ThunderLoan Protocol Audit Report

Version 1.0

Agrim Sharma

August 5, 2024

ThunderLoan Protocol Audit Report

Agrim Sharma

August 5, 2024

Prepared by Auditor: - Agrim Sharma

Table of Contents

Protocol Summary

The ThunderLoan protocol is meant to do the following:

1. Give users a way to create flash loans
2. Give liquidity providers a way to earn money off their capital

Disclaimer

Agrim Sharma, as a solo auditor makes all effort to find as many vulnerabilities in the code in the given time period, but holds no responsibilities for the findings provided in this document. A security audit

by him is not an endorsement of the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the Solidity implementation of the contracts.

Risk Classification

		Impact		
		High	Medium	Low
Likelihood	High	H	H/M	M
	Medium	H/M	M	M/L
	Low	M	M/L	L

I use the CodeHawks severity matrix to determine severity. See the documentation for more details.

Audit Details

- Commit Hash: 8803f851f6b37e99eab2e94b4690c8b70e26b3f6
- In Scope:

```
#-- interfaces
|   #-- IFlashLoanReceiver.sol
|   #-- IPoolFactory.sol
|   #-- ITSwapPool.sol
|   #-- IThunderLoan.sol
#-- protocol
|   #-- AssetToken.sol
|   #-- OracleUpgradeable.sol
|   #-- ThunderLoan.sol
#-- upgradedProtocol
    #-- ThunderLoanUpgraded.sol
```

- Solc Version: 0.8.20
- Chain(s) to deploy contract to: Ethereum
- ERC20s:
  - USDC
  - DAI
  - LINK
  - WETH

Roles

- Owner: The owner of the protocol who has the power to upgrade the implementation.
- Liquidity Provider: A user who deposits assets into the protocol to earn interest.
- User: A user who takes out flash loans from the protocol.

Executive Summary

This one was interesting, worked on auditing of Failure to initialize, Storage collision, Centralization, Missing events, Bad Upgrade, Oracle & Price Manipulation

Issues found

Severity	Number of issues found
High	3
Medium	2
Low	7
Gas	3
Info	4
Total	19

Findings

High

[H-1] Unnecessary `updateExchangeRate` in `deposit` function incorrectly updates `exchangeRate` preventing withdraws and unfairly changing reward distribution

Description: In the thunderloan system, the `exchangeRate` is responsible for calculating the exchange rate btn assetTokens and underlying tokens. In a way, it's responsible for keeping track of how many fees to give to liquidity providers.

However, the deposit func updates this rate, without collecting any fees!

```
    function deposit(IERC20 token, uint256 amount) external
        revertIfZero(amount) revertIfNotAllowedToken(token) {
        AssetToken assetToken = s_tokenToAssetToken[token];
        uint256 exchangeRate = assetToken.getExchangeRate();
        uint256 mintAmount = (amount * assetToken.
            EXCHANGE_RATE_PRECISION()) / exchangeRate;
        emit Deposit(msg.sender, token, amount);
        assetToken.mint(msg.sender, mintAmount);
        uint256 calculatedFee = getCalculatedFee(token, amount);
        assetToken.updateExchangeRate(calculatedFee);
        token.safeTransferFrom(msg.sender, address(assetToken), amount)
            ;
    }
```

Impact: 1. The redeem function is blocked bcoz the protocol thinks the owed tokens is more than it has. 2. Rewards are incorrectly calculated, leading to liquidity providers potentially getting way more or less than deserved.

Proof of Concept: 1. LP deposits 2. User takes out a flash loan 3. It is now impossible for LP to redeem

Proof of Code

Place the following into `ThunderLoanTest.t.sol`

```
function testRedeemAfterLoan() public setAllowedToken hasDeposits {
        uint256 amountToBorrow = AMOUNT * 10;
        uint256 calculatedFee = thunderLoan.getCalculatedFee(tokenA,
            amountToBorrow);

        vm.startPrank(user);
        tokenA.mint(address(mockFlashLoanReceiver), calculatedFee);
        thunderLoan.flashloan(address(mockFlashLoanReceiver), tokenA,
            amountToBorrow, "");
        vm.stopPrank();

        uint256 amountToRedeem = type(uint256).max;
        vm.startPrank(liquidityProvider);
        thunderLoan.redeem(tokenA, amountToRedeem);
    }
```

Recommended Mitigation: Remove the incorrect updated exchange rate lines from `deposit`.

```
function deposit(IERC20 token, uint256 amount) external revertIfZero(
    amount) revertIfNotAllowedToken(token) {
        AssetToken assetToken = s_tokenToAssetToken[token];
        uint256 exchangeRate = assetToken.getExchangeRate();
        uint256 mintAmount = (amount * assetToken.
            EXCHANGE_RATE_PRECISION()) / exchangeRate;
        emit Deposit(msg.sender, token, amount);
```

```
6           assetToken.mint(msg.sender, mintAmount);
7   -        uint256 calculatedFee = getCalculatedFee(token, amount);
8   -        assetToken.updateExchangeRate(calculatedFee);
9           token.safeTransferFrom(msg.sender, address(assetToken), amount)
                ;
10      }
```

[H-2] By calling a flashloan and then `ThunderLoan::deposit` instead of `ThunderLoan::repay` users can steal all funds from the protocol

[H-3] Mixing up variable location can cause storage collisions in `ThunderLoan::s_flashLoanFee` and `ThunderLoan::s_currentlyFlashLoaning` freezing protocol

Description: `ThunderLoan.sol` has 2 variables in the following order:

```
1 uint256 private s_feePrecision;
2 uint256 private s_flashLoanFee;
```

However, the upgraded contract `ThunderLoanUpgraded.sol` has a different order:

```
1 uint256 private s_flashLoanFee;
2 uint256 public constant FEE_PRECISION = 1e18;
```

Due to how solidity works after the upgrade the `s_flashLoanFee` will have the value of `s_feePrecision`. You cannot adjust the position of storage variable and removing storage variables for constant variables breaks the storage locations as well.

Impact: After upgrade, the `s_flashLoanFee` will have value of `s_feePrecision`. This means that users who take out flash loans right after an upgrade will be charged wrong fee

Also, the `s_currentlyFlashLoaning` mapping with storage in the wrong storage slot.

Proof of Concept:

PoC

The following test shows the difference of gas fees a user has to pay before and after upgrade.

Place the following into `ThunderLoanTest.t.sol`

```
1 import { ThunderLoanUpgraded } from "../../src/upgradedProtocol/
      ThunderLoanUpgraded.sol";
2 .
3 .
4 .
5 function testUpgradeBreaks() public {
```

```
        uint256 feeBeforeUpgrade = thunderLoan.getFee();
        vm.startPrank(thunderLoan.owner());
        ThunderLoanUpgraded upgraded = new ThunderLoanUpgraded();
        thunderLoan.upgradeToAndCall(address(upgraded), "");
        uint256 feeAfterUpgrade = thunderLoan.getFee();
        vm.stopPrank();

        console2.log("Fee before: ", feeBeforeUpgrade);
        console2.log("Fee after: ", feeAfterUpgrade);
        assert(feeBeforeUpgrade != feeAfterUpgrade);
    }
```

Also, you can see storage layout difference by running `forge inspect ThunderLoan storage` and `forge inspect ThunderLoanUpgraded storage`

Recommended Mitigation: If you must remove the storage variable, leave it as blank as to not mess up the storage slots.

```
- uint256 private s_flashLoanFee;
- uint256 public constant FEE_PRECISION = 1e18;
+ uint256 private s_blank;
+ uint256 private s_flashLoanFee;
+ uint256 public constant FEE_PRECISION = 1e18;
```

Medium

[M-1] Centralization risk for trusted owners

Impact: Contracts have owners with privileged rights to perform admin tasks and need to be trusted to not perform malicious updates or drain funds.

Instances (2):

```
File: src/protocol/ThunderLoan.sol

223:     function setAllowedToken(IERC20 token, bool allowed) external
    onlyOwner returns (AssetToken) {

261:     function _authorizeUpgrade(address newImplementation) internal
     override onlyOwner { }
```

Contralized owners can brick redemptions by disapproving of a specific token

[M-2] Using TSwap as price oracle leads to price and oracle manipulation attacks

Description: The TSwap protocol is a constant product formula based AMM (automated market maker). The price of a token is determined by how many reserves are on either side of the pool. Because of this, it is easy for malicious users to manipulate the price of a token by buying or selling a large amount of the token in the same transaction, essentially ignoring protocol fees.

Impact: Liquidity providers will drastically reduced fees for providing liquidity.

Proof of Concept:

The following all happens in 1 transaction.

1. User takes a flash loan from `ThunderLoan` for 1000 `tokenA`. They are charged the original fee `fee1`. During the flash loan, they do the following:
    1. User sells 1000 `tokenA`, tanking the price.
    2. Instead of repaying right away, the user takes out another flash loan for another 1000 `tokenA`.
        1. Due to the fact that the way `ThunderLoan` calculates price based on the `TSwapPool` this second flash loan is substantially cheaper.

```
1   function getPriceInWeth(address token) public view returns (
       uint256) {
2    address swapPoolOfToken = IPoolFactory(s_poolFactory).
        getPool(token);
3 @>      return ITSwapPool(swapPoolOfToken).
       getPriceOfOnePoolTokenInWeth();
4   }
```

    3. The user then repays the first flash loan, and then repays the second flash loan.

I have created a proof of code located in my `audit-data` folder. It is too large to include here.

Recommended Mitigation: Consider using a different price oracle mechanism, like a Chainlink price feed with a Uniswap TWAP fallback oracle.

Low

[L-1]: Centralization Risk for trusted owners

Contracts have owners with privileged rights to perform admin tasks and need to be trusted to not perform malicious updates or drain funds.

6 Found Instances

- Found in src/protocol/ThunderLoan.sol Line: 244

```
1       function setAllowedToken(IERC20 token, bool allowed) external
            onlyOwner returns (AssetToken) {
```

- Found in src/protocol/ThunderLoan.sol Line: 270

```
1       function updateFlashLoanFee(uint256 newFee) external onlyOwner
            {
```

- Found in src/protocol/ThunderLoan.sol Line: 298

```
1       function _authorizeUpgrade(address newImplementation) internal
            override onlyOwner { }
```

- Found in src/upgradedProtocol/ThunderLoanUpgraded.sol Line: 238

```
1       function setAllowedToken(IERC20 token, bool allowed) external
            onlyOwner returns (AssetToken) {
```

- Found in src/upgradedProtocol/ThunderLoanUpgraded.sol Line: 264

```
1       function updateFlashLoanFee(uint256 newFee) external onlyOwner
            {
```

- Found in src/upgradedProtocol/ThunderLoanUpgraded.sol Line: 287

```
1       function _authorizeUpgrade(address newImplementation) internal
            override onlyOwner { }
```

[L-2]: Missing checks for `address(0)` when assigning values to address state variables

Check for `address(0)` when assigning values to address state variables.

1 Found Instances

- Found in src/protocol/OracleUpgradeable.sol Line: 16

```
1           s_poolFactory = poolFactoryAddress;
```

[L-3]: `public` functions not used internally could be marked `external`

Instead of marking a function as **`public`**, consider marking it as `external` if it is not used internally.

6 Found Instances

- Found in src/protocol/ThunderLoan.sol Line: 236

```
1       function repay(IERC20 token, uint256 amount) public {
```

- Found in src/protocol/ThunderLoan.sol Line: 282

```
1       function getAssetFromToken(IERC20 token) public view returns (
            AssetToken) {
```

- Found in src/protocol/ThunderLoan.sol Line: 286

```
1       function isCurrentlyFlashLoaning(IERC20 token) public view
            returns (bool) {
```

- Found in src/upgradedProtocol/ThunderLoanUpgraded.sol Line: 230

```
1       function repay(IERC20 token, uint256 amount) public {
```

- Found in src/upgradedProtocol/ThunderLoanUpgraded.sol Line: 275

```
1       function getAssetFromToken(IERC20 token) public view returns (
            AssetToken) {
```

- Found in src/upgradedProtocol/ThunderLoanUpgraded.sol Line: 279

```
1       function isCurrentlyFlashLoaning(IERC20 token) public view
            returns (bool) {
```

[L-4]: Event is missing `indexed` fields

Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Each event should use three indexed fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.

9 Found Instances

- Found in src/protocol/AssetToken.sol Line: 31

```
1       event ExchangeRateUpdated(uint256 newExchangeRate);
```

- Found in src/protocol/ThunderLoan.sol Line: 105

```
1       event Deposit(address indexed account, IERC20 indexed token,
            uint256 amount);
```

- Found in src/protocol/ThunderLoan.sol Line: 106

```
1       event AllowedTokenSet(IERC20 indexed token, AssetToken indexed
            asset, bool allowed);
```

- Found in src/protocol/ThunderLoan.sol Line: 107

```
1       event Redeemed(
```

- Found in src/protocol/ThunderLoan.sol Line: 110

```
1       event FlashLoan(address indexed receiverAddress, IERC20
           indexed token, uint256 amount, uint256 fee, bytes params);
```

- Found in src/upgradedProtocol/ThunderLoanUpgraded.sol Line: 105

```
1       event Deposit(address indexed account, IERC20 indexed token,
           uint256 amount);
```

- Found in src/upgradedProtocol/ThunderLoanUpgraded.sol Line: 106

```
1       event AllowedTokenSet(IERC20 indexed token, AssetToken indexed
            asset, bool allowed);
```

- Found in src/upgradedProtocol/ThunderLoanUpgraded.sol Line: 107

```
1       event Redeemed(
```

- Found in src/upgradedProtocol/ThunderLoanUpgraded.sol Line: 110

```
1       event FlashLoan(address indexed receiverAddress, IERC20
           indexed token, uint256 amount, uint256 fee, bytes params);
```

[L-5]: PUSH0 is not supported by all chains

Solc compiler version 0.8.20 switches the default target EVM version to Shanghai, which means that the generated bytecode will include PUSH0 opcodes. Be sure to select the appropriate EVM version in case you intend to deploy on a chain other than mainnet like L2 chains that may not support PUSH0, otherwise deployment of your contracts will fail.

8 Found Instances

- Found in src/interfaces/IFlashLoanReceiver.sol Line: 2

```
1  pragma solidity 0.8.20;
```

- Found in src/interfaces/IPoolFactory.sol Line: 2

```
1 pragma solidity 0.8.20;
```

- Found in src/interfaces/ITSwapPool.sol Line: 2

```
1 pragma solidity 0.8.20;
```

- Found in src/interfaces/IThunderLoan.sol Line: 2

```
1 pragma solidity 0.8.20;
```

- Found in src/protocol/AssetToken.sol Line: 2

```
1 pragma solidity 0.8.20;
```

- Found in src/protocol/OracleUpgradeable.sol Line: 2

```
1 pragma solidity 0.8.20;
```

- Found in src/protocol/ThunderLoan.sol Line: 64

```
1 pragma solidity 0.8.20;
```

- Found in src/upgradedProtocol/ThunderLoanUpgraded.sol Line: 64

```
1 pragma solidity 0.8.20;
```

[L-6]: Empty Block

Consider removing empty blocks.

2 Found Instances

- Found in src/protocol/ThunderLoan.sol Line: 298

```
1     function _authorizeUpgrade(address newImplementation) internal
          override onlyOwner { }
```

- Found in src/upgradedProtocol/ThunderLoanUpgraded.sol Line: 287

```
1     function _authorizeUpgrade(address newImplementation) internal
          override onlyOwner { }
```

[L-7]: Unused Custom Error

it is recommended that the definition be removed when custom error is unused

2 Found Instances

- Found in src/protocol/ThunderLoan.sol Line: 84

```
1      error ThunderLoan__ExhangeRateCanOnlyIncrease();
```

- Found in src/upgradedProtocol/ThunderLoanUpgraded.sol Line: 84

```
1      error ThunderLoan__ExhangeRateCanOnlyIncrease();
```

Informational

[I-1] Poor Test Coverage

```
1  Running tests...
2  | File                                | % Lines        | % Statements
      | % Branches    | % Funcs        |
3  | ----------------------------------- | -------------- | --------------
      | ------------- | -------------- |
4  | src/protocol/AssetToken.sol         | 70.00% (7/10)  | 76.92% (10/13)
      | 50.00% (1/2)  | 66.67% (4/6)   |
5  | src/protocol/OracleUpgradeable.sol | 100.00% (6/6)  | 100.00% (9/9)
      | 100.00% (0/0) | 80.00% (4/5)   |
6  | src/protocol/ThunderLoan.sol        | 64.52% (40/62) | 68.35% (54/79)
      | 37.50% (6/16) | 71.43% (10/14) |
```

[I-2] Not using `__gap[50]` for future storage collision mitigation

[I-3] Different decimals may cause confusion. ie: AssetToken has 18, but asset has 6

[I-4] Doesn't follow https://eips.ethereum.org/EIPS/eip-3156

Recommended Mitigation: Aim to get test coverage up to over 90% for all files.

Gas

[GAS-1] Using bools for storage incurs overhead

Use `uint256(1)` and `uint256(2)` for true/false to avoid a Gwarmaccess (100 gas), and to avoid Gsset (20000 gas) when changing from 'false' to 'true', after having been 'true' in the past. See source.

Instances (1):

```
File: src/protocol/ThunderLoan.sol

98:     mapping(IERC20 token => bool currentlyFlashLoaning) private
    s_currentlyFlashLoaning;
```

[GAS-2] Using `private` rather than `public` for constants, saves gas

If needed, the values can be read from the verified contract source code, or if there are multiple values there can be a single getter function that returns a tuple of the values of all currently-public constants. Saves **3406-3606 gas** in deployment gas due to the compiler not having to create non-payable getter functions for deployment calldata, not having to store the bytes of the value outside of where it's used, and not adding another entry to the method ID table

Instances (3):

```
File: src/protocol/AssetToken.sol

25:     uint256 public constant EXCHANGE_RATE_PRECISION = 1e18;
```

```
File: src/protocol/ThunderLoan.sol

95:     uint256 public constant FLASH_LOAN_FEE = 3e15; // 0.3% ETH fee

96:     uint256 public constant FEE_PRECISION = 1e18;
```

[GAS-3] Unnecessary SLOAD when logging new exchange rate

In `AssetToken::updateExchangeRate`, after writing the `newExchangeRate` to storage, the function reads the value from storage again to log it in the `ExchangeRateUpdated` event.

To avoid the unnecessary SLOAD, you can log the value of `newExchangeRate`.

```
  s_exchangeRate = newExchangeRate;
- emit ExchangeRateUpdated(s_exchangeRate);
+ emit ExchangeRateUpdated(newExchangeRate);
```