SALUS SECURITY

JAN 2025

# CODE SECURITY ASSESSMENT

DEDERI V2

# Overview

## Project Summary

- Name: Dederi V2
- 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 V2 |
| **Version** | v4 |
| **Type** | Solidity |
| **Dates** | Jan 26 2025 |
| **Logs** | Dec 04 2024, Dec 17 2024, Jan 14 2025, Jan 26 2025 |

## Vulnerability Summary

| | |
|---|---|
| **Total High-Severity issues** | 0 |
| **Total Medium-Severity issues** | 8 |
| **Total Low-Severity issues** | 1 |
| **Total informational issues** | 2 |
| **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

## 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 | Sub asset check does not work | Medium | Business Logic | Resolved |
| 2 | Improper mark price calculation after the expiry time | Medium | Business Logic | Resolved |
| 3 | Incorrect mark price calculation | Medium | Business Logic | Resolved |
| 4 | Incorrect smooth mark price calculation | Medium | Business Logic | Resolved |
| 5 | GuardiansThreshold may be bypassed | Medium | Business Logic | Resolved |
| 6 | Missing chainlink sequencer status check | Medium | Business Logic | Resolved |
| 7 | Unmatched twap price check for mark price | Medium | Business Logic | Resolved |
| 8 | Centralization risk | Medium | Centralization | Mitigate |
| 9 | Lack of signature length check | Low | Business Logic | Resolved |
| 10 | Gas Optimization | Informational | Gas Optimization | Resolved |
| 11 | Inconsistent rounding of timeWeightedAverageTickS56. | Informational | Business Logic | Resolved |

## 2.2 Notable Findings

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

### 1. Sub asset check does not work

| Severity: Medium | Category: Business Logic |
| --- | --- |
| Target:<br>- core/StrategyManager/PortofolioMargin/lib/LibPM.sol | |

### Description

Users can allocate a portion of assets to another strategy. We use the checkSubAssets function to verify whether the split assets still belong to the original strategy.

The issue is that the checkSubAssets function returns a boolean indicating whether the split assets belong to the original strategy. We missed checking its return value.

Another issue is that we remove the split assets before checking the sub-assets, which causes the checkSubAssets function to always return false.

core/StrategyManager/PortofolioMargin/lib/LibPM.sol:L665-L673

```
function _split(uint256 _fromId, uint256 _toId, Asset[] memory _assets) internal {
    Layout storage s = layout();
    s.strategyStorage.removeAssets(_fromId, _assets);
    _addAssets(_toId, _assets);
    Strategy memory _fromStrategy = s.strategyStorage.getStrategy(_fromId);
    AssetLib.checkSubAssets(_fromStrategy.assets, _assets);
    PMRiskControlLib._riskCheckOfSplit(_toId);
    PMRiskControlLib._riskCheckOfSplit(_fromId);
}
```

### Recommendation

Check the sub-assets before removing them from the previous strategy, and ensure to validate the return value of the checkSubAssets function.

### Status

This issue has been resolved by the team with commit 3a6b15e.

## 2. Improper mark price calculation after the expiry time

| Severity: Medium | Category: Business Logic |
| --- | --- |

Target:
- core/StrategyManager/PortofolioMargin/lib/LibPM.sol

## Description

When checking a strategy's MM, we calculate the mark price of future legs, incorporating delta time in the calculation.

The issue occurs when a strategy leg reaches its expiry time, causing the expiry time to be less than the current timestamp. This results in a mark price calculation failure due to an underflow.

core/StrategyManager/PortofolioMargin/lib/PortofolioMarginLib.sol:L31-L55

```
function MM(Strategy memory _strategy) internal view returns (uint256) {
    LibPM.Layout storage s = LibPM.layout();
    ...
    uint256[] memory _F = PMAssetLib.getFutureMarkPrice(_strategy.assets);
    address _underlying = PMStrategyLib.getUnderlying(_strategy);
    SVIItems memory _sviItems = s.oracle.SVI(_underlying);
    ...
}
```

core/StrategyManager/PortofolioMargin/lib/LibPM.sol:L665-L673

```
function markPrice(address _underlying, uint256 _T, uint256 _St) public view returns
(uint256 _futureMarkP) {
    // Get abr
    int256 _abr = _ABR(_underlying, _T);
    uint256 _t = block.timestamp;
    int256 x = _abr * (_T - _t).toInt256() / Constant.YEAR_SECONDS_INT;
    uint256 _e_x = FixedPointMathLib.exp(x);

    return _St * _e_x / Constant.HIGH_DECIMALS;
}
```

## Recommendation

Exclude the expiry strategy legs when we calculate the MM.

## Status

This issue has been resolved by the team with commit 3a6b15e.

## 3. Incorrect mark price calculation

| Severity: Medium | Category: Business Logic |
| --- | --- |
| Target:<br>- core/asset/Future.sol | |

## Description

The Mark Price calculation should follow the design outlined in the documentation. According to the doc, after the expiration date at 7:30 UTC, we should use the IndexTWAP from 7:30 to the current time. Post-expiration, this value will serve as the underlying price for settlement.

The issue is that we continue using the IndexPrice even after the expiration date at 7:30 UTC.

core/asset/Future.sol:L38-L43

```
function markPrice(address _underlying, uint256 _expiration) public view returns
(uint256 markP) {
    uint256 _indexPrice = oracle.indexPrice(_underlying);
    return markPrice(_underlying, _expiration, _indexPrice);
}
```

core/asset/Future.sol:L155-L164

```
function markPrice(address _underlying, uint256 _T, uint256 _St) public view returns
(uint256 _futureMarkP) {
    // Get abr
    int256 _abr = _ABR(_underlying, _T);
    uint256 _t = block.timestamp;
    int256 x = _abr * (_T - _t).toInt256() / Constant.YEAR_SECONDS_INT;
    uint256 _e_x = FixedPointMathLib.exp(x);

    return _St * _e_x / Constant.HIGH_DECIMALS;
}
```

## Recommendation

Update the mark price calculation to align with the documentation.

## Status

This issue has been resolved by the team with commit 3a6b15e.

## 4. Incorrect smooth mark price calculation

| Severity: Medium | Category: Business Logic |
| --- | --- |
| Target:<br>- core/asset/Future.sol | |

## Description

According to the documentation, the calculation of the SmoothMarkPrice for futures assets follows different methods based on the time period:

1. Before the expiration date (UTC 7:40), the `SmoothMarkPrice` is calculated using the `SmoothIndexPrice` as the value of `St` in the `MarkPrice` formula.

2. After the expiration date (UTC 7:40), the `SmoothMarkPrice` is equal to the `MarkPrice`.

However, the contract implementation does not currently handle the correct calculation for the period after the expiration date (UTC 7:40).

contracts/core/asset/Future.sol:L31-L55

```
function SmoothMarkPrice(Asset memory _asset) public view returns (uint256 SmoothMarkP)
{
    (address _underlying, uint256 _expiration) =
FutureAssetEncoder.decode(_asset.assetId);
    uint256 _indexTWAPPrice = oracle.indexTWAP(_underlying);
    return markPrice(_underlying, _expiration, _indexTWAPPrice);
}
```

## Recommendation

Correctly implement the calculation method from the design document in the `SmoothMarkPrice` function.

## Status

This issue has been resolved by the team with commit 3a6b15e.

## 5. GuardiansTheshold may be bypassed

| Severity: Medium | Category: Business Logic |
| --- | --- |
| Target:<br>- vault/Vault.sol | |

## Description

When withdrawing funds, multiple signatures from guardians are required to ensure security. A guardian threshold is in place to verify that signatures come from different signers. However, a potential issue arises: a malicious signer could generate multiple valid signatures for the same message by manipulating the **k** parameter in the ECDSA algorithm.

The Python script below demonstrates this vulnerability by using the same message to produce two distinct yet valid signatures.

```
from ecdsa import SigningKey, SECP256k1
from ecdsa.util import randrange_from_seed__trytryagain
from hashlib import sha256
from eth_utils import keccak, to_bytes, to_checksum_address

sk = SigningKey.generate(curve=SECP256k1)
vk = sk.verifying_key

public_key_bytes = b"\x04" + vk.to_string()
public_key_hash = keccak(public_key_bytes[1:])[12:]
signer_address = to_checksum_address(public_key_hash)

def generate_signature(message, seed):
   prefix = f"\x19Ethereum Signed Message:\n{len(message)}".encode()
   message_hash = keccak(prefix + message)
   k = randrange_from_seed__trytryagain(seed, sk.curve.order)
   signature = sk.sign(message_hash, k=k)
   r = int.from_bytes(signature[:32], "big")
   s = int.from_bytes(signature[32:], "big")
   v = 27
   signature_with_v = signature + bytes([v])
   return signature_with_v, message_hash
message = b"message to sign"
signature1, message_hash1 = generate_signature(message, b"random_seed_1")
signature2, message_hash2 = generate_signature(message, b"random_seed_2")
print(f"Signer Address: {signer_address}")
print(f"Message Hash: {message_hash1.hex()}")
print(f"Signature 1: {signature1.hex()}")
print(f"Signature 2: {signature2.hex()}")
```

contracts/vault/Vault.sol:L486-L502

```
   function _verifyGuardianLargeWithdrawalSignature(
       WithdrawConfirmation calldata confirmation,
       bytes calldata confirmorSignature,
       bytes[] memory signature
   ) internal view {
       uint256 signaturesLength = signature.length;
       require(signaturesLength >= guardiansThreshold && signaturesLength > 0,
Vault_InsufficientGuardians());
       bytes32 messageHash = _getGuardianSignatureMessageHash(confirmation,
```

```
confirmorSignature);
        for (uint256 i; i < signaturesLength; ++i) {
            if (i < signaturesLength - 1) {
                require(keccak256(signature[i]) < keccak256(signature[i + 1]),
Vault_InvalidSignature());
            }
            address guardian = messageHash.recover(signature[i]);
            require(guardiansSet.contains(guardian), Vault_InvalidGuardian(guardian));
        }
    }
```

## Recommendation

Enhance the security check to make sure that signatures come from different signers.

## Status

This issue has been resolved by the team with commit a09c513.

## 6. Missing chainlink sequencer status check

| Severity: Medium | Category: Business Logic |
| --- | --- |
| Target:<br>- core/oracle/ChainlinkOracle.sol | |

## Description

In ChainlinkOracle, we will fetch asset prices from Chainlink oracle. The problem is that when the L2 sequencer is down and up, some prices will not be accurate. According to https://docs.chain.link/data-feeds/l2-sequencer-feeds#example-code, we should check the sequencer's uptime.

contracts/core/oracle/ChainlinkOracle.sol:L90-L112

```
function _getChainlinkPrice(address asset) internal view returns (uint256) {
    ChainlinkOracleStorage storage $ = _getChainlinkOracleStorage();
    TokenFeedConfig memory tokenConfig = $.tokenFeedConfigs[asset];
    AggregatorV3Interface feed = AggregatorV3Interface(tokenConfig.feed);

    uint256 maxStalePeriod = tokenConfig.maxStalePeriod;
    (, int256 answer,, uint256 updatedAt,) = feed.latestRoundData();

    if (answer <= 0) {
        revert InvalidAnswer(asset, answer, updatedAt, block.timestamp);
    }

    //arb error avoid
    if (block.timestamp > updatedAt) {
        if ((block.timestamp - updatedAt) > maxStalePeriod) {
            revert InvalidAnswer(asset, answer, updatedAt, block.timestamp);
        }
    }

    // Chainlink USD-denominated feeds store answers at 8 decimals, mostly
    uint256 decimalDelta = 18 - feed.decimals();
    return uint256(answer) * (10 ** decimalDelta);
}
```

## Recommendation

Check the L2 sequencer's uptime feed and make sure the L2 sequencer is stable.

## Status

This issue has been resolved by the team with commit 7e5a5c5.

## 7. Unmatched twap price check for mark price

| Severity: Medium | Category: Business Logic |
| --- | --- |
| Target:<br>- core/oracle/OracleChecker.sol | |

## Description

When updating the settlement price, we compare the updated TWAP price with the Uniswap V3 TWAP price. However, a potential issue arises due to mismatched time slots: Uniswap V3 uses a 30-minute TWAP window, while the off-chain TWAP is calculated over the interval between 7:30 and block.timestamp. These differing time frames can result in discrepancies between the TWAP values, which may ultimately lead to the price check failing and causing the transaction to revert.

contracts/core/oracle/UniTWAPOracle.sol:L127-L145

```
function _checkSettleTWAPRange(address token, uint256 offChainPrice) internal view {
    uint256 onChainTWAPPrice;
    OracleCheckerStorage storage $ = _getOracleCheckerStorage();
    if (!$.enableUniswapCheck) {
        return;
    }
    if (token == Constant.USDC) {
        return;
    } else if (token == Constant.WBTC || token == Constant.WETH) {
        onChainTWAPPrice = getUniswapTWAP(token, 30 minutes);
    } else {
        revert OracleChecker_TokenNotSupported();
    }
    ...
}
```

## Recommendation

Make sure that both the offchain twap price and uniswap twap price have the same time slot.

## Status

This issue has been resolved by the team with commit 7e5a5c5.

## 8. Centralization risk

| Severity: Medium | Category: Centralization |
| --- | --- |
| Target:<br>- contracts/oracle/Oracle.sol<br>- contracts/vault/Vault.sol | |

## Description

There are some privileged owner roles, for example, default admin role, oracle signers, etc. These roles will set the exchange router, set the assets' index price and some other key functions.

Should the owner's private key be compromised, an attacker could withdraw all yield distribution.

Since the privileged account is a plain EOA account, this can be worrisome and pose a risk to the other users.

contracts/oracle/Oracle.sol: L183-L185

```
function addWhitelist(address user) external onlyRole(DEFAULT_ADMIN_ROLE) {
    _signerWhitelist.add(user);
}
```

contracts/vault/Vault.sol: L357-L359

```
function setSwapRouter(address _swapRouter) external onlyRole(DEFAULT_ADMIN_ROLE) {
    swapRouter = ISwapRouter(_swapRouter);
}
```

## 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

The team has employed an MPC solution to mitigate this issue.

## 9. Lack of signature length check

| Severity: Low | Category: Business Logic |
| --- | --- |
| Target:<br>- core/vault/Vault.sol | |

## Description

When users withdraw cash from the vault, they must provide signatures signed by guardians. A signature threshold is in place, and we need to ensure that the number of signatures is never less than the guardiansThreshold. Failing to do so would bypass the guardiansThreshold limitation.

contracts/vault/Vault.sol: L530-L546

```
function _verifyGuadianPersonalSignature(bytes32 messageHash, bytes[] memory signature)
internal view {
    uint256 signaturesLength = signature.length;
    // require(signaturesLength >= guardiansThreshold, Vault_NotEnoughtGuardians());
    address[] memory guardians = new address[](signaturesLength);
    for (uint256 i; i < signaturesLength; ++i) {
        address guardian = messageHash.recover(signature[i]);
        require(guardiansSet.contains(guardian), Vault_InvalidGuardian(guardian));
        guardians[i] = guardian;
    }
    _insertionSort(guardians);
    for (uint256 i; i < signaturesLength - 1; ++i) {
        if (guardians[i] == guardians[i + 1]) {
            revert Vault_DuplicateSign();
        }
    }
}
```

## Recommendation

Verify that the number of signatures meets the guardiansThreshold.

## Status

This issue has been resolved by the team with commit 6ac972f.

# 2.3 Informational Findings

## 10. Gas Optimization

| Severity: Informational | Category: Gas Optimization |
| --- | --- |
| Target:<br>- application/rfq/StandardPMRFQ.sol | |

### Description

In the `_completeTheRFQInternal` function, when the premium is 0, it still enters the else logic and adds an assert with a units of 0 to the strategy. This wastes gas and does not cause any state changes.

application/rfq/StandardPMRFQ.sol:L276-L298

```
function _completeTheRFQInternal(
    StandardPMRFQDataTypes.CompleteTheRFQMakerParams calldata makerParams,
    StandardPMRFQDataTypes.CompleteTheRFQTakerParams calldata takerParams
) internal returns (uint256 takerStrategyId, uint256 makerStrategyId) {
    ...
    if (takerPremium > 0) {
        premiumAsset = Asset({
            assetType: cashAssetType,
            assetId: CashAssetEncoder.encode(Constant.USDC),
            units: takerPremium,
            extra: bytes32(0)
        });
        ...
    } else {
        premiumAsset = Asset({
            assetType: cashAssetType,
            assetId: CashAssetEncoder.encode(Constant.USDC),
            units: -takerPremium,
            extra: bytes32(0)
        });
        Asset[] memory takerTransferAssets = new Asset[](1);
        takerTransferAssets[0] = premiumAsset;
        strategyManager.transferCash(takerStrategyId, makerStrategyId,
takerTransferAssets);
    }
    ...
}
```

### Recommendation

When `takerPremium` is 0, the premium transfer is not executed.

### Status

This issue has been resolved by the team with commit 6ac972f.

## 11. Inconsistent rounding of timeWeightedAverageTickS56

| Severity: Informational | Category: Business Logic |
| --- | --- |

Target:
- contracts/oracle/lib/UniTWAPOracle.sol

## Description

After calling `IUniswapV3Pool::observe` to obtain `tickCumulatives`, if the difference `tickCumulatives[1] - tickCumulatives[0]` is negative, the subsequent division will round towards zero. This can cause the tick value to be higher than expected, leading to an overestimation of the price.

contracts/oracle/lib/UniTWAPOracle.sol:L191-L199

```
function _getUniswapTwap(TokenTWAPConfig memory config, uint32 toAgos, uint32
anchorPeriod)
    internal
    view
    returns (uint256)
{
    ...
    int56 timeWeightedAverageTickS56 = (tickCumulatives[1] - tickCumulatives[0]) /
anchorPeriod__;

    if (timeWeightedAverageTickS56 < TickMath.MIN_TICK || timeWeightedAverageTickS56 >
TickMath.MAX_TICK) {
        revert TWAPOracleTickNotInRange();
    ...
}
```

## Recommendation

When `tickCumulatives[1] - tickCumulatives[0]` is less than 0 and cannot be evenly divided by `anchorPeriod__`, the resulting `timeWeightedAverageTickS56` should be decreased by 1.

## Status

This issue has been resolved by the team with commit 7e5a5c5.

# Appendix

## Appendix 1 - Files in Scope

This audit covered the following files in commit 834b213:

| File | SHA-1 hash |
|---|---|
| Future.sol | 3b737b7873889b33e80b441bd6e95598a124ada0 |
| Option.sol | 5bbd9de075fb875ee3519dd3b03b97b958ad08d8 |
| Cash.sol | 203f0d0173da924f33515b0993ec7d58a2137c93 |
| LibPM.sol | 5e6d1777b2c4fe13da6bf44e76d60981aedc9066 |
| PMError.sol | 53655e772443fd8b2c24df789213eae33deaa222 |
| Config.sol | a990440e3bcf6129a88aa2bf604d6c0912a9f69a |
| EquityLib.sol | eb49c58721b4183521bc863e5c9ee37c6f161994 |
| PMEvent.sol | c4ebc60323bb4faa4cd8b372db229d8c01186f4d |
| PortfolioMarginLib.sol | b33a14d9b9ee35bb68d6774db1c7ba2e01855e4a |
| PMStrategyLib.sol | 932c4f62d8c44cf2c38e67b6adfe5f1e1a5b8343 |
| PMRiskControlLib.sol | d4c909a9618d92affaa6048ae84f749562002965 |
| PMAssetLIb.sol | 07dee8580df72a1a0b58f2fd8e95d4d4dfabd070 |
| PMInitializeFacet.sol | ed5425c4bacae6def8b31843d42ad467739aee72 |
| PMOwnershipFacet.sol | cae5ccd10ae826b8489e6e4ed51f9d4d1ec06c36 |
| PMWithdrawCashFacet.sol | 462a47eca36492573a9d2390ba4e29db8e3dad97 |
| PMTransferStrategyFacet.sol | 850622b2563a2fd109f60d12b67d551b3f60183e |
| PMADLFacet.sol | 407783bb835b73ac0e0102e1fe2c191ba786a53a |
| PMMergeFacet.sol | bdd7696359a798361b169bdfa3d1a8a919a21a40 |
| PMTransferCashFacet.sol | 2455795fe737a51c719990170d16f6719297859b |
| PMSplitFacet.sol | 94344763157a393f7f9bdd1232eb7324b9c0d515 |
| PMReadFacet.sol | 2de56c392309cab2d010efc878495d9a1cd21b03 |
| PMSettleFacet.sol | 0086d39a459972727419955072ee26cf76998041 |
| PMMintFacet.sol | 90b99c47ea0deb5f014b133109361562e858437c |

| | |
|---|---|
| PMLiquidationFacet.sol | 30425bc8f5e9e81374d8d732c801a2541a4fc835 |
| PMDepositCashFacet.sol | 40eb7746a31e6939e16a245ba287082a1f9b48a1 |
| dual/lib/LibDM.sol | 5480e00e06de42e5ee0598d0234a04fb4c298397 |
| dual/lib/Config.sol | 2787ed9c1740324372a31f76a9756ba4a5d64da7 |
| dual/lib/DMError.sol | e62e44671e42a43da3512f812bb4c1b8dcd8afb1 |
| dual/lib/DMAssetLib.sol | f94be0e546d2cf8d9ec0ce91253f4d7948b21045 |
| dual/lib/DMEvent.sol | c9862aa6b3c1fee1d037acedc443c0d9d565cea4 |
| dual/lib/DMRiskControlLib.sol | 441e4651bf0a2f22acc670ea580e7fbc994e22d5 |
| dual/facet/DMSettleFacet.sol | 2c9b0a636ea84292817a55c17e3154f259719ded |
| dual/facet/DMInitializeFacet.sol | 85f0510e4fe57a2e6e3f3464f12bbdbe712b2acc |
| dual/facet/DMTransferStrategyFacet.sol | 05fc4b14de1e217ba4c916eb977ba6139a16d86c |
| dual/facet/DMOwnershipFacet.sol | 8a148a86752d96e58781e16368a7885d462c83d1 |
| dual/facet/DMMintFacet.sol | c99284c6cdf745a3b4764113d347215917dd47fb |
| dual/facet/DMReadFacet.sol | f8eb7e9bcb57d646ca8f9814eb090ec5c5ec2fdc |
| dual/facet/DMDepositCashFacet.sol | fc6269f589640f26cf5e422d2a093c3f9ae92fc0 |
| StrategyStorage.sol | a6d26eae49ffae4b74f4d4f02145e6c4f1af9b5a |
| StrategyLib.sol | 18e7e7465f7eab229d56a4c27ecec5bf4f9d78f2 |
| AssetLib.sol | a5bc35a3e594a3f7a05d5d2b9a5a9a509df5b66b |
| Oracle.sol | 1a9cf122d5dcebd353208dfd33e508c35ad1c747 |
| OraclePermission.sol | 20570abfd80af0bcd3e134bb8d61b55e6aeb2d7b |
| Constant.sol | 8eb662166aaed1fdac1bbbbee86b7938497efbdf |
| AggregateAction.sol | bb21abfe63947ce20cd16e8b4dc6f00d1145565e |
| OptionAssetEncoder.sol | 009263b99b718bffaa4f880d8d5228f0ebccf4f4 |
| CommonEncoder.sol | ed1de819020d45ba6ee9dde5ed6788bf5e7c5192 |
| FutureAssetEncoder.sol | 8a91d7c3fb5416f00aeeaac867256183ce8e8182 |
| CashAssetEncoder.sol | 670acfb83d81be98c970354a61055b252e3417a5 |
| BlackFormula.sol | 98238dd5b2b505c7c9d406b05d7d3e07cb68f878 |
| SVI.sol | a1b44e7f26b96ff6e87ddf488df9275ff88de7f0 |
| FixedPointMathLib.sol | 1e12ce2dece2d54a053f798fb8f9da56f6ec4ed7 |

| | |
|---|---|
| ABR.sol | 074e5bd59139eb709da47b8347bb8213831d401b |
| SignedDecimalMath.sol | 6c9c8d4dd4464e55b51aa7c4709601de104cb404 |
| DecimalMath.sol | ada84a6a5ee020af6a096bafc0b3c56dc6910dc1 |
| TimestampCheck.sol | 43c7f41d5c400526bb8262a8488ae60c0133b412 |
| Types.sol | af110286814a121ba63176b5891d4f249ac2ed3f |
| DualRFQDataTypes.sol | 5849c1c6f060c9c0e13a5f622d5e6c9c561a81bf |
| StandardPMRFQDataTypes.sol | 6a01884e0ccc9d6de5e96364bbe179cc38d32ce4 |
| EIP712Lib.sol | 10236ff7d4c070fc3493b8c1422231a409e201ce |
| StandardSign.sol | 9b87537ddd1a79c37998eb3c607fdf1b91c30c2e |
| DualSign.sol | fafa9f3426eab3b6011ea05d3231b4f6e00e886e |
| StrategyQuery.sol | 2c93bd7656c2b271db4a3a4b012a334c3c431b47 |
| DualRFQ.sol | c92ae44dbe6e71e94c81627200a6bebf1410a7d4 |
| RFQPermission.sol | 69fedfd523552cfecdd7c987725a422cd44be320 |
| StandardPMRFQ.sol | cba3450f5391d8c95c128c42e94417bc0f3030c9 |
| VaultPermission.sol | 9a0d779ed0a29f1fd49603064266e42d9058fca9 |
| Vault.sol | 4859669539c47015e081d50a799f51f35a7b2181 |
| ExchangeCore.sol | f83cb7ebe566d70f5fdd52fefb5575d0d526bc21 |

And we audited the commit 63d3af2 that introduced new code optimization:

| File | SHA-1 hash |
|---|---|
| DualSign.sol | 5ea18c7bd270877fec24df1c7c0c1183781e68c5 |
| RFQPermission.sol | c5bd2f858dd2ab4d7bd6f356109db2f867c58f15 |
| StandardSign.sol | 429fbed8c131ad28bb8a592be09b13c576b8f61e |
| EIP712Lib.sol | cca1778a5746c252d0c3fc4ffccde659dfd0f782 |
| DualRFQ.sol | 693bb2fd97d2887fc6447c2d5f348a9ed4a8d60a |
| StandardPMRFQ.sol | 5b0862c834feed67e43d61e1bb3cb9922b35d4d8 |
| PMInitializeFacet.sol | 29608c679c17f73f92b5c3e835c8c29b15b3b3da |
| LibPM.sol | d7100518e95862c018f74b60da76ae6d35703af4 |
| PortfolioMarginLib.sol | 5c7c684c81b0a31e51c53d9dfc7a289445d8e4c0 |

| | |
|---|---|
| DMInitializeFacet.sol | 37e1b9dae52214281917ffe816a486ddeb71f9fb |
| LibDM.sol | 5aa56cea1975fe39055e995c52a4f35d8c72b63e |
| Oracle.sol | bd43b10a521d51aabfca81a4cadf3359d0a1a92f |
| IOracle.sol | eb3c2fda33c95d351a4f9fdf2192ec513ed08243 |
| OraclePermission.sol | 94c05c40041d46f8fec12b66706710225119f1f0 |
| EIP712Lib.sol | cca1778a5746c252d0c3fc4ffccde659dfd0f782 |
| SafeTransferHelper.sol | 0844276013b720bd86ca28a5d078dd55fc0bda0e |
| SafeVault.sol | 8dcd1d1e36494931a6486e689ab3622019bb5b71 |
| Vault.sol | 1c3f073cbe80de677f4ef50981fd817cacb073d0 |
| IVault.sol | 06db9c44ef23ec626db9b2483e5697c3e3bb01be |
| ExchangeCore.sol | cbe62638e195fb876f839333117bc661ac1a5d23 |
| TokenDecimals.sol | a9c78d792bfdc93c932722d9243e816324c4d027 |
| VaultPermission.sol | 38303cec4e7dba17461390932be3b656721e3289 |

And we audited the commit a09c513 that introduced new code optimization:

| | |
|---|---|
| UniTWAPOracle.sol | f236df9ecab834dc3f1c7b11bcea3706918d1cac |
| OraclePermission.sol | 94c05c40041d46f8fec12b66706710225119f1f0 |
| ChainlinkOracle.sol | 67bf97d89172e160bcf33ae636ad3ad87db27de1 |
| OracleChecker.sol | 03fe3027d510f32d2b5c6a2826d6516425fbd383 |