CANTINA

Se7en Seas

Security Review

Cantina Managed review by:

Jonah1005, Lead Security Researcher

Defsec, Security Researcher

Rappie, Associate Security Researcher

April 19, 2024

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

Seven Seas Capital is a collective of seasoned professionals in blockchain, data science, and finance committed to making crypto markets more efficient and transparent.

From Apr 1st to Apr 12th the Cantina team conducted a review of boring-vault on commit hash 939c77e2. The team identified a total of **28** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 8
- Low Risk: 7
- Gas Optimizations: 0
- Informational: 13

2.1 Trust Assumptions

Se7en Seas specifically calls out that the strategist is expected to act in the best interest of the `BoringVault`, and that users interacting with `BoringVault`'s should be aware of this.

3 Findings

3.1 Medium Risk

3.1.1 `UniswapV3DecoderAndSanitizer` allows strategies to drain the vault through providing malicious positions on UniV3

Severity: Medium Risk

Context: UniswapV3DecoderAndSanitizer.sol#L47-L56, UniswapV3DecoderAndSanitizer.sol#L58-L71

Description: The `UniswapV3DecoderAndSanitizer` supports two methods for providing liquidity. The first is through `mint`, where the provided tokens are checked. The second is through `increaseLiquidity`, where it verifies that the owner of the position is the `boringVault`. However, this validation is not sufficient to prevent malicious strategies from draining the `boringVault`

- For `increaseLiquidity`, since the `increaseLiquidity` function only verifies that the owner of the current position is the `boringVault`, an exploiter can send a malicious UniswapV3 position to the `boringVault` and trigger `increaseLiquidity` to have the `boringVault` provide liquidity to the malicious position, thereby enabling the exploiter to profit.

  For example, let's assume the base token of the `Vault` is USDT. The exploiter can first create a fake token, `FAKE`, and establish a `FAKE-USDT` market on UniswapV3. By sending a dust position of `FAKE-USDT` to the `boringVault`, the `UniswapV3DecoderAndSanitizer` would consider it a valid position and proceed to provide liquidity to the `FAKE-USDT` market. The exploiter can then profit by buying all USDT on the `FAKE-USDT` market.

- For `mint`, since the `UniswapV3DecoderAndSanitizer` only checks the tokens of the market. The malicious strategy can provide liquidity at a malicious tick. Providing liquidity at a malicious tick is similar to selling tokens at a really bad price. It creates a arbitrage space for the exploiter to further take profit by selling tokens on the market.

Recommendation: Consider validating the `tick`, `fee`, and `token` when providing liquidity to Uniswap V3 markets.

Se7en Seas: Fixed in commit d400d074 by adding the `operator token0 token1` to the merkle tree, and completely prevents the `FAKE-USDT` attack vector.

Additionally it is important to note that the fix does not address a strategist being able to choose a bad tick range when adding liquidity to Uniswap V3. But using the trust assumption that the strategist will act in the best interest of the BoringVault, this is no longer an issue.

Cantina Managed: Fixed.

3.1.2 The slippage protection in the `MicroManager` does not account for the harvested rewards

Severity: Medium Risk

Context: DexAggregatorUManager.sol#L114, DexSwapperUManager.sol#L131-L136

Description: The `boringVault` allows strategies to execute trades to manage the portfolio. To prevent strategies from draining the BoringVault, the `MicroManager` includes a slippage check to ensure that the value withdrawn is not smaller than the value deposited.

- DexSwapperUManager.sol#L126-L136

```
ERC20 tokenOut = path[path.length - 1];
uint256 tokenOutBalanceDelta = tokenOut.balanceOf(boringVault);

// Make the manage call.
manager.manageVaultWithMerkleVerification(manageProofs, decodersAndSanitizers, targets, targetData,
↪  values);

tokenOutBalanceDelta = tokenOut.balanceOf(boringVault) - tokenOutBalanceDelta;

uint256 tokenOutQuotedInTokenIn = priceRouter.getValue(tokenOut, tokenOutBalanceDelta, path[0]);

if (tokenOutQuotedInTokenIn < amountIn.mulDivDown(1e4 - allowedSlippage, 1e4)) {
    revert DexSwapperUManager__Slippage();
}
```

Since the slippage protection only considers the value of tokens withdrawn and the tokens increased within the swaps, exploiters can steal the harvested rewards.

Let's assume the boringVault holds a large position in Pendle Finance's YT Token and is nearing the settlement time. The exploiter can execute the following steps:

1. Initiate a trade that simply transfers tokens to the exploiter's wallet.
2. Claim rewards for the `boringVault`. Similar to most claimRewards functions, Pendle Finance allows anyone to claim interests for other users through `redeemDueInterestAndRewards`.
3. As the tokens are increased, the slippage protection is bypassed.

Recommendation: Consider sanitizing the entire swap path of a trade to prevent the initiator from gaining control of the flow.

Se7en Seas: Acknowledged. Given the current scope of the audit, as well as how the architecture will be configured for foreseeable launches, this exploit will never be economically viable for the strategist, as management fees will greatly outpace the harvestable rewards that are exploitable. If the strategist were to perform this attack, users would be able to easily see it, and decide to leave the vault as the strategist clearly does not have the BoringVault's best interest in mind. This leads to a decrease in fees earned making it impractical from an economic standpoint.

Cantina Managed: Acknowledged.

3.1.3 Malicious strategies can bypass slippage protection to drain the `BoringVault` with Reentrancy attacks

Severity: Medium Risk

Context: DexAggregatorUManager.sol#L114, DexSwapperUManager.sol#L131-L136

Description: The `boringVault` allows strategies to execute trades to manage the portfolio. To prevent strategies from draining the BoringVault, the `MicroManager` includes a slippage check to ensure that the value withdrawn is not smaller than the value deposited.

- DexSwapperUManager.sol#L126-L136

```
ERC20 tokenOut = path[path.length - 1];
uint256 tokenOutBalanceDelta = tokenOut.balanceOf(boringVault);

// Make the manage call.
manager.manageVaultWithMerkleVerification(manageProofs, decodersAndSanitizers, targets, targetData,
↪  values);

tokenOutBalanceDelta = tokenOut.balanceOf(boringVault) - tokenOutBalanceDelta;

uint256 tokenOutQuotedInTokenIn = priceRouter.getValue(tokenOut, tokenOutBalanceDelta, path[0]);

if (tokenOutQuotedInTokenIn < amountIn.mulDivDown(1e4 - allowedSlippage, 1e4)) {
    revert DexSwapperUManager__Slippage();
}
```

However, due to the lack of re-entrancy protection, the exploiter can interact with protocol within the swap execution and change the token balance, the slippage protection may be off.

A malicious payload could exploit the following scenario:

1. Execute a malicious swap that transfers all tokens to the attacker's wallet.
2. Deposit tokens into the boringVault and receive vault shares.
3. As the balance of the boringVault increases, the slippage protection is bypassed.
4. Withdraw the vault shares and obtain the profit.

Recommendation: Add a protocol-wise reentrancy protection to prevent this attack.

Se7en Seas: Fixed in commit ddf67b36.

The core problem of the present issue and the issue "The slippage protection in the MicroManager does not account for the harvested rewards" is that there exists a permissionless way to increase the token balance of the BoringVault using funds already owned by the BoringVault. The fix above closes off the attack vector where a deposit is made to the BoringVault in order to increase its token balance. Given the current scope of the audit, as well as how the architecture will be configured for foreseeable launches, this exploit is mitigated. However in the future it is possible that a new protocol integration could be vulnerable to this issue, so extra caution should be used when adding new protocols.

Cantina Managed: Fixed in commit ddf67b36.

The fix verifies that the `totalSupply` does not change during the swap. Following the trust assumption of the protocol, the strategist key is a multi-signature wallet and wouldn't do malicious things. Also, as the strategist role is granted to a multisig-wallet with limited functionality, permission actions can not be triggered within a swap transaction. Thus, the risks of such issues are largely limited.

It's important to note that the re-entrancy attack vector is mitigated under the safely configured assumption. The project team would configure the strategist role to a trusted multi-signature wallet with no capability of executing permission actions within the 1Inch execution.

If another permission action is invoked within the swap transaction, the strategist can still drain the vault. For instance, in a swap transaction targeting `wETH`, the exploiter can deposit 1000 `wETH` and withdraw 1000 `stETH`. In this scenario, the strategist can bypass the slippage protection. Similarly, if the strategist initiates a trade within another trade, the slippage protection would still be circumvented.

3.1.4 Inconsistency in minimum value out for native deposits via receive function

Severity: Medium Risk

Context: TellerWithMultiAssetSupport.sol#L310

Description: The current implementation of the `receive()` function calls the `deposit()` function with a hardcoded minimum value out of 0 for native deposits. This design could potentially lead to situations where users making native deposits do not have the opportunity to specify a minimum mint amount, differing from the ERC20 deposit flow where a minimumMint value is required. This inconsistency might result in unexpected outcomes for users, especially in volatile market conditions where the value of shares could change significantly before the transaction is processed.

```
/**
 * @dev Depositing this way means users can not set a min value out.
 */
receive() external payable {
    deposit(ERC20(NATIVE), msg.value, 0);
}
```

Recommendation: To ensure consistency and protect users from market volatility, it is recommended to delete `receive()` function.

Se7en Seas: After discussing with the team, as well as the macro auditor, the function has been removed in commit af82ce27, it just doesn't really add that much value for the baggage it has.

Cantina Managed: Fixed.

3.1.5 Re-entrancy risk with ERC777 tokens in multiple functions

Severity: Medium Risk

Context: TellerWithMultiAssetSupport.sol#L226-L255

Description: The `deposit` and `depositWithPermit` functions in the contract allow users to deposit ERC20 tokens into the contract. However, these functions do not consider the potential re-entrancy risks associated with ERC777 tokens.

ERC777 tokens have additional hooks or callbacks that can be triggered during token transfers. These hooks, if not properly handled, can lead to re-entrancy vulnerabilities, where an attacker can potentially re-enter the contract and execute malicious code before the original execution is complete.

If an attacker deposits an ERC777 token and exploits the re-entrancy vulnerability in either the `deposit` or `depositWithPermit` function, they may be able to circumvent certain security checks or manipulate the contract state in unintended ways. For example, an attacker could potentially bypass the share lock period by using the token's transfer hooks to transfer the newly minted shares to another address before the lock period is enforced.

```
/**
 * @notice Allows users to deposit into the BoringVault, if this contract is not paused.
 */
function deposit(ERC20 depositAsset, uint256 depositAmount, uint256 minimumMint)
   public
   payable
   requiresAuth
   returns (uint256 shares)
{
   if (isPaused) revert TellerWithMultiAssetSupport__Paused();
   if (!isSupported[depositAsset]) revert TellerWithMultiAssetSupport__AssetNotSupported();

   if (address(depositAsset) == NATIVE) {
       if (msg.value == 0) revert TellerWithMultiAssetSupport__ZeroAssets();
       nativeWrapper.deposit{value: msg.value}();
       depositAmount = msg.value;
       shares = depositAmount.mulDivDown(ONE_SHARE, accountant.getRateInQuoteSafe(nativeWrapper));
       if (shares < minimumMint) revert TellerWithMultiAssetSupport__MinimumMintNotMet();
       // `from` is address(this) since user already sent value.
       nativeWrapper.safeApprove(address(vault), depositAmount);
       vault.enter(address(this), nativeWrapper, depositAmount, msg.sender, shares);
   } else {
       if (msg.value > 0) revert TellerWithMultiAssetSupport__DualDeposit();
       shares = _erc20Deposit(depositAsset, depositAmount, minimumMint, msg.sender);
   }

   _afterPublicDeposit(msg.sender, depositAsset, depositAmount, shares, shareLockPeriod);
}

/**
 * @notice Allows users to deposit into BoringVault using permit.
 */
function depositWithPermit(
   ERC20 depositAsset,
   uint256 depositAmount,
   uint256 minimumMint,
   uint256 deadline,
   uint8 v,
   bytes32 r,
   bytes32 s
) external requiresAuth returns (uint256 shares) {
   if (isPaused) revert TellerWithMultiAssetSupport__Paused();
   if (!isSupported[depositAsset]) revert TellerWithMultiAssetSupport__AssetNotSupported();

   try depositAsset.permit(msg.sender, address(vault), depositAmount, deadline, v, r, s) {}
   catch {
       if (depositAsset.allowance(msg.sender, address(vault)) < depositAmount) {
           revert TellerWithMultiAssetSupport__PermitFailedAndAllowanceTooLow();
       }
   }
   shares = _erc20Deposit(depositAsset, depositAmount, minimumMint, msg.sender);

   _afterPublicDeposit(msg.sender, depositAsset, depositAmount, shares, shareLockPeriod);
}
```

An example Token list can be seen from below :

- PNT
- IMBTC
- Superfluid Token
- XDAI Token

Recommendation: To mitigate the potential re-entrancy risk with ERC777 tokens, it is recommended to implement a re-entrancy guard or a mutability check before any state-changing operations in both the `deposit` and `depositWithPermit` functions.

Se7en Seas: Fixed by adding nonReentrant to `TellerWithMultiAssetSupport:deposit`, `TellerWithMultiAssetSupport:depositWithPermit` and `TellerWithMultiAssetSupport:bulkDeposit` in commit d1b4b522.

Cantina Managed: Fixed.

3.1.6 Missing checks in the `bulkDeposit` and `bulkWithdraw` functions

Severity: Medium Risk

Context: TellerWithMultiAssetSupport.sol#L282

Description: The `bulkDeposit` function in the contract is responsible for allowing users with the "onramp" role to deposit ERC20 tokens into the contract. However, this function lacks important checks that are present in other functions within the contract.

Note : the `bulkWithdraw` is also missing `isSupported` check.

Specifically, the `bulkDeposit` function does not have the following checks:

- **Pause Check:** The function does not check if the contract is currently paused before allowing deposits. This means that deposits can be made even when the contract is in a paused state.
- **Token Support Check:** The function does not verify if the provided depositAsset is a supported token by the contract. This could potentially allow deposits of unsupported tokens, leading to unexpected behavior or potential vulnerabilities.

```
/**
 * @notice Allows on ramp role to deposit into this contract.
 * @dev Does NOT support native deposits.
 */
function bulkDeposit(ERC20 depositAsset, uint256 depositAmount, uint256 minimumMint, address to)
    external
    requiresAuth
    returns (uint256 shares)
{
    shares = _erc20Deposit(depositAsset, depositAmount, minimumMint, to);
    emit BulkDeposit(address(depositAsset), depositAmount);
}
```

Recommendation: To address these issues, it is recommended to add the following checks to the `bulkDeposit` function:

- Add a check to ensure that the contract is not paused before allowing deposits.
- Add a check to verify that the provided `depositAsset` is a supported token by the contract.

Se7en Seas: Fixed by adding the appropriate `isSupported` check to `TellerWithMultiAssetSupport:bulkDeposit` in commit 546ff962 and to `bulkWithdraw` in commit 59a18048. It is intentional for `bulkDeposit` to not check if the Teller is paused. Pausing is intended only to stop general public deposits, but permissioned deposits using the AtomicQueue should still be allowed.

Cantina Managed: Fixed.

3.1.7 Minting of fake shares for non-existent ERC20 tokens

Severity: Medium Risk

Context: BoringVault.sol#L70

Description: The protocol utilizes solmate's `SafeTransferLib` for transferring ERC20 tokens during the enter function. However, solmate's SafeTransferLib does not check whether the token address provided is a valid contract or not. This contrasts with OpenZeppelin's `SafeERC20`, which includes a check to ensure that the provided address is a contract.

If a non-existent token address (e.g., `0x0000000000000000000000000000000000000000`) is provided during the enter function, the transaction will still succeed without any errors. This could potentially lead to the minting of fake shares for a non-existent ERC20 token.

While this issue alone may not directly cause fund loss, as the non-existent token cannot be withdrawn or transferred, it could lead to accounting discrepancies and potential vulnerabilities if the non-existent token contract is later deployed at the same address.

```
/**
 * @notice Allows minter to mint shares, in exchange for assets.
 * @dev If assetAmount is zero, no assets are transferred in.
 */
function enter(address from, ERC20 asset, uint256 assetAmount, address to, uint256 shareAmount)
    external
    requiresAuth
{
    // Transfer assets in
    if (assetAmount > 0) asset.safeTransferFrom(from, address(this), assetAmount);

    // Mint shares.
    _mint(to, shareAmount);

    emit Enter(from, address(asset), assetAmount, to, shareAmount);
}
```

Proof Of Concept:

```
function testSolmateVulnerability(uint256 amount) external {
    amount = bound(amount, 0.0001e18, 10_000e18);

    uint256 fakeToken_amount = amount.mulDivDown(1e18, IRateProvider(FAKEToken_RATE_PROVIDER).getRate());

    ERC20 nonExistentToken = ERC20(0x0000000000000000000000000000000000000000);
    teller.bulkDeposit(nonExistentToken, fakeToken_amount, 0,address(this));

}
```

Output:

```
[PASS] testSolmateVulnerability(uint256) (runs: 257, : 132983, ~: 133078)
Traces:
  [133080] TellerWithMultiAssetSupportTest::testSolmateVulnerability(73882507095584939857760549393070179058599 ↲
  ↪  2162620060501269
  ↪  [7.388e56])
    [0] console::log("Bound Result", 6649807209276899681100 [6.649e21]) [staticcall]
      ← [Stop]
    [30252] 0xCd5fE23C85820F7B72D0926FC9b05b43E359b7ee::getRate() [staticcall]
      [25369] 0xe629ee84C1Bd9Ea9c677d2D5391919fCf5E7d5D9::getRate() [delegatecall]
        [20043] 0x308861A430be4cce5502d0A12724771Fc6DaF216::amountForShare(1000000000000000000 [1e18])
   ↪  [staticcall]
          [15157] 0x403ba4cd327293A4d23beE172982509d37310cEF::amountForShare(1000000000000000000 [1e18])
   ↪  [delegatecall]
            [7246] 0x35fA164735182de50811E8e2E824cFb9B6118ac2::totalShares() [staticcall]
              [2363] 0x1B47A665364bC15C28B05f449B53354d0CefF72f::totalShares() [delegatecall]
                ← [Return] 0x000000000000000000000000000000000000000000006a88575d625a4366d78b
              ← [Return] 0x000000000000000000000000000000000000000000006a88575d625a4366d78b
            ← [Return] 1032844804788224332 [1.032e18]
          ← [Return] 1032844804788224332 [1.032e18]
        ← [Return] 1032844804788224332 [1.032e18]
      ← [Return] 1032844804788224332 [1.032e18]
    [88091] TellerWithMultiAssetSupport::bulkDeposit(0x0000000000000000000000000000000000000000,
   ↪  6438341151011921562763 [6.438e21], 0, TellerWithMultiAssetSupportTest:
   ↪  [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496])
      [7405] RolesAuthority::canCall(TellerWithMultiAssetSupportTest:
   ↪  [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496], TellerWithMultiAssetSupport:
   ↪  [0xF62849F9A0B5Bf2913b396098F7c7019b51A820a],
   ↪  0x9d57442000000000000000000000000000000000000000000000000000000000) [staticcall]
        ← [Return] true
      [5165] AccountantWithRateProviders::getRateInQuoteSafe(0x0000000000000000000000000000000000000000)
   ↪  [staticcall]
        ← [Return] 1000000000000000000 [1e18]
      [62133] BoringVault::enter(TellerWithMultiAssetSupportTest:
   ↪  [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496], 0x0000000000000000000000000000000000000000,
   ↪  6438341151011921562763 [6.438e21], TellerWithMultiAssetSupportTest:
   ↪  [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496], 6438341151011921562763 [6.438e21])
        [7405] RolesAuthority::canCall(TellerWithMultiAssetSupport:
   ↪  [0xF62849F9A0B5Bf2913b396098F7c7019b51A820a], BoringVault:
   ↪  [0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f],
   ↪  0x39d6ba3200000000000000000000000000000000000000000000000000000000) [staticcall]
          ← [Return] true
        [0] 0x0000000000000000000000000000000000000000::transferFrom(TellerWithMultiAssetSupportTest:
   ↪  [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496], BoringVault:
   ↪  [0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f], 6438341151011921562763 [6.438e21])
          ← [Stop]
        emit Transfer(from: 0x0000000000000000000000000000000000000000, to:
   ↪  TellerWithMultiAssetSupportTest: [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496], amount:
   ↪  6438341151011921562763 [6.438e21])
        emit Enter(from: TellerWithMultiAssetSupportTest: [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496],
   ↪  asset: 0x0000000000000000000000000000000000000000, amount: 6438341151011921562763 [6.438e21], to:
   ↪  TellerWithMultiAssetSupportTest: [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496], shares:
   ↪  6438341151011921562763 [6.438e21])
        ← [Stop]
      emit BulkDeposit(asset: 0x0000000000000000000000000000000000000000, depositAmount:
   ↪  6438341151011921562763 [6.438e21])
      ← [Return] 6438341151011921562763 [6.438e21]
    ← [Stop]
```

Recommendation: To mitigate this issue, it is recommended to use OpenZeppelin's SafeERC20 library instead of solmate's SafeTransferLib. OpenZeppelin's implementation includes a check to ensure that the provided token address is a contract before proceeding with the transfer.

Se7en Seas: Acknowledged: Fixing the issue "Missing checks in the `bulkDeposit` and `bulkWithdraw` functions" now makes this issue much less likely to happen. The Owner role must mess up calling `addAsset`, and `setRateProviderData`, and even if both of those happens we can simply:

1) Pause deposits.
2) Refund any deposits that used this exploit to mint fake shares.
3) Correct the mistakes made when calling `addAsset`, `setRateProviderData`.
4) Unpause deposits.

Cantina Managed: Acknowledged.

3.1.8 Hardcoded deadline in the DEX interaction

Severity: Medium Risk

Context: DexSwapperUManager.sol#L118

Description: In the `DexSwapperUManager.sol` contract, specifically in the `swapWithUniswapV3` function, the deadline parameter for the `IUniswapV3Router.ExactInputParams` struct is set to `block.timestamp`. This means that the deadline for the trade is set to the current block timestamp, which could lead to issues if the trade execution is delayed due to network congestion or other factors.

Hardcoding the deadline in this manner can result in failed transactions if the deadline is exceeded before the trade is executed. This can cause users to lose their funds or incur additional transaction fees when attempting to resubmit the trade.

```
IUniswapV3Router.ExactInputParams memory params = IUniswapV3Router.ExactInputParams({
    path: packedPath,
    recipient: boringVault,
    deadline: block.timestamp,
    amountIn: amountIn,
    amountOutMinimum: amountOutMinimum
});
```

Recommendation: Instead of hardcoding the deadline parameter to block.timestamp, it is recommended to pass the deadline as a separate parameter to the `swapWithUniswapV3` function. This will allow the caller to set an appropriate deadline based on their requirements and expectations for trade execution time.

Here's an example of how the function signature and implementation could be modified:

```
function swapWithUniswapV3(
    bytes32[][] calldata manageProofs,
    address[] calldata decodersAndSanitizers,
    ERC20[] memory path,
    uint24[] memory fees,
    uint256 amountIn,
    uint256 amountOutMinimum,
    uint256 deadline
) external requiresAuth {
    // ... (existing code)
    IUniswapV3Router.ExactInputParams memory params = IUniswapV3Router.ExactInputParams({
        path: packedPath,
        recipient: boringVault,
        deadline: deadline, // Use the provided deadline parameter
        amountIn: amountIn,
        amountOutMinimum: amountOutMinimum
    });
    // ... (remaining code)
}
```

Se7en Seas: Fixed in commit ddf67b36.

Cantina Managed: Fixed.

3.2 Low Risk

3.2.1 Front-running attacks in `repay` function due to missing validation logic

Severity: Low Risk

Context: MorphoBlueDecoderAndSanitizer.sol

Description: In the current implementation of the `MorphoBlueDecoderAndSanitizer` contract's `repay` function, there is a noticeable absence of validation for key operation parameters, specifically for the assets and shares. According to the protocol documentation and function signature, either assets or shares must be zero when calling the repay function, to ensure the correct and intended functionality of repaying loans within the protocol. This critical validation check is missing in the sanitizer function, potentially allowing calls to the repay function with both assets and shares being non-zero, contradicting the intended protocol logic and potentially leading to unexpected behavior or transaction failures.

Furthermore, the documentation highlights a security concern where an attacker could exploit the lack of validation by front-running a legitimate repay transaction with a small repay, causing the legitimate transaction to revert due to underflow. The current sanitizer implementation does not mitigate this risk, as it does not enforce the protocol's guidance of using shares for full position repayments to avoid rounding errors and potential reverts.

See this reference:

> function repay(MarketParams memory marketParams, uint256 assets, uint256 shares, address onBehalf, bytes memory data) external returns (uint256 assetsRepaid, uint256 sharesRepaid);
>
> Repays assets or shares on behalf of onBehalf, optionally calling back the caller's onMorphoReplay function with the given data.
>
> Either assets or shares should be zero. To repay max, pass the shares's balance of onBehalf.
>
> Repaying an amount corresponding to more shares than borrowed will revert for underflow.
>
> It is advised to use the shares input when repaying the full position to avoid reverts due to conversion roundings between shares and assets.
>
> An attacker can front-run a repay with a small repay making the transaction revert for underflow.

Recommendation: Incorporate a recommendation in the documentation for users to prefer the use of `shares` when repaying the full position.

Se7en Seas: Acknowledged. The strategist will only rebalance the BoringVault using private txs, but will keep this edge case in mind when interacting with MorphoBlue.

Cantina Managed: Acknowledged.

3.2.2 `ManagerWithMerkleVerification` and decoders do not verify the ETH value being transferred out.

Severity: Low Risk

Context: ManagerWithMerkleVerification.sol#L234-L237

Description: In the current protocol, the strategies create a payload for the boringVault and help manage the boringVault's portfolio. To avoid malicious strategies from ruining the vault, ManagerWithMerkleVerification verifies calldata. Thus, for each strategy, only a restricted set of actions can be triggered.

```
contract ManagerWithMerkleVerification {
    // ...
function _verifyManageProof(
    bytes32 root,
    bytes32[] calldata proof,
    address target,
    address decoderAndSanitizer,
    uint256 value,
    bytes4 selector,
    bytes memory packedArgumentAddresses
) internal pure returns (bool) {
    bool valueNonZero = value > 0;
    bytes32 leaf =
        keccak256(abi.encodePacked(decoderAndSanitizer, target, valueNonZero, selector,
↪   packedArgumentAddresses));
    return MerkleProofLib.verify(proof, root, leaf);
}
// ...
```

However, `_verifyManageProof` only verifies whether there's native value being sent, instead of the actual ETH amount. Thus, the strategies can unlimitedly pull native tokens from the `boringVault`.

Recommendation: The `ManagerWithMerkleVerification` contract uses a succinct method to validate call data by passing the payload directly to the sanitizer using `functionStaticCall`. However, it restricts the capability of the decoder to check the native value.

Thus, recommended to pass the native ETH value as the parameter to the decoder. So that the decoder can check the value when it's needed.

Se7en Seas: Acknowledged. Given the current scope of the audit, as well as how the architecture will be configured for foreseeable launches, being able to limit the amount of ETH transferred to targets will only greatly hinder the strategist's ability to properly rebalance the BoringVault, and does very little to actual add more security constraints. Because of this we opt to keep the current security check which simply controls whether or not any ETH can be transferred to some target.

Cantina Managed: Acknowledged.

3.2.3 `getRateInQuoteSafe` should round up when depositing

Severity: Low Risk

Context: TellerWithMultiAssetSupport.sol#L323-L323

Description: `TellerWithMultiAssetSupport` allows users to deposit from multiple assets. The amount of shares is calculated as

```
shares = depositAmount.mulDivDown(ONE_SHARE, accountant.getRateInQuoteSafe(depositAsset));
```

Since `accountant.getRateInQuoteSafe` always rounds down, the share amount can potentially be overestimated.

Recommendation: Consider having `accountant.getRateInQuoteSafe(depositAsset, roundingDirection)` to get a rounded-up rate for depositing.

Se7en Seas: Acknowledged. The discrepancy is on the order of less than 10 wei, which is not enough of an impact to warrant adding additional more complex logic. Additionally even if this issue lead to a much larger discrepancy, we can always refund the deposit.

Cantina Managed: Acknowledged

3.2.4 Lack of upper bound for `minimumUpdateDelayInHours`

Severity: Low Risk

Context: AccountantWithRateProviders.sol#L168

Description: The `updateDelay` function in the contract allows updating the minimum time delay (in hours) between calls to the `updateExchangeRate` function. However, there is no explicit upper bound check for the `minimumUpdateDelayInHours` parameter, which could potentially lead to an indefinite pause of the contract's account state.

If the `minimumUpdateDelayInHours` is set to an excessively large value, it may prevent the `updateExchangeRate` function from being called for an extended period, effectively freezing the contract's exchange rate and preventing fee calculations.

```
/**
 * @notice Update the minimum time delay between `updateExchangeRate` calls.
 * @dev There are no input requirements, as it is possible the admin would want
 *      the exchange rate updated as frequently as needed.
 */
function updateDelay(uint8 minimumUpdateDelayInHours) external requiresAuth {
    uint8 oldDelay = accountantState.minimumUpdateDelayInHours;
    accountantState.minimumUpdateDelayInHours = minimumUpdateDelayInHours;
    emit DelayInHoursUpdated(oldDelay, minimumUpdateDelayInHours);
}
```

Recommendation: To mitigate this issue, it is recommended to introduce an upper bound check for the `minimumUpdateDelayInHours` parameter in the `updateDelay` function. This check should ensure that the provided value does not exceed a reasonable maximum delay.

Se7en Seas: Fixed in commit 6df88ee0 by adding a maximum update delay of 14 days.

Cantina Managed: Fixed.

3.2.5 Consider optimal curve pool for token swaps

Severity: Low Risk

Context: CurveDecoderAndSanitizer.sol

Description: Although the current implementation does not pose a direct risk, as the off-chain operations will handle the pool selection, it is worth considering the potential impact of not utilizing the optimal Curve pool for token swaps.

The Curve protocol supports multiple liquidity pools for different token pairs. If the implementation relies solely on the first pool returned by the `Curve.Registry.find_pool_for_coins` function, it may not necessarily be the most optimal pool for the given token pair. The first pool might have lower liquidity, higher slippage, or higher fees compared to other available pools, resulting in suboptimal trade execution and potentially lower returns for the users.

Recommendation: While the off-chain operations will handle the pool selection, it is recommended to consider implementing a mechanism to select the most optimal Curve pool for token swaps. This can be achieved by leveraging the `pool_index` parameter in the `find_pool_for_coins` function provided by the Curve Registry.

Se7en Seas: Acknowledged. Determining optimal swap paths is a very complex problem, that is practically impossible to reasonably do in smart contracts. We could optionally continually update the merkle root to only include the best pools for swapping, but this would lead to a lot of merkle root changes which is not optimal for users trying to track exactly what the strategist is able to do. A better option is to trust the strategist as their wants align with users wants, i.e. if the strategist makes decisions that benefit the BoringVault, more users will deposit which increases the TVL, and increases fees earned.

Cantina Managed: Acknowledged.

3.2.6 Failure to clean deposit history mapping in `bulkWithdraw`

Severity: Low Risk

Context: TellerWithMultiAssetSupport.sol#L294

Description: The `bulkWithdraw` function in the contract does not clean up the `publicDepositHistory` mapping after a successful withdrawal. This mapping is used to store the deposit history of users, and it is intended to be cleaned up after a deposit is refunded or withdrawn.

```
/**
 * @notice Allows off ramp role to withdraw from this contract.
 */
function bulkWithdraw(ERC20 withdrawAsset, uint256 shareAmount, uint256 minimumAssets, address to)
    external
    requiresAuth
    returns (uint256 assetsOut)
{
    if (shareAmount == 0) revert TellerWithMultiAssetSupport__ZeroShares();
    assetsOut = shareAmount.mulDivDown(accountant.getRateInQuoteSafe(withdrawAsset), ONE_SHARE);
    if (assetsOut < minimumAssets) revert TellerWithMultiAssetSupport__MinimumAssetsNotMet();
    vault.exit(to, withdrawAsset, assetsOut, msg.sender, shareAmount);
    emit BulkWithdraw(address(withdrawAsset), shareAmount);
}
```

Recommendation: To mitigate this issue, it is recommended to add a step in the `bulkWithdraw` function to clean up the `publicDepositHistory` mapping for the corresponding deposit entry. This can be achieved by deleting the entry from the mapping after a successful withdrawal.

Se7en Seas: Acknowledged. It is potentially leaving some gas savings on the table, but implementing this is a relatively complex task that is not worth the gas savings.

Cantina Managed: Acknowledged.

3.2.7 Deflationary asset tokens are not handled uniformly across the protocol

Severity: Low Risk

Context: BoringVault.sol#L70

Description: The code base do not support rebasing/deflationary/inflationary asset tokens whose balance changes during transfers or over time. The necessary checks include at least verifying the amount of tokens transferred to contracts before and after the actual transfer to infer any fees/interest.

```
/**
 * @notice Allows minter to mint shares, in exchange for assets.
 * @dev If assetAmount is zero, no assets are transferred in.
 */
function enter(address from, ERC20 asset, uint256 assetAmount, address to, uint256 shareAmount)
    external
    requiresAuth
{
    // Transfer assets in
    if (assetAmount > 0) asset.safeTransferFrom(from, address(this), assetAmount);

    // Mint shares.
    _mint(to, shareAmount);

    emit Enter(from, address(asset), assetAmount, to, shareAmount);
}
```

Recommendation: Consider checking previous balance/after balance equals to amount for any rebasing/inflation/deflation asset tokens.

Se7en Seas: Acknowledged. There are no plans to support fee on transfer assets.

Cantina Managed: Acknowledged.

3.3 Informational

3.3.1 Strategist that can initiate flash loan can execute other flash loan strategies through `receiveFlashloan` callback

Severity: Informational

Context: ManagerWithMerkleVerification.sol#L148-L189

Description: Strategists can initiate various strategies with `manageVaultWithMerkleVerification` as long as they prove their permissions through the correct Merkle proof. For strategies that use flash loans, they have to send a payload to the `boringVault` that triggers `ManagerWithMerkleVerification.flashloan`. In the `flashloan` function, the contract calls `BalancerVault.flashloan`. Finally, the callback function `receiveFlashLoan` would execute the actual payload.

To correctly configure the settings, the flashLoan function should accept calls from the boringVault, and all strategies initiated from receiveFunction should have the same permission levels. This leads to concerns about permission settings:

1. If there were multiple `ManagerWithMerkleVerification` instances deployed in the protocol, strategists from one manager could initiate strategies from another manager. Since the `flashLoan` function only checks if `msg.sender == balancerVault`, it doesn't differentiate who the real initiator is.
2. Strategists that can initiate one of the flash loan strategies can initiate other strategies. In the `receiveFlashLoan` function, the `Manager` contract verifies the payload by calling `this.manageVaultWithMerkleVerification`. Regardless of who the initiator of the flash loan strategy is, the `msg.sender` of this recursive call is the `Manager` contract, thus sharing the same `strategistManageRoot`.

Recommendation: Be aware of the potential issues. Flash loan strategies can impose different risks to the portfolio. Alternatively, the `BalancerV2DecoderAndSanitizer` should do a more aggressive check to make sure the payload is legal.

Se7en Seas: Acknowledged. Given the current scope of the audit, as well as how the architecture will be configured for foreseeable launches, there will only be one strategist rebalancing the BoringVault, so this issue will not come up.

Cantina Managed: Acknowledged.

3.3.2 Old shares would automatically unlock when a lower `shareLockPeriod` is set

Severity: Informational

Context: TellerWithMultiAssetSupport.sol#L176-L178, TellerWithMultiAssetSupport.sol#L331-L346

Description: Users depositing through the public deposit function must lock their funds for a period determined by currentShareLockPeriod. Whenever the beforeTransfer hook is triggered, it verifies whether the user's shares are still within the lock period:

```
function beforeTransfer(address from) external view {
      if (shareUnlockTime[from] >= block.timestamp) revert TellerWithMultiAssetSupport__SharesAreLocked();
 }
```

Since the share lock period is set by user address at TellerWithMultiAssetSupport.sol#L338:

```
shareUnlockTime[user] = block.timestamp + currentShareLockPeriod;
```

If the currentShareLockPeriod has been lowered since the last deposit, the shareUnlockTime[user] may be lower after the deposit.

Recommendation: Document the behaviors and be aware of this when `currentShareLockPeriod` changes.

Se7en Seas: Fixed in commit e7be8477.

Cantina Managed: Fixed. The risks are accepted and the behavior is documented.

3.3.3 Consider checking `msg.sender == address(balancerVault)` on `receiveFlashLoan` function for readability

Severity: Informational

Context: ManagerWithMerkleVerification.sol#L148-L189

Description: The `receiveFlashloan` function of `ManagerWithMerkleVerification` is a callback function triggered when strategies initiate flash loans from Balancer. This function must only be called by the `Balancer`, or the caller can potentially drain the `boringVault`.

In the current design, the owner should properly set the authority so that the `requireAuth` modifier would revert when triggered by any other address.

Recommendation: Since the function should only be called by `BalancerVault`, consider checking `msg.sender == address(BalancerVault)` to improve readability.

Se7en Seas: Fixed in commit 2921e0cd.

Cantina Managed: Verified.

3.3.4 Absence of liquidate function in `MorphoBlueDecoderAndSanitizer`

Severity: Informational

Context: MorphoBlueDecoderAndSanitizer.sol

Description: The current implementation of the `MorphoBlueDecoderAndSanitizer` contract lacks support for the liquidate function, an operation within the `Morpho` protocol that allows for the liquidation of a borrower's position under specific market conditions. The liquidate function requires that either `seizedAssets` or `repaidShares` be zero to ensure a single action per operation—either seizing assets or repaying shares. However, the absence of this function in the sanitizer contract means there's no validation mechanism to enforce this rule, potentially leading to transactions that violate these constraints.

See this reference:

> function liquidate(MarketParams memory marketParams, address borrower, uint256 seizedAssets, uint256 repaidShares, bytes memory data) external returns (uint256, uint256);
>
> Liquidates the given repaidShares of debt asset or seize the given seizedAssets of collateral on the given market marketParams of the given borrower's position, optionally calling back the caller's onMorphoLiquidate function with the given data.
>
> Either seizedAssets or repaidShares should be zero.
>
> Seizing more than the collateral balance will underflow and revert without any error message.
>
> Repaying more than the borrow balance will underflow and revert without any error message.
>
> An attacker can front-run a liquidation with a small repay making the transaction revert for underflow.

Recommendation: Introduce a sanitizing function for `liquidate` operations within the `MorphoBlueDecoderAndSanitizer` contract.

Se7en Seas: Acknowledged. The BoringVault will not be calling the `liquidate` function.

Cantina Managed: Acknowledged.

3.3.5 Slippage risk in `borrow` function due to lack of validation for `assets` and `shares` parameters

Severity: Informational

Context: MorphoBlueDecoderAndSanitizer.sol

Description: The `borrow` function in the `MorphoBlueDecoderAndSanitizer` allows users to borrow either `assets` or `shares` on behalf of another account, directing the borrowed amount to a specified receiver. The function is designed such that either the assets or shares parameter must be zero, to prevent ambiguity and ensure precise operation.

It has been identified in the corresponding sanitizer function, where there is no validation check to enforce that one of these parameters (`assets` or `shares`) is indeed zero. This absence of validation not only deviates from the intended functionality but also introduces a slippage risk. Specifically, if an attempt is made to borrow an amount of shares without this validation, the operation may result in borrowing fewer assets than expected due to market slippage.

See this reference:

> function borrow(MarketParams memory marketParams, uint256 assets, uint256 shares, address onBehalf, address receiver) external returns (uint256 assetsBorrowed, uint256 sharesBorrowed);
>
> Borrows assets or shares on behalf of onBehalf to receiver.
>
> Either assets or shares should be zero. Most usecases should rely on assets as an input so the caller is guaranteed to borrow assets of tokens, but the possibility to mint a specific amount of shares is given for full compatibility and precision.
>
> msg.sender must be authorized to manage onBehalf's positions.
>
> Borrowing a large amount can revert for overflow.
>
> Borrowing an amount of shares may lead to borrow fewer assets than expected due to slippage. Consider using the assets parameter to avoid this.

Recommendation: Enhance the sanitizer function associated with the `borrow` operation to include a check that eliminates slippage risk.

Se7en Seas: Acknowledged. We will keep this in mind when interacting with MorphoBlue markets.

Cantina Managed: Acknowledged.

3.3.6 Missing `idToMarketParams` Support in `MorphoBlueDecoderAndSanitizer` Contract

Severity: Informational

Context: MorphoBlueDecoderAndSanitizer.sol

Description: The `MorphoBlueDecoderAndSanitizer` contract, designed for sanitizing and validating inputs across various operations in the Morpho protocol, does not currently support the `idToMarketParams` mapping. This mapping is critical for efficiently translating an ID to its corresponding market parameters (`loanToken`, `collateralToken`, `oracle`, `irm`, and `lltv`).

See this reference:

> function idToMarketParams(Id id) external view returns (address loanToken, address collateralToken, address oracle, address irm, uint256 lltv);
>
> The market params corresponding to id.
>
> This mapping is not used in Morpho. It is there to enable reducing the cost associated to calldata on layer 2s by creating a wrapper contract with functions that take id as input instead of marketParams.

Recommendation: Integrate support for the `idToMarketParams` mapping within the `MorphoBlueDecoderAndSanitizer` contract.

Se7en Seas: Acknowledged. The strategist can handle going between ids and market params off-chain, so this functionality is not needed.

Cantina Managed: Acknowledged.

3.3.7 Lack of "Zero-Value" Check for assets or shares Parameters in `MorphoBlueDecoderAndSanitizer`

Severity: Informational

Context: MorphoBlueDecoderAndSanitizer.sol#L13

Description: The `MorphoBlueDecoderAndSanitizer` contract's supply function is designed to validate and sanitize inputs for the corresponding supply function in the `Morpho` protocol, which enables users to supply assets or shares into the market. According to the `Morpho` protocol documentation, when calling the supply function, either the `assets` or `shares` parameter must be zero to ensure the transaction's integrity.

See this reference:

> Supplies assets or shares on behalf of onBehalf, optionally calling back the caller's onMorpho-Supply function with the given data.
>
> Either assets or shares should be zero. Most usecases should rely on assets as an input so the caller is guaranteed to have assets tokens pulled from their balance, but the possibility to mint a specific amount of shares is given for full compatibility and precision.
>
> Supplying a large amount can revert for overflow.
>
> Supplying an amount of shares may lead to supply more or fewer assets than expected due to slippage. Consider using the assets parameter to avoid this.

Recommendation: Introduce a validation step in the supply function of the `MorphoBlueDecoderAndSanitizer` contract.

Se7en Seas: Acknowledged. The MorphoBlue contracts perform this check when interacting with them, so if a strategist violated this constraint the rebalance will revert, which is desired behavior.

Cantina Managed: Acknowledged.

3.3.8 Consider using `poolId` instead of pool address in `BalancerV2DecoderAndSanitizer`

Severity: Informational

Context: BalancerV2DecoderAndSanitizer.sol#L97-L99

Description: In the `BalancerV2DecoderAndSanitizer`, the pool address is extracted from the poolId. However, Balancer V2 does not revert when the same address registers as several different pools. (ref: PoolRegistry.sol#L68C14-L88).

Recommendation: This is the edge case scenario. We can either upgrade the `BalancerV2DecoderAndSanitizer` if we find such cases in the Balancer ecosystem; or, consider to check `poolId` directly.

Se7en Seas: Acknowledged.

Cantina Managed: Acknowledged

3.3.9 Use named error for revert statement to maintain code consistency

Severity: Informational

Context: DexSwapperUManager.sol#L110

Description: The `DexSwapperUManager` contract contains a standard revert statement with an error message string. This pattern is inconsistent with the rest of the code that employs named errors for revert conditions.

```
if (path.length - 1 != fees.length) revert("Bad path/fees");
```

Recommendation: Use a named error for the revert statement to maintain consistency with conventions throughout the rest of the codebase.

```diff
- if (path.length - 1 != fees.length) revert("Bad path/fees");
+ if (path.length - 1 != fees.length) revert DexSwapperUManager__BadPathOrFees();
```

Se7en Seas: Fixed in commit ddf67b36.

Cantina Managed: Fixed.

3.3.10 Event is missing indexed fields

Severity: Informational

Context: Global scope

Description: 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).

Recommendation: Add indexes on the events.

Se7en Seas: Fixed in commit 224613ec.

Cantina Managed: Fixed.

3.3.11 Upgrade `pragma` to solidity 0.8.22

Severity: Informational

Context: Global scope

Description: The contracts in the project are currently using Solidity version 0.8.21. The latest version of Solidity (0.8.22) introduces an optimization for loop increments that can improve gas efficiency and code readability.

Recommendation: Upgrade the Solidity version used in the project to 0.8.22 to take advantage of the loop increment optimization.

Se7en Seas: Acknowledged. We prefer to continue using Solidity version 0.8.21 as it meets all of our criteria, and has worked for us for multiple deployments.

Cantina Managed: Acknowledged.

3.3.12 Missing removal of unused allowances in `DexSwapperUManager.swapWithUniswapV3`

Severity: Informational

Context: DexSwapperUManager.sol#L92-L140, DexAggregatorUManager.sol#L123-L139

Description: An operational gap exists concerning handling leftover token allowances after performing a swap with the `swapWithUniswapV3` function in the `DexSwapperUManager` contract. The `swapWith1Inch` function in the `DexAggregatorUManager` contract has a mechanism to deal with unused token allowances. However, `swapWithUniswapV3` lacks this functionality.

Removing unused allowances will prevent strategists from being able to use external protocols in any unwanted way.

Recommendation: Integrate the mechanism used by `swapWith1Inch` into `swapWithUniswapV3` to remove any leftover allowance after performing a DEX swap.

Se7en Seas: Fixed in commit 82710395.

Cantina Managed: Fixed.

3.3.13 Move duplicate code from micro-managers to abstract parent contract

Severity: Informational

Context: DexAggregatorUManager.sol#L75-L79, DexAggregatorUManager.sol#L145-L164, DexSwapperUManager.sol#L75-L79, DexSwapperUManager.sol#L145-L164

Description: The `DexAggregatorUManager` and `DexSwapperUManager` contracts contain duplicated code across multiple functions. Specifically, the functions `setAllowedSlippage` and `revokeTokenApproval` are present in both contracts.

This redundancy enlarges the codebase and increases the risk of bugs in potential future refactors. Moving shared logic to an abstract parent contract reduces duplicate code and increases the codebase's maintainability.

Recommendation: Move all shared logic between the existing micro-managers to an abstract parent contract.

Se7en Seas: Fixed in commit 513e3147.

Cantina Managed: Fixed.

4 Appendix

4.1 Fuzzing Summary

Code:

- Vault, Teller, and accountant - Report commit hash
- Vault, Teller, and accountant - Latest code at the time of writing
- DexSwapperUManager - Report commit hash
- DexSwapperUManager - Latest code at the time of writing

Methodology: This report contains the results of two distinct fuzzing approaches.

1. **Stateful Fuzzing for the Vault, Teller, and Accountant Contracts:** Implement a stateful fuzzing suite employing Echidna and Medusa. Define and test multiple invariants to assess the protocol's security and robustness.
2. **Differential Fuzzing for the Manager and Micro-Manager Contracts:** Implement a differential fuzzing suite to compare simulated results of an on-chain Uniswap V3 swap against swaps executed using the `DexSwapperUManager` contract. This approach aims to identify discrepancies and vulnerabilities in the swapping mechanism.

Invariants

Invariant	Description
DEPOSIT-01	Actor asset balance decreases by the deposited amount after a successful deposit
DEPOSIT-02	Vault asset balance increases by the deposited amount after a successful deposit
DEPOSIT-03	Actor vault balance increases by the amount of shares minted after a successful deposit
DEPOSIT-04	The amount of shares minted is greater than or equal to the provided minimum amount of shares
DEPOSIT-05	Depositing reverts if Accountant is paused
DEPOSIT-06	Depositing does not unexpectedly revert
WITHDRAW-01	The amount of assets withdrawn is equal to the expected precalculated amount
WITHDRAW-02	Actor asset balance increases by the withdrawn amount after a successful withdrawal
WITHDRAW-03	Vault asset balance decreases by the withdrawn amount after a successful withdrawal
WITHDRAW-04	Actor vault balance decreases by the shares burned after a successful withdrawal
WITHDRAW-05	The amount of assets withdrawn is greater than or equal to the provided minimum amount of asse
WITHDRAW-06	Withdrawing reverts if Accountant is paused
WITHDRAW-07	Withdrawing reverts when the amount of assets withdrawn is less than the provided minimum amo
WITHDRAW-08	Bulk withdrawing does not unexpectedly revert
EXRATE-01	Fees owed increase after successfully updating the exchange rate
EXRATE-02	Updating exchange rate reverts when Accountant is paused
EXRATE-03	Updating exchange rate does not unexpectedly revert
CLAIMFEES-01	Payout account balance increases by fees owed after successfully claiming fees
CLAIMFEES-02	Fees owed are zero after successfully claiming fees
CLAIMFEES-03	Claiming fees reverts when Accountant is paused
CLAIMFEES-04	Claiming fees does not unexpectedly revert
GLOBAL-01	All actors are always able to withdraw all owned funds
GLOBAL-02	Actors cannot increase total balance above the gained yield.