SALUS SECURITY

JUL 2025

CODE SECURITY ASSESSMENT

DEDERI

Overview

Project Summary

- Name: Dederi - Incremental Audit
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
    - https://github.com/Dederi-Finance/dederi-contracts-v2
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Dederi - Incremental Audit
Version	v3
Type	Solidity
Dates	Jul 07 2025
Logs	Jul 01 2025, Jul 03 2025, Jul 07 2025

Vulnerability Summary

Total High-Severity issues	1
Total Medium-Severity issues	5
Total Low-Severity issues	2
Total informational issues	3
Total	11

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction **4**

1.1 About SALUS 4

1.2 Audit Breakdown 4

1.3 Disclaimer 4

Findings **5**

2.1 Summary of Findings 5

2.2 Notable Findings 6

1. Borrow interest can be manipulated 6
2. Slippage protection failure in the liquidate function 7
3. Approval is not reset after the swap. 8
4. Missing updateVaultLend after some balance changes 9
5. Improper rounding direction in withdrawWithUSDC 10
6. Centralization risk 11
7. Missing _validateBeforeBorrow check in the borrow function 12
8. Incorrect approval amount may cause borrow to fail 13

2.3 Informational Findings 14

9. The runADLPartial does not consume traceId 14
10. Some upgradeable contracts are not initialized 15
11. Approval is not reset after removing the module 16

Appendix **17**

Appendix 1 - Files in Scope 17

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Borrow interest can be manipulated	High	Business Logic	Resolved
2	Slippage protection failure in liquidate function	Medium	Business Logic	Resolved
3	Approval is not reset after the swap	Medium	Business Logic	Resolved
4	Missing updateVaultLend after some balance changes	Medium	Business Logic	Resolved
5	Improper rounding direction in withdrawWithUSDC	Medium	Business Logic	Resolved
6	Centralization risk	Medium	Centralization	Acknowledged
7	Missing _validateBeforeBorrow check in the borrow function	Low	Business Logic	Resolved
8	Incorrect approval amount may cause borrow to fail	Low	Business Logic	Resolved
9	The runADLPartial does not consume traceId	Informational	Business Logic	Resolved
10	Some upgradable contracts are not initialized	Informational	Business Logic	Resolved
11	Approval is not reset after removing the module	Informational	Business Logic	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Borrow interest can be manipulated	
Severity: High	Category: Business Logic
Target: - contracts/lendPool/lib/LibUser.sol	

Description

When accruing interest, the system calculates the current variable borrow rate, which is then used to determine the accrued interest for the previous time slot.
However, this design has a vulnerability: the variable borrow rate can be manipulated. A malicious user could perform a flash loan to temporarily inflate the total borrowed amount in AAVE, thereby artificially increasing the borrow rate just before interest is accrued. As a result, all borrowers may end up paying more interest than expected, even though the manipulation only occurred briefly.
contracts/lendPool/lib/LibUser.sol: L154-L174

```
function settleUserLendInterest(uint256 lastInterestUpdateTime) internal returns
(uint256 userInterest) {
    userInterest = calcUserInterest(lastInterestUpdateTime);
}
function calcUserInterest(uint256 lastInterestUpdateTime) internal view returns
(uint256) {
    uint256 _borrowBorrow = Math.min(LibLP._getLendBorrow(), _getUserHolding());
    return _borrowBorrow * (SystemLib._blockTimestamp() - lastInterestUpdateTime) *
_calculateLendPoolInterestRate()
        / Constant.LOW_DECIMALS / Constant.YEAR_SECONDS;
}
```

contracts/lendPool/lib/LibUser.sol: L182-L204

```
function _calculateLendPoolInterestRate() internal view returns (uint256) {
    uint256 _missingRate = LibLP._getMissingRate();
    uint256 _aaveUSDCBorrowInterestRate =
IAAVEV3Module(ContractAddress.AAVE_V3_MODULE).getVariableBorrowRate(ContractAddress.USDC
) / 1e22;
}
```

Recommendation

Revisit the borrow interest rate model.

Status

This issue has been resolved by the team with commit 96d20a7.

2. Slippage protection failure in the liquidate function

Severity: Medium	Category: Business Logic
Target: - contracts/core/StrategyManager/PortfolioMargin/lib/LibLendPool.sol	

Description

The `liquidate` function allows users to liquidate unhealthy strategies using USDC. The `maxUSDCAmount` parameter is intended to act as a slippage control, capping the maximum amount of USDC that can be spent to acquire the specified `wantAssets`.

However, when the price of a `wantAsset` increases and the provided USDC is insufficient, the function does not revert. Instead, it silently adjusts `wantAssets[i].units` downward to match the available USDC. As a result, users may unknowingly complete the liquidation at a less favorable price, receiving fewer `wantAssets` than expected, and effectively bypassing the intended slippage protection.

contracts/core/StrategyManager/PortfolioMargin/lib/LibLendPool.sol: L71-L81

```
function liquidate(uint256 strategyId, Asset[] memory wantAssets, uint256 maxUSDCAmount)
internal {
    ...
    uint256 _residualValue = maxUSDCAmount;
    for (uint256 i = 0; i < wantAssets.length; i++) {
        address _token = CashAssetEncoder.getUnderlying(wantAssets[i].assetId);
        uint256 _price = IOracle(ContractAddress.ORACLE).indexPrice(_token) *
Constant.LOW_DECIMALS / (_rewardRate);
        uint256 _payUSDCAmount = wantAssets[i].units.toUint256() * _price /
Constant.HIGH_DECIMALS;
        if (_payUSDCAmount > _residualValue) {
            wantAssets[i].units = (_residualValue * Constant.HIGH_DECIMALS /
_price).toInt256();
            _residualValue = 0;
        } else {
            _residualValue -= _payUSDCAmount;
        }
    }
    ...
}
```

Recommendation

It is recommended to revert when `_residualValue` is insufficient, instead of adjusting `wantAssets`.

Status

This issue has been resolved by the team with commit 96d20a7.

3. Approval is not reset after the swap.

Severity: Medium	Category: Business Logic

Target:

- contracts/core/StrategyManager/dual/lib/LibDM.sol

Description

During DM settlement, if an option results in a loss, the underlying token must be swapped for USDC to cover the deficit. Prior to the swap, the entire balance of the underlying token is approved to the `` `swapModule` ``, even though only a portion (`` `amountIn` ``) may be used in the actual swap.

However, the remaining approval is not revoked after the swap, and since the `` `swapTokenForExactUSDC` `` function lacks access control, a malicious actor could exploit the leftover approval to steal the remaining tokens.

contracts/core/StrategyManager/dual/lib/LibDM.sol: L387-L412

```
function settle(uint256 _strategyId, bytes memory swapPath) internal {
    ...
    if (optionType == 0 && option.units < 0) {
        ...

        if (_usdcAmount > 0) {
            ...
        } else if (_usdcAmount < 0) {
            ...
            uint256 underlyingAmount = StrategyLib.getSpecifyCashAmount(_strategy,
underlying).toUint256();
            uint256 underlyingAmountWithNativeDecimals =
underlyingAmount.from18Decimals(underlying);
            IVault(ContractAddress.VAULT).approveTokenForModule(
                ContractAddress.SWAP_MODULE, underlying,
underlyingAmountWithNativeDecimals
            );
            (bool success,, uint256 amountIn) =
ISwapModule(ContractAddress.SWAP_MODULE).swapTokenForExactUSDC(
                underlyingAmountWithNativeDecimals, _absUSDCAmountWithNativeDecimals,
swapPath, 0, underlying
            );
            ...
        }
    }
    ...
}
```

Recommendation

It is recommended to clear the approval after the swap.

Status

This issue has been resolved by the team with commit 96d20a7.

4. Missing updateVaultLend after some balance changes

Severity: Medium	Category: Business Logic
Target: - contracts/vault/Vault.sol	

Description

The functions `increaseCash` and `adjustTotalDeposit` modify `dederibalanceUSDC`, which is a critical component for determining the system's `lendBorrow` and `missingRate`.

However, these functions do not call `updateVaultLend` afterward, leaving both `lendBorrow` and `missingRate` in an outdated state. This can lead to several issues:

- Stale `missingRate` may cause interest rates to be outdated or inaccurately calculated.
- Incorrect `lendBorrow` data could mislead the borrowing and repayment logic for Aave, potentially resulting in inefficient or incorrect fund management.

contracts/vault/Vault.sol: L185-L192

```
function increaseCash(address user, address token, uint256 amountWith18Decimals)
    external
    whenNotPaused
    onlyRole(STRATEGY_MANAGER_ROLE)
{
    _addUserBalance(user, token, amountWith18Decimals);
    // _updateLendBalance();
}
```

contracts/vault/Vault.sol: L338-L345

```
function adjustTotalDeposit(address token, int256 amountWith18Decimals) public
onlyRole(STRATEGY_MANAGER_ROLE) {
    VaultStorage storage $ = _getVaultStorage();
    if (amountWith18Decimals > 0) {
        $.totalDeposit[token] += amountWith18Decimals.toUint256();
    } else {
        $.totalDeposit[token] -= (-amountWith18Decimals).toUint256();
    }
}
```

Recommendation

It is recommended to call `updateVaultLend` after these balance changes to recalculate `lendBorrow` and the `missingRate`

Status

This issue has been resolved by the team with commit 96d20a7.

5. Improper rounding direction in withdrawWithUSDC

Severity: Medium	Category: Business Logic

Target:
- contracts/lendPool/lib/LibUser.sol

Description

The `withdrawWithUSDC` function facilitates exiting the LendPool by burning shares and returning USDC to the vault. However, when converting the asset amount to the corresponding number of shares to burn, the calculation rounds down.

This rounding behavior is unfavorable to the protocol. In cases where `_getShareRate()` is large, small withdrawal amounts may result in zero shares being burned, effectively allowing users to withdraw USDC without giving up any shares — a form of free withdrawal that can lead to value leakage over time.

contracts/lendPool/lib/LibUser.sol: L64-L66

```
function withdrawWithUSDC(uint256 amount) internal {
    LibLP._settleInterest();
    _withdraw(amount * Constant.HIGH_DECIMALS / _getShareRate(), amount);
}
```

Recommendation

It is recommended to round up when calculating the number of shares to be burned.

Status

This issue has been resolved by the team with commit 96d20a7.

6. Centralization risk

Severity: Medium	Category: Centralization
Target: - contracts/vault/Vault.sol	

Description

In dederi contracts, there exists some privileged roles, e.g. `APPROVER_ROLE`, `DEFAULT_ADMIN_ROLE`. These roles have the authority to execute some key functions such as `approveTokenForModule`, `addTrustedModule` and `addGuardians`, etc.

If these roles' private keys are compromised, an attacker could trigger these functions to steal all funds.

contracts/vault/Vault.sol: L401-L407

```
function approveTokenForModule(address module, address token, uint256 amount) external
onlyRole(APPROVER_ROLE) {
    _approveTokenForModule(module, token, amount);
}

function addTrustedModule(address module) external onlyRole(DEFAULT_ADMIN_ROLE) {
    _addTrustedModule(module);
}
```

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

This issue has been acknowledged by the team.

7. Missing _validateBeforeBorrow check in the borrow function

Severity: Low	Category: Business Logic
Target: - contracts/vault/modules/AAVEV3Module.sol	

Description

The `_validateBeforeBorrow` function is designed to ensure that the amount of USDC being borrowed does not exceed the allowed limit. This validation acts as a safeguard against over-borrowing based on the current missing rate.

While this check is correctly applied in the `supplyAndBorrow` function, it is missing in the standalone `borrow` function. As a result, a privileged admin can bypass the missing rate restriction and borrow an arbitrary amount of USDC, potentially violating the protocol's risk controls.

contracts/vault/modules/AAVEV3Module.sol: L136-L139

```
function borrow(address asset, uint256 amount) external onlyAuthorized nonReentrant {
    _borrow(asset, amount);
    emit BorrowSuccess(asset, amount);
}
```

Recommendation

It is recommended to add the `_validateBeforeBorrow` check in the `borrow` function.

Status

This issue has been resolved by the team with commit 96d20a7.

8. Incorrect approval amount may cause borrow to fail

Severity: Low	Category: Business Logic
Target: - contracts/vault/modules/AAVEV3Module.sol	

Description

The AAVEV3Module borrows USDC from Aave, but the debt is recorded on the Vault. Therefore, the Vault needs to call `approveDelegation` to authorize the debt allowance for the AAVEV3Module.

However, in the `borrow` function, if there is already some existing `borrowAllowance` for the desired borrow amount, the code calls `approveDelegation` with `amount - borrowAllowance`.

This logic is incorrect because `approveDelegation` sets the allowance directly instead of increasing it. As a result, the allowance is set to `amount - borrowAllowance` rather than `amount`, causing the borrow operation to fail.

contracts/vault/modules/AAVEV3Module.sol: L531-L533

```
function _borrow(address asset, uint256 amount) internal {
    _updateDebtStateBeforeAction();
    AAVEV3ModuleStorage storage $ = _getAAVEV3ModuleStorage();
    uint256 borrowAllowance = _getBorrowAllowance();
    if (borrowAllowance < amount) {
        _approveDelegation(amount - borrowAllowance);
    }
    ...
}
```

Recommendation

It is recommended to set the approval amount to `amount` instead of `amount - borrowAllowance`.

Status

This issue has been resolved by the team with commit 96d20a7.

2.3 Informational Findings

9. The runADLPartial does not consume traceId	
Severity: Informational	Category: Business Logic
Target: - contracts/core/StrategyManager/PortfolioMargin/lib/LibADL.sol	

Description

The `runADLPartial` function requires a `traceId` parameter, which is generated off-chain and serves as a unique identifier for tracking the operation. However, the `runADLPartial` function does not call the `consumeTraceId` function to consume this `traceId`, which may lead to the reuse of the same `traceId`.

contracts/core/StrategyManager/PortfolioMargin/lib/LibADL.sol: L531-L533

```
function runADLPartial(uint256 traceId, ADLBatchParam[] calldata partialParam) internal
{
    // Check if it is already marked for partial ADL
    uint256 strategyId = partialParam[0].adlParam.strategyId;
    ...
}
```

Recommendation

It is recommended to call `consumeTraceId` to consume the `traceId` and prevent its reuse.

Status

This issue has been resolved by the team with commit 96d20a7.

10. Some upgradeable contracts are not initialized

Severity: Informational	Category: Business Logic

Target:
- contracts/vault/Vault.sol
- contracts/vault/modules/AAVEV3Module.sol

Description

The `Vault` contract inherits from `PausableUpgradeable`, but its initializer function does not call `__Pausable_init()` to initialize `PausableUpgradeable`.

Similarly, the `AAVEV3Module` contract inherits from `ReentrancyGuardUpgradeable`, but its initializer function does not call `__ReentrancyGuard_init()` to initialize `ReentrancyGuardUpgradeable`.

contracts/vault/modules/AAVEV3Module.sol: L84-L87

```
function initialize() external initializer {
    __AccessControlEnumerable_init();
    _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
}
```

contracts/vault/Vault.sol: L338-L345

```
function initialize(address _feeTo) external initializer {
    VaultStorage storage $ = _getVaultStorage();
    $.feeTo = _feeTo;
    // Initialize all inherited contracts
    __Guardian_init();
    __Nonces_init();
    __ModuleManager_init(); // Initialize module manager
    _grantRole(DEFAULT_ADMIN_ROLE, _msgSender());
}
```

Recommendation

It is recommended to initialize the inherited upgradeable contract components in the initializer functions.

Status

This issue has been resolved by the team with commit eebf7b9.

11. Approval is not reset after removing the module

Severity: Informational	Category: Business Logic
Target: - contracts/vault/Vault.sol	

Description

The `Vault` contract can be extended with trusted modules, which may be authorized to use funds held by the `Vault`.

The `removeTrustedModule` function is used to remove a trusted module. However, after removal, the `Vault` does not revoke its possible token approvals for the module, which may allow the removed module to continue accessing `Vault` funds.

contracts/vault/Vault.sol: L407-L409

```
function removeTrustedModule(address module) external onlyRole(DEFAULT_ADMIN_ROLE) {
    _removeTrustedModule(module);
}
```

Recommendation

It is recommended to revoke all token approvals from the `Vault` to the module after it is removed.

Status

This issue has been resolved by the team with commit 96d20a7.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 749f4f6:

File	SHA-1 hash
Future.sol	839a207f3265ada27923bb4cb6e54424b0db24b6
Option.sol	1c67ea98581f022186cec6463ad8917e9eef2280
MultiToken.sol	97852df822d60e5688adeb3b0bcc3f7ea6da68ec
Cash.sol	89daa3d6dc364ca547aa8287f0fcf7423980e244
LibPM.sol	c9ea97e22e0c2de94d7bb3edde572e51d52b1cdd
LibLendPool.sol	3d34ebdcf9e140b9207fbb135a0b3754f5d06fe5
Config.sol	5366da8b98a664e8b2c16e94391916c6ea4ec9ab
EquityLib.sol	3a2db03d655c8fb1d99c9fe52becc9286d82018c
LibADL.sol	2e8d2fcdcde8681cd9ab59308eedbcab59d0f00b
PortfolioMarginLib.sol	a5d99be9e6deb7ca7783b648552da6fa73980f58
LibSwap.sol	10e9cf4bdd605341bd093d8cb2537b2a9fb53b3a
PMRiskControlLib.sol	34e3db4759b541c825079ee3ea5dda086606d5d8
LibStandardSign.sol	adbc3993ac152016def8f2591d8382f0401ab0fe
LibLiquidation.sol	5c6a954ea383f0b54f4ea88abdfb90cc95944d93
PMInitializeFacet.sol	36da7d41f929b29451d26da91076db3b1164861f
PMWithdrawCashFacet.sol	1d81685c6c587ff10bbbf5ebe37be09eb26f0e46
PMTransferStrategyFacet.sol	297621bcc9231413c14e9dcd94d178d75b7b3564
PMADLFacet.sol	49f8f7aa0eec150a61dccc5c40e9640f5522f85b
PMSwapFacet.sol	e3b6d3de501ea1684e72370682fc6ecff5904f2d
PMLendFacet.sol	3d284a4a279d322f013849e2af11aed039d6a05a
PMAccessControlFacet.sol	2142b80f456ef75c4d0152e81bbdebb8b898412b
PMSettleFacet.sol	18cb7667e6c59b9b67774323c7ab63ba9ea0b2d5
PMCompleteTheRFQFacet.sol	13226b28c27b9160490c29e93657d62c3f9216d4

PMSplitStrategyFacet.sol	44ab3be32c7f811a5719bda3d3ab21432ac3a5aa
PMLiquidationFacet.sol	e458ad231f54e257c2bd6e6866726bebf47bc8a0
PMMergeStrategyFacet.sol	3332d419c98591f5b3d51132d217127d5d54782d
PMCancelQuoteFacet.sol	6708b9456e135b6dd75a131a9affade3712d8e0e
PMDepositCashFacet.sol	0b3b423135d7c2da406f694ca5a20174cebf2753
LibDM.sol	42b867c3a34a23ed9e7f48d9bb25398a5f4083ee
Config.sol	550ae46c5a88c7632094ea2928edd9360baa9b3b
LibDualSign.sol	c48d4cc2fd41b3aaa522e8494f84872d1da13027
DMRiskControlLib.sol	c748147783e79c1463cbd0f2ef8c12b67c294003
DMCompleteTheRFQFacet.sol	814a5cbfda80bb827c55767538be46b02838b626
DMSettleFacet.sol	704cf1da481e2d704009732fc0c996b37cb2d19e
DMInitializeFacet.sol	a3649e1d78f2b7c37a53d499bb8e45f918625abc
DMTransferStrategyFacet.sol	5e592689c5625d0258591702a81ceaa0c150f1df
DMAccessControlFacet.sol	5a34f5a56aad0c369151862c9c4afc601a7b54ea
DMCancelQuoteFacet.sol	c3d77e3e8e1651eb85d46ed837c5f1a2bc70e3c5
StrategyStorage.sol	931b93edae1cd54c0739188ef11800250d476639
LibSignature.sol	f7ad77096d28ee9e36831aa6424745cb36616cbc
LibAccessControl.sol	85e4724b0e01f91dc02350cefb690b6411414765
LibTraceId.sol	c7e5cfc1822fc83333b19588575a3b464659b41c
Oracle.sol	b853403ab40383657ee30afbd9cc3ba6e1b4aeef
UniTWAPOracle.sol	77361756b5ddec4b074c4ca3b8b7ec409c5a1afd
OraclePermission.sol	ec16801dd4ffaacb737bcaf6e4196d55b7ea387f
ChainlinkOracle.sol	00f03ed67cffc7696fb5c74c8729194e2085ddfb
OracleChecker.sol	abf3d5e7a7fcee3d712580c709343ddf70d4ec42
Constant.sol	135edef8efcd2af271f12f03211b5c9a15999f37
AggregateAction.sol	936987757ab2d16254d92d2a14d04a4d5219eed4
SystemLib.sol	7642083a7abb95f3e3632bf9672bca0331a54929
OptionAssetEncoder.sol	fe25a1a248e0c54accf9229cbc9465b9e1b0a9ca
CommonEncoder.sol	dd9e9117c85d4d3c540aa5001e233b6a192a829c

FutureAssetEncoder.sol	11b89549a2eb2bc5c20073ca4f55f48fae4f2048
CashAssetEncoder.sol	01e3bb7501b260211fac363352398cc9de8a26c2
StrategyLib.sol	4a72c134431da682f90a393a1fdcecd1d124d16e
BlackFormula.sol	383b1cbe991093fa7eef418ad7e8f9efb6febce3
SVI.sol	bafad2d75a8bdf4c1d758935f27d3622a87b2ff0
FixedPointMathLib.sol	1e12ce2dece2d54a053f798fb8f9da56f6ec4ed7
ABR.sol	c1d7ebda14068a968c73398da91a19d429005a9e
SignedDecimalMath.sol	6c9c8d4dd4464e55b51aa7c4709601de104cb404
DecimalMath.sol	ada84a6a5ee020af6a096bafc0b3c56dc6910dc1
SafeTransferHelper.sol	734274c80cf0d8e0e13ec752a205cb43b6fa7ddb
TimestampCheck.sol	09a468b7f420dfdea47ff761f663b2a4bc063bd4
ContractAddress.sol	9798c0a61ef04533d1cd2be70177436d74f9360e
AssetLib.sol	d22c10f51987a33c05c186a74f1c285f5a0765e8
EIP712Lib.sol	cc9558fc0dd6149e1676ae0b11c4f95b6ecc8e45
Types.sol	e6a5da912e6f86360232e2704dbc5b675ab4ee7a
StrategyQuery.sol	2a67290947242d9bca7dd783938bb9f367f50433
LibUser.sol	5fc3059109c7a777e7e896367238f8dbc5d2d6d9
Config.sol	d9f1a1461c50fa4940feb215a7af5a2a01253c3c
LibLP.sol	c0138b605e7d44e3cd7778809578ef8f5afe12a7
LibAave.sol	6bdad68713b77a2fdfc8eda1db233c0c72aedefc
LPViewFacet.sol	a447be4caef4c3182500929b447a17e8167334bd
LPUpdateBorrowFacet.sol	101611ffcf2b02b313c01dce51d178801ca02c40
LPWithdrawFacet.sol	afe5a0a3c76e7a60fc29a51c00380aad2d573ac7
LPVaultBorrowFacet.sol	ca24e4a6c675690ed68c893808b263821a3ea7f7
LPInitializeFacet.sol	71dd84859dc85c4b81f5b30b19d3f32e3f272875
LPUpdateInterestFacet.sol	86313911dc9b7d6e42e24e2517d0ddad62e0b698
LPAAVEFacet.sol	f75da71ce0f6bc2938946fc6c4098a7a2287972f
LPDepositFacet.sol	f87b8d2e9cca07fe1cd79aed9af0e97f0c72ad6c
SafeVault.sol	ff4408ad89ac514c7578326effbff3e7eba00d00

Vault.sol	9270e5609673f36721e21ac2d9407540d58841b0
ModuleManagerUpgradeable.sol	a57b1aa6a780e640d3eae35520c9afeebe6a0f9d
GuardianUpgradeable.sol	0c97a200b914fbb3a556ec84e11d6aa971023e62
TokenDecimals.sol	a9c78d792bfdc93c932722d9243e816324c4d027
SwapModule.sol	75713a72fa133d5814a4e4b59160d5fd9556b870
AAVEV3Module.sol	91c9e7a05ea373757636369a53cf4347859bc3d7