CERTIK

Security Assessment

Exactly

Jun 12th, 2022

Table of Contents

Summary

This report has been prepared for Exactly to discover issues and vulnerabilities in the source code of the Exactly project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	Exactly
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/exactly-finance/protocol
Commit	889bfe9fd7b4500820700f82afcc7665e2cd2bcc

Audit Summary

Delivery Date	Jun 12, 2022 UTC
Audit Methodology	Static Analysis, Manual Review

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Mitigated	Partially Resolved	Resolved
Critical	0	0	0	0	0	0	0
Major	2	0	0	1	0	0	1
Medium	1	0	0	1	0	0	0
Minor	5	0	0	2	0	0	3
Optimization	0	0	0	0	0	0	0
Informational	4	0	0	2	0	0	2
Discussion	0	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
AUD	Auditor.sol	eee68afea397390175e9185a048bbbdb43565e77d823ca7ed4ee9df18a694bba
EOB	ExactlyOracle.sol	e7cb706e5840905b6c410308fc8669462d32774e4274771cfbb5aad91a85f41d
FLE	FixedLenderETHRouter.sol	711f2ddb876a523a9399effe011ff42c699f0f2c51fd38dc047e4798ded24bf4
IOB	interfaces/IOracle.sol	c98e2349b1854f3b2caed3c264e4572cb7742463ffe079f1bd62a0d055869275
IAB	interfaces/IAuditor.sol	bc4ea45d11cafe3c14740cf3b9e9f48c51b94a77d4d44f21c44f3b392783202c
IIR	interfaces/IInterestRateModel.sol	edf2f8407f48257e3cb74fa69aa3949550aacc2738fe8a453d89a7154959fa76
TSU	utils/TSUtils.sol	e2760084b1a58c3fb1d1d0ad23270a236187b6232762da7aa4c5ddc0b36ada2e
PLB	utils/PoolLib.sol	f8aef551fda847a5e0faf4f0cfbdf226102f537637caed78285db8b0f0ee34db
FLB	FixedLender.sol	c9900a0a009eac5e919c9c721180efc049ecaa6ffc78e8bb6caf81822da53fee
IRM	InterestRateModel.sol	aa837b122d6346313e6113afc20e66cf2ecb663c2a72d8fbdd8aa9afda494780
PAB	PoolAccounting.sol	6c32e07bba9f0e4f98d38698814f3c494248afa24c913b4645d1755448d74711

Understanding

External Dependencies

The scope of the audit treats third-party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets.

There are a few depending injection contracts or addresses in the current project:

- `chainlinkFeedRegistry` for contract `ExactlyOracle`;

In addition, the contract is serving as the underlying entity to interact with third-party contracts and interfaces:

- `ERC4626`, `FixedPointMathLib`, `ReentrancyGuard`, `AccessControl`, `Pausable` for contract `FixedLender`;
- `FixedPointMathLib`, `AccessControl` for contract `Auditor`;
- `FeedRegistryInterface`, `AccessControl` for contract `ExactlyOracle`;
- `WETH` for contract `FixedLenderETHRouter`;
- `FixedPointMathLib`, `AccessControl` for contract `InterestRateModel`.
- `FixedPointMathLib`, `AccessControl` for contract `PoolAccounting`;

We assume these vulnerable actors and implement proper logic to collaborate with the current project.

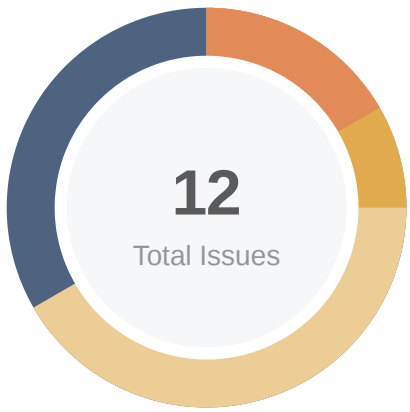
Privileged Roles

The following roles are adopted to enforce the access control:

- Role `DEFAULT_ADMIN_ROLE` is adopted to update configurations of the contract `FixedLender`.
- Role `DEFAULT_ADMIN_ROLE` is adopted to update configurations of the contract `ExactlyOracle`.
- Role `DEFAULT_ADMIN_ROLE` is adopted to update configurations of the contract `Auditor`.
- Role `DEFAULT_ADMIN_ROLE` is adopted to update configurations of the contract `InterestRateModel`.
- Role `DEFAULT_ADMIN_ROLE` is adopted to update configurations of the contract `PoolAccounting`.

To improve the trustworthiness of the project, dynamic runtime updates in the project should be notified to the community. Any plan to invoke the aforementioned functions should be also considered to move to the execution queue of `Timelock` contract.

Findings



Critical	**0** (0.00%)
Major	**2** (16.67%)
Medium	**1** (8.33%)
Minor	**5** (41.67%)
Informational	**4** (33.33%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
AUD-01	Lack Of Input Validation	Volatile Code	Informational	Acknowledged
CON-01	Centralization Related Risks	**Centralization / Privilege**	**Major**	Acknowledged
CON-02	Potential Incorrect Fee Distribution	Logical Issue	Major	Resolved
CON-03	Potential Revert In `Liquidate`	Logical Issue	Medium	Acknowledged
CON-04	Third Party Dependencies	Volatile Code	Minor	Acknowledged
FLB-01	Incorrect Error Thrown	Logical Issue	Minor	Resolved
FLB-02	Incompatibility With Deflationary Tokens	Volatile Code	Minor	Acknowledged
FLB-03	Potential Reentrancy Issues	Logical Issue	Minor	Resolved
FLB-04	Discussion On Liquidation Process	Logical Issue	Informational	Acknowledged
FLE-01	Mistakenly Transferred ETH Could Be Locked In The Contract	Volatile Code	Informational	Resolved
PAB-01	Potential Rounding Error	Logical Issue	Minor	Resolved
PLB-01	Inconsistent Comment And Code	Inconsistency	Informational	Resolved

AUD-01 | Lack Of Input Validation

Category	Severity	Location	Status
Volatile Code	Informational	Auditor.sol (v2): 166~189	Acknowledged

Description

The `enableMarket()` enables a certain FixedLender market of the protocol. However, there is no explicit validation in `enableMarket` function to ensure the `collateralFactor` is properly set.

```
  function enableMarket(
    FixedLender fixedLender,
    uint128 collateralFactor,
    string memory symbol,
    string memory name,
    uint8 decimals
  ) external onlyRole(DEFAULT_ADMIN_ROLE) {
```

Recommendation

The auditors notice that validations on `collateralFactor` has been applied in the function `setCollateralFactor`.

```
  /// @notice sets the collateral factor for a certain fixedLender.
  /// @dev Value can only be set between 90% and 30%.
  /// @param fixedLender address of the market to change collateral factor for.
  /// @param collateralFactor collateral factor for the underlying asset.
  function setCollateralFactor(FixedLender fixedLender, uint128 collateralFactor)
    external
    onlyRole(DEFAULT_ADMIN_ROLE)
  {
    if (collateralFactor > 0.9e18 || collateralFactor < 0.3e18) revert
InvalidParameter();
    markets[fixedLender].collateralFactor = collateralFactor;
    emit CollateralFactorUpdated(fixedLender, collateralFactor);
  }
```

We advise also adding an explicit validation in `enableMarket` function as implemented in the `setCollateralFactor()` function:

```
function enableMarket(
    FixedLender fixedLender,
```

```
    uint128 collateralFactor,
    string memory symbol,
    string memory name,
    uint8 decimals
  ) external onlyRole(DEFAULT_ADMIN_ROLE) {
    if (collateralFactor > 0.9e18 || collateralFactor < 0.3e18) revert
InvalidParameter();
    ...
    ...
  }
```

Alleviation

[Exactly]: The team acknowledged this issue and will not update the code in current stage.

CON-01 | Centralization Related Risks

Category	Severity	Location	Status
Centralization / Privilege	**Major**	FixedLender.sol (v2): 230~258; ExactlyOracle.sol (v2): 46~52; Auditor.sol (v2): 146~225; InterestRateModel.sol (v2): 56~60, 101~126; PoolAccounting.sol (v2): 74~96	Acknowledged

Description

In the contract `FixedLender`, the role `DEFAULT_ADMIN_ROLE` has authority over the following functions:

- `setMaxFuturePools` will set the protocol's max future weekly pools for borrowing and lending.
- `setAccumulatedEarningsSmoothFactor` will set the factor used when smoothly accruing earnings to the smart pool.

the role `PAUSER_ROLE` has authority over the following functions:

- `pause` will set the `_pause` state to true in case of emergency
- `unpause` will set the `_pause` state to false when the threat is gone

Any compromise to the `DEFAULT_ADMIN_ROLE` and `PAUSER_ROLE` accounts may allow a hacker to take advantage of this authority.

In the contract `ExactlyOracle`, the role `DEFAULT_ADMIN_ROLE` has authority over the following functions:

- `setAssetSources` will set or replace the sources of assets.

Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow a hacker to take advantage of this authority.

In the contract `Auditor`, the role `DEFAULT_ADMIN_ROLE` has authority over the following functions:

- `setOracle` will set Oracle's to be used
- `setLiquidationIncentive` will set liquidation incentive for the whole ecosystem
- `enableMarket` will enable a certain FixedLender market
- `setCollateralFactor` will set the collateral factor for a certain fixedLender
- `setMarketBorrowCaps` will set the given borrow caps for the given fixedLender markets

Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow a hacker to take advantage of this authority.

In the contract `InterestRateModel`, the role `DEFAULT_ADMIN_ROLE` has authority over the following functions:

- `setCurveParameters` will update this model's curve parameters.
- `setSPFeeRate` will update `spFeeRate`

Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow a hacker to take advantage of this authority.

In the contract `PoolAccounting`, the role `DEFAULT_ADMIN_ROLE` has authority over the following functions:

- `setInterestRateModel` will set the interest rate model to be used by this PoolAccounting
- `setPenaltyRate` will set the penalty rate per second.
- `setSmartPoolReserveFactor` will set the percentage that represents the smart pool liquidity reserves that can't be borrowed.

Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow a hacker to take advantage of this authority.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multi-signature wallets.

Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND

- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement;
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles;
  OR
- Remove the risky functionality.

Noted: Recommend considering the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

[Exactly]: We have implemented the suggested short term solution, where we have a multisig (10 addresses, 5 signers) and we have a timelock. The separation of concerns for the multisig and the timelock are the following:

1. `PAUSER_ROLE`: this role is assigned to the multisig. This will allow the multisig to execute a total hold of the depositing and borrowing actions for the protocol, but not for that of withdraw. This feature is only in case of emergency, where some major vulnerability would put user funds at risk.
2. `DEFAULT_ADMIN_ROLE`: this role is assigned to the `TimelockController` (openZeppelin). The minimum value for executing anything is 7 days, and the proposer and the executor is only the multisig. In the deploy script, the `ADMIN_ROLE` is revoked from the deployer as soon as the `TimelockController` contract is granted authority.

We're aware for the mid term solution which is the DAO approach and it's highly likely that we move in this direction. A possible iteration would be to re-assign the proposer and executor on the timelock to the newly created DAO. In the current stage of the project, we're working to discover the parameters that would make

the most sense. We do understand the permanent solution, but it's not something that we can properly discuss at the moment.

CON-02 | Potential Incorrect Fee Distribution

Category	Severity	Location	Status
Logical Issue	Major	PoolAccounting.sol (v2): 144~151, 246~253; utils/PoolLib.sol (v2): 161~168	Resolved

Description

According to the Exactly project logic, the protocol maintains two pools: smart pool and maturity pool.

The `distributeEarningsAccordingly` function is designed to distribute a portion of fee to the smart pool.

```
  /// @notice Returns what proportional of earnings would amountFunded represent
considering suppliedSP the total
  /// @param earnings amount to be distributed as earnings between the two participants
  /// @param suppliedSP current supply of the smart pool.
  /// @param amountFunded amount that will be checked if it came from smart pool or not.
  function distributeEarningsAccordingly(
    uint256 earnings,
    uint256 suppliedSP,
    uint256 amountFunded
  ) internal pure returns (uint256 earningsA, uint256 earningsB) {
    earningsB = earnings.fmul(amountFunded - Math.min(suppliedSP, amountFunded),
amountFunded);
    earningsA = earnings - earningsB;
  }
```

According to the comment in L159, `suppliedSP` parameter is the current supply of the smart pool. Therefore, the variable `earningsA` may represent the fees that belong to the smart pool and `earningsB` represent the leftover fees.

In the function `borrowMP`, the `distributeEarningsAccordingly` function will be invoked to distribute the fees. The two return values (`borrowVars.newUnassignedEarnings`, `borrowVars.earningsSP`) are respectively added to the corresponding values `pool.earningsUnassigned` and `smartPoolEarningsAccumulator`.

However, the return value might have been used in the **opposite** way. That is, the **second** return value in L144 `borrowVars.earningsSP` is added to the smart pool earning records, which might conflict with the above understanding on `distributeEarningsAccordingly()` that the **first** return value should represent the fees that belong to the smart pools.

```
(borrowVars.newUnassignedEarnings, borrowVars.earningsSP) =
PoolLib.distributeEarningsAccordingly(
      borrowVars.fee,
      pool.smartPoolBorrowed(),
      amount
    );
    smartPoolEarningsAccumulator += borrowVars.earningsSP;
    pool.earningsUnassigned += borrowVars.newUnassignedEarnings;
```

The same issue might exist in the function `withdrawMP`:

```
 (uint256 earningsUnassigned, uint256 newEarningsSP) =
PoolLib.distributeEarningsAccordingly(
      amount - redeemAmountDiscounted,
      pool.smartPoolBorrowed(),
      redeemAmountDiscounted
    );
    pool.earningsUnassigned += earningsUnassigned;
    earningsSP += newEarningsSP;
```

Therefore, it could lead to incorrect fees distributed to the smart pool.

Recommendation

We hope to discuss with the team if the above return values of `distributeEarningsAccordingly()` function are incorrectly used? Is the following refactored code aligned with the project design?

```
 (borrowVars.earningsSP, borrowVars.newUnassignedEarnings) =
PoolLib.distributeEarningsAccordingly(
      borrowVars.fee,
      pool.smartPoolBorrowed(),
      amount
    );
    smartPoolEarningsAccumulator += borrowVars.earningsSP;
    pool.earningsUnassigned += borrowVars.newUnassignedEarnings;
```

```
 (uint256 newEarningsSP, uint256 earningsUnassigned) =
PoolLib.distributeEarningsAccordingly(
      amount - redeemAmountDiscounted,
      pool.smartPoolBorrowed(),
      redeemAmountDiscounted
    );
    pool.earningsUnassigned += earningsUnassigned;
    earningsSP += newEarningsSP;
```

Alleviation

[Exactly]: The team confirmed that:

1. Every time a borrow in a maturity uses Smart Pool assets, the fees generated by the loan will be assigned to `unnassignedEarnings`. This is because this earnings have not been accrued yet, and are available to be taken by a Maturity Pool Depositor
2. Every time a borrow in a maturity is generated by Maturity Pool assets, those earnings will be automatically assigned to the Smart Pool participants as earnings (`earningsSP` return value in the function mentioned).
3. The logic behind the naming convention is: `earningsSP` are earnings to be immediately sent to the smart pool as earnings, and `earningsUnassigned` are earnings that can be taken by a maturity pool depositor.
4. Over time, if no depositor comes to take the `earningsUnassigned`, then all those earnings will be accrued to the smart pool.

CON-03 | Potential Revert In `Liquidate`

Category	Severity	Location	Status
Logical Issue	Medium	Auditor.sol (v2): 346~360; FixedLender.sol (v2): 278~283	Acknowledged

Description

The `liquidate` function is designed to liquidate an uncollateralized position. It will call `auditor.liquidateCalculateSeizeAmount` to get the number of tokens to be liquidated and call `_seize` or `collateralFixedLender.seize` to seize a certain amount of tokens.

According to the logic of `liquidateCalculateSeizeAmount` function, we have the following equation:

$$seizeTokens = \frac{actualRepayAmount * priceBorrowed * 10^{markets[fixedLenderCollateral].decimals} * liquidationIncentive}{priceCollateral * 10^{markets[fixedLenderBorrowed].decimals} * 1e18}$$

```
function liquidateCalculateSeizeAmount(
    FixedLender fixedLenderBorrowed,
    FixedLender fixedLenderCollateral,
    uint256 actualRepayAmount
  ) external view override returns (uint256) {
    // Read oracle prices for borrowed and collateral markets
    uint256 priceBorrowed =
oracle.getAssetPrice(FixedLender(fixedLenderBorrowed).assetSymbol());
    uint256 priceCollateral =
oracle.getAssetPrice(FixedLender(fixedLenderCollateral).assetSymbol());

    uint256 amountInUSD = actualRepayAmount.fmul(priceBorrowed,
10**markets[fixedLenderBorrowed].decimals);
    // 10**18: usd amount decimals
    uint256 seizeTokens = amountInUSD.fmul(10**markets[fixedLenderCollateral].decimals,
priceCollateral);

    return seizeTokens.fmul(liquidationIncentive, 1e18);
  }
```

In order to simplify the calculation, assuming that there are the same decimals and the `liquidationIncentive` is 1e18, so the equation will be :

$$seizeTokens = \frac{actualRepayAmount * priceBorrowed}{priceCollateral}$$

When the borrower is delayed, there are more and more penalties and the amount of `actualRepayAmount` will increase. When the `actualRepayAmount` increases, the `seizeTokens` will also increase.

If the `seizeTokens` is greater than the balance of the collateral asset, the `liquidate` function will revert:

```
    uint256 seizeTokens = auditor.liquidateCalculateSeizeAmount(this,
collateralFixedLender, assets);
```

```
    // Revert if borrower collateral token balance < seizeTokens
    (uint256 balance, ) = collateralFixedLender.getAccountSnapshot(borrower, maturity);
    if (balance < seizeTokens) revert BalanceExceeded();
```

In this case, no one can liquidate this uncollateralized position anymore.

Recommendation

To minimize the potential loss resulting from a collateral price drop, the revert during the liquidation is unexpected and unwanted in most situations. Therefore, we would like to open a discussion with the team if the revert is intended?

Alleviation

[Exactly]: The team acknowledged the potential revert in a liquidation function when the growing penalties or price drop make the seized token calculated bigger than the balance held by the user.

CON-04 | Third Party Dependencies

Category	Severity	Location	Status
Volatile Code	Minor	ExactlyOracle.sol (v2): 56~59; FixedLenderETHRouter.sol (v2): 4	Acknowledged

Description

The contract is serving as the underlying entity to interact with third-party `chainlink` and `WETH` protocols. The scope of the audit treats 3rd party entities as black boxes and assumes their functional correctness. However, in the real world, 3rd parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of 3rd parties can possibly create severe impacts.

For example, if the price of chainlink oracle is inaccurate, it could lead to the loss of the Exactly protocol.

Reference: https://twitter.com/CertiKAlert/status/1524968003049332756

Recommendation

We understand that the business logic of `ExactlyOracle` requires interaction with `chainlink` and `WETH`. We encourage the team to constantly monitor the statuses of 3rd parties to mitigate the side effects when unexpected activities are observed.

Alleviation

[Exactly]: The team acknowledged this issue and will not update the code in current stage.

FLB-01 | Incorrect Error Thrown

Category	Severity	Location	Status
Logical Issue	Minor	FixedLender.sol (v2): 283, 490	Resolved

Description

```
if (balance < seizeTokens) revert BalanceExceeded();
```

Per of the context of the function `liquidate`, and the `if` condition, i.e. `balance < seizeTokens`, the error to be thrown should be `BalanceInsufficient()` instead of `BalanceExceeded()`.

Recommendation

Consider renaming the error `BalanceExceeded()` to make it match the logic of the `if` branch.

Alleviation

[Exactly]: The team resolved this issue in commit 4d843eb7e3b26b61e5b40a1d84ca380e48d03304

FLB-02 | Incompatibility With Deflationary Tokens

Category	Severity	Location	Status
Volatile Code	Minor	FixedLender.sol (v2): 132	Acknowledged

Description

When transferring standard ERC20 deflationary tokens, the input amount may not be equal to the received amount due to the charged transaction fee. For example, if a user stakes 100 deflationary tokens (with a 10% transaction fee) in a `FixedLender`, only 90 tokens actually arrived in the contract. However, the user can still withdraw 100 tokens from the contract, which causes the contract to lose 10 tokens in such a transaction.

Recommendation

We advise the client to regulate the set of pool tokens supported and add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens.

Alleviation

[Exactly]: At the current moment, the team acknowledge that our system will not support tokens with fees, and will have to properly vet all assets added to the system.

FLB-03 | Potential Reentrancy Issues

Category	Severity	Location	Status
Logical Issue	Minor	FixedLender.sol (v2): 171, 180, 213, 218	Resolved

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

The `FixedLender` inherits from the `ERC4626` contract, which implementation is unknown. Furthermore, the following functions will invoke corresponding functions inherits from the `ERC4626` contract` and are possibly interact with external assets:

- `withdraw()`
- `redeem()`
- `transfer()`
- `transferFrom()`

For example, if the attacker is able to successfully trigger `redeem()` invocation inside the `withdraw()` invocation, before contact states are resolved, it could lead to a potential reentrancy attacker loophole.

Recommendation

We recommend also applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

[Exactly]: We've reviewed internally and externally these contracts, and we cannot reproduce the claim about the potential reentrancy attack in the mentioned functions.

The functions seem to correctly follow the recommended pattern. We understand the concept of adding extra checks, but we feel that those might look slightly odd since the implementation for EIP4626 comes from SolMate (these contracts are currently being audited by them), and the proposed solution involves adding OpenZeppelin's reentrancy guards.

[CertiK]: The auditors confirmed there is no reentrancy attack vector in the current codebase if the external dependency EIP4626 is correctly implemented. However, as the EIP4626 is without current audit scope, this serves as a note to warn of the potential risk introduced by EIP4626.

FLB-04 | Discussion On Liquidation Process

Category	Severity	Location	Status
Logical Issue	Informational	FixedLender.sol (v2): 266	Acknowledged

Description

In the Exactly protocol's liquidation logic, the liquidator could repay the debt of the original borrower to get the original borrow's assets.

However, if the liquidation is not processed in time, or there is a sharp price drop on certain asset. It could lead to the borrowed assets' value much larger than the collateral's value. Therefore, the protocol might suffer unexpected loss.

Recommendation

We hope to discuss with the team on how will the project ensure the liquidation will be triggered in time.

Alleviation

[Exactly]: The team stated liquidation in under development and will introduce more features in the later stage.

FLE-01 | Mistakenly Transferred ETH Could Be Locked In The Contract

Category	Severity	Location	Status
Volatile Code	Informational	FixedLenderETHRouter.sol (v2): 29	Resolved

Description

```
receive() external payable {}
```

The `receive` function allows the contract to receive native tokens (ETH).

However, if ETH is mistakenly transferred by anyone to the current contract, it could be locked.

Recommendation

We recommend only allowing `WETH` address to transfer tokens to the contract. For example,

```
receive() external payable {
  require(msg.sender == address(weth),"Not from WETH");
}
```

Alleviation

[Exactly]: The team resolved this issue by adding a check in commit 372df7290030f4fe352b9b4822a946179fe9c9f8

PAB-01 | Potential Rounding Error

Category	Severity	Location	Status
Logical Issue	Minor	PoolAccounting.sol (v2): 303, 343~347	Resolved

Description

In the `repayMP` function, the following calculation may lead to a rounding error.

```
debtCovered = repayAmount.fmul(repayVars.position.principal + repayVars.position.fee,
repayVars.amountOwed);
```

If the rounding error occurs, the `debtCovered` will be smaller than expected. Therefore, after reducing the debt, there could be some leftover in the position amd the following condition in the `if` branch will never be met:

```
    repayVars.position.reduceProportionally(debtCovered);
    if (repayVars.position.principal + repayVars.position.fee == 0) {
      delete mpUserBorrowedAmount[maturity][borrower];
      userMpBorrowed[borrower] = userMpBorrowed[borrower].clearMaturity(maturity);
    } else {
      // we proportionally reduce the values
      mpUserBorrowedAmount[maturity][borrower] = repayVars.position;
    }
```

Hence, as the code in the `if` branch will not be executed due to rounding error, the position will never be fully repaid and closed.

Recommendation

We hope to discuss with the team if the above situation will cause any actual issues to the project.

Alleviation

[Exactly]: The team resolved this issue by redesigning the calculation of `debtCovered` in commit 719ebafd63a654922310464a1429a9ea87a8fa87.

PLB-01 | Inconsistent Comment And Code

Category	Severity	Location	Status
Inconsistency	Informational	utils/PoolLib.sol (v2): 218~221	Resolved

Description

The comment of the function `hasMaturity` states this function will clean the user's position, which does not match the function implementation.

```
/// @dev Cleans user's position from encoded maturity pools
/// @param encoded encoded maturity dates where the user borrowed
/// @param maturity maturity date
function hasMaturity(uint256 encoded, uint256 maturity) internal pure returns (bool) {
```

Recommendation

Consider changing the comment of the function `hasMaturity`.

Alleviation

[Exactly]: The team resolved this issue by changing the comment in commit aed4293bf1652f4cabbf686a649e8e974c9500ce

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK’s prior written consent in each instance.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED “AS IS” AND

“AS AVAILABLE” AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER’S OR ANY OTHER PERSON’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK’S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER’S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED “AS IS” AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK’S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.