SALUS SECURITY

DEC 2024

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	v2
Type	Solidity
Dates	Dec 17 2024
Logs	Dec 04 2024, Dec 17 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	5
Total Low-Severity issues	1
Total informational issues	1
Total	7

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	Centralization risk	Medium	Centralization	Mitigate
6	Lack of signature length check	Low	Business Logic	Resolved
7	Gas Optimization	Informational	Gas Optimization	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: - 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: - 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: - 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. Centralization risk

Severity: Medium	Category: Centralization
Target: - contracts/oracle/Oracle.sol - 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.

6. Lack of signature length check

Severity: Low	Category: Business Logic
Target: - 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

7. Gas Optimization

Severity: Informational	Category: Gas Optimization
Target: - 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.

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