Fyde Security Review

Pashov Audit Group

Conducted by: peanuts, T1MOH, unforgiven

March 8th 2024 - March 14th 2024

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

3. Introduction

A time-boxed security review of the **FydeTreasury/core-refactored** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About Fyde

Fyde Protocol is a treasury management solution which incentivizes users to deposit into a diversified pool of assets, while allowing for the retention of governance rights and preserving the control over their allocations. Tokens compose a portfolio management strategy tracking the market performance of each asset and implementing a dynamic weight mechanism for the pool.

5. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hash - 0ae92bfe60c660f56747009a6c3ef3b4013eb0ee

fixes review commit hash - 5ddd2edd31ad44c022652fc05f5bcb9f910a8178

Scope

The following smart contracts were in scope of the audit:

- `RelayerV2`
- `TaxModule`
- `OracleModule`
- `QuarantineList`
- `UniswapAdapter`
- `ChainlinkAdapter`
- `Ownable`
- `AccessControl`

7. Executive Summary

Over the course of the security review, peanuts, T1MOH, unforgiven engaged with Fyde to review Fyde. In this period of time a total of **19** issues were uncovered.

Protocol Summary

Protocol Name	Fyde
Repository	https://github.com/FydeTreasury/core-refactored
Date	March 8th 2024 - March 14th 2024
Protocol Type	Asset management strategy

Findings Count

Severity	**Amount**
Critical	1
High	1
Medium	11
Low	6
Total Findings	**19**

Summary of Findings

ID	Title	Severity	Status
[C-01]	Attacker can manipulate the protocols AUM	Critical	Resolved
[H-01]	User can divide big deposit/withdraw into small parts to reduce tax	High	Resolved
[M-01]	Attacker can inflate share price and cause loss for users	Medium	Acknowledged
[M-02]	User may lose funds if there are duplicate tokens in the deposit list	Medium	Resolved
[M-03]	Slippage check may be inefficient in deposit() when _keepGovRights=True	Medium	Acknowledged
[M-04]	AUM value update can be sandwiched to extract value from the protocol	Medium	Acknowledged
[M-05]	Guardian can't unpause the RelayerV2	Medium	Resolved
[M-06]	Centralization risk that causes fund loss or lock	Medium	Acknowledged
[M-07]	OracleModule assumes that all Chainlink feeds have a heartbeat of 24 hours	Medium	Acknowledged
[M-08]	Users cannot burn their TRSY in exchange for assets when RelayerV2.withdraw() is paused	Medium	Acknowledged
[M-09]	Assets with more than 18 decimals are incorrectly handled in UniswapAdaptor	Medium	Resolved

[M-10]	Attacker can reenter RelayerV2 with ERC777 tokens and steal funds	Medium	Acknowledged
[M-11]	Code doesn't validate stale AUM values	Medium	Acknowledged
[L-01]	Protocol can be exploited by malicious Keeper	Low	Acknowledged
[L-02]	Dangerous use of payable functions when interacting with RelayerV2	Low	Resolved
[L-03]	ChainlinkAdapter does not check for round completeness, which may lead to stale data	Low	Acknowledged
[L-04]	First depositor will always pay taxes	Low	Acknowledged
[L-05]	Target concentration for deposits and withdraws is not checked to be equal to 100%	Low	Acknowledged
[L-06]	When Chainlink price isn't available, an attacker can extract value from the protocol	Low	Acknowledged

8. Findings

8.1. Critical Findings

[C-01] Attacker can manipulate the protocols AUM

Severity

Impact: High

Likelihood: High

Description

In the new design, the AUM value will be set and updated by off-chain bot. The issue is that the bot defines the absolute value of the AUM and the Fyde contract changes the current AUM based on the bot-provided value so the attacker can change the current AUM by front-running and causing the wrong AUM value when the bot's transaction executes. This is POC:

1. Suppose the current AUM is $100K and the attacker deposited $20K into the protocol.
2. The AUM decreases to $98K and the off-chain bot sends a transaction to update the AUM to $98K.
3. The attacker was watching the mempool and saw the AUM update transaction and front-run it and withdrew $4K from the protocol. Now the protocol AUM is $96K and the real AUM is $94K.
4. When the AUM update transaction executes the AUM would be set to $98K and the difference between protocol AUM and real AUM would be `98K - 94K = 4K`.
5. So while in the beginning, the protocol's AUM had $2K difference(about 2%) in the end it has $4K(more than 4%) difference.
6. The attacker can perform this attack again and again for each AUM update and cause more differences between the protocol's AUM and real AUM. As

their difference increases protocol would calculate fewer shares for the attacker when he withdraws tokens.

In fact, each user would have an incentive to withdraw before the AUM update transaction because AUM is getting decreased. So the protocol would be in an unstable situation and the difference between real AUM and the protocol's AUM would increase.

Recommendations

The code should allow for AUM differential updates. For example, the off-chain bot should set the amount that AUM should be decreased or increased, in this way the AUM value always changes in the correct direction.

8.2. High Findings

[H-01] User can divide big deposit/withdraw into small parts to reduce tax

Severity

Impact: Medium

Likelihood: High

Description

Tax to pay depends on how much deposit/withdraw makes current asset concentration deviate from the target value, therefore big amounts will pay bigger tax. However, users can supply a request array with the same assets, which can significantly reduce the tax to pay. For example, when asset concentration is too low, the tax to pay on deposit for this asset can equal to 0 to incentivize deposits. Part of fee calculation that uses delta concentration of deposit works incorrectly when an array with the same assets is supplied. Because initial concentration is used, it's not updated upon iteration:

```
function _computeDepositTaxableAmount(
    ProcessParam memory processParam,
    uint256 protocolAUM,
    uint256 totalUsdDeposit
  ) internal pure returns (ProcessParam memory) {
    int256 deltaConc = protocolAUM.toInt()
      * (processParam.currentConc.toInt() - processParam.targetConc.toInt
      ()) / 1e20;
```

As a result, delta concentration is undervalued which results in lower tax than intended.

You can see the difference in amounts in these tests.

Recommendations

Add duplicate check in `deposit()` and `withdraw()`.

8.3. Medium Findings

[M-01] Attacker can inflate share price and cause loss for users

Severity

Impact: High

Likelihood: Low

Description

When the share amount is calculated and `totalSupply` is zero, the code uses the deposit amount and doesn't add extra precision, this will give the attacker the ability to inflate the share price up to `1000 * 1e18` and cause a loss for depositors because of rounding error.

```
function _convertToShares
    (uint256 _usdValue, uint256 _usdAUM, uint256 totalTrsySupply)  {
    return totalTrsySupply == 0 ? _usdValue :
      (_usdValue * totalTrsySupply) / _usdAUM;
  }
```

This is one example that this could happen:

1. Suppose there is one token(T1) with 100% weight in the Fyde token list and there is no tax. T1 price is 1.
2. Attacker deposits `10 * 1e18` T1 tokens into the Fyde and receive `10 * 1e18` shares. The Fyde's AUM will be `10 * 1e18`.
3. Now the price of token T1 increased 10% and its 1.1 and Fyde's AUM isn't updated yet and AUM is still `10 * 1e18`.
4. Attacker would withdraw `9.1 * 1e18` T1 tokens and it's value with be `1.1 * 9.09 * 1e18 = 10 * 1e18` which is equal to AUM so code would burn `10 * 1e18` share from attacker.
5. Now `totalSupply` is equal to 0 but real AUM is `1e18`.
6. Attacker would deposit 1 wei T1 token and the share price would be about `1e18`.

Recommendations

Add more precision to the share function _convertToShares(uint256 _usdValue, uint256 _usdAUM, uint256 totalTrsySupply) { return totalTrsySupply == 0 ? 10^18 * _usdValue : (_usdValue * totalTrsySupply) / _usdAUM; }

[M-02] User may lose funds if there are duplicate tokens in the deposit list

Severity

Impact: High

Likelihood: Low

Description

RelayerV2 doesn't check that token lists don't have duplicate items and doesn't support duplicate items in the list. If token lists have duplicate items then users may lose funds because some logics won't work as designed. For example when users set `_keepGovRights` as True, code loops through the tokens list and unstake all the RelayerV2 remaining balance, and it will cause the user to not receive the duplicate amounts. This is POC:

1. User would call `deposit()` with token list `[T1, T1]` and `_keepGovRights = True`.
2. Code would deposit tokens into Fyde contract and receive `2* sharesAfterTax` sTrsy tokens.
3. Now code would loop through token list to transfer user sTrsy and also unstake the tax sTrsy.
4. For the first T1 code would transfer `sharesAfterTax` and also unstake all the remaining balance which is `sharesAfterTax`.
5. For the second T2 because the balance is 0 code won't transfer any token to the user. So User would receive `sharesAfterTax` while he should have received `2 * sharesAfterTax`.

Recommendations

Make code to support duplicate items or have some checks to ensure the list has no duplicate items or warn users about this risk.

[M-03] Slippage check may be inefficient in `deposit()` when `_keepGovRights=True`

Severity

Impact: Medium

Likelihood: Medium

Description

In `deposit()` function code checks that `slippageChecker` with the sum of the minted shares `sharesToMint`, the issue is that when `_keepGovRights = True` then the code sends users different sTrsy tokens and there is no slippage check for their values. So even so some of the tokens may be higher than `slippageChecker` but the user may receive unfavorable sTrsy tokens. This is an issue because the token prices in the Fyde can be higher/lower than real token prices and the code would mint more/less sTrsy tokens for those tokens.

Also, because the code use `mint(contract balance, sharesAfterTax)` to calculate sTrsy transfer amount, so the total transferred amount can be lower than `sharesToMint` and the code should check real transferred amounts with `slippageChecker`.

Recommendations

Users should be able to add a slippage amount for each individual sTrsy they receive.

[M-04] AUM value update can be sandwiched to extract value from the protocol

Severity

Impact: High

Likelihood: Low

Description

Protocol requires the off-chain bot to update AUM value so it can compute the share amount in deposited/withdrawal function and the attacker can use this to his advantage and perform the sandwich attack to extract value from the protocol. This is the POC:

1. Suppose the price of T1 token is 1 and protocol's `AUM` is $98K and `totalShare` is $98K.
2. Attacker monitoring mempool saw the off-chain bot AUM update transaction to set AUM as $102K.
3. Now attacker would perform this sandwich attack:
4. Before AUM update attacker would deposit 2K T1 tokens and receive 2K shares and `totalShare` and `AUM` will be $100K.
5. Then AUM update tx will be executed and AUM would be set as $102K.
6. Now attacker would withdraw 2.04K T1 tokens and code would calculate shares as `shares = token * price * totalShare / AUM = 2.04 * 1 * 100K / 102K = 2K` and transfer 2.04K T1 token while burning user's 2K shares.
7. At the end user received 0.04K tokens more.

Recommendations

Don't allow withdraw and deposit in the same block for each user to make the attack harder. Use the same AUM value for the whole block. Add delay for withdrawal or deposit.

[M-05] Guardian can't unpause the RelayerV2

Severity

Impact: Medium

Likelihood: Medium

Description

According to the docs, Guardian should be able to pause/unpause the protocol.

> Guard : Able to pause/unpause the protocol and quarantine assets.

The issue is that the unpause functions access controls don't allow Guardian to unpause the protocol.

```
///@notice Unpause the protocol
  function unpauseProtocol() external onlyOwner {
    paused = false;
    emit Unpause(block.timestamp);
  }

  ///@notice Unpause the swaps
  function unpauseSwap() external onlyOwner {
    swapPaused = false;
    emit Unpause(block.timestamp);
  }
```

Recommendations

Allow Guardian to unpause the protocol.

[M-06] Centralization risk that causes fund loss or lock

Severity

Impact: High

Likelihood: Low

Description

There are multiple roles that are crucial for the protocol to work properly like the off-chain bot that updates AUM and the price setter role. If these roles comprised or work expectedly then contract crucial features would not work and users may lose funds. For example:

1. If an off-chain AUM updater bot is comprised then the attacker can increase/decrease AUM by 2% in each block, so after 10min can increase AUM by 150% and practically withdraw all of the contract funds. To avoid

this, the code should add more limits for AUM changes, like don't let AUM be changed more than 10% in 5 min.

2. If an off-chain price setter is comprised, then it can set the wrong manual price for tokens and the attacker can steal protocol funds by interacting with the protocol. To avoid this the min/max limit price for each token should be updated frequently so the manual price setter can't cause more damage.

Recommendations

Add max/min limit for what off-chain operator can set and also update them frequently. Also add longer period price change detection, for example, don't let AUM be changed more than 10% in 5 min.

[M-07] OracleModule assumes that all Chainlink feeds have a heartbeat of 24 hours

Severity

Impact: High

Likelihood: Low

Description

In `OracleModule.sol`, there is a function to set the stale period of the chainlink feed.

```
function setStalePeriod(uint32 _stalePeriod) public onlyOwner {
    stalePeriod = _stalePeriod;
  }
```

This stale period will check whether the `updatedAt` variable returned from the `latestRoundData()` call is up to date.

```
function _isStale(uint256 timestamp) internal view returns (bool) {
    return block.timestamp - timestamp <= stalePeriod ? false : true;
  }
```

The `stalePeriod` is set at 90000 seconds (25 hours). OracleModule.sol is hardcoded to 25 hours, assuming most of the Chainlink data feeds have 24 hours deviation threshold:

```
///@notice Period for checking if chainlink data is stale
  ///@dev At init set at 25 hours, most of the chainlink feed have an heartbeat
  // of 24h
  uint32 public stalePeriod;
```

However, some assets have 1-hour heartbeat, the most significant is ETH/USD: according to DefiLlama 43% of Fyde TVL is WETH.

The issue is that some Chainlink feeds which are used by the protocol, like ETH-USD and BTC-USD have a heartbeat of (1 hour). In those cases, if the prices derived from Chainlink are stale, the protocol will still assume that the prices are healthy because it sets the `stalePeriod` as 25 hours, which will lead to incorrect accounting when depositing assets for TRSY or burning TRSY for assets.

Recommendations

The protocol has the ability to choose the assets that interact with the protocol and will potentially be using many Chainlink feeds. Consider saving all the Chainlink AggregatorV3Interface into a mapping and using different `stalePeriod` for different heartbeats instead of interacting directly with the AccessControlledOffchainAggregator.

[M-08] Users cannot burn their TRSY in exchange for assets when RelayerV2.withdraw() is paused

Severity

Impact: High

Likelihood: Low

Description

In RelayerV2.sol, the guard, which is appointed by the keeper, can pause the protocol through `pauseProtocol()`.

```
function pauseProtocol() public onlyGuard {
    paused = true;
    emit Pause(block.timestamp);
  }
```

When the protocol is paused, functions with the `whenNotPaused` modifier, such as `withdraw()` and `governanceWithdraw()` cannot be used.

```
function withdraw(UserRequest[] calldata _userRequest, uint256 _maxTRSYToPay)
    external
    payable
>   whenNotPaused
    onlyUser
  {
```

Most of the time, users should always be able to withdraw their funds even if the protocol is paused. They should not depend on the owner/guard to be able to withdraw their funds.

Recommendations

Recommend removing the `whenNotPaused` modifier in the `withdraw()` functions.

[M-09] Assets with more than 18 decimals are incorrectly handled in UniswapAdaptor

Severity

Impact: High

Likelihood: Low

Description

Function `_getUniswapPrice()` is supposed to handle cases when the quote token has more than 18 decimals by downscaling the price. However, it will just revert on the first line because `factor` is `uint`:

```
function _getUniswapPrice
    (address asset, AssetInfo calldata assetInfo, uint32 twapPeriod)
    internal
    view
    returns (uint256)
  {
    uint256 baseAmount = 10 ** assetInfo.assetDecimals;
@>  uint256 factor = (18 - assetInfo.quoteTokenDecimals); // 18 decimals
    uint256 finalPrice;

    uint32[] memory secondsAgos = new uint32[](2);
    secondsAgos[0] = twapPeriod;
    secondsAgos[1] = 0;

    try IUniswapV3Pool(assetInfo.uniswapPool).observe(secondsAgos) returns (
      int56[] memory tickCumulatives, uint160[] memory
    ) {
      int24 tick = _computeTick(tickCumulatives, twapPeriod);

      int256 price = OracleLibrary.getQuoteAtTick(
        tick, baseAmount.to128(), asset, assetInfo.uniswapQuoteToken
      ).toInt();

      finalPrice = factor > 0
        ? price.upscale(factor).toUint()
@>      : price.downscale((-factor.toInt()).toUint()).toUint();
      return finalPrice;
    } catch {
      return finalPrice;
    }
  }
```

Recommendations

Refactor to:

```
uint256 baseAmount = 10 ** assetInfo.assetDecimals;
-   uint256 factor = (18 - assetInfo.quoteTokenDecimals); // 18 decimals
+   int256 factor = 18 - int256(int8
+ (assetInfo.quoteTokenDecimals)); // 18 decimals
    uint256 finalPrice;

    uint32[] memory secondsAgos = new uint32[](2);
    secondsAgos[0] = twapPeriod;
    secondsAgos[1] = 0;

    try IUniswapV3Pool(assetInfo.uniswapPool).observe(secondsAgos) returns (
      int56[] memory tickCumulatives, uint160[] memory
    ) {
      int24 tick = _computeTick(tickCumulatives, twapPeriod);

      int256 price = OracleLibrary.getQuoteAtTick(
        tick, baseAmount.to128(), asset, assetInfo.uniswapQuoteToken
      ).toInt();

      finalPrice = factor > 0
-       ? price.upscale(factor).toUint()
-       : price.downscale((-factor.toInt()).toUint()).toUint();
+       ? price.upscale(factor.toUint()).toUint()
+       : price.downscale((-factor).toUint()).toUint();
      return finalPrice;
    } catch {
      return finalPrice;
    }
  }
```

[M-10] Attacker can reenter RelayerV2 with ERC777 tokens and steal funds

Severity

Impact: High

Likelihood: Low

Description

There's no reentrancy guard in the RelayerV2 contract and if one of the deposit or withdraw tokens had a hook (ERC777 or ...) then it would be possible to reenter the RelayerV2 contract again and steal the funds while the state is wrong. There are multiple ways that attackers can exploit this, one way is to set previous caching prices for tokens for the current operation. This is the POC:

1. Suppose token T1 is ERC777 and has a hook in transfers and it's used in the Fyde contract.

2. First Attacker would deposit T1 token by calling `deposit()`.
3. Code would call `_enableOracleCache(T1)` and OracleModule would set `cacheEnabled = true`.
4. Now when `deposit()` wants to transfer T1 token, the hook function will be called and execution will reach the attacker contract.
5. Now attacker would call `deposit()` and use T2 and T3 tokens to deposit.
6. In this `deposit()` call when code reaches the `OracleModule.useCache()`, because `cacheEnabled = true` so cache price will be used and set for T2 and T3 (`getPriceInUSD()` would return the cache price)
7. In the end attacker was able to set old cache prices for tokens T2 and T3. Attackers can use this to steal funds whenever the old cached prices show very wrong value (For example a token price was very high in the past and it's still in the cache).

There could be other methods to exploit this too.

Recommendations

Add reentrancy guard for RelayerV2 contract

[M-11] Code doesn't validate stale AUM values

Severity

Impact: High

Likelihood: Low

Description

In the new design of the system, the value of the AUM should be updated by the off-chain bot. The issue is that there is no validation time for the AUM value and the code assumes that AUM is valid, this would cause issues when the off-chain bot can't update the AUM value for any reason. There are multiple reasons for AUM to not be updated, for example when the Ethereum network is busy when there is a bug in the off-chain bot, or when it's compromised. This is the POC for this issue:

1. Suppose the AUM is $100K and the off-chain bot updated the protocol AUM value.
2. Suppose the off-chain bot is down and real AUM has reached $120K but the AUM value in protocol state is still $100K.
3. Now Attacker can deposit tokens into the protocol and receive more shares (because AUM is lower in the protocol).
4. Protocol allows users to interact with the contracts even when AUM hasn't been updated for a long time.

Recommendations

Like the manual price in the Oracle which is set by the off-chain operator and has an expiration time the AUM set by the off-chain bot should have a valid period(expiration time) too and the code should check it when using the AUM value. If the AUM value is stale then the code shouldn't allow interactions.

8.4. Low Findings

[L-01] Protocol can be exploited by malicious Keeper

The protocol team is interested to know the worst outcome when Keeper becomes malicious. Let's observe how malicious Keeper can exploit protocol.

Part 1. Keeper is allowed to update Assets Under Management (AUM), however, update must not exceed max allowed deviation from the configured value. let it be N. Part 2. How tax is calculated in deposit/withdraw? A flat tax is always applied, i.e. fixed percent. Also If assets exceed target asset concentration after operation, additional tax is applied. But assume the attacker won't unbalance protocol, so only flat tax is applied on deposit and withdrawal. Part 3. Attack example. Suppose the following scenario:

1. Attacker takes flashloan. Flashloan potential on Mainnet is around 1 Billion USD, for comparison current Fyde TVL is 2 Million USD. Because of step 1 he receives an increased amount of TRSY on deposit to what is intended.
2. Attacker deposits assets in the right portions to minimize tax, so only `flatTax` is applied.
3. Keeper increases AUM to the max allowed value.
4. Now attacker's TRSY tokens after the deposit are more valuable than before because AUM has increased while TRSY supply is the same.
5. Attacker withdraws his TRSY and pays `flatTax`.

Profit depends on the current values of `flatTax` and `AUMDeviationThreshold`. If `flatTax` is relatively big to `AUMDeviationThreshold`, the attack is not profitable. Conduct economic analysis to pick out the right values for them.

[L-02] Dangerous use of payable functions when interacting with RelayerV2

Some functions that the users interact with, like `deposit()`, `withdraw()` and `swap()` , have a payable modifier. The protocol does not seem to interact with

native tokens. If users were to accidentally send native tokens together with their assets, the native tokens would be stuck in the contract.

```
function withdraw(UserRequest[] calldata _userRequest, uint256 _maxTRSYToPay)
    external
>   payable
    whenNotPaused
    onlyUser
  {
```

Consider removing the payable function or having a way to withdraw their native token that is stuck in the contract.

[L-03] ChainlinkAdapter does not check for round completeness, which may lead to stale data

When querying the results from `latestRoundData()`, round completeness is not validated. Not checking for round completeness could lead to stale prices and wrong price return value, or outdated price.

```
try FeedRegistryInterface(clRegistry).latestRoundData
      (asset, clQuoteToken) returns (
      uint80, int256 clPrice, uint256, uint256 updatedAt, uint80
    ) {
```

Validate the data feed for round completeness for all the functions that call `latestRoundData()`.

```
try FeedRegistryInterface(clRegistry).latestRoundData
      (asset, clQuoteToken) returns (
+
+      uint80 roundID, int256 clPrice, uint256, uint256 updatedAt, uint80 answeredInRo
    ) {
+   require(answeredInRound >= roundID, "round not complete");
```

[L-04] First depositor will always pay taxes

For all assets, there will be a target concentration and a current concentration. When protocol AUM is at 0, the asset's current concentration will be at zero. If the first depositor deposits an asset and the target concentration is not at 100%,

then they will incur taxes. The only way for them not to incur taxes is to deposit assets equal to all the target concentration set.

Ensure that the user knows the risk and target concentration of an asset before depositing to prevent taxes from incurring.

```
function _getTargetConcentrationDeposit
    (address _asset) internal view returns (uint256) {
     //@audit - make sure that the targetConcDeposit is at 100% for the first
     // depositor
>    return uint256(taxParams[_asset].targetConcDeposit);
  }
```

The best way to prevent this issue is for the protocol themselves to fund the treasury first and be the first depositor.

[L-05] Target concentration for deposits and withdraws is not checked to be equal to 100%

The owner will set the Tax Params for a whitelisted asset and will indicate the `targetConcDeposit` and `targetConcWithdraw` for each asset in the `setTaxParams()` function.

```
function setTaxParams
    (address[] calldata _assets, TaxParams[] calldata _taxParams)
    external
    onlyOwner
  {
    for (uint256 i; i < _assets.length; ++i) {
      taxParams[_assets[i]] = _taxParams[i];
    }
  }
```

`setTaxParams()` does not check that the targetConcentration of all assets must add up to 100%. For example, if WBTC, WETH, and USDC are whitelisted, then the target concentration set for each should add up to 100%, eg 40%, 40%, 20%. Since there is no check in the function, the owner can potentially set the target concentration of each asset as 100%, 100%, 100% for a total of 300%, which will break the accounting of the protocol.

Ensure that `targetConcDeposit` and `targetConcWithdraw` of all assets always equal to 100%.

[L-06] When Chainlink price isn't available, an attacker can extract value from the protocol

When Chainlink price feed is not available, OracleModule would use Uniswap TWAP price or a manually specified price. During this time if the price of the token has sudden price changes then the attacker can extract value from the protocol. This is the POC:

1. Suppose Chainlink price feed for T1 token is down and Uniswap TWAP price is 1 and protocol uses this price.
2. Suddenly the real-time price of the T1 drops to 0.9 but Uniswap TWAP price is still 1.
3. Now attacker can deposit T1 token into the protocol and the protocol will calculate a higher value for deposited T1 tokens(because TWAP price is behind) and mint more shares for the attacker.
4. The attack would be more impactful if during this time the off-chain bot updates AUM and decreases it based on the new T1 price.

The attack would be possible with manual price too (attacker can front-run manual price updates)

There is no easy fix for this. Check for a Shorter TWAP period (like 30 seconds) to be in the range of 30min TWAP price.