Protocol Audit Report

Prepared by: Bug Pirate

Table of Contents

- Table of Contents
- Disclaimer
- Risk Classification
- Audit Details
  - Scope
- Protocol Summary
  - Roles
- Executive Summary
  - Issues found
- Findings
  - High
  - [H-1] Erroneous `ThunderLoan::updateExchangeRate` in the `deposit` function causes protocol to think it has more fee than it really does, which blocks redemption and incorrectly sets the exchange rate.
  - [H-2] Mixing up viariable location causes storage collision in `thunderloan::s_flashLoneFee` and `ThunderLoan::s_currentlyFlashLoanding`, freezing protocol.
  - [H-3] By calling a flasloan and then `ThundeLoan::deposit` instead of `ThunderLoan::repay` users can steal all funds from the protocol.
  - [H-4] getPriceOfOnePoolTokenInWeth uses the Tswap price which doesnt account for decimals, aslo fee precision is 18 detaimals.
  - Medium
  - [M-1] Using Tswap as a price oracle leads to price and oracle manipulation attack
  - Low
  - [L-1]: Missing checks for `address(0)` when assigning values to address state variables
  - L-2: `public` functions not used internally could be marked `external`
  - L-3: Event is missing `indexed` fields
  - L-4: PUSH0 is not supported by all chains
  - L5: Empty Block
  - L-6: Unused Custom Error
  - L-7: Unused Imports
  - L-8: State variable changes but no event is emitted.

Disclaimer

The YOUR_NAME_HERE 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

The findings described in this document correspond the following commit hash:

```
commit hash ` 8803f851f6b37e99eab2e94b4690c8b70e26b3f6`
```

Scope

```
src/interfaces/IFlashLoanReceiver.sol
src/interfaces/IPoolFactory.sol
src/interfaces/ITSwapPool.sol
src/interfaces/IThunderLoan.sol
src/protocol/AssetToken.sol
src/protocol/OracleUpgradeable.sol
src/protocol/ThunderLoan.sol
src/upgradedProtocol/ThunderLoanUpgraded.sol
```

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!

What is a flash loan?

A flash loan is a loan that exists for exactly 1 transaction. A user can borrow any amount of assets from the protocol as long as they pay it back in the same transaction. If they don't pay it back, the transaction reverts and the loan is cancelled.

Users additionally have to pay a small fee to the protocol depending on how much money they borrow. To calculate the fee, we're using the famous on-chain TSwap price oracle.

We are planning to upgrade from the current `ThunderLoan` contract to the `ThunderLoanUpgraded` contract. Please include this upgrade in scope of a security review.

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

Issues found

Severity	Number of issues found
High	4
Medium	1
Low	7
Info	0
Gas	0
Total	12

Findings

High

[H-1] Erroneous `ThunderLoan::updateExchangeRate` in the `deposit` function causes protocol to think it has more fee than it really does, which blocks redemption and incorrectly sets the exchange rate.

Description: In `ThunderLoan::updateExchangeRate` the protocol calculates the exchange rate by dividing the total fee by the total deposits. However, the protocol does not account for the fact that the fees re not yet distributed to the liquidity providers. This mean the exhange rate has been calculated as if the fees have already been distributed, which causes the exchange rate to be high, making the protocol think it has more money than it really does, which block redemption and incorrectly sets the exchange rate.

```
 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 serveral impacts of this bug.

1. The `redeem` function is blocked, because the protocol think the owned token is more than what actually has.
2. Rewards are incorrctly calculated, which leads to users potencially gettingway more or lee 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

```
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 exchnage 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);
        assetToken.mint(msg.sender, mintAmount);

-       uint256 calculatedFee = getCalculatedFee(token, amount);

-        assetToken.updateExchangeRate(calculatedFee);


        token.safeTransferFrom(msg.sender, address(assetToken), amount);
    }
```

[H-2] Mixing up viariable location causes storage collision in `thunderloan::s_flashLoneFee` and `ThunderLoan::s_currentlyFlashLoanding`, freezing protocol.

Description`THunderLoan.sol` has two viraible in the following order;

```
  uint256 private s_feePrecision;

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

However, the 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 storage works in solidty, the `s_flashLoanFee` will have the value of `s_feePrecision`. You can not change the position of storage viriables, and removing storage viariables for the constant variables, breakes the storage locations as well.

ImpactAfter the upgraded, the `s_flashLoanFee` will have the value of `s_feePrecision`. This means that users who take out flash loans right after the upgrade will be changed wrong fees.

proof of Concept

▶ POC

```
import { ThunderLoanUpgraded} from
"src/upgradedProtocol/ThunderLoanUpgraded.sol";
.
.
.
 function testUpgradeBreaks() public{
        uint256 feebeforeUpgrade = thunderLoan.getFee();
        vm.startPrank(thunderLoan.owner());
        ThunderLoanUpgraded upgrade = new  ThunderLoanUpgraded();
        thunderLoan.upgradeToAndCall(address(upgrade), "");
        uint256 feeAfterUpgrade = thunderLoan.getFee();
        vm.stopPrank();

        console2.log("fee Before Upgrade:", feebeforeUpgrade );
        console2.log("fee after Upgrade:", feeAfterUpgrade );

        assert(feebeforeUpgrade != feeAfterUpgrade);
 }

  //fee Before Upgrade: 3000000000000000
  //fee after Upgrade: 1000000000000000000
```

you can also see the storage layout different by running `forge inspect ThunderLoan stoarge` and `forge inspect ThunderLoanUpgraded storage`

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

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

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

[H-3] By calling a flasloan and then `ThundeLoan::deposit` instead of `ThunderLoan::repay` users can steal all funds from the protocol.

Decription The `ThunderLoan` contract is vulnerable to a malicious exploitation that allows an attacker to abuse the flashloan feature. Specifically, a user can:

1. Take a flashloan.
2. Instead of repaying the borrowed amount using the repay function, deposit the funds into the protocol using the deposit function, which doesn't correctly enforce repayment of flashloaned funds.
3. Redeem the "deposited" funds to withdraw more than what they originally flashloaned. This results in a net gain for the attacker, effectively stealing funds from the protocol. The root cause of this vulnerability lies in the lack of proper checks in the flashloan repayment mechanism and insufficient distinction between deposit and repay actions in the protocol logic.

Impact This vulnerability allows any malicious user to drain all funds from the ThunderLoan protocol. It poses an existential risk to the protocol as an attacker can:

- Drain user deposits.
- Manipulate the protocol’s balance to withdraw more than their fair share.

Proof of Concept The following steps outline how the exploit can be executed:

1. Deploy a malicious smart contract (HackerIn) that interacts with ThunderLoan.
2. Initiate a flashloan using the malicious contract as the borrower.
3. Inside the malicious contract:
4. Call deposit instead of repay with the flashloaned funds.
5. Claim deposited funds to withdraw the stolen funds.
6. Return from the flashloan call, leaving the protocol insolvent.

▶ POC

```
function testUserDepositInsteadOfRepayToStealFund()
        public
        setAllowedToken
        hasDeposits
    {
        vm.startPrank(user);

        uint256 amountToBorrow = 50e18;
        uint256 fee = thunderLoan.getCalculatedFee(tokenA, amountToBorrow);

        HackerIn hacker = new HackerIn(address(thunderLoan));

        uint256 extraFeeBuffer = 1e18; // Add 1 token buffer
        tokenA.mint(address(hacker), fee + extraFeeBuffer);

        thunderLoan.flashloan(address(hacker), tokenA, amountToBorrow, "");
```

```
        hacker.redeemMoney();
        vm.stopPrank();
        assert(tokenA.balanceOf(address(hacker)) > 50e18 + fee);
    }
```

[H-4] getPriceOfOnePoolTokenInWeth uses the Tswap price which doesnt account for decimals, aslo fee precision is 18 detaimals.

Medium

[M-1] Using Tswap as a price oracle leads to price and oracle manipulation attack

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

Impact Liquidity providers will drastically reduce fees for the liquidity pool.

Proof Of Concept

▶ POC

```
function test__priceManipulation() public {
        //1. set up contract
        thunderLoan = new ThunderLoan();
        tokenA = new ERC20Mock();
        weth = new ERC20Mock();
        proxy = new ERC1967Proxy(address(thunderLoan), "");

        BuffMockPoolFactory pf = new BuffMockPoolFactory(address(weth));
        // Create a Tswap Dex between WETH / TokenA
        address tswapPool = pf.createPool(address(tokenA));
        thunderLoan = ThunderLoan(address(proxy));
        thunderLoan.initialize(address(pf));

        //2 Fund Tswap
        vm.startPrank(liquidityProvider);
        tokenA.mint(liquidityProvider, 1000e18);
        tokenA.approve(address(tswapPool), 1000e18);
        weth.mint(liquidityProvider, 1000e18);
        weth.approve(address(tswapPool), 1000e18);
        BuffMockTSwap(tswapPool).deposit(
            1000e18,
            1000e18,
            1000e18,
            block.timestamp
        );
        vm.stopPrank();
```

```
        //3. Fund ThunderLOan
        //setAllowed
        vm.prank(thunderLoan.owner());
        thunderLoan.setAllowedToken(tokenA, true);
        //fund
        vm.startPrank(liquidityProvider);
        tokenA.mint(liquidityProvider, 1000e18);
        tokenA.approve(address(thunderLoan), 1000e18);
        thunderLoan.deposit(tokenA, 1000e18);
        vm.stopPrank();

        //4. We are going tp take out 2 flash loans
        // a. To nuke the price of the WETH/TokenA on tSawp
        // b. show that doing so greatly reduces the fees we pay on Tswap
        uint256 normalFeeCost = thunderLoan.getCalculatedFee(tokenA,
100e18);
        console2.log("Normal Fee is:", normalFeeCost);

        uint256 amountToBorrow = 50e18; // doing this twice to our hacker
brains hehe
        HackerOnFlee hack = new HackerOnFlee(
            address(tswapPool),
            address(thunderLoan),
            address(thunderLoan.getAssetFromToken(tokenA))
        );

        vm.startPrank(user);
        tokenA.mint(address(hack), 100e18);
        thunderLoan.flashloan(address(hack), tokenA, amountToBorrow, "");
        vm.stopPrank();

        uint256 attackFee = hack.Feeone() + hack.FeeTwo();
        console2.log("Attcak Fee is:", attackFee);
        assert(attackFee < normalFeeCost);
    }

    function testUserDepositInsteadOfRepayToStealFund()
        public
        setAllowedToken
        hasDeposits
    {
        vm.startPrank(user);

        uint256 amountToBorrow = 50e18;
        uint256 fee = thunderLoan.getCalculatedFee(tokenA, amountToBorrow);

        HackerIn hacker = new HackerIn(address(thunderLoan));

        uint256 extraFeeBuffer = 1e18; // Add 1 token buffer
        tokenA.mint(address(hacker), fee + extraFeeBuffer);

        thunderLoan.flashloan(address(hacker), tokenA, amountToBorrow, "");
        hacker.redeemMoney();
        vm.stopPrank();
```

```
        assert(tokenA.balanceOf(address(hacker)) > 50e18 + fee);
    }
```

Recommended Mitigation consider using a diffrent price oracle mechanism, like a chinlink prize feed with a uniswap TwAP fallback oracle.

Low

[L-1]: 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-2: `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-3: 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-4: 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;
```

L5: 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-6: 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-7: 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-8: 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 {
```