Lead Auditors:

- Dionis Xhaferi

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

Liquidity providers can `deposit` assets into `ThunderLoan` and be given `AssetTokens` in return. These `AssetTokens` gain interest over time depending on how often people take out flash loans!

Disclaimer

The DX team 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 the team 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
	High	H	H/M	M
Likelihood	Medium	H/M	M	M/L
	Low	M	M/L	L

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

Audit Details

- Commit Hash: 8803f851f6b37e99eab2e94b4690c8b70e26b3f6

Scope

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

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 was a very informational codebase. I learned alot from flash laons, storage collision, and oracle/reward manipulation.

Issues found

Severity	Number of issues found
High	2
Medium	2
Low	9
Gas	3
Info	0
Total	16

Findings

[H-1] The `ThunderLoan::UpdateExchangeRate` in the `deposit` function causes protocol to think it has more fees than it really does, whcih blocks redemptions and incorrectly sets the exchange rate.

Description: In the Thuderloan system, the `exchangeRate` os responsible for calculating the exchange rate between assetTokens and underlying tokens. In a way, its responsible for keeping track of how many fees to give to liquidity providers.

However, the `deposit` function updates the state 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: There are several impacts to this bug. The redeem function is blocked because the owed tokens is more than it has. Rewards are incorrectly calculated. This leads to Liquidity providers getting way more or less than deserved.

Proof of Concept:

1. LP deposits
2. User takes out flash loan
3. LP unable to claim fees

Place the following in `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

```
 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);
    }
```

[H-2] Mixing up variable location causes storage collisions in ThunderLoan:_flashLoanFee and ThunderLoan:_currentlyFlashLoaning

Description: ThunderLoan.sol has two variables in the following order:

```
uint256 private s_feePrecision;
uint256 private s_flashLoanFee; // 0.3% ETH fee
```

However, the expected upgraded contract ThunderLoanUpgraded.sol has them in a different order.

```
uint256 private s_flashLoanFee; // 0.3% ETH fee
uint256 public constant FEE_PRECISION = 1e18;
```

Due to how Solidity storage works, after the upgrade, the s_flashLoanFee will have the value of s_feePrecision. You cannot adjust the positions of storage variables when working with upgradeable contracts.

Impact: After upgrade, the s_flashLoanFee will have the value of s_feePrecision. This means that users who take out flash loans right after an upgrade will be charged the wrong fee. Additionally the s_currentlyFlashLoaning mapping will start on the wrong storage slot.

Proof of Code:

Code You can also see the storage layout difference by running forge inspect ThunderLoan storage and forge inspect ThunderLoanUpgraded storage

Recommended Mitigation: Do not switch the positions of the storage variables on upgrade, and leave a blank if you're going to replace a storage variable with a constant. In ThunderLoanUpgraded.sol:

```
-    uint256 private s_flashLoanFee; // 0.3% ETH fee
-    uint256 public constant FEE_PRECISION = 1e18;
+    uint256 private s_blank;
+    uint256 private s_flashLoanFee;
+    uint256 public constant FEE_PRECISION = 1e18;
```

[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.

```
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:
User sells 1000 tokenA, tanking the price.
Instead of repaying right away, the user takes out another flash loan for another 1000 tokenA.
```

```
Due to the fact that the way ThunderLoan calculates price based on the TSwapPool this second flash loan is
substantially cheaper.
    function getPriceInWeth(address token) public view returns (uint256) {
        address swapPoolOfToken = IPoolFactory(s_poolFactory).getPool(token);
@>      return ITSwapPool(swapPoolOfToken).getPriceOfOnePoolTokenInWeth();
    }
```

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

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

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: 239

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

- Found in src/protocol/ThunderLoan.sol Line: 265

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

- Found in src/protocol/ThunderLoan.sol Line: 292

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

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

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

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

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

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

```
    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

```
        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: 231

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

- Found in src/protocol/ThunderLoan.sol Line: 276

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

- Found in src/protocol/ThunderLoan.sol Line: 280

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

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

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

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

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

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

```
    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

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

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

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

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

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

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

```
    event Redeemed(
```

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

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

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

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

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

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

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

```
    event Redeemed(
```

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

```
    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

```
pragma solidity 0.8.20;
```

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

```
pragma solidity 0.8.20;
```

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

```
pragma solidity 0.8.20;
```

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

```
pragma solidity 0.8.20;
```

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

```
pragma solidity 0.8.20;
```

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

```
pragma solidity 0.8.20;
```

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

```
pragma solidity 0.8.20;
```

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

```
pragma solidity 0.8.20;
```

L-6: Empty Block

Consider removing empty blocks.

▶ 2 Found Instances

- Found in src/protocol/ThunderLoan.sol Line: 292

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

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

```
    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

```
    error ThunderLoan__ExhangeRateCanOnlyIncrease();
```

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

```
    error ThunderLoan__ExhangeRateCanOnlyIncrease();
```

L-8: Unused Imports

Redundant import statement. Consider removing it.

▶ 1 Found Instances

- Found in src/interfaces/IFlashLoanReceiver.sol Line: 4

```
import { IThunderLoan } from "./IThunderLoan.sol";
```

L-9: State variable changes but no event is emitted.

State variable changes in this function but no event is emitted.

▶ 4 Found Instances

- Found in src/protocol/ThunderLoan.sol Line: 140

```
    function initialize(address tswapAddress) external initializer {
```

- Found in src/protocol/ThunderLoan.sol Line: 265

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

- Found in src/upgradedProtocol/ThunderLoanUpgraded.sol Line: 140

```
    function initialize(address tswapAddress) external initializer {
```

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

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

[G-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;

[G-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;
```

[G-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);
```