Zellic

Prepared for
TempleDAO
TempleDAO

Prepared by
Katerina Belotskaia
Ayaz Mammadov
Zellic

January 26, 2024

Origami Finance

Smart Contract Security Assessment

Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Executive Summary

Zellic conducted a security assessment for TempleDAO from January 2nd to January 25th, 2024. During this engagement, Zellic reviewed Origami Finance's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.1. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- What unforeseen attack vectors could arise from users front- / back-running rebalances, and are the leverage restrictions sufficiently strict?
- Could the methods for handling rounding up/down issues potentially disadvantage the vault?
- How accurately does the protocol handle ERC-20 decimal conversions, and are there any risks associated with these conversions?

1.2. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

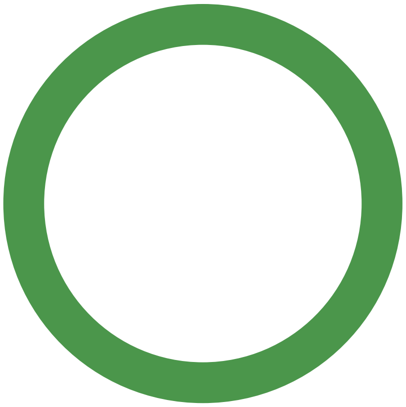
1.3. Results

During our assessment on the scoped Origami Finance contracts, we discovered three findings, all of which were low impact.

Additionally, Zellic recorded its notes and observations from the assessment for TempleDAO's benefit in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
🟥 Critical	0
🟧 High	0
🟨 Medium	0
🟩 Low	3
⬜ Informational	0



2. Introduction

2.1. About Origami Finance

Origami Finance is a protocol that provides targeted leverage for any whitelisted liquid-staking strategy through a simple vault UX. The flagship product is the Leveraged Origami Token Vault (lovToken), which connects users who wish to lever up on their favorite liquid-staking yield strategy and liquidity providers who will supply USDC to the oUSDC vault to earn interest for lending to one or more lovToken vaults. The lovToken vault will automatically lever up when its collateralization ratio is healthy and deleverage when that the same ratio deteriorates. The prevailing borrow APR for each lovToken vault is also dynamic and will fluctuate depending on the utilization of the debt ceiling set for each vault. To ensure capital efficiency, the USDC will only be lent out if the borrow APR paid by the vault exceeds the APY for the designated USDC idle strategy. Overall, the Origami protocol design is highly modular and composable to accommodate new liquid-staking strategies and new sources of liquidity to maximize returns.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards.

> We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

Origami Finance Contracts

Repository	https://github.com/TempleDAO/origami ↗
Version	origami: `9b23bec768bbc40d1dccde9c0d73dadf33aeec96`
Programs	• apps/protocol/contracts/common/* • apps/protocol/contracts/investments/OrigamiInvestment • apps/protocol/contracts/investments/OrigamiInvestment • apps/protocol/contracts/investments/OrigamiInvestmentVault • apps/protocol/contracts/investments/OrigamiOToken • apps/protocol/contracts/investments/OrigamiOTokenWithNative • apps/protocol/contracts/investments/lending/idleStrategy/* • apps/protocol/contracts/investments/lovToken/* • apps/protocol/contracts/investments/util/* • apps/protocol/contracts/libraries/*
Type	Solidity
Platform	EVM-compatible

2.4. Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of five and a half person-weeks. The assessment was conducted over the course of three calendar weeks.

Contact Information

The following project manager was associated with the engagement:

Chad McDonald
Engagement Manager
chad@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Katerina Belotskaia
Engineer
kate@zellic.io ↗

Ayaz Mammadov
Engineer
ayaz@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

January 2, 2024	Kick-off call
January 2, 2024	Start of primary review period
January 25, 2024	End of primary review period

3. Detailed Findings

3.1. Exit-fee arbitrage

Target	OrigamiLovToken		
Category	Business Logic	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

When a user exits the OrigamiLovToken, a percent of their reserves are kept as an exit fee. As such, the ratio of reserve to supply tokens is increased. This increases the price of the shares of the remaining users in the vault and can have undesired side effects.

```
function exitToToken(
    address /*account*/,
    IOrigamiInvestment.ExitQuoteData calldata quoteData,
    address recipient
) external virtual override onlyLovToken returns (
    uint256 toTokenAmount,
    uint256 toBurnAmount
) {
    ...
    // The entire amount of lovTokens will be burned
    // But only the non-fee portion is redeemed to reserves and sent to the
    user
    toBurnAmount = quoteData.investmentTokenAmount;
    uint256 reservesAmount = exitFeeRate.getRemainder(toBurnAmount);
    ...
    reservesAmount = _sharesToReserves(cache, reservesAmount);
    ...
}
```

```
  function exitToToken(
...
      if (lovTokenToBurn > 0) {
          _burn(address(_manager), lovTokenToBurn);
      }
  }
```

Impact

One potential form of arbitrage that is rendered possible by the increase in share price would be a just-in-time (JIT) liquidity arbitrage. This involves MEV front-running a user's `exitToToken` call, adding liquidity before the exit and removing liquidity after the exit. By doing this, the arbitrager can capture exit-fee funds that should be distributed to other pool holders. Consequently, the pool loses the opportunity to grow.

However, it is important to note that various factors make this very unlikely, as the asset/liability (A/L) limits prevent large entries and exits of the pool; otherwise, arbitrageurs risk losing funds on gas prices and the exit fee they will face when exiting.

Recommendations

Ensure that A/L limits are sufficiently strict and that exit-fee rates are sufficiently low as to render these attacks unprofitable. Otherwise, implement a slow distribution of exit-fee tokens, rendering these attacks impossible.

Remediation

The exit fee system has now been redesigned in commit 92874a88 ↗, changing dynamically with respect to the spot price against the historical price. The best case scenario for an exit arbitrageur is when the spot price is greater than the historical price, as this is when the deposit fee is the lowest (minFeeBps) and the exit fee bps is the highest. However, this fee is also capped to the possible prices that the oracle can return, furthermore reducing risk of such attacks during volatile moments.

Furthermore, it is very unlikely that a large enough exit is possible to render such an arbitrage profitable considering gas, A/L limits will be hit, daily circuit breakers will go off and the maximum dynamic fee possible as a consequence of the oracle price range checks.

3.2. Rebalance asset/liability slippage

Target	OrigamiLovTokenErc4626Manager		
Category	Business Logic	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

When a rebalance is performed, there is no guarantee that the asset/liability (A/L) ratio is changed as much as expected, as the A/L checks in the `rebalance` functions only verify that the A/L remains above the floor in case of `_rebalanceDown` and below the ceiling in the case of `_rebalanceUp`. This can be seen in the `_validateALRatio` function.

```
function _validateALRatio(Range.Data storage validRange,
    uint128 ratioBefore, uint128 ratioAfter, AlValidationMode alMode)
    internal virtual {
    if (alMode == AlValidationMode.LOWER_THAN_BEFORE) {
        // Check that the new A/L is not below the floor
        // In this mode, the A/L may be above the ceiling still, but should
    be decreasing
        // Note: The A/L may not be strictly decreasing in this mode since
    the liabilities (in reserve terms) is also
        // fluctuating
        if (ratioAfter < validRange.floor) revert ALTooLow(ratioBefore,
    ratioAfter, validRange.floor);
    } else {
        // Check that the new A/L is not above the ceiling
        // In this mode, the A/L may be below the floor still, but should be
    increasing
        // Note: The A/L may not be strictly increasing in this mode since
    the liabilities (in reserve terms) is also
        // fluctuating
        if (ratioAfter > validRange.ceiling) revert ALTooHigh(ratioBefore,
    ratioAfter, validRange.ceiling);
        }
}
```

Impact

The performance of the fund may be impacted as not as much leveraged interest is accumulated and the per-borrower interest rate might be set with the expectation of a certain A/L ratio being reached.

Recommendations

Implement a slippage check for the A/L range that verifies that the A/L has moved in the expected direction and that it is within a margin of the expected A/L.

Remediation

This was remediated in commit d3ed0724 ↗ by modifying the rebalance functions to accept a `params` parameter that contains new members such as `minNewAL` and `maxNewAL` which enforce the slippage of the A/L rebalance when calling `rebalanceDown` or `rebalanceUp`.

```
{
    if (alRatioAfter <= alRatioBefore) revert ALTooLow(alRatioBefore,
    alRatioAfter, alRatioBefore);
    if (alRatioAfter < params.minNewAL) revert ALTooLow(alRatioBefore,
    alRatioAfter, params.minNewAL);
    if (alRatioAfter > params.maxNewAL) revert ALTooHigh(alRatioBefore,
    alRatioAfter, params.maxNewAL);
}
```

3.3. Seed-deposit mispricing

Target	OrigamiAbstractLovTokenManager		
Category	Business Logic	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The first seed deposit into the lov vault is priced at a 1:1 ratio regardless of the current Dai/USDC price.

```
function _reservesToShares(Cache memory cache, uint256 reserves)
    private view returns (uint256) {
if (cache.totalSupply == 0) {
    return reserves;
}
```

However, when this seed deposit is removed, it is affected by the current `userRedeemableReserves`, which is impacted by the current Dai/USDC price provided by the oracle.

Impact

As a result, the withdrawal of the seed deposit could cause a loss of reserves for other users as the initial deposit is better priced than the market rate. Alternatively, it could cause a loss for the initial deposit, which received a worse rate than the market rate.

Recommendations

Ensure that the initial deposit will not be removed.

Remediation

The TempleDAO team has ensured that the initial deposit will not be removed.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. Possibles share-price inflation

The vault will be initialized with a seed deposit by the TempleDAO team, but in the event that it is not, there are possible pathways to an inflated share price. This could be enabled by several factors:

- Truncation caused by a change in the reserve/share, possibly due to fluctuations in the oracle price.
- Exit-fee donation — a user could repeatedly inflate the share price by entering and exiting, inflating the share price by 1.005x each time.

However, these conditions come with the prerequisite that the vault must initially be empty; otherwise, the attacker is only donating their reserves to other vault participants. This is not possible as the TempleDAO team will be seeding their vaults with a seed deposit.

Even in the case the vault's share price becomes inflated, the only attack rendered viable is a variant of the first deposit attack where the share price can be increased to a point to cause a truncation in a following victim's deposit such that a portion of the deposit is stolen. However, this is thwarted by slippage checks that revert if a user does not receive the expected amount of share back; therefore, it would only be viable if the share price were inflated and the victim did not correctly set their slippage parameter.

4.2. Lack of validation

The following outlines areas in the scoped contracts that lack validation.

- The `constructor` of multiple contracts lacks essential input-parameter validation, such as verifying that the addresses of the contracts are not zero addresses. This absence of validation could lead to the deployment of contracts with invalid configurations. In some contracts, the owner has the flexibility to update global variables. However, in the `reserveToken` contract, for example, the `reserveToken` address is declared as immutable, making it unchangeable after deployment.
- The `investWithToken()` and `exitToToken()` functions of the OrigamiInvestmentVault contract utilize `quoteData` generated by the `investQuote()` and `exitQuote()` functions, correspondingly. These functions generate `InvestQuoteData` or `ExitQuoteData` data, which include the `underlyingInvestmentQuoteData` field. It is assumed that this field duplicates and encodes all information present in other fields of the `quoteData`. For example, the `quote-`

Data.underlyingInvestmentQuoteData.investmentTokenAmount should be identical to quoteData.investmentTokenAmount, and so forth. However, the investWithToken() and exitToToken() functions do not explicitly validate these data duplications, which poses potential security risks.

For example, it is assumed that the reserveToken contract should utilize all tokens approved by the OrigamiInvestmentVault contract during the investment process. Therefore, if users try to use different token addresses for underlyingInvestmentQuoteData.fromToken and quoteData.fromToken calling the investWithToken() function, the transaction should revert because reserveToken.investWithToken() will not be able to transfer invested tokens from the OrigamiInvestmentVault contract. But if the quoteData is improperly prepared by a user, it can lead to the underutilization of approved tokens, and the reserveToken contract will have the excessive approval from the OrigamiInvestmentVault for the invested tokens.

This scenario presents an opportunity for exploitation by a malicious user. The user could leverage the excess tokens to conduct investments using their own quoteData, where the fields quoteData.underlyingInvestmentQuoteData.fromToken and quoteData.fromToken differ. This could result in the investment of arbitrary tokens into the OrigamiInvestmentVault contract, while approved tokens will be utilized in the reserveToken contract.

- The reservesVestingDuration from the RepricingToken contract determines how long in seconds the new reserve's funds will be under vesting. But the contract does not specify the maximum value to which the reservesVestingDuration can be limited. Therefore, a user with an access to the setReservesVestingDuration function can update this value to any up to the max(uint256).

Remediation

This issue has been acknowledged by TempleDAO, and a fix was implemented in commit 46423d0b ↗.

4.3. Exemplary codebase

We want to applaud the TempleDAO team for their exemplary codebase, which contains relevant documentation that is concise and succinct. They have an extensive testing suite with many unit tests compounded with a forked integration test based on already deployed contracts. They have also employed a series of fuzzing tests to ensure that important invariants (such as A/L checks that are maintained across rebases, circuit breaker checks, oracle stability checks, etc...) cannot be broken. On top of this, we would like to shine a light on their defensive programming, adding slippage checks on every relevant call; their attention to detail when rounding to avoid potential truncation issues; their novel use of circuit breakers, which limit daily volume and per-user volatility in case of unexpected events; and finally, their developer communication, explaining various mechanisms in simple fashion.

4.4. Potential price arbitrage

There is share-price movement when rebalancing happens. This is due to the change in liabilities that may increase/decrease the redeemable reserves, which affects the share price. As a result, entry and exit prices could be moved such that they may be profitable for an attacker to enter right before a rebalancing happens and then exit to make a profit. This is, however, not possible as the rebalancing bot will use flashbots protect and will have randomization; therefore, predicting the price movement becomes infeasible.

5. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

5.1. Module: LinearWithKinkInterestRateModel.sol

Function: `setRateParams(uint80 _baseInterestRate, uint80 _maxInterestRate, uint256 _kinkUtilizationRatio, uint80 _kinkInterestRate)`

This function allows the caller with access to update the rate parameters.

Inputs

- `_baseInterestRate`
  - **Constraints**: `_baseInterestRate` should be less than `_kinkInterestRate`.
  - **Impact**: Base interest rate, which is used for calculating the current interest rate in case `utilizationRatio` is less than or equal to the `kinkUtilizationRatio`.
- `_maxInterestRate`
  - **Constraints**: It should neither be more than `_kinkInterestRate` or `_baseInterestRate`.
  - **Impact**: This rate represents the interest rate applied when the utilization reaches 100%.
- `_kinkUtilizationRatio`
  - **Constraints**: It should neither be equal to zero nor less than `PRECISION`.
  - **Impact**: The utilization level at which the slope of the curve shifts. The current interest rate will be calculated using `_kinkInterestRate` when `utilizationRatio` is more than `_kinkUtilizationRatio`.
- `_kinkInterestRate`
  - **Constraints**: `_kinkInterestRate` should be less than `_maxInterestRate` and more than `_baseInterestRate`.
  - **Impact**: Interest rate at the `kinkUtilization`.

Branches and code coverage

Intended branches

- The new parameters have been set successfully
  - ☑ Test coverage

Negative behavior

- Non-whitelisted caller
    - ☑ Negative test
- Set invalid rate params
    - ☑ Negative test

5.2. Module: MintableToken.sol

Function: `addMinter(address account)`

Allows the caller who has access to the function to assign the address as the minter.

Inputs

- `account`
    - **Constraints**: No constraints.
    - **Impact**: The address will have an ability to mint/burn tokens.

Branches and code coverage

Intended branches

- The minter was successfully set
    - ☑ Test coverage

Negative behavior

- Non-whitelisted caller
    - ☑ Negative test

Function: `burn(address account, uint256 amount)`

Available only for minters. Allows to burn tokens from any account without limits.

Inputs

- `_to`
    - **Constraints**: `account != address(0)`.
    - **Impact**: The account from which tokens will be burned.
- `_amount`
    - **Constraints**: The `_to` account should have enough tokens to burn.
    - **Impact**: The number of tokens will be burned.

Branches and code coverage

Intended branches

- The minter was successfully burn tokens
  - ☑ Test coverage

Negative behavior

- Caller is not a minter.
  - ☑ Negative test
- `_to` contains fewer than the specified `amount` of tokens.
  - ☐ Negative test

Function: `mint(address _to, uint256 _amount)`

Available only for minters. Allows to mint new tokens without limit.

Inputs

- `_to`
  - **Constraints**: `account != address(0)`.
  - **Impact**: Receiver of new tokens.
- `_amount`
  - **Constraints**: No constraints.
  - **Impact**: The number of tokens will be minted.

Branches and code coverage

Intended branches

- The minter was successfully mint tokens
  - ☑ Test coverage

Negative behavior

- Caller is not a minter.
  - ☑ Negative test
- `_to` is zero address.
  - ☐ Negative test

Function: `recoverToken(address token, address to, uint256 amount)`

Allows the caller who has access to the function to transfer any tokens from the contract balance without restrictions.

Inputs

- `token`
    - **Constraints**: No constraints.
    - **Impact**: The address of the token that will be transferred — can be this contract address.
- `to`
    - **Constraints**: Not zero address.
    - **Impact**: The receiver of tokens.
- `amount`
    - **Constraints**: The contract should have enough amount of tokens.
    - **Impact**: The number of tokens will be transferred from the contract.

Branches and code coverage

Intended branches

- The tokens were successfully recovered
    - ☑ Test coverage

Negative behavior

- Non-whitelisted caller
    - ☑ Negative test

Function: `removeMinter(address account)`

Allows the caller who has access to the function to remove the address from the list of minters.

Inputs

- `account`
    - **Constraints**: No constraints.
    - **Impact**: The address will lose the ability to mint/burn tokens.

Branches and code coverage

Negative behavior

- Non-whitelisted caller
    - ☑ Negative test

5.3. Module: OrigamiAaveV3IdleStrategy.sol

Function: `allocate(uint256 amount)`

Allocates to Aave (internal).

Inputs

- `amount`
    - **Control**: Full.
    - **Constraints**: None.
    - **Impact**: Amount to supply Aave lending pool.

Branches and code coverage

Intended branches

- Allocate was successful
    ☑ Test coverage

Negative behavior

- Allocate zero amount
    ☑ Negative test

Function call analysis

- `SafeERC20.safeTransferFrom(this.asset, msg.sender, address(this), amount)`
    - **What is controllable?** `amount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** Discarded.
    - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this.lendingPool.supply(address(this.asset), amount, address(this), 0)`
    - **What is controllable?** `amount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** Discarded.
    - **What happens if it reverts, reenters or does other unusual control flow?** N/A.

Function: `withdraw(uint256 amount, address recipient)`

Withdraws from the Aave pool.

Inputs

- `amount`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Amount to withdraw.
- `recipient`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Recipient.

Branches and code coverage

Intended branches

- Withdraws from lending pool.
  - ☑ Test coverage

Negative behavior

- Verifies zero amount.
  - ☑ Negative test

Function call analysis

- `this.availableToWithdraw() -> this.aToken.balanceOf(address(this))`
  - **What is controllable?** Nothing.
  - **If the return value is controllable, how is it used and how can it go wrong?** Amount of aTokens.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this.availableToWithdraw() -> this.asset.balanceOf(address(this.aToken))`
  - **What is controllable?** Nothing.
  - **If the return value is controllable, how is it used and how can it go wrong?** Amount of reserve in Aave lending pool.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this.lendingPool.withdraw(address(this.asset), amount, recipient)`
  - **What is controllable?** `amount` and `recipient`.
  - **If the return value is controllable, how is it used and how can it go wrong?**

Amount to withdraw from lending pool.

- **What happens if it reverts, reenters or does other unusual control flow?** can revert if pool doesn't have enough funds to withdraw

5.4. Module: OrigamiAbstractLovTokenManager.sol

Function: `exitToToken(address, IOrigamiInvestment.ExitQuoteData quoteData, address recipient)`

The function can be called only from the lovToken contract. Allows investor to exit from the reserve token.

Inputs

- `quoteData.investmentTokenAmount`
  - **Constraints**: Is not validated here — all checks performed by the lovToken contract.
  - **Impact**: lovTokens will be burned from the manager account.
- `quoteData.toToken`
  - **Constraints**: There is verification that `toToken` can be equal to the `depositAsset` or `_reserveToken`; otherwise, the transaction will revert.
  - **Impact**: The `recipient` will receive these tokens in return — `toToken` should be an accepted ERC-20 token.
- `quoteData.maxSlippageBps`
  - **Constraints**: Is not used and is not validated.
  - **Impact**: The maximum allowed slippage of the `expectedToTokenAmount`.
- `quoteData.deadline`
  - **Constraints**: Is not used and is not validated.
  - **Impact**: N/A.
- `quoteData.expectedToTokenAmount`
  - **Constraints**: Is not used and is not validated.
  - **Impact**: N/A.
- `quoteData.minToTokenAmount`
  - **Constraints**: There is a check that `toTokenAmount` is not less than `quoteData.minToTokenAmount`.
  - **Impact**: The minimum amount of `toToken` to receive.
- `quoteData.underlyingInvestmentQuoteData`
  - **Constraints**: Is not used and is not validated.
  - **Impact**: N/A.
- `recipient`
  - **Constraints**: Cannot be zero address.
  - **Impact**: The receiver of the `toToken`.

Branches and code coverage

Intended branches

- `exitToToken` was successfully completed as expected.
  - ☑ Test coverage

Negative behavior

- Caller is not an approved lovToken contract.
  - ☑ Negative test
- The `toTokenAmount` is less than `minToTokenAmount`.
  - ☑ Negative test
- `toToken` is not supported.
  - ☑ Negative test
- `recipient` is zero address.
  - □ Negative test
- The pause state for exit is true
  - ☑ Negative test

Function call analysis

- `this.populateCache() -> this.liabilities() -> this.lendingClerk.borrowerDebt(address(this))`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** The value is used to calculate the debt converted to shares, which in turn is used for A/L ratio calculation.
  - **What happens if it reverts, reenters or does other unusual control flow?** The function returns the current debt of this manager contract. The balance of debt tokens can be changed only over `borrow/repay` function or by the minter of debt tokens, who can transfer debt tokens between accounts.
- `this.populateCache() -> this.liabilities() -> this._reserveToken.previewWithdraw(debtInDepositAsset)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns debt amount converted to the shares. The value is used for `cache.liabilities`.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this.populateCache() -> this.lovToken.totalSupply()`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns full number of minted lovTokens. The value is used for `cache.totalSupply`.

    - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this._assetToLiabilityRatio(cache) -> OrigamiMath.mulDiv(cache.assets, OrigamiAbstractLovTokenManager.PRECISION, cache.liabilities, Rounding.ROUND_DOWN)`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns current L/A ratio, which is calculated using total reserves balance `cache.assets`, which is equal to `reservesBalance()`, and debt shares, which is equal to the `cache.liabilities`.
    - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `BasisPointFraction.getRemainder(this.exitFeeRate, toBurnAmount)`
    - **What is controllable?** `toBurnAmount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the number of tokens minus the exit fee.
    - **What happens if it reverts, reenters or does other unusual control flow?** Can revert if `exitFeeRate` is more than `BASIS_POINTS_DIVISOR`.
- `this._sharesToReserves(cache, reservesAmount) -> OrigamiMath.mulDiv(shares, this._userRedeemableReserves(cache), cache.totalSupply, Rounding.ROUND_DOWN)`
    - **What is controllable?** `reservesAmount` == `shares`.
    - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
    - **What happens if it reverts, reenters or does other unusual control flow?** In case `cache.totalSupply` is not zero, the reserved amount will be calculated as follows: `shares.mulDiv(_userRedeemableReserves(cache), cache.totalSupply, OrigamiMath.Rounding.ROUND_DOWN);`.
- `this._sharesToReserves(cache, reservesAmount) -> this._userRedeemableReserves(cache)`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** Can return zero; if redeemable reserves are not available (`cache.assets` is less than liabilities with buffer), then the `_sharesToReserves` function result also will be equal to zero.
    - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this._redeemFromReserves(reservesAmount, quoteData.toToken, recipient) -> OrigamiLovTokenErc4626Manager._redeemFromReserves`
    - **What is controllable?** `quoteData.toToken` and `recipient`.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the amount of redeemed tokens.
    - **What happens if it reverts, reenters or does other unusual control flow?** The function will revert if `toToken` is neither `depositAsset` or `_reserve-`

Token. If equal to `_reserveToken`, `reservesAmount` of `_reserveToken` will be transferred to the `recipient`. Otherwise, the `reservesAmount` amount of tokens will be redeemed from the `_reserveToken` contract, the `reservesAmount` reserve tokens will be burned, and assets tokens will transferred to the `recipient`.

Function: `investWithToken(address account, IOrigamiInvestment.InvestQuoteData quoteData)`

The function can be called only from the LovToken contract, but the `LovToken:investWithToken`, which triggers this function, can be called by any user. The global `investmentsPaused` should not be true.

Inputs

- `account`
  - **Constraints**: If the global variable `allowAll` is true, the `account` is not validated. Otherwise, if the address is not a contract, the `allowedAccounts` should contain the `account` address.
  - **Impact**: The address of the caller of the `LovToken:investWithToken` function who initiated the invest process.
- `quoteData.fromToken`
  - **Constraints**: The address is validated inside the `_depositIntoReserves` function. If `fromToken` is equal to the `depositAsset`, then `depositAsset` will be deposited to the `_reserveToken`. If `fromToken` is equal to the `_reserveToken`, then its tokens are already deposited for the `_manager` contract; otherwise, the function will revert.
  - **Impact**: The token that will be invested.
- `quoteData.fromTokenAmount`
  - **Constraints**: Cannot be zero.
  - **Impact**: If `fromToken == depositAsset`, then `fromTokenAmount` tokens will be deposited to the `_reserveToken` contract using `deposit()`. If `fromToken == _reserveToken` and then `fromTokenAmount`, then tokens were already deposited.
- `quoteData.maxSlippageBps`
  - **Constraints**: Is not used and is not validated.
  - **Impact**: The maximum allowed slippage of the `expectedInvestmentAmount`.
- `quoteData.deadline`
  - **Constraints**: Is not used and is not validated.
  - **Impact**: The maximum deadline to execute the transaction.
- `quoteData.expectedInvestmentAmount`
  - **Constraints**: Is not used and is not validated.

    - **Impact**: The expected amount of this lovToken token to receive in return.
- `quoteData.minInvestmentAmount`
    - **Constraints**: There is a check that `investmentAmount` is not less than `quoteData.minInvestmentAmount`.
    - **Impact**: The minimum amount of lovToken to receive.
- `quoteData.underlyingInvestmentQuoteData`
    - **Constraints**: Is not used and is not validated.
    - **Impact**: Extra quote parameters.

Branches and code coverage

Intended branches

- `investWithToken` was successfully completed as expected.
    ☑ Test coverage

Negative behavior

- Caller is not approved lovToken contract.
    ☑ Negative test
- The `investmentAmount` is less than `minInvestmentAmount`.
    ☑ Negative test
- `fromToken` is not supported.
    ☑ Negative test
- The pause state for invest is true
    ☑ Negative test
- `quoteData.fromTokenAmount` is zero
    ☑ Negative test

Function call analysis

- `this.populateCache() -> this.liabilities() -> this.lendingClerk.borrowerDebt(address(this))`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** The value is used to calculate the debt converted to shares, which in turn is used for A/L ratio calculation.
    - **What happens if it reverts, reenters or does other unusual control flow?** The function returns current debt of this manager contract. The balance of debt tokens can be changed only over `borrow/repay` function or by the minter of debt tokens, who can transfer debt tokens between accounts.
- `this.populateCache() -> this.liabilities() -> this._reserveToken.previewWithdraw(debtInDepositAsset)`
    - **What is controllable?** N/A.

- **If the return value is controllable, how is it used and how can it go wrong?** Returns debt amount converted to the shares. The value is used for `cache.liabilities`.
- **What happens if it reverts, reenters or does other unusual control flow?** N/A.

- `this.populateCache() -> this.lovToken.totalSupply()`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns full number of minted lovTokens. The value is used for `cache.totalSupply`.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this._assetToLiabilityRatio(cache) -> OrigamiMath.mulDiv(cache.assets, OrigamiAbstractLovTokenManager.PRECISION, cache.liabilities, Rounding.ROUND_DOWN)`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns current L/A ratio, which is calculated using total reserves balance `cache.assets`, which is equal to `reservesBalance()`, and debt shares, which is equal to the `cache.liabilities`.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this._depositIntoReserves(quoteData.fromToken, quoteData.fromTokenAmount) -> OrigamiLovTokenErc4626Manager._depositIntoReserves`
  - **What is controllable?** `quoteData.fromToken` and `quoteData.fromTokenAmount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the number invested reserves.
  - **What happens if it reverts, reenters or does other unusual control flow?** If `fromToken == _reserveToken`, the function will return `quoteData.fromTokenAmount`. If `fromToken == depositAsset`, the function will return the result of the `_reserveToken.deposit()` function; otherwise, this function will revert.
- `this._reservesToShares(cache, newReservesAmount)`
  - **What is controllable?** `newReservesAmount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** If `cache.totalSupply == 0`, the function will return `newReservesAmount`, so `_reservesToShares` will be 1:1.
  - **What happens if it reverts, reenters or does other unusual control flow?** Calculate the number of shares for the corresponding `reserves`, `cache.totalSupply`, and `_redeemableReserves` amounts.

Function: `setExitFeeRate(uint256 _exitFeeRate)`

Sets the percentage of the exit fee. This percentage of tokens will be burned during the exit process, but it will not be included in the calculation of the amount of tokens received.

Inputs

- `_exitFeeRate`
  - **Constraints**: Cannot be more than `BasisPointFraction.BASIS_POINTS_DIVISOR`.
  - **Impact**: During the exit process, a certain percentage of tokens will be burned and excluded from the calculation of tokens received.

Branches and code coverage

Intended branches

- `exitFeeRate` was updated properly.
  - ☑ Test coverage

Negative behavior

- Caller has no access to this function.
  - ☑ Negative test
- `_exitFeeRate` is more than `BasisPointFraction.BASIS_POINTS_DIVISOR`.
  - ☑ Negative test

Function: `setRebalanceALRange(uint128 floor, uint128 ceiling)`

Allows to update `rebalanceALRange` by lower and upper bounds of A/L. These values are used by the `_validateALRatio` function during the `_rebalanceUp/_rebalanceDown` process so that the new A/L is still within the `rebalanceALRange`.

Inputs

- `floor`
  - **Constraints**: `floor` should be more than 1e18.
  - **Impact**: The lower bounds of A/L.
- `ceiling`
  - **Constraints**: N/A.
  - **Impact**: The upper bounds of A/L.

Branches and code coverage

Intended branches

- `rebalanceALRange` was updated properly.
  - ☑ Test coverage

Negative behavior

- Caller has no access to this function.
  - ☑ Negative test
- Floor is more than the ceiling.
  - ☑ Negative test

Function: `setRedeemableReservesBuffer(uint256 buffer)`

Updates `redeemableReservesBuffer`, which is defined as 100% plus `buffer`. When the user will initiate the exit process, this `redeemableReservesBuffer` will be used to calculate the available exit reserves amount.

Inputs

- `buffer`
  - **Constraints**: Cannot be more than `BasisPointFraction.BASIS_POINTS_DIVISOR`.
  - **Impact**: This percent of debt will be also held in addition to debt amount.

Branches and code coverage

Intended branches

- `redeemableReservesBuffer` was updated properly.
  - ☑ Test coverage

Negative behavior

- Caller has no access to this function.
  - ☑ Negative test
- `buffer` is more than `BasisPointFraction.BASIS_POINTS_DIVISOR`.
  - ☑ Negative test

Function: `setUserALRange(uint128 floor, uint128 ceiling)`

Allows to update `userALRange` by lower and upper bounds of A/L when users deposit/exit into lovToken. These values are used by the `_validateALRatio` function during the `investWithTo-`

`ken/exitToToken` process to validate A/L changes.

Inputs

- `floor`
  - **Constraints**: `floor` should be more than 1e18.
  - **Impact**: The lower bounds of A/L.
- `ceiling`
  - **Constraints**: N/A.
  - **Impact**: The upper bounds of A/L.

Branches and code coverage

Intended branches

- `userALRange` was updated properly.
  ☑ Test coverage

Negative behavior

- Caller has no access to this function.
  ☑ Negative test
- Floor is more than the ceiling.
  ☑ Negative test

5.5. Module: OrigamiCircuitBreakerAllUsersPerPeriod.sol

Function: `preCheck(address, uint256 amount)`

The function can be called only by a proxy contract, which is set during deployment and cannot be changed. Allows to verify that the new amount does not exceed the cap in the ongoing period. The function is invoked through a proxy `OrigamiCircuitBreakerProxy.preCheck()` function, which is, in turn, called by `OrigamiLendingClerk.borrow()` and `OrigamiLendingSupplyManager.exitToToken()`.

Inputs

- `amount`
  - **Constraints**: The number of tokens to be checked should not exceed the overall limit.
  - **Impact**: If this value does not exceed the cap, it will be added to the current time bucket for future cap verification.

Branches and code coverage

Intended branches

- The previous bucket was reset as expected.
  - ☑ Test coverage
- The current bucket was updated properly.
  - ☑ Test coverage

Negative behavior

- Caller is not proxy contract.
  - □ Negative test
- The amount exceeds the cap.
  - ☑ Negative test

Function: `setConfig(uint32 _periodDuration, uint32 _nBuckets, uint128 _cap)`

This function allows the caller access to set new values for `periodDuration`, `nBuckets`, and `cap`. Subsequently, `secondsPerBucket` will be recalculated based on the new values of `periodDuration` and `nBuckets`. Additionally, `bucketIndex` will be reset to zero, and all necessary buckets will be cleared.

Inputs

- `_periodDuration`
  - **Constraints**: `_periodDuration % _nBuckets` should be zero.
  - **Impact**: Borrowing within a `_periodDuration` window is limited to no more than the `cap`.
- `_nBuckets`
  - **Constraints**: `_nBuckets` should be less than `MAX_BUCKETS`.
  - **Impact**: The number of buckets into which the `periodDuration` should be divided.
- `_cap`
  - **Constraints**: No constraints.
  - **Impact**: The maximum allowed amount to be borrowed within each period.

Branches and code coverage

Intended branches

- Verify that the buckets have been reset correctly.
  - ☑ Test coverage
- `periodDuration`, `nBuckets`, and `cap` were updated.

☑ Test coverage

Negative behavior

- Non-whitelisted caller
  ☑ Negative test
- `_periodDuration % _nBuckets > 0`.
  ☑ Negative test
- `_nBuckets` more than `MAX_BUCKETS`'.
  ☑ Negative test

Function: `updateCap(uint128 newCap)`

This function allows the caller access to set new values for `cap`.

Inputs

- `newCap`
  - **Constraints**: No constraints.
  - **Impact**: The maximum allowed amount to be borrowed within each period.

Branches and code coverage

Intended branches

- The `cap` was updated.
  ☑ Test coverage

Negative behavior

- Non-whitelisted caller
  ☑ Negative test

5.6. Module: OrigamiCircuitBreakerProxy.sol

Function: `preCheck(address token, address onBehalfOf, uint256 amount)`

Performs a circuit breaker check, calling the correct circuit breaker.

Inputs

- `token`
  - **Control**: Full.
  - **Constraints**: None.

  - **Impact**: The token that is being transferred.
- `onBehalfOf`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Who is doing the transferring.
- `amount`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Amount being transferred.

Branches and code coverage

Negative behavior

- Fails if an empty mapping is called.
  - ☑ Negative test

Function call analysis

- `this.circuitBreakers[_identifier][token].preCheck(onBehalfOf, amount)`
  - **What is controllable?** Nothing.
  - **If the return value is controllable, how is it used and how can it go wrong?** Discarded.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.

5.7. Module: OrigamiCrossRateOracle.sol

Function: `latestPrice(OrigamiMath.Rounding roundingMode)`

Gets the latest price (rounded in ideal correction and scaled to 1e18).

Inputs

- `roundingMode`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: The direction to round.

Branches and code coverage

Intended branches

- Direction to round is correctly followed.
  - ☑ Test coverage
- Exchange rate is scaled to 1e18.
  - ☑ Test coverage

Negative behavior

- Verifies floor and ceiling requirements.
  - ☑ Negative test

Function: `_chainlinkPrice(IAggregatorV3Interface oracle, uint256 scalar, bool scaleDown, uint256 stalenessThreshold, OrigamiMath.Rounding roundingMode)`

Gets latest Chainlink price.

Inputs

- `oracle`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: The oracle to use.
- `scalar`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Amount to scale price by.
- `scaleDown`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Scale direction.
- `stalenessThreshold`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Freshness threshold of the oracle price.
- `roundingMode`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Rounding direction.

Branches and code coverage

Intended branches

- Scales and rounds in specified directions.
  - ☑ Test coverage

Negative behavior

- Verify that the response is fresh enough.
  - ☑ Negative test

5.8. Module: OrigamiDebtToken.sol

Function: `burnAll(address _debtor)`

Available only for minters. Allows to reset the full debt (`principal` + `interest`) of the `_debtor` account.

Branches and code coverage

Intended branches

- The `_debtor`'s debt is zero.
  - ☑ Test coverage

Negative behavior

- Caller is not a minter.
  - ☑ Negative test
- `_debtor` is zero address
  - ☑ Negative test

Function: `burn(address _debtor, uint256 _burnAmount)`

Available only for minters. Allows to burn debt tokens from any `_debtor` account.

Inputs

- `_debtor`
  - **Constraints**: `_debtor != address(0)`.
  - **Impact**: The debtor account from which debt tokens will be burned.
- `_burnAmount`
  - **Constraints**: Cannot be zero.
  - **Impact**: The number of debt tokens will be burned.

Branches and code coverage

Intended branches

- The `burn` was executed properly
  - ☑ Test coverage

Negative behavior

- Caller is not a minter.
  - ☑ Negative test
- The `_burnAmount` is invalid
  - ☑ Negative test
- The `_debtor` is zero address
  - ☑ Negative test
- `burn` zero amount
  - ☑ Negative test

Function: `checkpointDebtorsInterest(address[] _debtors)`

Updates the current interest for `_debtors` accounts. The `interestCheckpoint` will be updated only for debtors for whom time has passed since the last `timeCheckpoint`. Also, global `estimatedTotalInterest` will be increased by total interest.

Function: `mint(address _debtor, uint256 _mintAmount)`

Available only for minters. Allows to add a new debt position to any `_debtor` account. The debtor's `principal` will be increased by `amount`, and also `totalPrincipal` will be increased.

Inputs

- `_debtor`
  - **Constraints**: Cannot be zero address.
  - **Impact**: The receiver of debt tokens.
- `_mintAmount`
  - **Constraints**: No constraints.
  - **Impact**: The debt amount.

Branches and code coverage

Intended branches

- The `_debtor`'s `principal` was increased.
  - ☑ Test coverage

- totalPrincipal was updated properly.
  - ☑ Test coverage

Negative behavior

- Caller is not a minter.
  - ☑ Negative test
- _mintAmount is zero.
  - ☑ Negative test
- _debtor address is zero.
  - ☑ Negative test

Function: `recoverToken(address token, address to, uint256 amount)`

Allows the caller who has access to the function to transfer any tokens from the contract balance without restrictions.

Inputs

- token
  - **Constraints**: No constraints.
  - **Impact**: The address of the token that will be transferred — can be this contract address.
- to
  - **Constraints**: Not zero address.
  - **Impact**: The receiver of tokens.
- amount
  - **Constraints**: The contract should have enough amount of tokens.
  - **Impact**: The number of tokens will be transferred from the contract.

Branches and code coverage

Negative behavior

- Non-whitelisted caller
  - ☑ Negative test

Function: `setInterestRate(address _debtor, uint96 _rate)`

Available only for minters or callers with an access to this function. Allows to update the compounding interest rate for a debtor. Before the update of debtor.rate, the _debtor.interestCheckpoint will be updated by the current rate, and only after, the new one will be set.

Inputs

- `_debtor`
    - **Constraints**: No constraints.
    - **Impact**: The `_debtor` for which the interest rate will be updated.
- `_rate`
    - **Constraints**: `_rate` cannot be more than `MAX_INTEREST_RATE`.
    - **Impact**: The rate at which interest will be accrued to the debtor.

Branches and code coverage

Intended branches

- Validate that `interestCheckpoint` was updated using an old rate.
    ☑ Test coverage
- The rate was set properly.
    ☑ Test coverage

Negative behavior

- Non-whitelisted caller
    ☑ Negative test
- The `rate` is invalid
    ☑ Negative test

Function: `transferFrom(address from, address to, uint256 amount)`

Available only for minters, so accounts with debt cannot freely transfer debt tokens. Also, minters do not need to have an approve to transfer debt tokens from any accounts. The debt tokens will be removed from the debtor. If the debtor's debt is less than `amount`, the transaction will revert. At first, the debtor's interest will be covered by `amount` and the rest of the tokens will decrease the `principal`. Then, the same amount of debt tokens will be added to the `to` debtor position, the debtor's `principal` will be increased by `amount`, and `totalPrincipal` will be increased.

Branches and code coverage

Intended branches

- `totalPrincipal` was updated properly.
    ☑ Test coverage
- The `from`'s `principal` and `interestCheckpoint` were updated properly.
    ☑ Test coverage
- the `to`'s `principal` was increased.
    ☑ Test coverage

Negative behavior

- Caller is not a minter.
  - ☑ Negative test
- `from` account does not have debt.
  - ☑ Negative test
- `to` is zero address.
  - ☑ Negative test
- `from` is zero address.
  - ☑ Negative test
- `amount` is zero.
  - ☑ Negative test

Function: `transfer(address to, uint256 amount)`

Available only for minters, so accounts with debt cannot freely transfer debt tokens. Also, minters do not need to have an approve to transfer debt tokens from any accounts. The debt tokens will be burned from the debtor. If the debtor's debt is less than `amount`, the transaction will revert. At first, the debtor's interest will be covered by `amount` and the rest of the tokens will decrease the `principal`. Then, the same amount of debt tokens will be minted to the `to` debtor, the debtor's `principal` will be increased by `amount`, and `totalPrincipal` will be increased.

Branches and code coverage

Intended branches

- `totalPrincipal` was updated properly.
  - ☑ Test coverage
- The `msg.sender`'s `principal` and `interestCheckpoint` were updated properly.
  - ☑ Test coverage
- The `to`'s `principal` was increased.
  - ☑ Test coverage

Negative behavior

- Caller is not a minter.
  - ☑ Negative test
- `msg.sender` does not have debt.
  - ☑ Negative test
- `to` is zero address.
  - ☑ Negative test
- `amount` is zero.
  - ☑ Negative test

5.9. Module: OrigamiElevatedAccessBase.sol

Function: `setExplicitAccess(address allowedCaller, ExplicitAccess[] access)`

Sets the access for a certain caller.

Inputs

- `allowedCaller`
  - **Control**: Full.
  - **Constraints**: `!= 0`.
  - **Impact**: The caller to be given the access.
- `access`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: The function selector to allow/deny.

Branches and code coverage

Intended branches

- Sets the specified access.
  - ☑ Test coverage

Negative behavior

- Does not allow zero address.
  - ☑ Negative test

5.10. Module: OrigamiIdleStrategyManager.sol

Function: `allocate(uint256 amount)`

Allocates funds to the idleStrategy.

Inputs

- `amount`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: The amount to allocate to the idle strategy.

Branches and code coverage

Intended branches

- Idle strategy funds increased.
  - ☑ Test coverage

Function call analysis

- `SafeERC20.safeTransferFrom(this.asset, msg.sender, address(this), amount)`
  - **What is controllable?** `amount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** Discarded.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this.asset.balanceOf(address(this))`
  - **What is controllable?** Nothing.
  - **If the return value is controllable, how is it used and how can it go wrong?** Amount of assets pulled in.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this.idleStrategy.allocate(underlyingAllocation)`
  - **What is controllable?** Nothing.
  - **If the return value is controllable, how is it used and how can it go wrong?** Discarded.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.

Function: `withdraw(uint256 amount, address recipient)`

Withdraw funds from an idle strategy, accounting for the buffer to avoid small withdraw.

Inputs

- `amount`
  - **Control**: None, `lendingClerk` only.
  - **Constraints**: `!= 0`.
  - **Impact**: Amount to withdraw.
- `recipient`
  - **Control**: Full.
  - **Constraints**: `!= 0`.
  - **Impact**: Recipient.

Branches and code coverage

Intended branches

- Funds are withdrawn from the idle strategy.
  - ☑ Test coverage

Negative behavior

- Revert if not enough balance is withdrawn.
  - ☑ Negative test
- `!= 0` checks are respected.
  - ☑ Negative test

Function call analysis

- `this.asset.balanceOf(address(this))`
  - **What is controllable?** Nothing.
  - **If the return value is controllable, how is it used and how can it go wrong?** Prewithdrawal balance.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `_idleStrategy.withdraw(withdrawnFromIdleStrategy, address(this))`
  - **What is controllable?** Nothing.
  - **If the return value is controllable, how is it used and how can it go wrong?** Amount pulled out.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `SafeERC20.safeTransfer(this.asset, recipient, amount)`
  - **What is controllable?** `recipient`.
  - **If the return value is controllable, how is it used and how can it go wrong?** Nothing.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.

5.11. Module: OrigamiInvestmentVault.sol

Function: `exitToToken(ExitQuoteData quoteData, address recipient)`

Allows users to withdraw investment funds from the Origami investment vault.

### Inputs

- `quoteData.investmentTokenAmount`
  - **Constraints**: Cannot be zero, and the caller must own a sufficient amount.
  - **Impact**: The amount of shares to sell. Shares will be burned from the caller

account.

- `quoteData.toToken`
    - **Constraints**: Can be equal to the `reserveToken` or approved ERC-20.
    - **Impact**: The `recipient` will receive these tokens in return — `toToken` should be an accepted ERC-20 token or `reserveToken`.
- `quoteData.underlyingInvestmentQuoteData`
    - **Constraints**: No constraints.
    - **Impact**: Extra quote parameters that will be provided to the `reserveToken.exitToToken()`.
- `recipient`
    - **Constraints**: Cannot be zero address.
    - **Impact**: The receiver of the `toToken`.

Branches and code coverage

Negative behavior

- Verify that result does not depend on user's `underlyingQuoteData.underlyingExitQuoteData.investmentTokenAmount` (the user controls this field, but it will be overwritten by the function thus, the user's value should not affect the result).
    - □ Negative test
- `toToken` is not approved.
    - □ Negative test
- The caller owns fewer than `quoteData.investmentTokenAmount` shares tokens.
    - ☑ Negative test

Function call analysis

- `this._redeemReservesFromShares(quoteData.investmentTokenAmount, msg.sender, quoteData.minToTokenAmount, recipient)`
    - **What is controllable?** `quoteData.investmentTokenAmount`, `quoteData.minToTokenAmount`, and `recipient`.
    - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
    - **What happens if it reverts, reenters or does other unusual control flow?** The function converts user's shares amount to reserve tokens and burns this `sharesAmount` from the user balance. If `quoteData.toToken == reserveToken`, this contract will transfer `reserveToken` to receiver. Also, the function performs a slippage check that `reserveTokenAmount` is not less than expected `minReserveTokenAmount`.
- `IOrigamiInvestment(this.reserveToken).exitToToken(underlyingQuoteData.underly recipient)`
    - **What is controllable?** `underlyingQuoteData.underlyingExitQuoteData`

and `recipient`.
- **If the return value is controllable, how is it used and how can it go wrong?** Returns the number of tokens received by the recipient. If `toTokenAmount < quoteData.minToTokenAmount`, the function will revert.
- **What happens if it reverts, reenters or does other unusual control flow?** The function can revert if `underlyingQuoteData.underlyingExitQuoteData.toToken` is not accepted by the `reserveToken` contract.

Function: `investWithToken(InvestQuoteData quoteData)`

Allows users to invest `reserveToken` or approved ERC-20 tokens. If the global variable `allowAll` is true, the `msg.sender` is not validated. Otherwise, if the address is not a contract, the `allowedAccounts` should contain the `msg.sender` address. In exchange, `msg.sender` will receive the Origami investment tokens.

Inputs

- `quoteData.fromToken`
  - **Constraints**: It can be `reserveToken` or approved ERC-20 token to invest to `reserveToken` contract.
  - **Impact**: If `quoteData.fromToken == reserveToken`, then `reserveToken` is enough to just transfer `fromTokenAmount` to this contract from `msg.sender`. Otherwise, `fromToken` will be transferred to this contract, and after that, invested to the `reserveToken` contract.
- `quoteData.fromTokenAmount`
  - **Constraints**: Cannot be equal to zero.
  - **Impact**: The amount of tokens to invest.
- `quoteData.underlyingInvestmentQuoteData`
  - **Constraints**: N/A.
  - **Impact**: Extra quote parameters that will be provided to the `reserveToken.investWithToken()` function for investing `fromToken` tokens to the `reserveToken` contract.

Branches and code coverage

Negative behavior

- Non-whitelisted caller
  - ☑ Negative test
- `quoteData.fromToken` is not an approved token address.
  - ☐ Negative test
- The caller does not own enough `fromToken` tokens.

  - ☑ Negative test
- `investmentAmount` is less than `quoteData.minInvestmentAmount`.
  - ☑ Negative test
- `quoteData.fromToken` and `quoteData.underlyingInvestmentQuoteData.fromToken` are different.
  - □ Negative test
- `quoteData.fromTokenAmount` is less than `quoteData.underlyingInvestmentQuoteData.fr`
  - □ Negative test

Function call analysis

- `SafeERC20.safeTransferFrom(IERC20(this.reserveToken), msg.sender, address(this), reservesAmount)`
  - **What is controllable?** `reservesAmount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** No return value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Can revert if `msg.sender` does not have enough `reserveToken` to transfer. The function `investWithToken` has non-reentrant modifier.
- `SafeERC20.safeTransferFrom(IERC20(quoteData.fromToken), msg.sender, address(this), quoteData.fromTokenAmount)`
  - **What is controllable?** `quoteData.fromToken` and `quoteData.fromTokenAmount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** No return value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Can revert if `msg.sender` does not have enough `fromToken` to transfer. The function `investWithToken` has non-reentrant modifier.
- `IOrigamiInvestment(this.reserveToken).investWithToken(underlyingQuoteData)`
  - **What is controllable?** `underlyingQuoteData`.
  - **If the return value is controllable, how is it used and how can it go wrong?** Return amount of received tokens in exchange of invested tokens.
  - **What happens if it reverts, reenters or does other unusual control flow?** Can revert if `quoteData.underlyingInvestmentQuoteData.fromToken` is not an approved token address. Also revert if `quoteData.underlyingInvestmentQuoteData.fromTokenAmount` is more than `quoteData.fromTokenAmount`.
- `this._issueSharesFromReserves(reservesAmount, msg.sender, quoteData.minInvestmentAmount)`
  - **What is controllable?** `reservesAmount` and `quoteData.minInvestmentAmount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** Return the minted shares amount — in case it is less than `minSharesAmount`, the function will revert.

- **What happens if it reverts, reenters or does other unusual control flow?** N/A.

Function: `setPerformanceFee(uint256 _performanceFee)`

This function allows the caller with access to update the vault performance fee.

Inputs

- `_performanceFee`
  - **Constraints**: Should not be more than `BasisPointFraction.BASIS_POINTS_DIVISOR`.
  - **Impact**: This value is used to calculate the amount that the protocol takes from harvested rewards prior to their compounding into reserves.

Branches and code coverage

Negative behavior

- Non-whitelisted caller
  - ☑ Negative test

5.12. Module: OrigamiLendingClerk.sol

Function: `addBorrower(address borrower, address interestRateModel, uint256 debtCeiling)`

Allows the caller who has access to this function to add new borrower and borrow configuration for them.

Inputs

- `borrower`
  - **Constraints**: Cannot be zero address — should not be already added.
  - **Impact**: The borrower address.
- `interestRateModel`
  - **Constraints**: Cannot be zero address.
  - **Impact**: The address of the interest rate model; the `calculateInterestRate` function of the contract returns the latest borrower-specific interest rate.
- `debtCeiling`
  - **Constraints**: No constraints.

- **Impact**: The debt limit — if the debt balance of the borrower is more than `debtCeiling`, the `_availableToBorrow` returns zero.

Branches and code coverage

Intended branches

- `borrowerConfig` was updated.
  - ☑ Test coverage
- New borrower was added.
  - ☑ Test coverage

Negative behavior

- Non-whitelisted caller
  - ☑ Negative test
- Borrower was already added.
  - ☑ Negative test
- `borrower` is zero address
  - ☑ Negative test
- `interestRateModel` is zero address
  - ☑ Negative test

Function: `borrowMax(address recipient)`

The same as the `borrow` function, but `_borrow` is called with the full available amount that can be borrowed.

Function: `borrow(uint256 amount, address recipient)`

Allows approved borrower to borrow funds. If borrower is not approved, the `_getBorrowerConfig` function reverts because `msg.sender` does not pass verification. Also, function reverts if `globalBorrowPaused` is true or borrow is paused for `msg.sender`. The debt tokens will be transferred to the `msg.sender`, and lent funds will be transferred to the recipient.

Inputs

- `amount`
  - **Constraints**: The `circuitBreakerProxy.preCheck` function will revert if new utilization ratio is more than capacity.
  - **Impact**: Amount to borrow.
- `recipient`
  - **Constraints**: Should not be zero address.

- **Impact**: Receiver of lending funds.

Branches and code coverage

Negative behavior

- `msg.sender` is not an approved as a trusted borrower.
  - ☑ Negative test
- `amount` exceeds available borrow amount.
  - ☑ Negative test
- the pause state is true
  - ☑ Negative test

Function call analysis

- `circuitBreakerProxy.preCheck(address(asset),msg.sender,borrowAmount)`
  - **What is controllable?** `borrowAmount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters or does other unusual control flow?** Revert if new utilization ratio exceeds the capacity.
- `debtToken.safeTransferFrom(address(idleStrategyManager), borrower, borrowAmount.scaleUp(_assetScalar))`
  - **What is controllable?** `borrowAmount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters or does other unusual control flow?** No problems.
- `idleStrategyManager.withdraw(borrowAmount, recipient);`
  - **What is controllable?** `borrowAmount` and `recipient`.
  - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters or does other unusual control flow?** Can revert if there are not enough idle funds for lending.

Function: `deposit(uint256 amount)`

Allows SupplyManager to deposit `asset` tokens. These tokens will be allocated to the trusted strategy contract, and the appropriate amount of `debtToken` tokens will be minted for IdleStrategyManager.

Inputs

- `amount`
    - **Constraints**: No constraints.
    - **Impact**: The amount to deposit.

Branches and code coverage

Intended branches

- The `deposit` was executed properly.
    ☑ Test coverage

Negative behavior

- Caller is not SupplyManager.
    ☑ Negative test

Function call analysis

- `asset.safeTransferFrom(msg.sender, address(this), amount)`
    - **What is controllable?** `amount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
    - **What happens if it reverts, reenters or does other unusual control flow?** Transfer invested assets from SupplyManager to this contract. Can revert if callers do not own enough asset tokens, but this is unlikely, since previous contracts in the call chain verify that enough tokens have been provided by the user.
- `idleStrategyManager.allocate(amount)`
    - **What is controllable?** `amount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
    - **What happens if it reverts, reenters or does other unusual control flow?** Transfer asset tokens from this contract to IdleStrategyManager contract.
- `debtToken.mint(address(idleStrategyManager), amount.scaleUp(_assetScalar))`
    - **What is controllable?** `amount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
    - **What happens if it reverts, reenters or does other unusual control flow?** No problem.

Function: `recoverToken(address token, address to, uint256 amount)`

Allows the caller who has access to the function to transfer any tokens from the contract balance without restrictions.

Inputs

- `token`
  - **Constraints**: No constraints.
  - **Impact**: The address of the token that will be transferred — can be this contract address.
- `to`
  - **Constraints**: Not zero address.
  - **Impact**: The receiver of tokens.
- `amount`
  - **Constraints**: The contract should have enough amount of tokens.
  - **Impact**: The number of tokens will be transferred from the contract.

Branches and code coverage

Negative behavior

- Non-whitelisted caller
  - ☑ Negative test

Function: `repay(uint256 amount, address borrower)`

Allows an approved borrower to pay down debt if `globalRepayPaused` is not paused and repay for `borrower` is not paused.

Inputs

- `amount`
  - **Constraints**: Cannot be more than the borrower debt token balance.
  - **Impact**: The amount to repay.
- `borrower`
  - **Constraints**: Should be an approved borrower.
  - **Impact**: The borrower to repay.

Branches and code coverage

Intended branches

- The `repay` was executed properly
  - ☑ Test coverage

Negative behavior

- `borrower` address is not an approved borrower.
  - ☑ Negative test
- `amount` is more than available.
  - □ Negative test
- `msg.sender` does not own enough asset tokens to return debt.
  - □ Negative test
- `repay` is paused
  - ☑ Negative test

Function call analysis

- `debtToken.safeTransferFrom(borrower, address(idleStrategyManager), debtToTransfer)`
  - **What is controllable?** `borrower`.
  - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters or does other unusual control flow?** Transfer debt tokens from borrower to strategy manager.
- `asset.safeTransferFrom(from, address(this), repayAmount);`
  - **What is controllable?** `repayAmount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters or does other unusual control flow?** Transfer lent funds from `msg.sender` to this contract.

Function: `setBorrowerDebtCeiling(address borrower, uint256 newDebtCeiling)`

Allows the caller who has access to this function to update `debtCeiling` for borrower. This also initiates the update process of borrower's interest rate as a result of changing the current borrower's utilization ratio.

Inputs

- `borrower`
  - **Constraints**: If `_borrowersSet` does not contain `borrower`, the `_getBorrowerConfig` function reverts.
  - **Impact**: The address of existing borrower.

- `newDebtCeiling`
    - **Constraints**: No constraints.
    - **Impact**: The debt limit — if the debt balance of the borrower is more than `debtCeiling`, the `_availableToBorrow` returns zero. Also, it affects the borrower's utilization ratio.

Branches and code coverage

Intended branches

- New borrower's interest rate was calculated properly.
    - ☑ Test coverage
- Borrower's `debtCeiling` was updated properly.
    - ☑ Test coverage

Negative behavior

- Non-whitelisted caller
    - ☑ Negative test
- The borrower has not been added before.
    - ☑ Negative test

Function: `setBorrowerInterestRateModel(address borrower, address interestRateModel)`

Allows the caller who has access to this function to update the interest rate model contract address for the borrower.

Inputs

- `borrower`
    - **Constraints**: If `_borrowersSet` does not contain `borrower`, the `_getBorrowerConfig` function reverts.
    - **Impact**: The address of the existing borrower.
- `interestRateModel`
    - **Impact**: The address of the interest rate model — the `calculateInterestRate` function of the `interestRateModel` contract returns the latest interest rate calculated using the current utilization ratio.

Branches and code coverage

Intended branches

- New borrower's interest rate was calculated according to the new model.

- ☑ Test coverage

Negative behavior

- Non-whitelisted caller
  - ☑ Negative test
- The borrower has not been added before.
  - ☑ Negative test
- The `interestRateModel` is zero address.
  - ☑ Negative test

Function: `setBorrowerPaused(address borrower, bool pauseBorrow, bool pauseRepay)`

Allows the caller who has access to this function to set borrower's states `borrowPaused` and `repayPaused`.

Inputs

- `borrower`
  - **Constraints**: No constraints.
  - **Impact**: For this `borrower` address, the `borrowPaused` and `repayPaused` states will be set.
- `pauseBorrow`
  - **Constraints**: No constraints.
  - **Impact**: Change the `borrowPaused` state for `borrower`. If `globalBorrowPaused` is true, the `borrow` action is unavailable for `borrower`.
- `pauseRepay`
  - **Constraints**: No constraints.
  - **Impact**: Change the `repayPaused` state for `borrower`. If `repayPaused` is true, the `repay` action is unavailable for `borrower`.

Branches and code coverage

Intended branches

- If `borrowPaused` is true for `borrower`, the `borrow` function that is called by `borrower` reverts.
  - ☑ Test coverage
- If `repayPaused` is true for `borrower`, the `repay` function that is called by `borrower` reverts.
  - ☑ Test coverage

Negative behavior

- Non-whitelisted caller
  - ☑ Negative test
- The borrower is not enabled
  - ☑ Negative test

Function: `setGlobalPaused(bool _pauseBorrow, bool _pauseRepay)`

Allows the caller who has access to this function to set states of `globalBorrowPaused` and `globalRepayPaused`.

Inputs

- `_pauseBorrow`
  - **Constraints**: No constraints.
  - **Impact**: Change `globalBorrowPaused` state. If `globalBorrowPaused` is true, the `borrow` action is unavailable.
- `_pauseRepay`
  - **Constraints**: No constraints.
  - **Impact**: Change `globalRepayPaused` state. If `globalRepayPaused` is true, the `repay` action is unavailable.

Branches and code coverage

Intended branches

- If `globalRepayPaused` is true, the `borrow` function reverts.
  - ☑ Test coverage
- If `globalRepayPaused` is true, the `repay` function reverts.
  - ☑ Test coverage

Negative behavior

- Non-whitelisted caller
  - ☑ Negative test

Function: `shutdownBorrower(address borrower)`

Allows caller with an access to this function to revoke the user’s approval for the borrow. The `borrower` will be deleted from the `borrowers` array, the current debt (`principal` + `interest`) will be reset, and debt tokens will be burned for `borrower`.

Function: `withdraw(uint256 amount, address recipient)`

Allows SupplyManager to withdraw assets from strategy. The appropriate amount of debt tokens will be burned.

Inputs

- `amount`
    - **Constraints**: `amount` should not be more than available tokens (not on loan).
    - **Impact**: Amount of asset tokens to withdraw.
- `recipient`
    - **Constraints**: No constraints.
    - **Impact**: Receiver of asset tokens.

Branches and code coverage

Intended branches

- The `withdraw` was executed properly
    ☑ Test coverage

Negative behavior

- Caller is not a trusted SupplyManager.
    ☑ Negative test
- `amount` is more than `_globalAvailableToBorrow()`.
    ☑ Negative test

Function call analysis

- `this.debtToken.burn(address(this.idleStrategyManager), OrigamiMath.scaleUp(amount, this._assetScalar))`
    - **What is controllable?** `amount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
    - **What happens if it reverts, reenters or does other unusual control flow?** Can revert if IdleStrategyManager does not own enough debt tokens.
- `this.idleStrategyManager.withdraw(amount, recipient)`
    - **What is controllable?** `amount` and `recipient`.
    - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
    - **What happens if it reverts, reenters or does other unusual control flow?** Can revert if `amount` is zero. If `recipient` address is zero, can revert due to an error during the withdraw process from the Strategy contract.

5.13. Module: OrigamiLendingRewardsMinter.sol

Function: `checkpointDebtAndMintRewards(address[] debtors)`

Checkpoints the debt and distributes rewards.

Inputs

- `debtors`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Debtors to be checkpointed.

Branches and code coverage

Intended branches

- Correctly mint according to the checkpoints and the accumulated interest.
  - ☑ Test coverage
- Split the interest into fees.
  - ☑ Test coverage

Function call analysis

- `this.debtToken.checkpointDebtorsInterest(debtors)`
  - **What is controllable?** Everything.
  - **If the return value is controllable, how is it used and how can it go wrong?** Discarded.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this._mintRewards() -> this.debtToken.estimatedCumulativeInterest()`
  - **What is controllable?** Nothing.
  - **If the return value is controllable, how is it used and how can it go wrong?** Estimated cumulative interest.
  - **What happens if it reverts, reenters or does other unusual control flow?** Estimated cumulative interest (borrow rates should be up to date).
- `this._mintRewards() -> this.ovToken.performanceFee()`
  - **What is controllable?** Nothing.
  - **If the return value is controllable, how is it used and how can it go wrong?** Performance fee in BP.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this._mintRewards() -> this.oToken.mint(this.feeCollector, feeAmount)`

    - **What is controllable?** Nothing.
    - **If the return value is controllable, how is it used and how can it go wrong?** Discarded.
    - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this._mintRewards() -> this.oToken.mint(address(this), newReservesAmount)`
    - **What is controllable?** Nothing.
    - **If the return value is controllable, how is it used and how can it go wrong?** Discarded.
    - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this._mintRewards() -> SafeERC20.safeIncreaseAllowance(this.oToken, address(this.ovToken), newReservesAmount)`
    - **What is controllable?** Nothing.
    - **If the return value is controllable, how is it used and how can it go wrong?** Discarded.
    - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `this._mintRewards() -> this.ovToken.addPendingReserves(newReservesAmount)`
    - **What is controllable?** Nothing.
    - **If the return value is controllable, how is it used and how can it go wrong?** Discarded.
    - **What happens if it reverts, reenters or does other unusual control flow?** N/A.

5.14. Module: OrigamiLendingSupplyManager.sol

Function: `exitToToken(address account, IOrigamiInvestment.ExitQuoteData quoteData, address recipient)`

This function is called by the oToken contract when user exits from oToken.

Inputs

- `account`
    - **Constraints**: No constraints.
    - **Impact**: The account that initiates the exit process.
- `quoteData.toToken`
    - **Constraints**: Verify that `toToken` is equal to the `asset` address.
    - **Impact**: The tokens that `account` expects to receive as a result of the exit process.
- `quoteData.investmentTokenAmount`

    - **Constraints**: The new `Utilisation` should not exceed the cap.
    - **Impact**: The shares amount to exit.
- `recipient`
    - **Constraints**: No constraints.
    - **Impact**: Receiver of `asset` tokens.

Branches and code coverage

Intended branches

- The `exitToToken` was executed properly
    - ☑ Test coverage

Negative behavior

- `quoteData.investmentTokenAmount` exceeds the `cap`.
    - ☑ Negative test
- The caller is not oToken.
    - ☑ Negative test
- `quoteData.toToken` is not `asset`.
    - ☑ Negative test
- `_paused.exitsPaused` is true.
    - ☑ Negative test

Function call analysis

- `this.circuitBreakerProxy.preCheck(address(this.oToken), account, quoteData.investmentTokenAmount)`
    - **What is controllable?** `quoteData.investmentTokenAmount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
    - **What happens if it reverts, reenters or does other unusual control flow?** Will revert if the new `Utilisation` exceeds the current capacity.
- `this.lendingClerk.withdraw(toTokenAmount, recipient)`
    - **What is controllable?** `toTokenAmount` and `recipient`.
    - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
    - **What happens if it reverts, reenters or does other unusual control flow?** The function will revert if `toTokenAmount` is more than the available debt tokens (not borrowed).

Function: `investWithToken(address account, IOrigamiInvestment.InvestQuoteData quoteData)`

This function is called by the oToken contract during the investing process.

Inputs

- `account`
  - **Constraints**: If the global variable `allowAll` is true, the `account` is not validated. Otherwise, if the address is not a contract, the `allowedAccounts` should contain the `account` address or `account` should be equal to the `ovToken` address.
  - **Impact**: The address of the caller of the `OToken:investWithToken` function who initiated investing process.
- `quoteData.fromToken`
  - **Constraints**: Should be equal to `asset`.
  - **Impact**: The address of the token that will be invested.
- `quoteData.fromTokenAmount`
  - **Constraints**: No constraints.
  - **Impact**: The invested token amount.

Branches and code coverage

Intended branches

- The `investWithToken` was executed properly
  - ☑ Test coverage

Negative behavior

- `fromToken` is not approved.
  - ☑ Negative test
- `account` is not allowed.
  - ☑ Negative test
- `_paused.investmentsPaused` is true.
  - ☑ Negative test

Function call analysis

- `this.lendingClerk.deposit(quoteData.fromTokenAmount)`
  - **What is controllable?** `quoteData.fromTokenAmount`
  - **If the return value is controllable, how is it used and how can it go wrong?** N/A.

- **What happens if it reverts, reenters or does other unusual control flow?** The function transfers `asset` tokens (should be equal to the `fromToken`, otherwise it reverts) from this contract to the `lendingClerk` and allocates them to the IdleStrategyManager contract and mints debt tokens for strategy.

Function: `recoverToken(address token, address to, uint256 amount)`

Allows the caller who has access to the function to transfer any tokens from the contract balance without restrictions.

Inputs

- `token`
  - **Constraints**: No constraints.
  - **Impact**: The address of the token that will be transferred — can be this contract address.
- `to`
  - **Constraints**: Not zero address.
  - **Impact**: The receiver of tokens.
- `amount`
  - **Constraints**: The contract should have enough amount of tokens.
  - **Impact**: The number of tokens will be transferred from the contract.

Branches and code coverage

Negative behavior

- Non-whitelisted caller
  - ☑ Negative test

Function: `setLendingClerk(address _lendingClerk)`

Allows caller with access to this function to change the `lendingClerk` contract address. The `lendingClerk` contract is triggered during investment and exit-token processes for deposit/withdraw tokens. The `asset` tokens' approve for the old `lendingClerk` contract will be reset to zero, and the new `lendingClerk` will get full access to `asset` tokens, which are owned by this contract.

Inputs

- `_lendingClerk`

    - **Constraints**: Cannot be zero address.
    - **Impact**: This contract is responsible for managing borrowing, repayments, and debt of borrowers.

Branches and code coverage

Intended branches

- The `lendingClerk` was updated properly
  - ☑ Test coverage

Negative behavior

- Non-whitelisted caller
  - ☑ Negative test
- The `_lendingClerk` is zero address
  - ☑ Negative test

5.15. Module: OrigamiLovTokenErc4626Manager.sol

Function: `_depositIntoReserves(address fromToken, uint256 fromTokenAmount)`

Deposits into reserves (converting tokens if they are not the reserve token).

Inputs

- `fromToken`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: The `from` token (sDAI or DAI).
- `fromTokenAmount`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Amount of tokens.

Branches and code coverage

Intended branches

- Converts DAI into sDAI if token is DAI.
  - ☑ Test coverage

Negative behavior

- Reverts if token is not DAI or sDAI.
    - ☑ Negative test

Function: `_maxUserReserves()`

The internal view function which calculate the max amount of reserves which can be added. The function is used by extrernal view function `maxInvest`.

Branches and code coverage

Intended branches

- Scales and rounds correctly depending on the oracle config.
    - ☑ Test coverage
- Returns `maxTokenAmount` if debt is zero.
    - ☑ Test coverage

Function: `_rebalanceDown(uint256 borrowAmount, bytes swapData, uint256 minReservesOut, bool force)`

Rebalances A/L down (borrow).

Inputs

- `borrowAmount`
    - **Control**: Full.
    - **Constraints**: None.
    - **Impact**: Amount to borrow.
- `swapData`
    - **Control**: Full.
    - **Constraints**: None.
    - **Impact**: Swap specific data for the swap debt to deposit.
- `minReservesOut`
    - **Control**: Full.
    - **Constraints**: None.
    - **Impact**: The slippage check.
- `force`
    - **Control**: Full.
    - **Constraints**: None.
    - **Impact**: Force ignore A/L limits.

Branches and code coverage

Intended branches

- Slippage checks are respected.
  - ☑ Test coverage
- Force parameter is respected.
  - ☑ Test coverage

Negative behavior

- A/L cannot go over ceiling or below floor when force is not set.
  - ☑ Negative test

Function: `_rebalanceUp(uint256 depositAssetsToWithdraw, uint256 minReserveAssetShares, bytes swapData, uint256 minDebtAmountToRepay, bool force)`

Rebalances up implementation.

Inputs

- `depositAssetsToWithdraw`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Amount to withdraw.
- `minReserveAssetShares`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Slippage check.
- `swapData`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Swapper-specific data for swapping.
- `minDebtAmountToRepay`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Slippage check.
- `force`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Ignore A/L limits and force rebalance.

Branches and code coverage

Intended branches

- Slippage checks are respected.
  - ☑ Test coverage
- Force parameter is respected.
  - ☑ Test coverage

Negative behavior

- A/L cannot go over ceiling or below floor when force is not set.
  - ☑ Negative test

Function: `_redeemFromReserves(uint256 reservesAmount, address toToken, address recipient)`

Redeems from reserves.

Inputs

- `reservesAmount`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Amount of reserves to redeem.
- `toToken`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Desired `toToken` (sDAI or DAI).
- `recipient`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: Recipient.

Branches and code coverage

Intended branches

- Redeems if token is DAI.
  - ☑ Test coverage

Negative behavior

- Invalid if tokens are neither DAI or sDAI.
  - ☑ Negative test

5.16. Module: OrigamiLovToken.sol

Function: `collectPerformanceFees()`

This function charges a performance fee for `feeCollector`. The function can only be called at a certain interval, no more often than `PERFORMANCE_FEE_FREQUENCY`.

Branches and code coverage

Intended branches

- The `feeCollector` received the proper amount of lovTokens.
  - ☑ Test coverage

Negative behavior

- The function cannot be called twice during the `PERFORMANCE_FEE_FREQUENCY` period.
  - ☑ Negative test

Function: `exitToNative(ExitQuoteData, address payable)`

Investing with native tokens is not supported. Accordingly, the exit to native tokens is not supported either. The function will revert.

Branches and code coverage

Negative behavior

- The function reverts.
  - ☑ Negative test

Function: `exitToToken(ExitQuoteData quoteData, address recipient)`

Allows users to exit from lovToken investing and receive an approved ERC-20 token in exchange.

Inputs

- `quoteData.investmentTokenAmount`
  - **Constraints**: Cannot be zero, and the caller must own a sufficient amount.
  - **Impact**: The amount of lovTokens to sell. Tokens will be transferred from the caller to the manager contract. At the end of transactions, tokens will be burned from the manager account.
- `quoteData.toToken`
  - **Constraints**: There is verification inside the `man-`

  ager._redeemFromReserves() function that toToken can be equal to the depositAsset or _reserveToken; otherwise, the transaction will revert.
  - **Impact**: The recipient will receive these tokens in return — toToken should be an accepted ERC-20 token.
- quoteData.maxSlippageBps
  - **Constraints**: Is not used and is not validated.
  - **Impact**: The maximum allowed slippage of the expectedToTokenAmount.
- quoteData.deadline
  - **Constraints**: Is not used and is not validated.
  - **Impact**: N/A.
- quoteData.expectedToTokenAmount
  - **Constraints**: Is not used and is not validated.
  - **Impact**: N/A.
- quoteData.minToTokenAmount
  - **Constraints**: There is a check inside _manager.exitToToken(msg.sender, quoteData, recipient) that toTokenAmount is not less than quoteData.minToTokenAmount.
  - **Impact**: The minimum amount of toToken to receive.
- quoteData.underlyingInvestmentQuoteData
  - **Constraints**: Is not used and is not validated.
  - **Impact**: N/A.
- recipient
  - **Constraints**: Cannot be zero address.
  - **Impact**: The receiver of the toToken.

Branches and code coverage

Intended branches

- recipient receives expected amount of toToken.
  - ☑ Test coverage
- The caller spent investmentTokenAmount of lovTokens.
  - ☑ Test coverage

Negative behavior

- The caller does not have enough lovTokens.
  - □ Negative test
- The toTokenAmount is less than minToTokenAmount.
  - □ Negative test
- toToken is not supported.
  - □ Negative test
- recipient is zero address.

  - □ Negative test
- `quoteData.investmentTokenAmount` is zero.
  - ☑ Negative test

Function call analysis

- `_manager.exitToToken(msg.sender, quoteData, recipient)`
  - **What is controllable?** `quoteData` and `recipient`.
  - **If the return value is controllable, how is it used and how can it go wrong?** If `lovTokenToBurn` is less than `investmentTokenAmount`, part of lovTokens will not be burned; otherwise, if `lovTokenToBurn` is more than `investmentTokenAmount`, more lovTokens will be burned.
  - **What happens if it reverts, reenters or does other unusual control flow?** Can revert if `toTokenAmount` is less than `minToTokenAmount` or if `toToken` is not supported.

Function: `investWithNative(InvestQuoteData)`

Investing with native tokens is not supported. The function will revert.

Branches and code coverage

Negative behavior

- The function reverts.
  - ☑ Negative test

Function: `investWithToken(InvestQuoteData quoteData)`

Allows users to invest in accepted ERC-20 tokens and receive lovToken in return.

Inputs

- `quoteData.fromToken`
  - **Constraints**: There are no constraints here. But the address is validated inside the `_depositIntoReserves` function of the OrigamiLovTokenManager contract (`_manager.investWithToken(msg.sender, quoteData)` -> `_depositIntoReserves`). If `fromToken` is equal to the `depositAsset`, then `depositAsset` will be deposited to the `_reserveToken`. If `fromToken` is equal to the `_reserveToken`, then its tokens are already deposited for the `_manager` contract; otherwise, the function will revert.
  - **Impact**: The token that will be invested.

- `quoteData.fromTokenAmount`
  - **Constraints**: Cannot be zero.
  - **Impact**: The caller transfers the `fromTokenAmount` of approved ERC-20 tokens to the OrigamiLovTokenManager contract.
- `quoteData.maxSlippageBps`
  - **Constraints**: Is not used and is not validated.
  - **Impact**: The maximum allowed slippage of the `expectedInvestmentAmount`.
- `quoteData.deadline`
  - **Constraints**: Is not used and is not validated.
  - **Impact**: The maximum deadline to execute the transaction.
- `quoteData.expectedInvestmentAmount`
  - **Constraints**: Is not used and is not validated.
  - **Impact**: The expected amount of this lovToken token to receive in return.
- `quoteData.minInvestmentAmount`
  - **Constraints**: There is a check inside `_manager.investWithToken(msg.sender, quoteData)` that `investmentAmount` is not less than `quoteData.minInvestmentAmount`.
  - **Impact**: The minimum amount of lovTokens to receive.
- `quoteData.underlyingInvestmentQuoteData`
  - **Constraints**: Is not used and is not validated.
  - **Impact**: Extra quote parameters.

Branches and code coverage

Intended branches

- The expected amount of lovTokens was minted for the `msg.sender`.
  - ☑ Test coverage

Negative behavior

- `quoteData.fromToken` is not a trusted contract.
  - □ Negative test
- The caller does not own enough `fromToken` tokens.
  - □ Negative test
- `quoteData.fromTokenAmount` is zero
  - ☑ Negative test

Function call analysis

- `SafeERC20.safeTransferFrom(IERC20(quoteData.fromToken), msg.sender, address(_manager), quoteData.fromTokenAmount)`

  - **What is controllable?** `quoteData.fromToken` and `quoteData.fromTokenAmount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** No return value.
  - **What happens if it reverts, reenters or does other unusual control flow?** Can revert if `msg.sender` does not have enough `fromToken` to transfer. The function `investWithToken` has a nonreentrant modifier.
- `_manager.investWithToken(msg.sender, quoteData)`
  - **What is controllable?** `quoteData`.
  - **If the return value is controllable, how is it used and how can it go wrong?** The function returns the number of lovTokens that will be minted for the caller. The `investmentAmount` is not less than `quoteData.minInvestmentAmount`.
  - **What happens if it reverts, reenters or does other unusual control flow?** The `_manager` is a trusted contract that is not controlled by the caller.

5.17. Module: OrigamiOToken.sol

Function: `amoBurn(address _account, uint256 _amount)`

Allows caller access to this function to burn AMO tokens — but not more than was minted.

Inputs

- `_account`
  - **Constraints**: `account != address(0)`.
  - **Impact**: The account from which tokens will be burned.
- `_amount`
  - **Constraints**: Cannot be more than `_amoMinted`.
  - **Impact**: The amount of AMO tokens that will be burned.

Branches and code coverage

Intended branches

- `amoMinted` was decreased by `_amount`.
  - ☑ Test coverage

Negative behavior

- Non-whitelisted caller
  - ☑ Negative test
- `_account` is zero address.
  - □ Negative test

- `_account` contains fewer than the specified `amount` of tokens.
  - ☑ Negative test
- `_amount` is more than `_amoMinted`.
  - ☑ Negative test

Function: `amoMint(address _to, uint256 _amount)`

This function allows the caller with access to mint new oTokens as part of its automated market operations.

Inputs

- `_to`
  - **Constraints**: `account != address(0)`.
  - **Impact**: Receiver of new tokens.
- `_amount`
  - **Constraints**: No constraints.
  - **Impact**: The number of tokens that will be minted.

Branches and code coverage

Intended branches

- `amoMinted` was increased by `_amount`.
  - ☑ Test coverage

Negative behavior

- Non-whitelisted caller
  - ☑ Negative test
- `_to` is zero address.
  - □ Negative test

Function: `exitToNative(ExitQuoteData, address payable)`

The function is not supported in this contract. See the `OrigamiOTokenWithNative.exitToNative()` function.

Branches and code coverage

Intended branches

- The `investWithNative` function reverts

☑ Test coverage

Function: `exitToToken(ExitQuoteData quoteData, address recipient)`

Allows to sell oTokens and receive one of the accepted ERC-20 tokens in return. Firstly, the oToken will be transferred to the `_manager` account from the caller's account. Then, `_manager` will process these tokens and determine the amount of `quoteData.toToken` to be received in exchange and send to the `recipient`. Finally, this calculated amount of `quoteData.toToken` will be burned from the manager's account.

Inputs

- `quoteData.investmentTokenAmount`
  - **Constraints**: Cannot be zero, and the caller must own a sufficient amount.
  - **Impact**: The amount of oTokens to sell. Tokens will be transferred from the caller to the manager contract. At the end of transactions, tokens will be burned from the manager account.
- `quoteData.toToken`
  - **Constraints**: There is no verification, but the `manager.exitToToken()` function verifies that `quoteData.toToken` is equal to the `asset` address; otherwise, it reverts.
  - **Impact**: The `recipient` will receive these tokens in return — `toToken` should be an accepted ERC-20 token.
- `recipient`
  - **Constraints**: Cannot be zero address.
  - **Impact**: The receiver of the `toToken`.

Branches and code coverage

Intended branches

- Successful `exitToToken` execution
  ☑ Test coverage

Negative behavior

- `toToken` is not approved.
  □ Negative test
- `quoteData.investmentTokenAmount` is zero.
  ☑ Negative test
- The caller owns fewer than the `quoteData.investmentTokenAmount` of `oToken`.
  ☑ Negative test

Function: `investWithNative(InvestQuoteData)`

The function is not supported in this contract. See the `OrigamiOTokenWithNative.investWithNative()` function.

Branches and code coverage

Intended branches

- The `investWithNative` function reverts
  - ☑ Test coverage

Function: `investWithToken(InvestQuoteData quoteData)`

Allows to invest in accepted ERC-20 tokens and receive oTokens in return. Firstly, the `quoteData.fromToken` will be transferred to the `_manager` account. Then, `_manager` will process these tokens and determine the amount of oToken to be received in exchange. Finally, this calculated amount of oToken will be minted to the caller's account.

Inputs

- `quoteData.fromTokenAmount`
  - **Constraints**: Should not be zero.
  - **Impact**: The number of `quoteData.fromToken` transferred to the `_manager` account.
- `quoteData.fromToken`
  - **Constraints**: There is no verification, but the `manager.investWithToken()` function verifies that `quoteData.fromToken` is equal to the `asset` address; otherwise, it reverts.
  - **Impact**: The address of the token that will be invested in exchange for oToken. So it should be only an approved ERC-20 token address.

Branches and code coverage

Intended branches

- Successful investment
  - ☑ Test coverage

Negative behavior

- `fromToken` is not approved.
  - □ Negative test
- `quoteData.fromTokenAmount` is zero
  - ☑ Negative test

- The caller owns fewer than the `quoteData.fromTokenAmount` of `quoteData.fromToken`.
  - ☑ Negative test

5.18. Module: RepricingToken.sol

Function: `addPendingReserves(uint256 amount)`

Allows the caller who has access to the function to add more reserve tokens. The new tokens will be vested.

Inputs

- `amount`
  - **Constraints**: The caller should have enough number of `reserveTokens` to transfer.
  - **Impact**: The `amount` will increase `pendingReserves`, and `totalReserves()` is gradually increasing as the vesting period ends.

Branches and code coverage

Negative behavior

- Non-whitelisted caller
  - ☑ Negative test
- `amount` is zero.
  - ☑ Negative test

Function: `checkpointReserves()`

Updates global variables `vestedReserves`, `pendingReserves`, and `lastVestingCheckpoint` after the period of the `VestingDuration` is fully completed.

Branches and code coverage

Intended branches

- `vestedReserves`, `pendingReserves`, and `lastVestingCheckpoint` are updated properly.
  - ☑ Test coverage

Negative behavior

- Vesting period is not completed.

☑ Negative test

Function: `recoverToken(address _token, address _to, uint256 _amount)`

Allows the caller who has access to the function to transfer any tokens from the contract balance without restrictions in case token is not `reserveToken`; otherwise, only the surplus reserves can be recovered.

Inputs

- `token`
  - **Constraints**: No constraints.
  - **Impact**: The address of the token that will be transferred — can be `reserveToken` contract address.
- `to`
  - **Constraints**: Not zero address.
  - **Impact**: The receiver of tokens.
- `amount`
  - **Constraints**: The contract should have enough amount of tokens — if token is `reserveToken`, no more than surplus reserves.
  - **Impact**: The number of tokens will be transferred from the contract.

Branches and code coverage

Intended branches

- Tokens are recovered properly.
  ☑ Test coverage

Negative behavior

- Non-whitelisted caller
  ☑ Negative test
- Exceed the limit if `_token` is `reserveToken`.
  ☑ Negative test

5.19. Module: TokenPrices.sol

Function: `setTokenPriceFunction(address token, bytes fnCalldata)`

The function is available only for the owner of the contract. Allows the owner of the contract associates a token address with a function's calldata that specifies the method for retrieving the price.

Inputs

- `token`
    - **Constraints**: No constraints.
    - **Impact**: The address of the token associated with the function.
- `fnCalldata`
    - **Constraints**: No constraints.
    - **Impact**: The calldata that will be possible to execute over the `runPriceFunction` function. It is used for `tokenPrices` and `tokenPrice` functions that are intended solely for utility purposes.

Branches and code coverage

Negative behavior

- Caller is not an owner.
    - ☑ Negative test

6. Assessment Results

At the time of our assessment, the reviewed code was not deployed to the Ethereum Mainnet.

During our assessment on the scoped Origami Finance contracts, we discovered three findings, all of which were low impact. TempleDAO acknowledged all findings and implemented fixes.

6.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.