PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Singularity DAO (DynaVaults)

30 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 Global Issues 8

1.3.2 VaultTokens 8

1.3.3 VaultTokensLib 8

1.3.4 VaultGovernance 9

1.3.5 VaultGovernanceLib 9

1.3.6 VaultFeesLib 9

1.3.7 VaultStrategiesLib 9

1.3.8 VaultConfigLib 10

1.3.9 DynaVaultManager 10

1.3.10 VaultManagerLib 10

1.3.11 DynaVault 10

1.3.12 DynaVaultLib 11

1.3.13 VaultSimulatorLib 11

1.3.14 VaultManagerSimulatorLib 11

2 Findings 12

2.1 Global Issues 12

2.1.1 Issues & Recommendations 13

2.2 VaultTokens 15

2.2.1 Issues & Recommendations 16

2.3 VaultTokensLib 17

2.3.1 Issues & Recommendations 19

2.4 VaultGovernance 26
2.4.1 Issues & Recommendations 28
2.5 VaultGovernanceLib 29
2.5.1 Issues & Recommendations 30
2.6 VaultFeesLib 31
2.6.1 Issues & Recommendations 32
2.7 VaultStrategiesLib 33
2.7.1 Issues & Recommendations 35
2.8 VaultConfigLib 38
2.8.1 Issues & Recommendations 40
2.9 DynaVaultManager 41
2.9.1 Issues & Recommendations 42
2.10 VaultManagerLib 43
2.10.1 Issues & Recommendations 45
2.11 DynaVault 49
2.11.1 Issues & Recommendations 51
2.12 DynaVaultLib 56
2.11.1 Issues & Recommendations 58
2.13 VaultSimulatorLib 65
2.13.1 Issues & Recommendations 67
2.14 VaultManagerSimulatorLib 68
2.14.1 Issues & Recommendations 70

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 DynaVaults 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/e35e854a8724ba4a0a1fa5e7e89ad6856c0eb090
Resolution	https://github.com/Singularity-DAO/DynaVaults/commit/08dd423aac6fc0634b9de721d110017590a8fe3d

1.2 Contracts Assessed

Contract	Address	Live Match
VaultTokens	Dependency	MATCH
VaultStrategiesLib	0xf22Bd338Ccb6446a25920d1858E50D709B663e9E	MATCH
VaultGovernance	Dependency	MATCH
VaultGovernanceLib	Dependency	MATCH
VaultFeesLib	0xb130375b543Bd919D85351B73636439C28D90Db6	MATCH
DynaVaultManager	0x95cF606f7e499549D83BD3c8a1e5D97FDF36688B	MATCH
DynaVault	0xC6B3345612aD9F385d5C8ffd1eCe98f7318ECf52	UNMATCHED
DynaVaultLib	0x3B94AfA809b113eF118C6D4ffC009FdF1A51ee08	UNMATCHED
VaultManagerLib	0x7B273c267B7De25811883D69D26BfFaEbf71C184	UNMATCHED
VaultConfigLib	0x3CE426BB2fCB9E1300BbE03E2D34D7b3B7347Db5	MATCH
VaultSimulatorLib	-	-
VaultManagerSimulatorLib	0x620978615e37f2c494cB9FcC2F9981029533fB39	UNMATCHED

DynaVault: https://www.diffchecker.com/gJd1eaDH/

DynaVaultLib: https://www.diffchecker.com/AdwTaDIh/

VaultManagerLib: https://www.diffchecker.com/j4Plnh9u/

VaultManagerSimulatorLib: https://www.diffchecker.com/vvu0zE6V/

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change)
Governance	2	1	-	1
High	2	2	-	-
Medium	3	3	-	-
Low	15	10	3	2
Informational	16	15	1	-
Total	**38**	**31**	**4**	**3**

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 Global Issues

ID	Severity	Summary	Status
01	GOV	High centralisation risk	ACKNOWLEDGED
02	LOW	Update to latest Solady `FixedPointMathLib`	✓ RESOLVED
03	INFO	Typography issues	PARTIAL

1.3.2 VaultTokens

No issues found.

1.3.3 VaultTokensLib

ID	Severity	Summary	Status
04	GOV	No limitation enforced for the value of a tokens `watermarkDuration`	✓ RESOLVED
05	HIGH	The amount variable is shadowed in `decreaseTokenDebt`	✓ RESOLVED
06	LOW	Insufficient safety checks to prevent DOS of some functions	✓ RESOLVED
07	LOW	During token removal, the strategies are not checked to be empty	PARTIAL
08	LOW	Approvals are not reset upon removing reserve tokens	✓ RESOLVED
09	INFO	Insufficient validation	✓ RESOLVED
10	INFO	Events are not emitted during important state changes	✓ RESOLVED
11	INFO	Typographical issues	✓ RESOLVED

1.3.4 VaultGovernance

ID	Severity	Summary	Status
12	INFO	Typographical issues	✓ RESOLVED

1.3.5 VaultGovernanceLib

No issues found.

1.3.6 VaultFeesLib

ID	Severity	Summary	Status
13	INFO	Lack of re-initilizer check	✓ RESOLVED

1.3.7 VaultStrategiesLib

ID	Severity	Summary	Status
14	LOW	Strategy not revoked properly	ACKNOWLEDGED
15	INFO	Insufficient validation	✓ RESOLVED
16	INFO	Events are not emitted during important state changes	✓ RESOLVED

1.3.8 VaultConfigLib

ID	Severity	Summary	Status
17	INFO	Typographical issues	✓ RESOLVED
18	INFO	Events are not emitted during important state changes	✓ RESOLVED

1.3.9 DynaVaultManager

ID	Severity	Summary	Status
19	INFO	Typographical issues	✓ RESOLVED

1.3.10 VaultManagerLib

ID	Severity	Summary	Status
20	MEDIUM	Fee shares are not minted during rebalancing	✓ RESOLVED
21	LOW	Investing in strategies is possible even on revoked strategies	PARTIAL
22	LOW	`reportReserve` does not check if token has been removed	✓ RESOLVED
23	LOW	`investStrategy` does not report reserve before updating strategy ratios	ACKNOWLEDGED

1.3.11 DynaVault

ID	Severity	Summary	Status
24	HIGH	Reserves are not reported during minting	✓ RESOLVED
25	LOW	Lack of minimum deposit assets check in proportional redeem	✓ RESOLVED
26	LOW	Vault is susceptible to the first depositor attack	✓ RESOLVED
27	INFO	Insufficient validation	✓ RESOLVED

1.3.12 DynaVaultLib

ID	Severity	Summary	Status
28	MEDIUM	Redemption fees are not always deducted	✓ RESOLVED
29	MEDIUM	Transferred redemption fees are not deducted from `tokenIdle`	✓ RESOLVED
30	LOW	CEI pattern not followed inside `transferProportional`	✓ RESOLVED
31	LOW	Not all losses are considered when checking for the `maxLoss` threshold before burn	PARTIAL
32	INFO	Gas optimizations	✓ RESOLVED
33	INFO	Insufficient validation	✓ RESOLVED
34	INFO	Events are not emitted during important state changes	✓ RESOLVED

1.3.13 VaultSimulatorLib

ID	Severity	Summary	Status
35	INFO	Typographical issues	✓ RESOLVED

1.3.14 VaultManagerSimulatorLib

ID	Severity	Summary	Status
36	LOW	The manager report reserve simulation does not properly simulate the actual function	✓ RESOLVED
37	LOW	Inaccurate rebalance simulation	✓ RESOLVED
38	LOW	Inaccurate report simulation	✓ RESOLVED

2 Findings

2.1 Global Issues

The issues listed in this section apply to the protocol as a whole. Please read through them carefully and take care to apply the fixes across the relevant contracts.

2.1.1 Issues & Recommendations

Issue #01	High centralisation risk
Severity	GOVERNANCE
Description	The entire infrastructure is governed by the `VaultGovernance` contract, which holds authority over critical operations and management. For example, if the governance or management keys are compromised, the entire architecture and its dependent contracts are at risk. This creates a single point of failure with potentially catastrophic consequences.
Recommendation	Consider having all the roles of governance under timelocks and a trusted multi-signature set-up. Some of them might need a stronger signature threshold—for example, governance can be under a ⅝ quorum while guardian can be under a ⅗ quorum.
Resolution	ACKNOWLEDGED

Issue #02	Update to latest Solady `FixedPointMathLib`
Severity	LOW SEVERITY
Description	The protocol uses Solady 0.0.192, while Solady has been gone through various security reviews. Consider updating to the latest version of the Solady which might cover edge-cases in the `FixedPointMathLib`.
Recommendation	Update to the latest `FixedPointMathLib`.
Resolution	RESOLVED

Issue #03	Typography issues
Severity	INFORMATIONAL
Description	There are multiple `console.logs` throughout the codebase. Make sure to remove them before the actual deployment, since they increase the contract size unnecessarily. — There are 2 to `TODO` notes in the following places: 1. `VaultSimulatorLib L196` - a simulation of strategy withdraw should be added to the simulatedCalcRedeemProportional() function 2. `VaultManagerSimulatorLib L229` - add _simulateHarvestStrategy which uses simulateReport(Strategy) Make sure that those TODOs are handled, so that important parts of the logic are not missed. — Typographical errors throughout all the contracts sometimes make it hard to understand the comments or variables. We did not include all these issues in the report to reduce the number of issues. We advise the team to go through all the contracts to fix these issues. — The conventional naming of several variables is not respected, e.g. `managementfee` and `net_performance_fee`.
Recommendation	Consider implementing the above recommendations.
Resolution	PARTIALLY RESOLVED

2.2 VaultTokens

`VaultTokens` serves as the primary interface layer for all token-related operations within the DynaVaults ecosystem, inheriting fee management capabilities while delegating comprehensive token portfolio management to its underlying library implementation for seamless multi-asset vault operations.

It provides complete token lifecycle management through functions that enable dynamic addition and removal of reserve tokens beyond the primary deposit asset, while maintaining strict governance controls and validation to ensure only authorized assets can be integrated into the vault's investment portfolio with configurable limits on total token count to prevent operational complexity.

The contract implements sophisticated asset allocation tracking that distinguishes between idle tokens (available for immediate use) and debt tokens (deployed to investment strategies), while maintaining accurate accounting of deposit debt ratios that represent target allocation percentages across the entire portfolio to ensure proper rebalancing mechanics.

Through its comprehensive querying capabilities, it provides real-time visibility into token statistics including watermarks for performance tracking, profit/loss accumulation, strategy deployment ratios, and historical reporting timestamps, enabling both automated systems and management interfaces to monitor portfolio health and make informed allocation decisions.

The system coordinates closely with strategy management by maintaining token-specific strategy queues that define withdrawal order during liquidations, while handling complex debt accounting updates that occur during token swaps to ensure target allocation ratios are properly maintained as portfolio composition changes through rebalancing operations.

Its watermark management functionality allows configurable performance measurement periods per token to account for varying volatility characteristics of different assets, ensuring fair performance fee calculations across diverse portfolio components while supporting the vault's sophisticated fee structure that accounts for both time-based management fees and performance-based rewards.

As the token coordination hub of the DynaVaults project, it seamlessly integrates portfolio diversification, target allocation management, strategy coordination, and comprehensive asset tracking while providing the foundational infrastructure for multi-token investment strategies that maintain proper debt accounting and enable sophisticated rebalancing mechanisms across diverse asset portfolios.

2.2.1 Issues & Recommendations

No issues found.

2.3 VaultTokensLib

`VaultTokensLib` serves as the sophisticated multi-token portfolio management engine for DynaVaults, implementing a revolutionary deposit debt ratio system that enables precise target allocation tracking across diverse asset portfolios while maintaining accurate accounting for both idle reserves and strategy-deployed capital.

It establishes a deposit token as the primary asset for user interactions while enabling management to add configurable reserve tokens that expand investment opportunities, where each token maintains comprehensive statistics including idle balances (immediately available), debt balances (deployed to strategies), profit/loss tracking, and performance watermarks for fair fee calculations across volatile assets.

The library implements an innovative target allocation mechanism through deposit debt ratios that represent desired portfolio weights - for example, starting with 10,000 USDC (100% allocation) and swapping 5,000 USDC to WBTC creates target ratios of 50% USDC and 50% WBTC, which remain stable even as new deposits temporarily shift actual holdings toward the deposit token until rebalancing occurs.

Its sophisticated swap accounting system handles three distinct scenarios: deposit-to-reserve swaps (expanding diversification), reserve-to-deposit swaps (consolidating positions), and reserve-to-reserve swaps (rebalancing between alternatives), each maintaining precise debt ratio adjustments while tracking any losses that occur during swap execution and updating token balances accordingly.

The watermark system provides configurable performance measurement periods per token, allowing governance to set longer durations for volatile assets to prevent excessive performance fee collection, while automatically updating watermarks when sufficient time has elapsed and maintaining separate tracking for reporting frequency versus performance measurement intervals.

Through its strategy queue management, the library maintains ordered lists of investment strategies per token that define withdrawal priority during liquidations, enabling sophisticated capital allocation while ensuring proper coordination between vault operations and external investment protocols through comprehensive debt tracking and strategy integration.

As the portfolio coordination engine of DynaVaults, it seamlessly manages complex multi-token portfolios with target allocation preservation, automated rebalancing support, performance-based fee calculations, and comprehensive asset tracking while providing the foundational infrastructure for sophisticated investment strategies that maintain proper debt accounting across diverse asset types and deployment mechanisms.

2.3.1 Issues & Recommendations

Issue #04	**No limitation enforced for the value of a tokens `watermarkDuration`**
Severity	GOVERNANCE
Description	The `setWatermarkDuration()` duration is a governance function that is used to update the intervals at which vault token prices get watermarked (fetching and caching the latest price from the oracle). The watermarks are an important mechanism in the protocol that is used to track price changes in vault tokens and determine if profit/loss has occurred. More precisely, this happens in `VaultManagerLib._reportProfitLoss()`, where the watermark value is used to determine the baseline price against which the current price is being compared: `uint256 baseline_value = Math.max(lastReportedValue, watermark)` In case governance gets compromised or turns malicious, `watermarkDuration` can be set to a very large value which would result in a stale watermark price that never gets updated. This would adversely affect the profit/loss calculations and the normal function of the system.
Recommendation	Make sure to add a reasonable MAX value for the duration that can be set inside `setWatermarkDuration()`.
Resolution	RESOLVED

Issue #05	The `amount` variable is shadowed in `decreaseTokenDebt`
Severity	HIGH SEVERITY
Description	In the `decreaseTokenDebt` function, there is a local variable `amount` that shadows the function parameter `amount`. This causes the function to use the uninitialized local variable instead of the intended parameter, resulting incorrect check and updates in the `tokenStats.tokenDebt`.
Recommendation	Consider removing the `amount` variable.
Resolution	RESOLVED

```
function decreaseTokenDebt(address tokenAddress, uint256
amount) external {
    // cached storage
    TokenStorage storage _storage = tokenStorage();

    // cached tokenDebt
    uint256 tokenDebtamount =
_storage.tokenStats[tokenAddress].tokenDebt;

    // cached amount
    uint256 amount;
    if (tokenDebtamount > amount) {
        // unchecked to save gas
        unchecked {
            amount = tokenDebtamount - amount;
    }
    } else {
            amount = 0;
    }

    // assign tokenDebt value
    _storage.tokenStats[tokenAddress].tokenDebt = amount;
}
```

Issue #06	Insufficient safety checks to prevent DOS of some functions
Severity	LOW SEVERITY
Description	Inside `withdrawIdle()`, add an additional check to make sure that `deltaDebt` is not bigger than `reserveDebt` before subtracting it. This is a safety measure that would prevent a DOS of the function in cases that there are miscalculations in the accounting of `reserveDebt`, which is not impossible given the high amount of state transitions that happen throughout the protocol. `withdrawIdle()` is used in important flows of the protocol which is why it is very important to ensure it cannot DOS.

Recommendation	Consider refactoring the code like this:
	`function withdrawIdle(address tokenAddress, uint256 assets) external {`
	`[...]`
	`if (tokenAddress != _storage.tokens[0]) {`
	`_storage.reserveDebt -= deltaDebt > _storage.reserveDebt ? _storage.reserveDebt : deltaDebt`
	`}`
	`}`
	
	This would prevent potential underflows, which are undesirable.
	
	Another function that needs additional safety check is `updateDebtAfterSwap()`, more precisely the following lines:
	
	L614—refactor it to how it is used in the other code blocks:
	`deltaDebt = Math.min( FixedPointMathLib.fullMulDiv(curDepositDebt, soldRatio, MAX_BPS), statstokenIn.depositDebt)`
	
	L625
	`_storage.reserveDebt -= deltaDebt;`
	
	to
	`_storage.reserveDebt -= Math.min(deltaDebt, _storage.reserveDebt );`

Resolution	RESOLVED

Issue #07	During token removal, the strategies are not checked to be empty
Severity	LOW SEVERITY
Description	`removeReserveToken()` is used to remove a supported token from the Vault. Each token has a corresponding array of strategies that the governance can add so that the available token balances could be invested into those strategies, which can generate yield and get distributed as profit to shareholders. The removal function ensures that the token does not have any idle or debt balances, before removing it from the tokens array. However, the function should also loop through all attached strategies to make sure there are no leftover assets in them as well. The main reason for such discrepancy might be a miscalculation in the report profit/loss flow. This would be an additional safety measure toe ensure that the token is ready to be withdrawn and nothing depends on it Additionally, also consider clearing the `_storage.tokenStrategies` array once the token has been removed. For example, `getTokenStrategies()` still returns all strategies even after token removal, which can produce unexpected behavior since it is used in different flows.
Recommendation	Make sure to verify that all strategies have been exited fully and then also remove them from the array so that the token is properly removed
Resolution	PARTIALLY RESOLVED Strategies balances are not checked.

Issue #08	Approvals are not reset upon removing reserve tokens
Severity	LOW SEVERITY

Description	When a new token is added as a reserve, the allowance is increased to `type(uint256).max`.
	
	`function addReserveToken(address tokenAddress) external {`
	`  [...]`
	`  DynaVaultAPI(VaultGovernanceLib.vault()).`
	`  approveAddedToken(tokenAddress);`
	`}`
	
	The `approveAddedToken` function approves `type(uint256).max` for the manager:
	`function approveAddedToken(address tokenAddress) external {`
	`  address _manager = configStorage().manager;`
	`  require(msg.sender == _manager, "!authorized");`
	`  if (IERC20(tokenAddress).allowance(address(this), _manager) == 0) {`
	`  IERC20(tokenAddress).safeIncreaseAllowance(_manager, type(uint256).max);`
	`  }`
	`}`
	
	This should be reset when a token is removed inside `removeReserveToken` to avoid leaving unnecessary dangling approvals, which might pose a risk if an attack vector somehow manages to exploit these approvals.
	
	As of now, we do not see a risk that is why this issue is marked as low severity.

Recommendation	Consider creating a function in the `VaultConfigLib` that can reset approvals for a token and that can be called only by the `DynaVaultManager`.
Resolution	RESOLVED

Issue #09	Insufficient validation
Severity	INFORMATIONAL
Description	`setWaterMarkDuration()` does not check if the token exists, so it is possible to update the watermark for non-existing tokens as well. Add a call to the `tokenExists()` function before updating the variable. — Within `updateDebtAfterSwap()`, add a top-level check after the `address(0)` validation to make sure that `tokenIn > statstokenIn.tokenIdle` and revert early instead at the end after all operations.
Recommendation	Consider implementing the above recommendations
Resolution	RESOLVED

Issue #10	Events are not emitted during important state changes
Severity	INFORMATIONAL
Description	The following function do not emit any events after updating storage variables: - `setWatermarkDuration()` - `setMaxTokens()` - `addReserveToken()` - `removeReserveToken()` - `reportProfitLossBalance()` Emitting events is a good practise that allows anyone to transparently track the configuration and state changes in a contract and is especially important in the case of DAOs.
Recommendation	Consider emitting events in the above state changing functions
Resolution	RESOLVED

Issue #11	Typographical issues
Severity	INFORMATIONAL
Description	The `IERC20` import is not used.
Recommendation	Consider fixing all typographical issues.
Resolution	RESOLVED

2.4 VaultGovernance

`VaultGovernance` serves as the foundational access control layer for the DynaVaults ecosystem, implementing a hierarchical three-tier governance model that balances security, operational efficiency, and emergency response capabilities through distinct roles with carefully defined permissions and responsibilities.

It establishes governance as the most powerful role designed for trusted multi-signature wallets that can modify core vault parameters, update fee structures, add/remove strategies, and make fundamental changes to vault operations, while management handles day-to-day operational decisions like strategy investments, liquidations, and routine rebalancing activities that require immediate response to market opportunities and risks.

The contract implements a guardian role specifically designed for emergency response scenarios, providing a lower-risk position that can trigger emergency shutdowns without having the authority to directly liquidate assets or modify core vault parameters, creating a safety mechanism that can halt operations during critical situations while preventing potential misuse of emergency powers.

Through its integration with `VaultGovernanceLib`, it provides comprehensive access control enforcement across all vault operations, ensuring that sensitive functions like strategy management, fee updates, and parameter modifications can only be executed by appropriately authorized roles while maintaining operational flexibility for time-sensitive management decisions.

The governance system supports role transitions and updates through secure setter functions that enforce proper authorization, enabling governance structures to evolve over time while maintaining security through the multi-signature governance requirement and separation of emergency powers from operational authority.

As the security foundation of the DynaVaults project, it ensures that vault operations maintain appropriate authorization levels while enabling responsive management of investment opportunities and comprehensive emergency protection through its carefully designed three-tier governance architecture that balances security, operational efficiency, and risk management.

2.4.1 Issues & Recommendations

Issue #12	Typographical issues
Severity	INFORMATIONAL
Description	In Solidity, it is considered a best practice to add an underscore to internal function, so that it is clear and easier to read that it is not public. Consider adding an underscore to `initializeGovernance()`. — Consider making `onlyGovernance()` and `onlyManagementOrGovernance()` `internal`, similar to the other "only.." functions as there is no point exposing them publicly.
Recommendation	Consider fixing the typographical issues.
Resolution	RESOLVED

2.5 VaultGovernanceLib

`VaultGovernanceLib` is a library serves as the core access control implementation engine for the DynaVaults ecosystem, providing a sophisticated three-tier authorization system that ensures proper separation of powers while enabling efficient vault operations through carefully designed permission hierarchies and role-based access controls.

It implements a governance-first architecture where governance represents the most privileged role (typically a multi-signature wallet) with authority to modify fundamental vault parameters, add or remove investment strategies and reserve tokens, update fee structures, and most importantly, manage role transitions by appointing new management and guardian addresses while maintaining the ability to update itself.

The library establishes management as the operational role designed for active vault administration, authorized to execute time-sensitive operations including asset swaps for rebalancing, strategy investments and liquidations, and day-to-day portfolio management decisions that require immediate market responsiveness, while being restricted from modifying core vault infrastructure or governance structures.

Its guardian implementation provides a specialized emergency response mechanism that grants authority to trigger emergency shutdowns across the entire vault system without providing access to asset liquidation or parameter modification functions, creating a safety valve that can halt operations during security incidents while preventing potential abuse of emergency powers.

The access control system provides granular permission checking through various modifier functions that enforce authorization at multiple levels, including vault-only operations (for internal contract calls), governance-or-strategy combinations (allowing strategies to self-revoke during emergencies), and emergency-authorized actions that combine management, governance, and guardian powers for critical situations.

As the security backbone of the DynaVaults project, it ensures that sensitive operations like strategy management (requiring governance approval), daily portfolio rebalancing (management authority), and emergency response (guardian powers) maintain appropriate authorization levels while providing the operational flexibility necessary for effective decentralized asset management through its carefully balanced three-tier governance architecture.

2.5.1 Issues & Recommendations

No issues found.

2.6 VaultFeesLib

`VaultFeesLib` has a crucial role in managing the fee structure associated with the vault's operations.

The library plays a pivotal role in the financial management of the vault, providing mechanisms for fee imposition and collection, which are integral to the vault's revenue model and operational sustainability.

2.6.1 Issues & Recommendations

Issue #13	Lack of re-initilizer check
Severity	INFORMATIONAL
Description	Similar to the logic of initializers in the other libraries, make sure to add a check in `initializeFees()` that prevents calling it a second time - e.g re-initializing it.
Recommendation	Add the following check: `require(_storage.managementFeeWallet == ZERO_ADDRESS , "ERR_REINIT")`
Resolution	RESOLVED

2.7 VaultStrategiesLib

`VaultStrategiesLib` serves as the comprehensive strategy orchestration engine for DynaVaults, implementing debt ratio management systems that enable precise capital allocation across multiple investment strategies while maintaining strict governance controls and risk management through configurable harvest limits and emergency shutdown mechanisms.

It establishes each strategy with configurable debt ratios (expressed in 18 decimals where 100e16 = 100%) that determine target capital allocation based on total available assets, while implementing safeguards through minimum and maximum debt per harvest limits that prevent excessive capital flows during individual harvest cycles, ensuring controlled and measured investment adjustments.

The library manages complete strategy lifecycle operations including governance-controlled additions with configurable parameters, seamless migrations that preserve debt ratios and performance history while transferring all positions to new strategy versions, and revocation mechanisms that safely wind down strategies by setting debt ratios to zero and preventing future deposits.

Its credit and debt calculation system continuously monitors each strategy's actual debt versus target allocation, determining available credit for new investments or outstanding debt requiring liquidation, while accounting for vault-level debt limits, token-specific constraints, and strategy-specific harvest boundaries to ensure optimal capital deployment.

The fee calculation engine implements dual-layer fee structures with both vault-level and strategy-level management fees (time-based on assets under management) and performance fees (percentage of gains), while applying management fee seniority principles that prevent double-charging and ensure fair distribution between vault operators and individual strategists.

Through its emergency shutdown functionality, the library provides critical risk management capabilities that can halt all strategy operations during market crises or security incidents, forcing immediate position exits while maintaining proper authorization controls that distinguish between emergency activation (management, governance, or guardian) and deactivation (governance only).

The performance tracking system maintains comprehensive historical data including total gains, losses, and returns while calculating expected future performance based on time-weighted historical returns, enabling informed decision-making about strategy allocations and providing transparency into strategy effectiveness across different market conditions.

As the strategic coordination backbone of DynaVaults, it seamlessly integrates capital allocation optimization, risk management, fee collection, and performance monitoring while providing the operational infrastructure for multi-strategy investment management with comprehensive governance controls, emergency safeguards, and precise debt ratio management that ensures efficient capital deployment across diverse investment opportunities.

2.7.1 Issues & Recommendations

Issue #14	Strategy not revoked properly
Severity	LOW SEVERITY
Description	The `VaultStrategiesLib` allows 3 main actions for strategies: - adding new strategies - revoking strategies - migrating strategies The idea behind revoking strategies is to either migrate their current balances and accounting to a new strategy or to prevent it from receiving any new deposits (in the case of emergencies). The issue here is that `_revokeStrategy()` which is responsible for decommissioning a strategy only sets its `debtRatio` to 0. Since this is not deterministic enough to make sure a strategy cannot be used anymore, some unexpected behavior is possible: - A revoked strategy could be migrated multiple times through `migrateStrategy()`, the only difference being that after the first migration, the `totalDebt` for all subsequent strategies will be 0. This is possible since activation is still true and all other fields remain as they were before the revoke - Parameters can be still updated on a revoked strategy - fees, min/maximum harvest etc. It would be a good practice to create an explicit flag in the Strategy struct called `migrated` and use it to execute particular checks so that it is clear that the strategy has been migrated. Then in the `migrateStrategy()` and all update functions for the strategy, that flag can be checked to prevent any actions on already migrated resources. This reduces the attack surface and the probability of unexpected outcomes
Recommendation	Add an additional boolean field called `migrated` to the Strategy struct and prevent further migrations or updates in case it is true. An exception here is to still allow withdrawals so that all debt can be taken out in case of an emergency/migration.

Resolution	ACKNOWLEDGED The client stated that this will be resolved in the strategies framework audit.

Issue #15	**Insufficient validation**
Severity	INFORMATIONAL
Description	`addStrategy()` does not check if the token of the strategy is part of the approved tokens for the vault. Add an additional check to make sure only strategies for approved tokens can be added. Such validation could also be added to `migrateStrategy()` to ensure that once a token is removed, strategies cannot be migrated to it — There is an inconsistency with the dev comments of `revokeStrategy()`. The comment states: "may only be called by governance, the guardian, or the Strategy". However, the `onlyGovernanceOrStrategy()` validation has been used. Consider using the `onlyEmergencyAuthorized()` in case the guardian should also be able to call it or update the comments. — Within `creditAvailable()`, check early if strategy is not active and return 0 in that case.
Recommendation	Consider implementing the above recommendations.
Resolution	RESOLVED

Issue #16	Events are not emitted during important state changes
Severity	INFORMATIONAL
Description	The following function do not emit any events after updating storage variables: - `addStrategy()` - `removeStrategyFromList()` Emitting events is a good practice that allows anyone to transparently track the configuration and state changes in a contract, which is especially important in the case of DAOs.
Recommendation	Consider emitting events in the above state changing functions.
Resolution	RESOLVED

2.8 VaultConfigLib

`VaultConfigLib` serves as the central configuration registry and external dependency coordinator for DynaVaults, managing all critical contract addresses and operational parameters while providing governance-controlled parameter management for risk settings and upgrade mechanisms for external integrations.

It handles vault initialization by establishing connections to the vault manager for operational control, dyna router for token swapping capabilities, and reference asset oracle for pricing data, while automatically setting up necessary token approvals and storing essential metadata like deposit token decimals for accurate calculations throughout the system.

The library implements governance-controlled parameter management for risk settings such as maximum loss thresholds that protect users during withdrawals, defaulting to 1% loss protection with a maximum configurable limit of 10%, ensuring that forced strategy liquidations during withdrawals cannot exceed acceptable loss levels for users.

Its secure upgrade mechanisms enable the vault to update external dependencies like oracle contracts and routing systems while maintaining validation to ensure reference asset consistency, preventing configuration errors that could disrupt vault operations or compromise pricing accuracy.

Through its approval management system, the library automatically grants maximum token allowances to the vault manager when new reserve tokens are added, streamlining token operations while maintaining proper authorization controls that prevent unauthorized token movements.

The configuration system provides read-only access to all stored parameters through getter functions, enabling other vault components to access critical

information like manager addresses, token details, oracle configurations, and risk parameters without requiring direct storage access.

As the foundational configuration layer of DynaVaults, it ensures proper integration between vault components and external protocols while maintaining strict governance controls over critical parameters, enabling the vault to operate seamlessly with external dependencies and maintain appropriate risk parameters throughout its lifecycle.

2.8.1 Issues & Recommendations

Issue #17	Typographical issues
Severity	INFORMATIONAL
Description	`MAX_BPS` is not being used inside the library and can be removed. — The comment for `MAX_LOSS_LIMIT` is incorrect, it should be 20%. — L120: *set the dyna router addrress* should be *sets dyna router address*.
Recommendation	Consider fixing the typographical issues.
Resolution	RESOLVED

Issue #18	Events are not emitted during important state changes
Severity	INFORMATIONAL
Description	The following function do not emit any events after updating storage variables: - `setRouter()` - `setReferenceAssetOracle()` - `setMaxLoss()` Emitting events is a good practice that allows anyone to transparently track the configuration and state changes in a contract, which is especially important in the case of DAOs.
Recommendation	Consider emitting events in the above state changing functions.
Resolution	RESOLVED

2.9 DynaVaultManager

`DynaVaultManager` serves as the primary orchestration interface for all vault management operations within the DynaVaults ecosystem, inheriting strategy management capabilities and coordinating between multiple specialized library modules. It acts as the central command center that handles strategy reporting cycles where strategies communicate gains, losses, and debt payments, while managing the investment and liquidation flows that adjust capital allocation to individual strategies based on management decisions.

The contract implements a sophisticated locked profit mechanism that gradually releases profits over time to prevent manipulation, requires recent reporting before allowing fee updates to ensure accurate accounting, and provides comprehensive rebalancing functionality that can redistribute capital across reserve tokens according to target allocations.

Additionally, it features an advanced simulation system that allows previewing the effects of rebalancing and other operations before execution, handles deposit token flows in coordination with the main vault, and maintains real-time tracking of free funds and unlocked profit ratios. As the executive layer of the DynaVaults project, it bridges user interactions with the underlying vault infrastructure while ensuring proper sequencing of operations, maintaining accounting integrity, and providing both operational execution and analytical capabilities for informed decision-making.

2.9.1 Issues & Recommendations

Issue #19	Typographical issues
Severity	INFORMATIONAL
Description	The DynaVaultAPI interface not used and can be removed. — L52: *admount* should be *amount.* — L251: *vaules* should be *values.* — `takeStrategiesSnapshot` is missing NatSpec comments.
Recommendation	Consider fixing the typographical issues.
Resolution	RESOLVED

2.10 VaultManagerLib

`VaultManagerLib` serves as the central orchestration engine for all DynaVaults operations, implementing sophisticated profit locking mechanisms that gradually release gains over configurable time periods to prevent manipulation while managing complex fee calculations and distributions across multiple recipient types. It coordinates the critical strategy reporting cycle where individual strategies communicate their performance through gain/loss reports and debt payments, automatically calculating and distributing management fees (time-based on assets under management) and performance fees (percentage of gains) to vault management, strategists, and performance fee wallets through newly minted vault shares.

The library maintains detailed accounting for reserve tokens by continuously monitoring their value changes against reference asset prices and watermarks, calculating profits/losses and applying appropriate fees while updating the locked profit ratios to ensure fair distribution timing. Its rebalancing functionality actively monitors deviations between actual token holdings and target debt ratios, automatically triggering swaps between deposit tokens and reserve assets while coordinating with strategies to withdraw necessary liquidity when idle balances are insufficient.

The system implements strict reporting requirements that prevent fee updates unless all tokens and strategies have reported within recent time limits, handles deposit/withdrawal flows by updating debt accounting and reporting reserve changes, and manages the complex interplay between idle assets, strategy-deployed capital, and locked profits to ensure accurate vault valuations. As the operational heart of the DynaVaults project, it seamlessly integrates profit recognition, fee collection, capital allocation, and risk management while maintaining precise accounting across multiple tokens and strategies through continuous monitoring and automated rebalancing mechanisms.



2.10.1 Issues & Recommendations

Issue #20	Fee shares are not minted during rebalancing
Severity	MEDIUM SEVERITY
Description	`rebalance()` is called by governance to readjust asset distribution of tokens and strategies based on their `targetDebtRatios`. In the beginning, `_reportReserve()` is called for all tokens. The issue here is that the internal `_reportReserve()` only calculates `reserveFees` and returns them without minting any. As a result, every time `rebalance()` is called, the accumulated fee shares for all tokens in the vault will not be minted.
Recommendation	Call `reportReserve()` instead of `_reportReserve()` so that the vault is not deprived of its fees.
Resolution	RESOLVED

Issue #21	Investing in strategies is possible even on revoked strategies
Severity	LOW SEVERITY
Description	Strategies created for tokens in the Vault could be revoked, which as explained in the dev docs should "set their debt limit to 0 and prevent them from receiving any future deposits". The `investStrategy` function of `VaultManagerLib` allows governance and management to update a strategy debt ratio by defining the investment amount. It makes sense not to allow this action for strategies that have already been revoked since they were either migrated or became too risky.
Recommendation	The only way to check if a strategy has been revoked is to check its `debtRatio`. However, as explained in another issue for the Strategy contracts, this cannot always be a clear indicator if the strategy has been revoked or just had its `debtRatio` decreased. It is better to introduce a separate flag that explicitly shows that a strategy was revoked.
Resolution	PARTIALLY RESOLVED The recommendation was not implemented for `investStrategy()`.

Issue #22	`reportReserve` does not check if token has been removed
Severity	LOW SEVERITY
Description	`reportReserve()` can be called by the governance to report the profit/loss for a particular token. However, there is no check to prevent calling it with a removed token. Since removed tokens have their values zeroed out and getting the token stats will not revert, this might lead to unexpected outcomes. The same recommendation applies to the `reportReserves()` function. In the case of `reportAllReservesFromVault()`, the validation is implicit since the tokens are not provided as input but rather fetched from the `VaultTokensLib.allTokens()` array which contains only valid tokens.
Recommendation	Call `tokenExists()` at the top of the function to make sure that the provided token address has not been removed and is approved.
Resolution	RESOLVED

Issue #23	**`investStrategy` does not report reserve before updating strategy ratios**
Severity	LOW SEVERITY
Description	The `investStrategy` function does not call `reportReserve()` before increasing the strategy debt ratio. In this case it is necessary, since `increaseStrategyDebtRatioByInvestmentAmount()` uses `tokenIdle` which gets updated to the current balance in `reportReserve()` and also mints the fee shares, adjusting the current `totalSupply`.
Recommendation	Call `reportReserve()` at the top of the function to make sure that vault state has been properly updated.
Resolution	ACKNOWLEDGED The client commented that this will be fixed in the strategy framework audit.

2.11 DynaVault

`DynaVault` serves as the primary user-facing interface and implementation of the ERC4626 tokenized vault standard within the DynaVaults ecosystem, extending ERC20 functionality to represent vault shares while coordinating all deposit, withdrawal, and investment operations through its specialized library modules.

It implements comprehensive deposit and withdrawal mechanisms that automatically trigger reserve token reporting before operations to ensure accurate pricing, while applying configurable deposit and redemption fees through a sophisticated fee calculation system that mints new shares to designated fee recipient wallets rather than transferring underlying tokens.

The contract features advanced slippage protection through ERC5143 implementation, allowing users to specify minimum expected shares on deposits or maximum tolerable fees on withdrawals, while also providing unique proportional redemption functionality that allows users to exit with a balanced portfolio of all vault's underlying assets rather than just the primary deposit token.

Its integrated swap management system enables vault management to rebalance portfolios and adjust target allocations through direct token swaps, with built-in preview functionality that queries multiple DEX routers to find optimal swap routes and pricing before execution, while maintaining proper debt ratio accounting across all operations.

The vault implements sophisticated simulation capabilities that allow previewing the effects of deposits, withdrawals, and rebalancing operations before execution, helping users and managers make informed decisions while the locked profit mechanism gradually releases performance gains over time to prevent manipulation and ensure fair distribution.

As the central hub of the DynaVaults project, it seamlessly coordinates between fee collection, asset management, strategy coordination, and user interactions while maintaining ERC4626 compliance and providing enhanced functionality for multi-token portfolio management with configurable risk parameters and comprehensive slippage protection mechanisms.

2.11.1 Issues & Recommendations

Issue #24	Reserves are not reported during minting
Severity	HIGH SEVERITY
Description	Before shares are minted or burned inside the vault, it is essential to call `reportAllReserves()` which does the following: - Calculates profit/loss generated by each vault token - Calculates fees and mints shares for them (increasing `totalSupply`) - Reports idleTokens and updates the watermark Calling this function before each deposit, withdraw operation is essential in order to make sure that the shares/assets calculations for the caller are executed based on the latest and actual state of the Vault. The issue here is that `mint()` (in contrast to deposit, redeem, withdraw) omits calling `reportAllReserves()`, which means that all share calculations will be inaccurate.
Recommendation	Add the `DynaVaultLib.reportAllReserves()` call at the top of the `mint()` function.
Resolution	RESOLVED

Issue #25	Lack of minimum deposit assets check in proportional redeem
Severity	LOW SEVERITY

Description	The standard `redeem()` flow contains a check that makes sure that after an account burns their shares, the assets that are left in the Vault are not below the configured minimum. More precisely, it calls the `checkRedeem()` function:
	
	—
	`function checkRedeem(uint256 shares, address owner) external view {`
	`  uint256 maxShares = maxRedeem(owner);`
	`  require(shares <= maxShares, "ERR_MAX_REDEEM");`
	`  uint256 remainingShares = maxShares - shares;`
	
	`  if (remainingShares != 0) {`
	`  require(_convertToAssetsGivenTotalSupply(`
	`  remainingShares, totalSupply(),`
	`  Math.Rounding.Down) >=`
	`  vaultStorage().minDepositAssets,`
	`  "ERR_MIN_REDEEM");`
	`  }`
	`}`
	
	However, the same check has not been applied in `redeemProportional()`, meaning it can be used to bypass the minimum amount requirement.

Recommendation	Add the following check inside `redeemProportional()` right after the `DynaVaultLib.reportAllReserves()` call: `DynaVaultLib.checkRedeem(sharesIncludingFees, owner);`
Resolution	RESOLVED

Issue #26	Vault is susceptible to the first depositor attack
Severity	LOW SEVERITY

Description	The first depositor attack or the so-called donation attack is a common vulnerability of ERC4626 vaults.
	
	In summary, it is an attack vector where the first depositor in a vault mints shares while there are none yet and then directly transfers (donates) a lot of vault tokens right before the next depositor, inflating the price so that the victims deposit is consumed without any shares getting minted.
	
	The attack has been described in detail by OpenZeppelin - https://docs.openzeppelin.com/contracts/5.x/erc4626#defending_with_a_virtual_offset
	
	Here is a short example of how this is possible in DynaVault:
	
	- The Vault has just been created and Bob deposits assets to mint 1 share
	- A bit of time passes and Alice also decides to invest assets to mint, let's say 2 shares
	- Bob front-runs her deposit by doing the following:
	  - He transfers a lot of tokens to the vault
	  - Supposing no minimum deposit has been set (the default), Bob calls `deposit` again with dust amount so that he can invoke `reportAllReserves()` which updates `tokenIdle` based on the current balance (which Bob inflated) so that `totalAssets` would be a very big number
	  - Alice's deposit call is executed but since the share price has been inflated significantly, she receives 0 shares and Bob withdraws everything
	
	It should be noted that the Dyna team already has some measures in place, such as configuring minimum amounts and also implementing a slippage check function, which should guard depositors from such exploits. However, under specific conditions, the exploit is possible.

Recommendation	OZ already has a battle tested implementation of the ERC 4626 standard, so it is always recommended to use it as a base and add the additional logic on top of it. This might be too resource intensive given that the Dyna team has already implemented their own ERC4626 vault. Another option here is to recommend to their users to use the slippage functions and also to mint some initial amount of shares to a dead address, which would make Bob's exploit too expensive to conduct.
Resolution	RESOLVED The team implemented ERC5143 which blocks any frontrunning attempts.

Issue #27	Insufficient validation
Severity	INFORMATIONAL
Description	Inside `initialize()`, validate that `ownerAddress` is not `address(0)`. — Make sure to add a top-level validation that the receiver and owner parameters provided to `deposit()`, `mint()`, `withdraw()`, `redeem()`, `redeemProportional()` are not `address(0)`. — Within `withdraw()`, check that `sharesToBurn` is not 0 before executing the rest of the flow. — Within `redeem()`, check that `sharesIncludingFees` is not 0.
Recommendation	Consider implementing the above recommendations.
Resolution	RESOLVED

2.12 DynaVaultLib

`DynaVaultLib` serves as the core implementation engine for ERC4626 and ERC5143 vault standards within the DynaVaults ecosystem, handling all fundamental deposit and withdrawal mechanics while managing complex share-to-asset conversions that account for locked profits and fee structures.

It implements sophisticated fee calculation mechanisms that distinguish between deposit fees (calculated on total incoming assets) and redemption fees (calculated on raw asset amounts), automatically routing fee payments to designated recipient wallets while ensuring proper accounting through iterative approximation algorithms that prevent share dilution during fee collection.

The library manages complex withdrawal scenarios where insufficient idle balances trigger automated strategy liquidation sequences, systematically withdrawing from multiple strategies in queue order while calculating and reporting any losses incurred during forced withdrawals, with built-in loss protection that reverts transactions exceeding configurable maximum loss thresholds.

Its advanced proportional redemption functionality allows users to exit with balanced portfolios of all vault tokens rather than just the primary deposit asset, automatically calculating proportional shares based on total token holdings and triggering strategy withdrawals when idle balances are insufficient, while maintaining accurate debt accounting across all operations.

The system implements dynamic asset-to-share conversion mechanisms that factor in unlocked fund ratios from the locked profit system, ensuring accurate valuations during periods when profits are being gradually released, while handling edge cases like zero total supply scenarios through fallback calculations based on deposit token precision.

During withdrawal operations, the library orchestrates complex multi-token swaps when users exit with deposit tokens but the vault holds diverse reserve assets, automatically querying optimal swap routes through integrated DEX routers and updating debt ratios to maintain accurate portfolio allocation tracking while protecting users from excessive slippage through configurable loss limits.

As the foundational calculation and execution layer of the DynaVaults project, it seamlessly integrates fee collection, share accounting, strategy coordination, and asset management while ensuring ERC4626 compliance and providing enhanced functionality for multi-token portfolio operations with comprehensive loss protection and sophisticated withdrawal optimization mechanisms.

2.11.1 Issues & Recommendations

Issue #28	Redemption fees are not always deducted
Severity	MEDIUM SEVERITY

Description	The protocol defines a redemption fee which should be deducted every time a user burns shares to withdraw the underlying assets. After all the calculations have been executed and reserve tokens swapped (if necessary), the fee percent is send to the `feeRecepient` at the bottom of `beforeBurn()`:
	
	L624-629
	`address feeRecipient = _manager.getFees().redemptionFeeWallet;`
	
	`if (fee > 0 && feeRecipient != address(this)) { SafeERC20.safeTransfer(IERC20(asset()), feeRecipient, fee); assetsToWithdraw = assetsToWithdraw > fee ? assetsToWithdraw - fee : 0; }`
	
	There is an `if` check that in case the defined `feeRecepient` is the Vault itself, the fees do not get transferred but instead remain in the contract.
	
	This is where the issue lies. The `assetsToWithdraw` variable holds `assetsNotIncludingFees + fee` and if there is a transfer, the `assetsToWithdraw` is deducted by the fee amount sent. However if the `feeRecipient` is the Vault, the `if` check does not get triggered and `assetsToWithdraw` does not get deducted with the fee that should remain in the Vault. As a result, the fee is not withheld and the full `assetsToWithdraw` is then sent back to the shares owner.

Recommendation	Move the fee deduction line outside the if-block so that fees are always taken: `if (fee > 0 && feeRecipient != address(this)) { SafeERC20.safeTransfer(IERC20(asset()), feeRecipient, fee); } assetsToWithdraw = assetsToWithdraw > fee ? assetsToWithdraw - fee : 0;`

Resolution	RESOLVED

Issue #29	Transferred redemption fees are not deducted from tokenIdle
Severity	MEDIUM SEVERITY

Description	`_withdraw()` in the DynaVault contract is executed when `withdraw` and `redeem` functions are invoked. Internally, `_withdraw()` calls the following library function in that order:
	1. `DynaVaultLib.beforeBurn()`
	2. `DynaVaultLib.afterBurn()`
	In `beforeBurn()`, the fees are deducted from the assets to sent and transferred to the `feeRecipient`:
	L624:629
	`if (fee > 0 && feeRecipient != address(this)) { SafeERC20.safeTransfer(IERC20(asset()), feeRecipient, fee); }`
	`beforeBurn()` returns `assetsToWithdraw` (the amount minus the fees) and `afterBurn()` uses it to send it to the recipient and decrease the `tokenIdle` by that amount:
	`manager().withdrawDepositToken(assetsNotIncludingFees); IERC20(asset()).safeTransfer(receiver, assetsNotIncludingFees);`
	The issue with this flow is that the fees that have been sent in the `beforeBurn()` are not accounted for in `tokenIdle`, or `manager().withdrawDepositToken()` does not get called after the fee is sent to the recipient, which should be done since the balances are changed.

Recommendation	Make sure to decrease `tokenIdle` upon fee transfer (call `manager().withdrawDepositToken()` at the end of `beforeBurn()`) just as it is done upon assets transfer to the share owner.
Resolution	RESOLVED

Issue #30	CEI pattern not followed inside `transferProportional`
Severity	LOW SEVERITY

Description	`transferProportional()` does not adhere to the CEI pattern when it executes the external call to transfer asset:
	
	L392:396
	`IERC20(token).safeTransfer(receiver, toRedeem[i] - fee);`
	`if (fee != 0 && feeRecipient != address(this)) {`
	`    (token).safeTransfer(feeRecipient, fee);`
	`}`
	`_manager.withdrawIdle(token, toRedeem[i]);`
	
	The issue here is that the `safeTransfer` calls are executed before the state updates to the `tokenIdle` value of the withdrawn token (`withdrawIdle()`).
	
	This issue is marked as low severity since the external function that calls it already has a re-entrancy guard in place. However this does not prevent read-only re-entrances and also it might be a problem in case the library function gets used in another function without the proper guards.

Recommendation	Make sure to first call `_manager.withdrawIdle(token, toRedeem[i]);` and only then execute the token transfers.
Resolution	RESOLVED

Issue #31	**Not all losses are considered when checking for the `maxLoss` threshold before burn**
Severity	LOW SEVERITY

Description	`beforeBurn()` executes a check after all swaps to the base token have been executed to make sure that the accumulated loss from strategy withdrawals are not above the accepted threshold. This is how the check looks:
	`if (totalLoss > FixedPointMathLib.fullMulDiv(maxLoss, depositTokensToWithdraw, MAX_BPS)) {`
	`  revert LossProtection(depositTokensToWithdraw, totalLoss, maxLoss);`
	`}`
	
	`totalLoss` is a local accumulator variable that gets incremented with the loss of each strategy inside the loop for every token.
	
	However, the variable does not consider another possible source of losses — this is the swaps executed from reserve to base tokens. As result the actual loss might be bigger than accounted for in `totalLoss` and thus the validation might not be entirely accurate.

Recommendation	Consider adding the potential losses from swaps as well to the `totalLoss` accumulator, to ensure a more precise validation.
	
	The swap loss can be calculated by subtracting `depositTokensAllocatedForWithdraw` (accumulates all `tokenOuts`) from the original withdraw amount `depositTokensToWithdraw`:
	
	`uint256 swapLoss = depositTokensAllocatedForWithdraw - depositTokensAllocatedForWithdraw`
	`maxLoss += swapLoss`
	
	... execute validation

Resolution	PARTIALLY RESOLVED

Issue #32	Gas optimizations
Severity	INFORMATIONAL

Description	Inside `calcRedeemProportional`, consider making the returned `toRedeem` array to store structs with the following form:
	
	`struct Redeem { amount: uint256, token: address }`
	
	After that, the array can be passed to `transferProportional()` and directly get the token addresses from within the array instead of having to execute the external calls to the manager contract all over again. If the number of tokens is bigger, this can save a reasonable amount of gas.
	
	—
	
	Inside `calcSharesForFeeAmountUsingGivenTotalSupplyAndTotalAssets()` calculate `deltaTotalAssetsInDeposit` like the below to save some gas if `deltaTotalAssets` is not 0:
	
	`uint256 deltaTotalAssetsInDeposit = (feeToken == depositToken) ? deltaTotalAssets : tokenValueInQuoteAsset(feeToken, deltaTotalAssets, depositToken);`
	
	—
	
	Inside `beforeBurn()`, first check if the `toRedeem` value is 0 and skip the gas intensive swap operation in the code block that executes the swap.
	
	—
	
	Inside `beforeBurn()`, the check can be moved to inside each loop of the token instead of validating the `maxLoss` at the end so that in case the loss is exceeded earlier, all further and gas intensive loops would be prevented:

	L571:574
	`if (tokenLoss != 0) {`
	`[...]`
	`totalLoss += (t > 0 && tokenLoss > 0) ? tokenValueInQuoteAsset(tokenAddress, tokenLoss, _asset) : tokenLoss;`
	`if (totalLoss > FixedPointMathLib.fullMulDiv(maxLoss, depositTokensToWithdraw, MAX_BPS)) { revert …}`
	`}`

Recommendation	Consider implementing the above recommendations.
Resolution	RESOLVED

Issue #33	**Insufficient validation**
Severity	INFORMATIONAL
Description	Inside `transferProportional()`, add a check to make sure that the length of `toRedeem` array is equal to `nrOfTokens`.
Recommendation	Consider implementing the above recommendations
Resolution	RESOLVED

Issue #34	Events are not emitted during important state changes
Severity	INFORMATIONAL
Description	The following function do not emit any events after updating storage variables: - `setDepositLimits()` Emitting events is a good practice that allows anyone to transparently track the configuration and state changes in a contract, which is especially important in the case of DAOs.
Recommendation	Consider emitting events in the above state changing functions
Resolution	RESOLVED

2.13 VaultSimulatorLib

`VaultSimulatorLib` serves as the comprehensive simulation engine for all vault operations within the DynaVaults ecosystem, providing accurate previews of deposits, withdrawals, and redemptions without requiring actual execution or gas costs by creating detailed snapshots of vault state and simulating the complete reporting cycle that would occur before real operations.

It implements sophisticated state capture mechanisms that record all critical vault parameters including total assets, total supply, locked profit ratios, token balances, debt positions, and strategy snapshots, enabling realistic simulation of complex operations by accounting for the reporting updates that would normally occur before user transactions.

The library provides precise share-to-asset conversion calculations that factor in simulated reporting results and unlocked fund ratios, ensuring users receive accurate previews of their transactions even when significant time has passed since the last reporting cycle, while handling fee calculations for both deposit and redemption scenarios through the same algorithms used in actual operations.

Its advanced proportional redemption simulation allows users to preview multi-token withdrawals by calculating exact token amounts they would receive based on current portfolio allocations and unlocked fund ratios, while accounting for redemption fees and potential strategy withdrawals that might be triggered during actual execution.

The system seamlessly integrates with the vault manager's simulation capabilities to provide comprehensive previews that include the effects of reserve token reporting, locked profit degradation, and fee share minting, ensuring users can make informed decisions about their transactions before committing gas costs.

By maintaining separate simulation pathways that mirror the exact logic of actual vault operations while operating on cached state snapshots, the library enables sophisticated preview functionality for complex multi-step operations like rebalancing and strategy reporting without affecting the actual vault state.

As the analytical layer of the DynaVaults project, it empowers users and management with accurate transaction previews, realistic slippage estimates, and detailed insights into vault operations while supporting advanced features like proportional redemptions and multi-token portfolio management through comprehensive simulation capabilities that account for all fees, locked profits, and reporting cycles.

2.13.1 Issues & Recommendations

Issue #35	Typographical issues
Severity	INFORMATIONAL
Description	The dev comment above `simulatedDeposit()` says "Simulates a withdrawal with fees" when it should be "Simulates a deposit with fees". Also, the additional tags to it should replace *withdraw* with *deposit*.
Recommendation	Consider implementing the above recommendations
Resolution	RESOLVED

2.14 VaultManagerSimulatorLib

`VaultManagerSimulatorLib` serves as the comprehensive simulation counterpart to VaultManagerLib within the DynaVaults ecosystem, providing accurate previews of all complex vault management operations including reserve token reporting, strategy performance tracking, fee calculations, and rebalancing scenarios without requiring actual execution or state changes.

It implements sophisticated locked profit simulation mechanisms that accurately model profit degradation over time using configurable degradation coefficients and time-based calculations, enabling precise previews of how locked profits will affect vault valuations and free fund availability during future operations.

The library provides detailed reserve token reporting simulation that calculates profit and loss scenarios based on current token balances versus historical watermarks and reference asset values, while computing accurate management and performance fee projections using the same algorithms employed in actual operations but operating on snapshot data rather than live state.

Its advanced strategy reporting simulation capabilities allow previewing the complete reporting cycle including gain registration, fee calculations across multiple recipient types (vault management, strategists, performance fees), and locked profit ratio updates that would occur during actual strategy harvest operations.

The system implements a comprehensive rebalancing simulation that analyzes current token allocations versus target debt ratios and calculates exact swap amounts and expected outputs through integrated DEX router previews, enabling management to assess rebalancing scenarios and slippage expectations before committing to actual operations.

Through integration with the vault's fee calculation systems, it accurately simulates fee share minting scenarios that account for dilution effects and properly model

how fees would impact total supply and share valuations during actual operations, while maintaining consistency with the iterative approximation algorithms used in production.

As the analytical simulation layer of the DynaVaults project, it empowers users and management with comprehensive previews of complex multi-step operations involving multiple tokens, strategies, and fee calculations, enabling informed decision-making about vault operations while providing realistic estimates of outcomes including profit unlocking, fee impacts, and rebalancing requirements through sophisticated modeling that mirrors actual vault mechanics without gas costs or state modifications.

2.14.1 Issues & Recommendations

Issue #36	The manager report reserve simulation does not properly simulate the actual function
Severity	LOW SEVERITY

Description	The simulator library contains functions that simulate real state-changing functions, that allow the caller to see how the state would be updated if the real function would be executed. This is why it is important that the simulation precisely mimics the operations in the real one.
	
	There is currently a discrepancy between `VaultManagerLib._reportProfitLoss()` and `VaultManagerSimulatorLib._simulatedReportProfitLoss()`.
	
	The discrepancy lies in the fact that the state-changing `VaultManagerLib._reportProfitLoss()` calls `VaultTokensLib.reportProfitLossBalance()` at the end which updates `tokenIdle`, `depositDebt` and `reserveDebt` of the token, where `tokenIdle` is set to the current balance of the vault.
	
	The simulator function on the other hand (`VaultManagerSimulatorLib._simulatedReportProfitLoss()`) skips the updates of the above fields since it does not modify state
	
	The function that uses `VaultManagerSimulatorLib._simulatedReportProfitLoss()` is `VaultManagerSimulatorLib.simulateTotalAssetsForToken()`:
	
	L226:230
	`function simulateTotalAssetsForToken(address token, VaultSimulatorAPI.VaultSnapshot memory snapshot) external view returns (uint256) {`
	
	`uint256 tokenIndex = VaultTokensLib.tokenIndex(token);`
	
	`(, snapshot) = _simulatedReportReserve(tokenIndex, snapshot);`
	
	`return snapshot.tokens[tokenIndex].tokenIdle + snapshot.tokens[tokenIndex].tokenDebt;`
	`}`

	The function returns `snapshot.tokenIdle` which however was not updated to the actual balance of the vault. The value of the snapshot can be a bit stale, since the initial snapshot is created through `VaultSimulatorLib.snapshot()` where it can be seen that `token.idle` is loaded from `tokenStats[token]`:
	
	`VaultSimulatorLib::L80:81`
	`tokens[i].tokenIdle = stats.tokenIdle;`
	
	This discrepancy might lead to potential differences between the simulation and the real result. An important consideration here is that the `simulateTotalAssetsForToken()` is used by the state-changing function `VaultManagerLib.liquidateStrategy()`.

Recommendation	Consider mimicking the logic of the original function precisely to make sure the result is as close as possible. Inside `VaultManagerSimulatorLib._simulatedReportProfitLoss()`, `tokenIdle`, `depositDebt` and `reserveDebt` can be updated in the snapshot struct to simulate storage writes.
Resolution	RESOLVED

Issue #37	Inaccurate rebalance simulation
Severity	LOW SEVERITY

Description	There are 2 inconsistencies inside `simulatedRebalance()`:
	
	The first one stems from calling `simulatedReportAllReserves()`, which as explained in a previous issue, does not update `tokenIdle`, `depositDebt` and `reserveDebt` as in the original function.
	
	`depositDebt` and `reserveDebt` get used right after the `simulatedReportAllReserves()` call, which as pointed out were not updated. Here is where they get used:
	
	L318:321
	`uint256 totalDepositDebt = VaultTokensLib.totalDepositDebt();`
	
	`uint256 targetDepositDebtRatio = VaultTokensLib.targetDepositDebtRatio(tokens[0]);`
	
	`uint256 targetDepositDebt = FixedPointMathLib.fullMulDiv(totalDepositDebt, targetDepositDebtRatio, MAX_TOKEN_DEBT_RATIO);`
	
	`uint256 depositDebt = VaultTokensLib.depositDebt(tokens[0]);`
	
	If we expand `VaultTokensLib.totalDepositDebt()`, it can be seen that it uses `reserveDebt`:
	
	`function totalDepositDebt() external view returns (uint256) { [...] _storage.tokenStats[depositToken].tokenIdle + _storage.tokenStats[depositToken].tokenDebt + _storage.reserveDebt; }`
	
	After `VaultTokensLib.totalDepositDebt()` at Line 321, `VaultTokensLib.depositDebt(tokens[0])` is called to get the `depositDebt` for the token .

	The second important issue to point out here is that even if the `simulatedReportAllReserves()` was accurate, `simulatedRebalance()` would still be incorrect, since the variables between L318-321 (`totalDepositDebt`, `targetDepositDebtRatio`, `depositDebt`) are all taken from storage when it should be taken from the snapshot, since the simulation function applies the changes there and not in actual storage.
Recommendation	In addition to updating the `_simulatedReportReserve()` implementation, also make sure not to use storage variables during simulation, since all the changes are applied in the snapshot and not in actual storage. This would prevent discrepancies in the calculations.
Resolution	RESOLVED

Issue #38	Inaccurate report simulation
Severity	LOW SEVERITY
Description	`simulateReport()` is not implemented properly because it deviates from the original version of the function in the following ways: - With the original `VaultManagerLib.report()` L359, it calls the internal version of `_reportReserve()` which reports the reserve only for one token, but more importantly it calculates profit/loss, without minting the fee shares and updating the `totalSupply`. - With the simulation function at `VaultManagerSimulatorLib.simulatedReport()` L422, it calls `simulatedReportAllReserves()` which simulates reporting for all vault tokens and it also simulates minting of the fee shares, thus changing `totalSupply`. - Another issue is that fee share minting is simulated 2 times—once in `simulatedReportAllReserves()` and then once again by calling `_simulatedIssueStrategyFeeShares()` at L440. As a result, `totalSupply` is increased 2 times. - Another difference is that after L444 of `simulatedReport()`, the `increase/decreaseDebt` operations from the original have been omitted, meaning that the snapshot will not have properly updated token debts.
Recommendation	Instead of `simulatedReportAllReserves()`, use `_simulatedReportReserve()` to closely match the logic of the state-changing `report()` function.
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY