SALUS SECURITY

MAY 2024

CODE SECURITY ASSESSMENT

DEDERI

Overview

Project Summary

- Name: Dederi
- Platform: Arbitrum
- Language: Solidity
- Repository:
    - https://github.com/Dederi-Finance/dederi-contracts
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Dederi
Version	v3
Type	Solidity
Dates	Jun 04 2024
Logs	May 24 2024; May 29 2024; Jun 04 2024

Vulnerability Summary

Total High-Severity issues	2
Total Medium-Severity issues	6
Total Low-Severity issues	4
Total informational issues	3
Total	15

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. Incorrect initial PnL for liquidator 6
2. Futures may be ignored when merging 8
3. Lack of input validation for settleStrategy() 9
4. Lack of strategy validation in the executeMergeStrategies() 10
5. The ADL process may be blocked because of underflow 11
6. Possible overflow because of unsafe casting 12
7. Incorrect function called 13
8. Centralization risk 14
9. The protocol can withdraw fees from questionable unsettled PnL 15
10. Signature verification failed because signers own the same index 17
11. An out-of-bound error may occur when verifying signatures 19
12. Lack of input validation for counterparties in adlExecute() 20
2.3 Informational Findings 21
13. Gas optimization suggestions 21
14. Inconsistency between comments and implementation 22
15. Typos 23
Appendix **24**
Appendix 1 - Files in Scope 24

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	Incorrect initial PnL for liquidator	High	Business Logic	Resolved
2	Futures may be ignored when merging	High	Business Logic	Resolved
3	Lack of input validation for settleStrategy()	Medium	Data Validation	Resolved
4	Lack of strategy validation in the executeMergeStrategies()	Medium	Data Validation	Resolved
5	The ADL process may be blocked because of underflow	Medium	Numerics	Resolved
6	Possible overflow because of unsafe casting	Medium	Numerics	Resolved
7	Incorrect function called	Medium	Business Logic	Resolved
8	Centralization risk	Medium	Centralization	Mitigated
9	The protocol can withdraw fees from questionable unsettled PnL	Low	Business Logic	Acknowledged
10	Signature verification failed because signers own the same index	Low	Business Logic	Resolved
11	An out-of-bound error may occur when verifying signatures	Low	Business Logic	Resolved
12	Lack of input validation for counterparties in adlExecute()	Low	Data Validation	Resolved
13	Gas optimization suggestions	Informational	Gas Optimization	Resolved
14	Inconsistency between comments and implementation	Informational	Inconsistency	Resolved
15	Typos	Informational	Code Quality	Resolved

2.2 Notable Findings

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

1. Incorrect initial PnL for liquidator

Severity: High	Category: Business Logic

Target:
- contracts/diamond/libraries/LibStrategyLiquidate.sol

Description

When one strategy is not healthy, the liquidator can liquidate this strategy. In the process of liquidation, the contract will calculate the liquidated premium for options legs in this strategy.

For example, the to-be-liquidated strategy owns one long call option. When the strategy is liquidated, the previous strategy owner will sell this long-call option to the liquidator with the liquidated premium. The previous strategy owner will take the liquidated premium as the profits and on the contrary, the liquidator will pay the liquidated premium.

The vulnerability is that the liquidator's initial PnL (option premium) is not correct. The correct value should be `-liquidatedPremium`.

contracts/diamond/libraries/LibStrategyLiquidate.sol:L120-L168

```
function _handleLiquidateStrategy(StrategyTypes.LiquidateStrategyRequest memory
liquidatorStrategyRequest)
    internal
    returns (uint256 strategyId)
{
    StrategyTypes.StrategyAllData memory liquidatedStrategy =
        LibStrategy.getStrategyAllData(liquidatorStrategyRequest.strategyId);
    // get liquidated option premium pnl
    int256 liquidatedPremium = LibPosition.getLiquidatedPremiumAndCheckSlippage(
        liquidatedStrategy.option, liquidatorStrategyRequest.option
    );
    ......
    // realizedPnl
    int256 liquidatedRealizedPnl = liquidatedPremium + LiquidatedFuturePnl;
    ......

    (strategyId,) = LibStrategyOpen._createOrMergeStrategy(
        StrategyTypes.StrategyRequest({
            ...
        }),
        liquidatedPremium,
        0
    );
}
```

Recommendation

Consider creating or merging liquidators' strategies with the correct option premium.

Status

This issue has been resolved by the team with commit b5e423d.

2. Futures may be ignored when merging

Severity: High	Category: Business Logic

Target:
- contracts/diamond/libraries/LibStrategyOpen.sol

Description

When trying to merge futures, the counter variable has been added twice, causing some futures to be ignored in the merge.

contracts/diamond/libraries/LibStrategyOpen.sol:L268-L273

```
for (uint256 i; i < newFuturesLen; ++i) {
        realizedPnl += LibPosition.tryMergeFuturePosition(_oldStrategy.positionIds,
_future[i]);
        unchecked {
            ++i;
        }
}
```

Recommendation

Consider removing one of the counter increments.

Status

This issue has been resolved by the team with commit a714f51.

3. Lack of input validation for settleStrategy()

Severity: Medium	Category: Data Validation
Target: - contracts/diamond/facets/StrategySettleFacet.sol	

Description

The settleStrategy() function aims to settle some positions in one strategy. Although this function can be called by CLEAR_ROLE, it is recommended to add related input parameters validation.

For example:

1. Double check that the strategy is active.
2. Double check that the positions belong to the strategy.

contracts/diamond/facets/StrategySettleFacet.sol:L25-L30

```
function settleStrategy(uint256 strategyId, uint256[] memory expiringPositionId)
external whenNotPaused {
   LibAccessControlEnumerable.checkRole(Constants.CLEAR_ROLE);
   uint256 settlementFee = LibSettle._handleSettle(strategyId, expiringPositionId);
   emit StrategySettled(strategyId, expiringPositionId, settlementFee);
}
```

Recommendation

Consider adding some on-chain input validation.

Status

This issue has been resolved by the team with commit a714f51.

4. Lack of strategy validation in the executeMergeStrategies()

Severity: Medium	Category: Data Validation
Target: - contracts/diamond/facets/StrategyMSFacet.sol	

Description

In the executeMergeStrategies() function, there is no strategy status and ownership validation, although these checks have been performed when requesting a merge.

Two possible scenarios are as below:

1. The strategy could be set to inactive in the adlRequest() function by the clear role after requesting and before executing. The inactive strategy should not be merged;
2. A user could send one of the strategies he wants to merge (let's say Strategy B) to another user after a merge request. After executing the merge, all profits in Strategy B will return to this user.

Recommendation

Consider double-checking the strategy status and ownership in the executeMergeStrategies() function.

Status

This issue has been resolved by the team with commit a714f51.

5. The ADL process may be blocked because of underflow

Severity: Medium	Category: Numerics

Target:
- contracts/diamond/facets/StrategyADLFacet.sol

Description

When one strategy's equity is less than 0, the clear role will trigger the ADL process to decrease the whole system risk. In the ADL process, the clear role will choose some counterparties to match the ADL strategy's positions to close them.

When the contract handles the futures, the contract will calculate realizedPnl via the difference between entryPrice and futureADLPrice. The vulnerability is that the transaction will be reverted if futureADLPrice is less than entryPrice because of the underflow.

contracts/diamond/facets/StrategyADLFacet.sol:L125-L150

```
function handleFuture(
    StrategyTypes.ADLStrategyRequest memory strategy,
    StrategyTypes.ADLStrategyRequest[] memory counterparties,
    LibPositionCore.Layout storage l
) internal returns (int256 realizedPnl) {
    uint256 futureADLPrice = LibPriceOracle._getADLPrice(strategy.positionId);
    StrategyTypes.Future storage adlFuture = l.futurePositions[strategy.positionId];
    uint256 reqLen = counterparties.length;
    uint256[] memory ADLSizes = new uint256[](reqLen);
    for (uint256 i; i < reqLen; ++i) {
        StrategyTypes.Future storage future =
l.futurePositions[counterparties[i].positionId];
    ...
        //ADLPnl
        realizedPnl =
            (int256(futureADLPrice - adlFuture.entryPrice) * int256(futureQty)) /
int256(Constants.SIZE_PRECISION);
        // update the counterpart's realized pnl
        LibStrategy.unpdateRealizedPnl(counterpartyStrategyId, -realizedPnl);
```

Recommendation

Consider casting from uint256 to int256 before subtraction calculation.

Status

This issue has been resolved by the team with commit b5e423d.

6. Possible overflow because of unsafe casting

Severity: Medium	Category: Numerics
Target: - contracts/diamond/libraries/LibStrategyMS.sol	

Description

If equity is negative, casting from int256 to uint256 may create an overflow, causing the check to be bypassed.

contracts/diamond/libraries/LibStrategyMS.sol:L115-L120

```
int256 equity = firstStrategyEquity + secondStrategyEquity;
if (uint256(equity) < mergeStrategyMargin.im) {
        LibMarketPricer._checkCollateralEnough(
        mergeStrategyMargin.im - uint256(equity), requestParam.collateralAmount
        );
}
```

Recommendation

Consider verifying that the value of equity is within the acceptable range for uint256.

Status

This issue has been resolved by the team with commit a714f51.

7. Incorrect function called

Severity: Medium	Category: Business Logic
Target: - contracts/diamond/facets/PausableFacet.sol	

Description

In the PausableFacet contract, instead of using _WithdrawPause() and _WithdrawUnpause() functions, withdrawPause()/withdrawUnpause() use _pause()/_unpause().

contracts/diamond/facets/PausableFacet.sol:L21-L31

```
function withdrawPause() external {
        // only MONITOR_ROLE can pause
        LibAccessControlEnumerable.checkRole(Constants.WITHDRAW_MONITOR_ROLE);
        _pause();
}

function withdrawUnpause() external {
        // only DEFAULT_ADMIN_ROLE can unpause
        LibAccessControlEnumerable.checkRole(Constants.DEFAULT_ADMIN_ROLE);
        _unpause();
}
```

In this case, withdrawPaused can not be set from false to true, and vice versa.

contracts/diamond/facets/VaultFacet.sol:L55

```
function withdraw(address recipient, uint256 amount) external override
whenWithdrawNotPaused nonReentrant
```

Recommendation

Consider changing _pause()/_unpause() to _WithdrawPause()/_WithdrawUnpause().

Status

This issue has been resolved by the team with commit 1ec2848.

8. Centralization risk

Severity: Medium	Category: Centralization

Target:
- contracts/diamond/facets/StrategyConfigFacet.sol

Description

In Dederi, there are several privileged roles: DEFAULT_ADMIN_ROLE, KEEP_ROLE, CLEAR_ROLE, and MONITOR_ROLE. These roles can configure some key factors, and execute some key actions.

If these roles' private keys are compromised, the hacker can modify some key factors, and execute some key actions to earn profits.

For example, set the minimum fees.

contracts/diamond/facets/StrategyConfigFacet.sol:L96-L103

```
function setFee(uint256 _minTakerFee, uint256 _minMakerFee) external {
   LibAccessControlEnumerable.checkRole(Constants.DEFAULT_ADMIN_ROLE);
   LibStrategyConfig.Layout storage l = LibStrategyConfig.layout();
   l.minTakerFee = _minTakerFee;
   l.minMakerFee = _minMakerFee;
   emit FeeSet(_minTakerFee, _minMakerFee);
}
```

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 mitigated by the team.

9. The protocol can withdraw fees from questionable unsettled PnL

Severity: Low	Category: Business Logic

Target:
- contracts/diamond/libraries/LibSettle.sol

Description

When the clear role settles positions in one strategy, contracts will calculate the realizedPnl and related unsettled balance. At the same time, the protocol can collect some protocol fees. The protocol fee amount is related to positions' size.

If there are some questionable unsettled balances, these unsettled balances will not be unlocked to the trader. However, the protocol can withdraw the protocol fees from the questionable trade. This is a little unfair for traders.

contracts/diamond/libraries/LibSettle.sol:L17-L46

```
function _handleSettle(uint256 strategyId, uint256[] memory expiringPositionId)
    internal
    returns (uint256 _totalSettlementFee)
{
    LibPositionCore.Layout storage l = LibPositionCore.layout();
    LibVault.Layout storage vl = LibVault.layout();

    int256 _totalRealizedPnl;
    for (uint256 i; i < expiringPositionId.length;) {
        int256 _realizedPnl;
        uint256 _settlementFee;
        // if option
        if (LibPosition._getType(expiringPositionId[i]) ==
StrategyTypes.AssetType.OPTION) {
            (_realizedPnl, _settlementFee) = _settleOption(strategyId,
expiringPositionId[i]);
        } else {
            (_realizedPnl, _settlementFee) = _settleFuture(strategyId,
expiringPositionId[i]);
        }
        _totalSettlementFee += _settlementFee;
        _totalRealizedPnl += _realizedPnl;
        unchecked {
            ++i;
        }
    }
    l.strategies[strategyId].realizedPnl += _totalRealizedPnl;
    l.strategies[strategyId].unsettled += Math.NonNegative(_totalRealizedPnl);
    vl.protocolFee += _totalSettlementFee;

    LibStrategy.closeStrategyIfNecessary(strategyId);
}
```

contracts/diamond/facets/VaultFacet.sol:L73-L82

```
function withdrawProtocolFee(address receiver) external {
    LibAccessControlEnumerable.checkRole(Constants.DEFAULT_ADMIN_ROLE);
    LibVault.Layout storage l = LibVault.layout();
```

```
    LibStrategyConfig.Layout storage cl = LibStrategyConfig.layout();

    uint256 protocolFee =
(l.protocolFee).convertFrom18(IERC20Metadata(cl.usdcToken).decimals());
    l.protocolFee = 0;
    (cl.usdcToken).safeTransfer(receiver, protocolFee);
    emit ProtocolFeeWithdraw(cl.usdcToken, receiver, protocolFee);
}
```

Recommendation

We recommend making use of a similar unsettle mechanism for the protocol fee.

Status

This issue has been acknowledged by the team.

10. Signature verification failed because signers own the same index

Severity: Low	Category: Business Logic

Target:
- contracts/diamond/facets/StrategyConfigFacet.sol

Description

When the keeper role updates the prices, some signers' signatures should be provided. The setPriceConfig() function can set/update valid signers.

The vulnerability is that counter i is cast from uint256 to uint8. Considering the length of signers is larger than 255. Several s_signers may share the same index.

When s_signers with the same index provide signatures to update one price, the transaction will be reverted because of duplicated signers.

contracts/diamond/facets/StrategyConfigFacet.sol:L188-L217

```
function setPriceConfig(address[] calldata signers) external {
    LibAccessControlEnumerable.checkRole(Constants.DEFAULT_ADMIN_ROLE);
    LibPriceOracle.Layout storage l = LibPriceOracle.layout();
    address[] memory signersArgs = signers;
    ......
    // add new signer addresses
    for (uint256 i; i < signersArgs.length;) {
        if (l.s_signers[signersArgs[i]].active) {
            revert RepeatedSignerAddress();
        }
        l.s_signers[signersArgs[i]] = LibPriceOracle.Signer({active: true, index:
uint8(i)});

        unchecked {
            ++i;
        }
    }
    l.s_signersList = signersArgs;

    emit PriceConfigUpdated(signers);
}
```

contracts/diamond/facets/StrategyConfigFacet.sol:L188-L217

```
function _checkSignatureOfReports(
    bytes[] calldata _reports,
    // ECDSA signatures
    bytes[] calldata _signatures
) internal view {
    ...
    LibPriceOracle.Signer memory signer;
    bool[] memory signed = new bool[](Constants.MAX_SIGNER_NUM);
    for (uint256 i; i < signLen;) {
        address signerAddress = _reportHash.recover(_signatures[i]);
        signer = l.s_signers[signerAddress];
        if (!signer.active) {
            revert InvalidSignature();
```

```
        }
        if (signed[signer.index]) {
            revert DuplicateSigner();
        }
        signed[signer.index] = true;
        unchecked {
            ++i;
        }
    }
}
```

Recommendation

Consider adding an input validation in the setPriceConfig() function if there is no need for more than 255 signers.

Status

This issue has been resolved by the team with commit a714f51.

11. An out-of-bound error may occur when verifying signatures

Severity: Low	Category: Business Logic

Target:
- contracts/diamond/libraries/LibPriceOracle.sol

Description

The signed array is used to record signed signers, assigning with length MAX_SIGNER_NUM (9). A signer's index is determined when setting the config, which could be greater than 8. If a signer with index 9 tries to sign the message, signed[signer.index] will fail because of an out-of-bound error.

contracts/diamond/libraries/LibPriceOracle.sol:L154-L184

```
function _checkSignatureOfReport(
        bytes memory _report,
        // ECDSA signatures
        bytes[] calldata _signatures
) internal view {
        LibPriceOracle.Layout storage l = LibPriceOracle.layout();
        ...

        LibPriceOracle.Signer memory signer;
        bool[] memory signed = new bool[](Constants.MAX_SIGNER_NUM);
        for (uint256 i; i < signLen;) {
        address signerAddress = _reportHash.recover(_signatures[i]);
        signer = l.s_signers[signerAddress];
        if (!signer.active) {
                revert InvalidSignature();
        }
        if (signed[signer.index]) {
                revert DuplicateSigner();
        }
        signed[signer.index] = true;
        unchecked {
                ++i;
        }
        }
}
```

Recommendation

If there won't be more than 9 signers, consider adding a check to ensure when configuring signers in the setPriceConfig() function.

Status

This issue has been resolved by the team with commit 77c8a95.

12. Lack of input validation for counterparties in adlExecute()

Severity: Low	Category: Data Validation

Target:
- contracts/diamond/libraries/LibPriceOracle.sol

Description

When one strategy enters ADL status, a clear role can execute ADL to close this strategy with some counterparties. The vulnerability is that it lacks enough input validation for counterparties. For example, it has to double-check that counterparties' positions have the same underlying, expiration time, etc. with the ADL position.

contracts/diamond/facets/StrategyADLFacet.sol:L54-L74

```
function adlExecute(
   StrategyTypes.ADLStrategyRequest memory strategy,
   StrategyTypes.ADLStrategyRequest[] memory counterparties
) external whenNotPaused {
   LibAccessControlEnumerable.checkRole(Constants.CLEAR_ROLE);
   LibPositionCore.Layout storage l = LibPositionCore.layout();

   StrategyTypes.PositionData memory positionData = l.positions[strategy.positionId];

   //check if can ADL.
   if (!LibStrategy.isADL(strategy.strategyId)) {
       revert StrategyIsNotAllowADL(strategy.strategyId);
   }

   if (positionData.assetType == StrategyTypes.AssetType.OPTION) handleOption(strategy,
counterparties, l);
   else handleFuture(strategy, counterparties, l);

   if (!LibStrategy.checkStrategyStatus(strategy.strategyId)) {
       l.strategyNFT.burn(strategy.strategyId);
   }
}
```

Recommendation

We recommend adding related input validation on-chain.

Status

This issue has been resolved by the team with commit a714f51.

2.3 Informational Findings

13. Gas optimization suggestions

Severity: Informational	Category: Gas Optimization
Target: - contracts/utils/TimestampCheck.sol	

Description

contracts/utils/TimestampCheck.sol:L10-L16

```
if (dayOfWeek < 5 || (dayOfWeek == 5 && currentTime % 1 days < 8 hours)) {
        // It's before this Friday 8:00 AM
        daysUntilNextFriday = (5 + 7 - dayOfWeek) % 7;
} else {
        // It's after this Friday 8:00 AM, calculate for next week
        daysUntilNextFriday = dayOfWeek == 5 ? 7 : (12 - dayOfWeek) % 7;
}
```

Based on the current calculation logic, only on Friday after 8 am is a special case, so the above code can be optimized to the code below.

```
if (dayOfWeek != 5 || (dayOfWeek == 5 && currentTime % 1 days < 8 hours)) {
    daysUntilNextFriday = (12 - dayOfWeek) % 7;
}
else {daysUntilNextFriday = 7;}
```

Recommendation

Consider updating the code based on the above suggestions.

Status

This issue has been resolved by the team with commit 1ec2848.

14. Inconsistency between comments and implementation

Severity: Informational	Category: Inconsistency

Target:
- contracts/diamond/facets/StrategyOpenFacet.sol

Description

In the createStrategy() function, the comment notes that the signer parameter is no longer needed, while it's still used in the current implementation.

contracts/diamond/facets/StrategyOpenFacet.sol:L35-L54

```
/**
 * @notice strategy open + close
 * @param signer No longer needed
 * @param signature maker's signed bytes
 * @param makerStrategy maker's signed struct
 * @param takerStrategy taker's struct
 */
function createStrategy(
        address signer,
        bytes memory signature,
        StrategyTypes.StrategyRequest memory makerStrategy,
        StrategyTypes.StrategyRequest memory takerStrategy
) external whenNotPaused {
        ...
        LibEIP712._checkSignatureUsed(signature);
        LibEIP712._verify(signer, signature, makerStrategy);
```

Recommendation

Consider fixing the inconsistency between comments and implementation.

Status

This issue has been resolved by the team with commit a714f51.

15. Typos

Severity: Informational	Category: Code Quality
Target: - contracts/diamond/libraries/LibStrategy.sol	

Description

contracts/diamond/libraries/LibStrategy.sol:L361

```
function _checkUderlyingSame(address _asset1, address _asset2) internal pure
```

Uderlying should be U**n**derlying.

contracts/diamond/libraries/LibStrategy.sol:L377

```
function unpdateRealizedPnl(uint256 _strategyId, int256 _realizedPnl) internal
```

u**n**pdate should be update.

Recommendation

Consider fixing the typos.

Status

This issue has been resolved by the team with commit a714f51.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit eb95636:

File	SHA-1 hash
contracts/MultiCall.sol	1fca744c9c3644c71f10a740f96608837341l9f8
contracts/utils/ConvertDecimals.sol	144b48a715a3cbc240afa1673964acdfdb858b54
contracts/utils/Constants.sol	7a2202e94021393c9c663e91131aa051e5be5838
contracts/utils/Math.sol	728a101971a3b14ec2b2ee4366ec3c304d023124
contracts/utils/TimestampCheck.sol	c8866ae6d115f93a76b59e413e36aef718068951
contracts/utils/CurrencyTransfer.sol	947bfb38ee4afba3579b53d5021fadb20d08c141
contracts/nft/StrategyNFT.sol	779012e6f9391d7ce65d456eca06d2486503bb9c
contracts/diamond/libraries/LibSettle.sol	e368e1b0344bc2f0468deee94e13b1bf41249659
contracts/diamond/libraries/LibStrategyLiquidate.sol	aca8e9b2f333c7e7d0703f928d283574b7132473
contracts/diamond/libraries/LibStrategyOpen.sol	79a3d89588d0dcc5e6a82b6c0bd75514cb43a492
contracts/diamond/libraries/LibDiamond.sol	8736533d496ca5053328e7308199b32555302119
contracts/diamond/libraries/LibCollateral.sol	4e37be836714212c37e8c267b56db30f49831546
contracts/diamond/libraries/LibAccessControlEnumerable.sol	2823345c77a307b1e92c3f41b84c8e9d66056aeb
contracts/diamond/libraries/LibStrategyMS.sol	df5a648b2fe835934ce2cf71a14abf069a619ab3
contracts/diamond/libraries/LibVault.sol	2a3ff07a1aac65b74a21ace9f39eda5e6e5f811d
contracts/diamond/libraries/LibStrategy.sol	c2f408391e9f7e65f0fc200b84ab63b94fc39044
contracts/diamond/libraries/LibSpotPriceOracle.sol	1fadad5d91d2782e4ea6011707e2fc510b809d62
contracts/diamond/libraries/LibUniTWAPOracle.sol	6bc8120d8b63c1984d6d4aa53847b60c39d78fb4
contracts/diamond/libraries/LibSettleTWAPOracle.sol	70510c35df38f6db9773bb4fee2fa77c6d5cd5a5
contracts/diamond/libraries/LibStrategyConfig.sol	5aa0c4b85b32a70aabe0e91d947e7bead9dc4609
contracts/diamond/libraries/LibStrategyADL.sol	7c4e76ccedb46b5c52623a74a71b6a2c38197543
contracts/diamond/libraries/StrategyTypes.sol	e6b7dcf38e2ea76e4d3814b0326b6ec26a13feb1
contracts/diamond/libraries/LibPosition.sol	e523efb7642693591858447401486364777730d8

contracts/diamond/libraries/LibEIP712.sol	f874386289d735e38767183074164bb003f4eac1
contracts/diamond/libraries/LibMarketPricer.sol	b7c04d68821cc5b4e1f5c6c60f7c7fbbb2f9a26e
contracts/diamond/libraries/LibPriceOracle.sol	021e60ced3f03da06db4a4922a7c7910babe4e50
contracts/diamond/libraries/LibPositionCore.sol	06f1d19d00999c29e5e95e416eee9745e1c6cb0c
contracts/diamond/StrategyManager.sol	4718483eb321da006c7cb1bb0d76afe8e22d1204
contracts/diamond/errors/GenericErrors.sol	9dd3e13a276d00dc1cf499563303d847a2288842
contracts/diamond/security/Pausable.sol	7332e6e0504681771bd07d0944aa2c5c0a091b0f
contracts/diamond/security/ReentrancyGuard.sol	8662782cbe458d80f4c1c54d272147a8c82a590c
contracts/diamond/interfaces/IStrategyLiquidate.sol	096ef8bdfafe83e51a9256752fbe27d6e8b8b0da
contracts/diamond/interfaces/IPositionCore.sol	2f2459a82629dbcdd7077ad94d2804955a80793c
contracts/diamond/interfaces/IStrategyOpen.sol	8f5fc8e54ee24d3f0c13cce08bbf0fb12e5803f7
contracts/diamond/interfaces/IDiamondLoupe.sol	cb3ca4ae767047fc0a9e0c6ce4858ce46d4d5367
contracts/diamond/interfaces/IAggregatorV3.sol	3efebe870a461786cec60300739159430ac6b921
contracts/diamond/interfaces/ISpotPriceOracle.sol	58f72ee80ffe6c2b92db25746286265af05427b2
contracts/diamond/interfaces/IStrategyConfig.sol	60083b5b42bab8e17d58b30eb8ece6414653b616
contracts/diamond/interfaces/IStrategyMS.sol	8695cdd610a6247e3efe3ef7a48cdd0a322fae9e
contracts/diamond/interfaces/IUniTWAPOracle.sol	3b4f5ef1a1a938d0b689ff6b7c06f526c771d076
contracts/diamond/interfaces/IVault.sol	72a6e3081bfd2b790e018477ccb328043aca715f
contracts/diamond/interfaces/IDiamondCut.sol	e6595d72a85e9413f42d8e0c91618d5f19925341
contracts/diamond/interfaces/IStrategySettle.sol	f114d02c40866d652636d8d35a83c7240ab5e73b
contracts/diamond/interfaces/ICollateral.sol	8433c9f0e7787ad2c7c4089558a016395139d4c5
contracts/diamond/upgradeInitializers/DiamondInit.sol	ecdc48a33f88ed51e020eb765a838f007e4cb24c
contracts/diamond/facets/AccessControlEnumerableFacet.sol	550b2587c54b39a618aff7ac9c3a55be9e059d4b
contracts/diamond/facets/UniTWAPOracleFacet.sol	5ae703390544f2fc83e9f2df0fabffbfa6677ff1
contracts/diamond/facets/ADLPriceOracleFacet.sol	659490977c0a601e77c537914f6f4ed59553bf0c
contracts/diamond/facets/DiamondCutFacet.sol	b27b164d3e8c255000bb55b376e8cf4a3e50922d
contracts/diamond/facets/StrategyOpenExFacet.sol	bcbe9fbc84f75b3abde99c20486be86cfcfd5f2c
contracts/diamond/facets/DiamondLoupeFacet.sol	3a10f5b0c6fcb79949e35c9b2ca1aed9230a0f86
contracts/diamond/facets/SettleTWAPOracleFacet.sol	e12e6782f1851cfa3f1c245f33a5b79ba1ed773e
contracts/diamond/facets/VaultFacet.sol	1b084a1eb72ce5b921a245fd3577bebaa1511a75
contracts/diamond/facets/StrategyADLFacet.sol	69558c50c02deeaa1f5325cd6df10a6531b80961

contracts/diamond/facets/PositionCoreFacet.sol	23e63ca4afb82f02d88fbecf32e7ebb91ff8762d
contracts/diamond/facets/EIP712Facet.sol	48445a109fc4e03a55eb625e0fe45b71c940c6bd
contracts/diamond/facets/SpotPriceOracleFacet.sol	5785a746ff1b2d5da1559ae9c67fdf07d8126662
contracts/diamond/facets/StrategyConfigFacet.sol	365882fa97cb54443342e856cd5bfd769483bdb8
contracts/diamond/facets/StrategyLiquidateFacet.sol	5637e6255e0c531d9c221c98a6014e98aeebddfc
contracts/diamond/facets/MarkPriceOracleFacet.sol	1c7edb8cc6a3352e3b5cc99ee36cdb587ab75e84
contracts/diamond/facets/CollateralFacet.sol	bef2cf284c5e014137a0286916780d1084df3970
contracts/diamond/facets/LiquidatePriceOracleFacet.sol	a61ee328f847d529d51877032d3eb579b9885224
contracts/diamond/facets/StrategySettleFacet.sol	8bba010453ba0c5a6f8d081385017fdfdee588b8
contracts/diamond/facets/PausableFacet.sol	36b935497cc59696b6919f340257dce68dd33325
contracts/diamond/facets/StrategyMSFacet.sol	8f704049322068298eb5e48a0b518b3bb652900c
contracts/diamond/facets/MarginOracleFacet.sol	7b22f6f3c128e437ea5115baef2220cd66be65ce
contracts/diamond/facets/StrategyOpenFacet.sol	435533b3b2c7d0717211deccbc7aabd789d2e5bb