PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Singularity DAO (Index Vaults)

29 May 2025

paladinsec.co

info@paladinsec.co

Table of Contents

Table of Contents 2

Disclaimer 4

1 Overview 5

1.1 Summary 5

1.2 Contracts Assessed 6

1.3 Findings Summary 7

1.3.1 DynaLiquidity 8

1.3.2 DynaLiquidityLib 8

1.3.3 DynaLiquidityProxyAssetManager 9

1.3.4 DynaLiquidityAssetManagerLib 9

1.3.5 DynaLiquidityFactory 9

1.3.6 DynaLiquidityRegistry 9

1.3.7 BalancerV2DynaZapper 10

1.3.8 VaultDynaZapper 10

2 Findings 11

2.1 DynaLiquidity 11

2.1.1 Privileged Functions 11

2.1.2 Issues & Recommendations 13

2.2 DynaLiquidityLib 16

2.2.1 Issues & Recommendations 17

2.3 DynaLiquidityProxyAssetManager 23

2.3.1 Privileged Functions 23

2.3.2 Issues & Recommendations 24

2.4 DynaLiquidityAssetManagerLib 30

2.4.1 Issues & Recommendations 31

2.5 DynaLiquidityFactory 32

2.5.1 Privileged Functions 32

2.5.2 Issues & Recommendations 33
2.6 DynaLiquidityRegistry 34
2.6.1 Privileged Functions 34
2.6.2 Issues & Recommendations 35
2.7 BalancerV2DynaZapper 36
2.7.1 Privileged Functions 36
2.7.2 Issues & Recommendations 37
2.8 VaultDynaZapper 41
2.8.1 Privileged Functions 41
2.8.2 Issues & Recommendations 42

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. Paladin is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. Paladin is furthermore allowed to claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for Singularity DAO's Index Vault contracts on the Arbitrum network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	Singularity DAO
URL	https://singularitydao.ai/
Platform	Base
Language	Solidity
Preliminary	https://github.com/Singularity-DAO/DynaVaults/commit/35bb164db6166799a893d8dadbf1ed262b26a240
Resolution	https://github.com/Singularity-DAO/DynaVaults/commit/08dd423aac6fc0634b9de721d110017590a8fe3d

1.2 Contracts Assessed

Contract	Address	Live Match
DynaLiquidity	-	UNDEPLOYED
DynaLiquidityAssetManagerLib	-	UNDEPLOYED
DynaLiquidityFactory	-	UNDEPLOYED
DynaLiquidityLib	-	UNDEPLOYED
DynaLiquidityProxyAssetManager	-	UNDEPLOYED
DynaLiquidityRegistry	0x82f22418DE1b9B30E04aFc0cE303cf7EbE8d1674	MATCH
BalancerV2DynaZapper	0x3A37527e7fcEf237A03af70B69aeF9f7A3C9de07	MATCH
VaultDynaZapper	0xeE60B535CD61Ec546A23D22E2144FA2bCA48fDE5	MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change)	Failed
Governance	0	-	-	-	-
High	4	4	-	-	-
Medium	4	2	1	-	1
Low	10	6	-	3	-
Informational	13	10	1	2	-
Total	**31**	**22**	**2**	**5**	**1**

Classification of Issues

Severity	Description
Governance	Issues under this category are where the governance or owners of the protocol have certain privileges that users need to be aware of, some of which can result in the loss of user funds if the governance's private keys are lost or if they turn malicious, for example.
High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
Medium	Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 DynaLiquidity

ID	Severity	Summary	Status
01	HIGH	Majority of the pool's AUM management fees will get locked inside `DynaLiquidity`	✓ RESOLVED
02	LOW	Liquidity governance could introduce arbitrage opportunities that hurt liquidity providers	✓ RESOLVED
03	INFO	The contract is not upgradeable	ACKNOWLEDGED
04	INFO	Typographical issues	✓ RESOLVED

1.3.2 DynaLiquidityLib

ID	Severity	Summary	Status
05	MEDIUM	`createManagedPool()` can cause all funds to be lost	✓ RESOLVED
06	LOW	Not using `safeTransferFrom` can cause DOS with USDT or other non-standard tokens	ACKNOWLEDGED
07	LOW	`joinWithSingleAsset()` will always revert for USDT or other weird tokens	ACKNOWLEDGED
08	LOW	Managed pool creation can be front-run	ACKNOWLEDGED
09	LOW	`tokenReferenceValue()` can report 0 and logic will not revert	
10	INFO	Insufficient validation	✓ RESOLVED
11	INFO	Typographical issues	✓ RESOLVED

1.3.3 DynaLiquidityProxyAssetManager

ID	Severity	Summary	Status
12	HIGH	Managed assets are not accounted for and could lead to assets not being returned to the pool from proxies	✓ RESOLVED
13	MEDIUM	Proxy migration might fail	FAILED
14	LOW	Use `safeTransfer` for ERC20 transfers	✓ RESOLVED
15	INFO	Insufficient validation	✓ RESOLVED
16	INFO	Typographical issues	PARTIAL

1.3.4 DynaLiquidityAssetManagerLib

ID	Severity	Summary	Status
17	INFO	Typographical issues	✓ RESOLVED

1.3.5 DynaLiquidityFactory

ID	Severity	Summary	Status
18	INFO	Emit events on state changing functions	✓ RESOLVED
19	INFO	Typographical issues	✓ RESOLVED

1.3.6 DynaLiquidityRegistry

ID	Severity	Summary	Status
20	INFO	Emit events on state changing functions	✓ RESOLVED

1.3.7 BalancerV2DynaZapper

ID	Severity	Summary	Status
21	MEDIUM	`BalancerV2Zapper` does not check if the used `DynaLiquidity` has an active flag in `DynaLiquidityRegistry`	PARTIAL
22	LOW	Arbitrary contracts can be called and given permission to	✓ RESOLVED
23	LOW	`BalancerV2Zapper` does not support swaps with pools that contain the BPT token of other pools	✓ RESOLVED
24	INFO	Encoding of router swap data could be skipped in an edge case	ACKNOWLEDGED
25	INFO	Typographical issues	✓ RESOLVED

1.3.8 VaultDynaZapper

ID	Severity	Summary	Status
26	HIGH	Usage of `transferFrom` with arbitrary from value	✓ RESOLVED
27	HIGH	`_redeem` attempts to take `tokenIn` amount twice from the caller of `swap()`	✓ RESOLVED
28	MEDIUM	`VaultZapper` can deposit funds into disabled vaults	✓ RESOLVED
29	LOW	Unnecessary approval of `DynaVault`	✓ RESOLVED
30	LOW	Hard coded `useWhiteList`	✓ RESOLVED
31	INFO	Typographical issues	✓ RESOLVED

2 Findings

2.1 DynaLiquidity

`DynaLiquidity` is a contract deployed by `DynaLiquidityFactory`. It manages all pool and asset manager operations. This contract deploys the managed pool, control the key parameters for this pool and move liquidity in and out.

2.1.1 Privileged Functions

- `setupLiquidity [onlyRole]`
- `createManagedPool [onlyRole]`
- `addInitialLiquidity [onlyRole]`
- `addToken [onlyRole]`
- `setAssetManager [onlyRole]`
- `invest [onlyRole]`
- `liquidate [onlyRole]`
- `registerLiquidityManager [onlyRole]`
- `updateWeightsGradually [onlyRole]`
- `removeToken [onlyRole]`
- `setEnableWhitelistForLPsFlag [onlyRole]`
- `addToWhitelistForLPs [onlyRole]`
- `removeFromWhitelistForLPs [onlyRole]`
- `updateManagementFee [onlyRole]`
- `collectManagementFees [onlyRole]`
- `updateFeeRecipient [onlyRole]`
- `updateSwapFee [onlyRole]`
- `setSwapEnabled [onlyRole]`

- joinWithSingleAsset [onlyRole]
- exitWithSingleAsset [onlyRole]
- grantRole [onlyRole]
- revokeRole [onlyRole]

2.1.2 Issues & Recommendations

Issue #01	Majority of the pool's AUM management fees will get locked inside DynaLiquidity
Severity	HIGH SEVERITY
Description	The AUM management fees generated by the pool will be sent to the owner on each `setManagementAumFeePercentage()`, `addToken()`, `removeToken()`, `swap()`, `joinPool()` and `exitPool()` calls according to BalancerV2's `ManagedPool.sol` contract (link). The owner of the pool is set to be the `DynaLiquidity` contract (this is done in `DynaLiquidityLib.createManagedPool()`). This means that only `collectAumManagementFees()` correctly handles the BPT tokens received and sends them to `_storage.feeRecipient`. The rest of the fee amounts will be locked in the contract since there is no way for them to be moved.
Recommendation	As `DynaLiquidity` is not meant to hold tokens other than the BPT AUM fees, consider implementing a role guarded function for transferring the BPT tokens from `DynaLiquidity` to the `feeReceiver`.
Resolution	RESOLVED

Issue #02	**Liquidity governance could introduce arbitrage opportunities that hurt liquidity providers**
Severity	LOW SEVERITY
Description	When `DynaLiquidity` adds a token to a pool, it relies on `LIQUIDITY_GOVERNANCE` to provide a reasonable weight and token amount. There are no safeguards that prevent `LIQUIDITY_GOVERNANCE` from setting a certain weight but to provide a token amount that is much higher or lower than the correct ratio that should be added to the pool. If the token amount provided is way off to make the ratio in the pool correct, this would introduce arbitrage opportunities in the pool and current liquidity providers would be hurt by this. One solution could be to introduce certain percentage limits that the provided amount could be off to minimize the possible arbitrage opportunity or to set gradual weights change in the same transaction with `addToken()`. Similarly, `LIQUIDITY_GOVERNANCE` can mint an unlimited amount of BPT tokens (or very little) for this token addition to the pool that could harm (or be a gain for) existing liquidity providers. Consider again introducing a certain percentage limit of how much the discrepancy between the provided funds and the deserved BPT tokens to be minted could be.
Recommendation	Consider implementing the recommendations mentioned above.
Resolution	RESOLVED

Issue #03	The contract is not upgradeable
Severity	INFORMATIONAL
Description	Considering that `DynaLiquidity` integrates with BalancerV2 vaults and the managed pool contracts which have very complex inheritance structures and all functions of the contract are already access controlled, it would make sense for `DynaLiquidity` to be upgradable to retain the flexibility to fix bugs or add missing features. For example, DynaLiquidity cannot call `pool.setJoinExitEnabled()`.
Recommendation	Consider implementing the recommendation mentioned above.
Resolution	ACKNOWLEDGED

Issue #04	Typographical issues
Severity	INFORMATIONAL
Description	It is considered best practice to have the letter "I" in front of the names of interfaces. Consider renaming `DynaLiquidityAPI` to `IDynaLiquidityAPI`. — Emit events on each state changing function. This applies to every contract. — The comments above `addToken()` have a typographical error. "The address of the toke to add" should be "The address of the token to be added".
Recommendation	Consider making the changes mentioned above.
Resolution	RESOLVED

2.2 DynaLiquidityLib

`DynaLiquidityLib` is a library that contains the main implementation logic for `DynaLiquidity`.

2.2.1 Issues & Recommendations

Issue #05	`createManagedPool()` can cause all funds to be lost
Severity	MEDIUM SEVERITY
Description	`createManagedPool()` should not be able to be called more than once. Each `DynaLiquidity` contract is meant to manage a particular managed pool and once configured it should not be overwritten. If `createManagedPool()` is called a second time, this would overwrite the `liquidityStorage().pool` value and `DynaLiquidity` will lose access to the initial pool, all the funds in it and to all asset managers of this pool.
Recommendation	Check if `storage.pool != address(0)` and prevent further calls to the function.
Resolution	RESOLVED

Issue #06	**Not using `safeTransferFrom` can cause DOS with USDT or other non-standard tokens**
Severity	LOW SEVERITY
Description	Within `DynaLiquidityLib`, there are several sections where `transferFrom` is used instead of `safeTransferFrom`. The issue is that some non-standard token implementations such as USDT on mainnet do not return a boolean on `transfer/transferFrom` which will cause the call to revert. `addInitialLiquidity()` uses `safeTransferFrom` which allows the owner to deploy capital into the pool but adding more or trying to exit the pool after that would not be possible since `transferFrom` is used there.
Recommendation	Consider using `safeTransferFrom` instead of `transferFrom` throughout the codebase.
Resolution	ACKNOWLEDGED TransferFrom is used on L242, L403, L433.

Issue #07	**`joinWithSingleAsset()` will always revert for USDT or other weird tokens**
Severity	LOW SEVERITY
Description	USDT and other similar tokens require the allowance to be zero before changing it to another non zero value. At first during the creation stage of the pool, `DynaLiquidity.addInitialLiquidity()` gets called by `DynaLiquidityFactory` and the allowance of DynaLiquidity would be set to the Balancer vault to `type(uint).max`. If the liquidity manager tries to provide liquidity with USDT again via `joinWithSingleAsset()`, then the call would revert since `approve` instead of `forceApprove` is used. Additionally, in `addInitialLiquidity()`, the `safeApprove()` function is used. This is now deprecated and should be replaced with `forceApprove()` as well.
Recommendation	Consider using `forceApprove` instead of `approve` or `safeApprove` throughout the codebase.
Resolution	ACKNOWLEDGED

Issue #08	Managed pool creation can be front-run
Severity	LOW SEVERITY
Description	`createManagedPool()` calls `ManagedPoolFactory.create()` to deploy the pool. `create()` is a permissioned function that can be called by anyone and uses `CREATE2` for pool deployments. This means that anyone can monitor the `createManagedPool()` transaction, copy the parameters and call `ManagedPoolFactory.create()` directly using these parameters. As result, `createManagedPool()` would fail since the pool would already be deployed. Since there is no significant economic incentive and it could be easily circumvented by providing a different salt value, this issue is marked as Low.
Recommendation	This is more of an inherent issue to the design, so in such a potential case the team could provide priority fees to make sure the transaction would be executed first or wait for some time and then try again.
Resolution	ACKNOWLEDGED

Issue #09	`tokenReferenceValue()` can report 0 and logic will not revert
Severity	LOW SEVERITY
Description	In `addToken()`, `tokenReferenceValue()` is used (within `previewAddToken()` -> `_getTokenReferenceAssetValues()`) to calculate the slippage that occured. The issue is that if the oracle malfunctions, the function logic will continue and not revert, it will return 0 as a reference value. In this scenario, the slippage will most likely make the `addToken` call revert but this can result in incorrect calculations if the `tokenReferenceValue()` function is used to calculate fees etc., and the method using it does not revert if it returns 0.
Recommendation	Consider making `tokenReferenceValue()` revert instead of returning 0 values.
Resolution	

Issue #10	Insufficient validation
Severity	INFORMATIONAL
Description	Within `removeToken()`, consider adding an additional parameter for the `burnAmount` during token removal instead of hardcoding it to 0. There is already such a param when adding tokens. — Within `joinWithSingleAsset()`, add an additional variable called `tokenExists` which should be set to `true` if it matches any of the tokens in the loop. This is an efficient way to return early and skip the external call if the token is not registered to the pool — Consider if it is intentional that the token loop starts iteration from index 0 (BPT token) instead of 1.
Recommendation	Consider implementing the recommended validations.
Resolution	RESOLVED

Issue #11	Typographical issues
Severity	INFORMATIONAL

Description	Remove the following lines as they are not used:
	
	`- import "../interfaces/DynaLiquidityAPI.sol";`
	
	`- import "../interfaces/MinimalBalancerAPI.sol";`
	
	`- error MissingLiquidityGovernance();`
	
	`- error TokenWeightNotZero(address tokenAddress);`

Recommendation	Consider fixing the issues mentioned above.
Resolution	RESOLVED

2.3 DynaLiquidityProxyAssetManager

`DynaLiquidityProxyAssetManager` is responsible for the management of one asset belonging to a particular managed pool. It allows DynaLiquidity to move liquidity out from the managed pool to a strategy.

2.3.1 Privileged Functions

- `initialize [onlyRole]`
- `setProxyImplementation [onlyRole]`
- `removeToken [onlyRole]`
- `withdrawFromPool [onlyRole]`
- `depositToPool [onlyRole]`

2.3.2 Issues & Recommendations

Issue #12	**Managed assets are not accounted for and could lead to assets not being returned to the pool from proxies**
Severity	HIGH SEVERITY
Description	Changing the proxy implementation is done via the `setProxyImplementation()` function: 1. Call `liquidate` on the current proxy, which returns the tokens to the Managed pool, so that they can be used by the new proxy. 2. Assign the new proxy The issue is that for point 1, the managed amount is used, which should point to the amount of tokens that have been outsourced for management to a 3rd party protocol. When `withdrawFromPool()` is called to transfer funds from the pool to the proxy, the managed assets are always set to 0. This means that there is no way to track the assets that have been transferred out from the pool to the proxy and since managed is always 0, the proxy will never return the funds upon `liquidation()`. The same issue applies to `removeToken()`, where the assets will not be deposited from the proxy to the pool before token removal. Furthermore, not having correct maintenance of the managed variable introduces arbitrage opportunities within the pool that could hurt liquidity providers. For example, if some funds are taken out of the pool to be invested in a strategy, the current logic will reduce the cash and will set managed to 0. This effectively means that liquidity providers lose value and will receive less in exchange for their BPT and also means that now the token that was invested has a reduced amount in the pool and it will be treated as the price of this token has changed. This will affect swaps because the external weighted math contract performs swap calculations based on the tokens cash + managed balances.

	Managed pools are meant to invest part of the pool's funds while maintaining a correctly managed balance in order not to affect the price of the tokens in the pool during investments.
Recommendation	The managed funds are not accounted for properly as they are reduced to 0 upon withdrawal. The idea of this variable is to keep track of the cash balances that have been taken out of the pool by the asset manager. The exact management of those balances depends on the specific goal of the protocol. Normally the straightforward approach would be to update the managed balances with the amount that has been taken out and then reduce it with the amounts that are deposited back (the managed balances would still have to be updated before deposit in case there are additional profits that would be added as well). This would make it a reliable source when deciding on the amount to liquidate from the to-be removed proxy.
Resolution	

Issue #13	Proxy migration might fail
Severity	MEDIUM SEVERITY
Description	When calling `setProxyImplementation()`, the old proxy is first liquidated and then `invest()` is called for the new proxy. There is an inconsistency here because the amount to invest (withdraw from the Pool) is based on the current balances of asset tokens sitting in the contract instead of the amounts that were returned back to the pool by the old proxy. If the balances in the pool are lower than the ones in the contract, the call would revert. A potential scenario for this might be if `invest()` is called while there is no proxy configured (which stores the tokens in the contract).
Recommendation	Overall the logic is not very consistent and makes it a bit difficult to track the token flow. Consider if it makes sense to change the proxy implementation flow to something like the following: 1. Liquidate the old Proxy 2. Invest the balances that were returned in point 1 back in the new proxy Alternatively, step 2 could be skipped, since `invest` can be called at any time after that with a specific amount. Also if the intent here is to deposit into the pool any leftover assets sitting in the contract, then `liquidate()` instead of `invest()` should be called for the new proxy.

Resolution

FAILED RESOLUTION

The issue is still present.
There are 2 main parts in `setProxyImplementation()` that are before the change of `proxyImplementation` and after.

Before the change, the managed balance is taken from the Balancer vault and `liquidate` is called with this amount. The managed balance in the balancer vault is a combination of the balance of the asset manager and the `proxyImplementation` (as per `depositToPool/withdrawFromPool` update operation).

Issue #1
The managed balance recorded in the vault can be higher than the funds the `proxyImplementation` (the strategy) has due to the strategy registering losses or just because the managed balance is a combination of the strategy funds and the asset manager balance (anybody can increase the asset manager token balance by transferring an amount to the asset manager in order to achieve DOS), then `proxyImplementation.liquidate()` will most likely revert because `setProxyImplementation()` will try to take strategy funds + the asset manager balance from the strategy. This will prevent governance from successfully changing the `proxyImplementation`.

The remove token function uses the same pattern and will revert as well.

Issue #2
In the second part of the function after changing `proxyImplementation` to the new proxy, the current balance of the asset manager is recorded and `invest()` is called with it.

This would be incorrect since `invest()` withdraws an amount from the pool and transfers it to the `proxyImplementation`. There is no connection between the asset manager balance and the amount that needs to be withdrawn from the pool and invested in the new proxy. Most probably, the intention was to record the balance of the old proxy and to invest this same amount to the new proxy but for this to happen the current code should be changed a bit.

Issue #14	Use `safeTransfer` for ERC20 transfers
Severity	LOW SEVERITY
Description	Within `removeToken()`, consider using `safeTransfer()` to transfer the token since it could be any token address. This is a best practice and standard approach for protocols that work with arbitrary tokens.
Recommendation	Use `safeTransfer` for all ERC20 token transfers.
Resolution	RESOLVED

Issue #15	Insufficient validation
Severity	INFORMATIONAL
Description	Inside the `initialize` function, check that `liquidityGovernance` and `assetManagement` addresses are not set to 0. — Within `removeToken()`, make sure that `tokenRecipient` is not `address(0)`. — Within `depositToPool()`, check if the amount to deposit is bigger than the actual balances in the contract and reduce them like this: `amount > balanceOf(asset) ? balanceOf(asset) : amount`
Recommendation	Consider making the changes mentioned above.
Resolution	RESOLVED

Issue #16	Typographical issues
Severity	INFORMATIONAL
Description	Remove the following line as it is not used: `import "../interfaces/MinimalBalancerAPI.sol";` — `depositToPool()` and `withdrawFromPool()` are public which implies that they could be called directly by somebody with the `ASSET_MANAGEMENT` role. This is not possible since `DynaLiquidity` has deployed the proxy asset manager and all roles initially were given to `DynaLiquidity`. There are no functions in `DynaLiquidity` that can give any of the roles in the proxy asset manager. — Consider making `depositToPool()` and `withdrawFromPool()` private.
Recommendation	Consider making the changes mentioned above.
Resolution	PARTIALLY RESOLVED `depositToPool()` and `withdrawFromPool()` are still public.

2.4 DynaLiquidityAssetManagerLib

`DynaLiquidityAssetManagerLib` is used by both `DynaLiquidity` and `DynaLiquidityLib` and contains logic for making calls to the main functions of `DynaLiquidityProxyAssetManager`.

2.4.1 Issues & Recommendations

Issue #17	Typographical issues
Severity	INFORMATIONAL
Description	`LIQUIDITY_AsSET_MANAGER_STORAGE_POSITION` variable should be renamed to `LIQUIDITY_ASSET_MANAGER_STORAGE_POSITION`. — `event UpdatedDynaLiquidityAssetManager(address tokenAddress, address assetManager);` is not used. — `initializeProxyAssetManager()`, `invest()` and `liquidate()` do not have NatSpec comments. Consider adding such to all relevant functions in the codebase that do not have them.
Recommendation	Consider fixing all typographical issues.
Resolution	RESOLVED

2.5 DynaLiquidityFactory

`DynaLiquidityFactory` is responsible for the initial configuration, implementation selection and deployment of DynaLiquidity.

2.5.1 Privileged Functions

- `setDynaLiquidityRegistry [onlyRole]`
- `setDynaLiquidityImplementation [onlyRole]`
- `setManagedPoolFactory [onlyRole]`
- `setProxyAssetManagerImplementation [onlyRole]`
- `initDynaLiquidity [onlyRole]`
- `grantRole [onlyRole]`
- `revokeRole [onlyRole]`

2.5.2 Issues & Recommendations

Issue #18	Emit events on state changing functions
Severity	INFORMATIONAL
Description	The following state changing functions do not emit events: - `setDynaLiquidityRegistry()` - `setDynaLiquidityImplementation()` It is considered best practice to emit events that mark changes in the state of the smart contract. It provides transparency to anyone observing how the contract state has changed and also allows querying by off-chain listeners when that is necessary.
Recommendation	Consider implementing the recommendations mentioned above.
Resolution	RESOLVED

Issue #19	Typographical issues
Severity	INFORMATIONAL
Description	Remove the following line as it is not used: `import "../interfaces/MinimalBalancerAPI.sol";`
Recommendation	Consider fixing the typographical issues.
Resolution	RESOLVED

2.6 DynaLiquidityRegistry

`DynaLiquidityRegistry` is responsible for keeping track of all DynaLiquidity contracts.

2.6.1 Privileged Functions

- `setLiquidityEnabled [onlyRole]`
- `setDynaLiquidityType [onlyRole]`
- `registerDynaLiquidity [onlyRole]`
- `grantRole [onlyRole]`
- `revokeRole [onlyRole]`

2.6.2 Issues & Recommendations

Issue #20	Emit events on state changing functions
Severity	INFORMATIONAL
Description	The following state changing functions do not emit events: - `setLiquidityEnabled()` - `setDynaLiquidityType()` - `registerDynaLiquidity()` It is considered best practice to emit events that mark changes in the state of the smart contract. It provides transparency to anyone observing how the contract state has changed and also allows querying by off-chain listeners when that is necessary.
Recommendation	Consider implementing the recommendations mentioned above.
Resolution	RESOLVED

2.7 BalancerV2DynaZapper

`BalancerV2DynaZapper` enables users to swap between ERC20 tokens and Balancer pool tokens. It identifies which tokens are Balancer pool tokens by allowing the admin to set the preferred token for each pool.

If the token in or the token out is not the preferred one for the user or the pool, a swap via one of the DynaRouters is executed.

2.7.1 Privileged Functions

- `initialize [onlyRole]`
- `setPreferredPoolAsset [onlyRole]`
- `grantRole [onlyRole]`
- `revokeRole [onlyRole]`

2.7.2 Issues & Recommendations

Issue #21	**`BalancerV2Zapper` does not check if the used `DynaLiquidity` has an active flag in `DynaLiquidityRegistry`**
Severity	MEDIUM SEVERITY
Description	`BalancerV2Zapper` calls `setSwapEnabled()` before and after `joinPool()` or `exitPool()`. `DynaLiquidity` contracts can be marked in `DynaLiquidityRegistry` as being active or not but this is not checked before `BalancerV2Zapper` makes calls to `DynaLiquidity`.
Recommendation	Consider introducing a check if the used `DynaLiquidity` is an active one on joins.
Resolution	PARTIALLY RESOLVED A check was introduced in `_joinWithSingleAsset()` but not in `_exitWithSingleAsset()`.

Issue #22	Arbitrary contracts can be called and given permission to
Severity	LOW SEVERITY
Description	The `_swapTokenToToken()` takes a router address, which is decoded from the `swapData` provided by the caller. The issue is that the router is not validated in any way and `BalancerV2Zapper` freely approves and calls it. The approval is up to the amount that `msg.sender` has already transferred in and the contract is not meant to store any funds. No serious issues have been detected from this, however calling and approving arbitrary addresses is rarely a good practice — it increases the attack surface and could potentially lead to unexpected behavior.
Recommendation	Consider making a call to `DynaRouterRegistry` to see if the router is a registered one. This ensures that only the expected routers can get called.
Resolution	RESOLVED

Issue #23	**`BalancerV2Zapper` does not support swaps with pools that contain the BPT token of other pools**
Severity	LOW SEVERITY
Description	The admin is responsible for setting the preferred token for each supported managed pool. If a pool does not have a preferred token set by the admin then this BPT token/pool is treated as a normal token. There is an edge case where the caller might want to swap a BPT token of pool X for the BPT token of pool Y. The logic in the swap function does not support this case as both checks that determine if the `_exitPool()` or the `_joinPool()` should be called will not pass ("`if (tokenInIsBpt && ! tokenOutIsBpt) {`" and "`if (!tokenInIsBpt && tokenOutIsBpt) {`") since both `tokenIn` and `tokenOut` will have a preferred token set by the admin. In this case, the call will simply return without making any state changes. Similar situation would happen if a managed pool contained the BPT token of another pool and the user wanted to swap BPT1 for BPT2 in this managed pool.
Recommendation	To support this edge case, consider introducing an argument to the `swap()` function that would allow the caller to determine if the swap is intended to be a join or an exit one instead of determining this based on if the admin have set the preferred token to `tokenIn` or `tokenOut`. This way the code would know if it should swap the `tokenIn` for another token that could then be used to join pool Y and get the other BPT token.
Resolution	ACKNOWLEDGED

Issue #24	Encoding of router swap data could be skipped in an edge case
Severity	INFORMATIONAL
Description	The check should be "`if (length > 0) {`" instead of "`if (length > 1) {`". If `routerSwapData.length` is 1 then the population of `swapData` will be skipped and only the router will be in included in the final `swapData` array. As of now, we did not find a router that would need a path with just one value in it. This seems to not to be an issue now but since new router implementations could introduce this issue in the future we decided to mention it.
Recommendation	Consider implementing the fix mentioned above.
Resolution	RESOLVED

Issue #25	Typographical issues
Severity	INFORMATIONAL
Description	`intitialBalanceOut` should be `initialBalanceOut`. — `import "../interfaces/DynaVaultAPI.sol";` is not used — `error MissingAddress();` is not used
Recommendation	Consider fixing the typographical issues.
Resolution	RESOLVED

2.8 VaultDynaZapper

`VaultDynaZapper` allows users to swap between ERC20 tokens and `DynaVault` shares. If the in or out token is not the desired one, it uses `DynaRouterMeta` to perform a swap.

2.8.1 Privileged Functions

- `initialize [onlyRole]`
- `grantRole [onlyRole]`
- `revokeRole [onlyRole]`

2.8.2 Issues & Recommendations

Issue #26	Usage of `transferFrom` with arbitrary `from` value
Severity	HIGH SEVERITY
Description	Both `BalancerV2Zapper` and `VaultZapper` use `safeTransferFrom` in multiple sections of the code. The issue is that before a user can use `swap()`, they need to approve the contract first for a certain amount. If this is not done in one transaction, then an attacker can front run the `swap()` call of the user and use their allowance to perform a swap with their tokens and specify an attacker owned address as the receiver. As a result, the assets of the user will be stolen.
Recommendation	Consider always using `safeTransferFrom` with `msg.sender` as the from value on all places throughout the codebase.
Resolution	RESOLVED

Issue #27	`_redeem` attempts to take `tokenIn` amount twice from the caller of `swap()`
Severity	HIGH SEVERITY
Description	The caller of `VaultZapper.swap()` will first approve `VaultZapper` for X amount. `VaultZapper` then transfers the tokens on L174. In `_redeem()` on L232, `DynaVaultAPI(vault).redeemCheckSlippage(sharesIn, receiver, owner, minAmountOut);` that is called will attempt to transfer the X amount again from the owner (that is the from address argument in the swap function). If there is not enough allowance, the call will revert. If there is enough allowance however, the result would be the first X amount of tokens to be left in the `VaultZapper` and the second X amount to be used for the redemption.
Recommendation	Replace the owner argument on L232 with `address(this)`. This way (according to the `DynaVault` implementation), `VaultZapper` will not need to approve the vault since the caller and the owner will be the same address and the amount that was initially transferred will be used for the redemption.
Resolution	RESOLVED

Issue #28	`VaultZapper` can deposit funds into disabled vaults
Severity	MEDIUM SEVERITY
Description	The `VaultRegistry` contract is responsible for keeping track of all Dyno vaults. These vaults can be registered by the registry admin but they can also be enabled/disabled via `setVaultEnabled()`. The issue is that despite the status of the vault, `VaultZapper` allows for new deposits to be successful.
Recommendation	Consider reverting new deposits to vaults that are disabled. The recommendation does not include a revert during redemptions since users should be able to take out their funds from the vault.
Resolution	RESOLVED

Issue #29	Unnecessary approval of `DynaVault`
Severity	LOW SEVERITY
Description	On L234 in `VaultZapper`, an approval is made from `VaultZapper` to the `DynaVaultAPI(vault)`. However, according to the `DynaVault` implementation, `redeemCheckSlippage()` does require an approval when caller and the owner are the same: https://github.com/Singularity-DAO/DynaVaults/blob/35bb164db6166799a893d8dadbf1ed262b26a240/contracts/dynavaults/DynaVault.sol#L343-L345.
Recommendation	Consider not approving `DynaVault` when the owner argument of `redeemCheckSlippage()` is `address(this)`.
Resolution	RESOLVED

Issue #30	Hard coded `useWhiteList`
Severity	LOW SEVERITY
Description	`VaultZapper` uses the `useWhiteList` variable to determine on each `swap()` whether to check if the `msg.sender` is allowed to receive `tokenOut`. There is no way for the admin to set the `useWhiteList` variable to `false` and this forces the contract to always check if the callers are eligible to receive the `tokenOut` asset.
Recommendation	Consider introducing a setter function for `useWhiteList` or allow the admin to set the value in the `initialize()` function.
Resolution	RESOLVED

Issue #31	Typographical issues
Severity	INFORMATIONAL
Description	A widely adopted convention is to have I in front of interface names. `DynaRouterAPI` should be `IDynaRouterAPI`. Also consider adding natspec comments to `DynaRouterAPI` interface functions. — Remove commented code. — On L71 instead of "The address fo the spender" it should be "The address of the spender". — On L202 `intitialBalanceDeposit` should be `initialBalanceDeposit`.
Recommendation	Consider fixing the typographical issues.
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY