ENIGMA DARK

Securing the Shadows



Security Review

Silo Finance

Silo V3

February, 2026

Contents

1. Summary
2. Engagement Overview
3. Risk Classification
4. Vulnerability Summary
5. Findings
6. Disclaimer

Summary

Enigma Dark
Enigma Dark is a web3 security firm leveraging the best talent in the space to secure all kinds of blockchain protocols and decentralized apps. Our team comprises experts who have honed their skills at some of the best auditing companies in the industry. With a proven track record as highly skilled white-hats, they bring a wealth of experience and a deep understanding of the technology and the ecosystem.

Learn more about us at enigmadark.com

Silo Finance: Silo V3
Silo Finance (Silo V3) is a decentralized, non-custodial lending protocol that uses isolated risk markets, enabling permissionless lending and borrowing while reducing contagion across assets.

The V3 release introduces a new defaulting liquidation mechanism as its primary feature, alongside the removal of same-asset borrowing and several minor fixes.

Engagement Overview

Over the course of 1.4 weeks (1 week and 2 days), beginning January 29 2026, the Enigma Dark team conducted a security review of the Silo Finance: Silo V3 project. The review was performed by two Lead Security Researchers: 0xWeiss & vnmrtz.

The following repositories were reviewed at the specified commits:

Repository	Commit
silo-finance/silo-contracts-v3	`d1827be...80ab80d`
silo-finance/silo-contracts-v3 (post-audit linter fixes)	`39d5d6e474010db85bbc49afc673e1c9fdde459a`

Scope

Modified files (existing files with changes):

File	Notes
`silo-core/contracts/Silo.sol`	Code removed only
`silo-core/contracts/SiloConfig.sol`	Code removed only
`silo-core/contracts/hooks/SiloHookV2.sol`	
`silo-core/contracts/hooks/defaulting/*`	
`silo-core/contracts/hooks/liquidation/PartialLiquidation.sol`	
`silo-core/contracts/incentives/SiloIncentivesController.sol`	
`silo-core/contracts/incentives/SiloIncentivesControllerFactory.sol`	

New files:

File
`silo-core/contracts/hooks/defaulting/*`

Risk Classification

Severity	Description
Critical	Vulnerabilities that lead to a loss of a significant portion of funds of the system.
High	Exploitable, causing loss or manipulation of assets or data.
Medium	Risk of future exploits that may or may not impact the smart contract execution.
Low	Minor code errors that may or may not impact the smart contract execution.
Informational	Non-critical observations or suggestions for improving code quality, readability, or best practices.

Vulnerability Summary

Severity	Count	Fixed	Acknowledged
Critical	0	0	0
High	0	0	0
Medium	0	0	0
Low	0	0	0
Informational	4	1	3

Findings

Index	Issue Title	Status
I-01	Looping through all incentive programs could DOS calls to `afterTokenTransfer`	Acknowledged
I-02	Missing event emission on share token fallback during liquidation	Acknowledged
I-03	Minor improvements to the code and comments	Acknowledged
I-04	Missing return data length check inside try-catch block	Fixed

Detailed Findings

High Risk

No issues found.

Medium Risk

No issues found.

Low Risk

No issues found.

Informational

I-01 - Looping through all incentive programs could DOS calls to `afterTokenTransfer`

Severity: Informational

Context:

- BaseIncentivesController.sol#L48
- SiloIncentivesController.sol#L90

Technical Details:

The `_createIncentiveProgram` function does not enforce a cap on the number of incentive programs that can be created:

```
  function _createIncentiveProgram(
        bytes32 _programId,
        DistributionTypes.IncentivesProgramCreationInput memory
_incentivesProgramInput
    ) internal virtual {
        require(_incentivesProgramInput.rewardToken != address(0),
InvalidRewardToken());
        require(_incentivesProgramIds.add(_programId),
IncentivesProgramAlreadyExists());
```

This becomes problematic because `afterTokenTransfer` iterates over every active incentive program and invokes `_handleAction` for both the sender and recipient:

```
  // iterate over incentives programs
>>       for (uint256 i = 0; i < numberOfPrograms; i++) {
            bytes32 programId = _incentivesProgramIds.at(i);

            if (_sender != address(0)) {
                _handleAction(programId, _sender, _totalSupply,
_senderBalance);
            }

            if (_recipient != address(0)) {
                _handleAction(programId, _recipient, _totalSupply,
_recipientBalance);
            }
        }
```

Impact:

If a sufficiently large number of incentive programs accumulate, the unbounded loop in `afterTokenTransfer` can exceed the block gas limit, effectively bricking all token transfers, including deposits, withdrawals, and liquidations, that rely on this hook.

Recommendation:

Enforce a hard cap on the number of active incentive programs at creation time to prevent a transfer-hook DoS.

Developer Response:

Acknowledged. We aware of that, we can always remove some programs.

I-02 - Missing event emission on share token fallback during liquidation

Severity: Informational

Context:

- PartialLiquidation.sol#L236

Technical Details:

Inside the `liquidationCall` function when the call to `ISilo(_silo).redeem()` reverts with `ReturnZeroAssets` error, the contract catches the error and it switches the settlement mode, transferring share tokens directly to the liquidator instead of underlying collateral assets.

No dedicated event is emitted when this fallback path is taken. The outer liquidation event does still fire with `withdrawCollateral == 0`, which implicitly signals that the settlement occurred in share tokens rather than assets.

While the outer liquidation event implicitly signals a share-token settlement when withdrawCollateral is zero, an explicit event would improve observability and reduce the need for off-chain systems to infer settlement type indirectly.

Recommendation:

Emit a dedicated event when the redeem fallback is triggered:

```
        } catch (bytes memory e) {
            if (_isToAssetsConvertionError(e)) {
                IERC20(_shareToken).transfer(msg.sender, _shares);
+                emit RedeemFallbackToShares(_silo, _shareToken, msg.sender,
_shares);
            } else {
                RevertLib.revertBytes(e, string(""));
            }
        }
```

Developer Response:

Acknowledged. This is a sufficiently rare edge case that we prefer not to introduce additional code changes at this time.

I-03 - Minor improvements to the code and comments

Severity: Informational

Context: See below.

Technical Details/Recommendations:

1. Whitelist.sol#L18: The `onlyAllowed` modifier in the `Whitelist` contract is unused and can be removed.
2. SiloHookV2.sol#L6C53-L6C58: Unused import of `ISilo` in `SiloHookV2`. Consider removing it.
3. PartialLiquidation.sol#L11C61-L11C74: Unused import of `IHookReceiver` in `PartialLiquidation`. Consider removing it.
4. PartialLiquidationLib.sol#L28: Typo in comment, "wai" should be "wei".
5. DefaultingRepayLib.sol#L44-L45: `DefaultingRepayLib.actionsRepay`, interest is accrued for the debt silo redundantly, `_fetchConfigs` already performs this accrual earlier in the same transaction.
6. PartialLiquidation.sol#L190, PartialLiquidationByDefaulting.sol#L288: Since the same silo can no longer act as both debt and collateral, the `collateralConfig.silo != debtConfig.silo` guard is always true and could be removed.

Developer Response:

1. Acknowledged.
2. Fixed at commit `2ad3aec`.
3. Fixed at commit `2ad3aec`.
4. Acknowledged.
5. Acknowledged.
6. Acknowledged.

I-04 - Missing return data length check inside try-catch block

Severity: Informational

Context:

- PartialLiquidation.sol#L245-L246

Technical Details:

The `_tryRedeem` function inside the `PartialLiquidation` contract uses a try-catch pattern when calling the `redeem()` function:

```
function _tryRedeem(
        address _silo,
        address _shareToken,
        uint256 _shares,
        ISilo.CollateralType _collateralType
    ) internal returns (uint256 withdrawCollateral) {
        if (_shares == 0) return 0;

        try ISilo(_silo).redeem({
            _shares: _shares,
            _receiver: msg.sender,
            _owner: address(this),
            _collateralType: _collateralType
        }) returns (uint256 assets) {
            withdrawCollateral = assets;
        } catch (bytes memory e) {
            if (_isToAssetsConvertionError(e)) {
                IERC20(_shareToken).transfer(msg.sender, _shares);
            } else {
                RevertLib.revertBytes(e, string(""));
            }
        }
    }
```

Within the catch block, `_isToAssetsConvertionError` is called to determine whether the revert was caused by a zero-asset conversion. However, the function does not validate that the error data is exactly 4 bytes before comparing it against the selector:

```
    /// @dev this method detects if error is caused by unable to convert
shares to assets eg 999 shares => 0 assets
    function _isToAssetsConvertionError(bytes memory _error) internal pure
returns (bool) {
        return bytes4(_error) == ISilo.ReturnZeroAssets.selector;
    }
```

Recommendation:

Add an explicit length check to ensure the error data is exactly 4 bytes before performing the selector comparison:

```
function _isToAssetsConvertionError(bytes memory _error) internal pure
returns (bool) {
+     if (_error.length != 4) return false;
        return bytes4(_error) == ISilo.ReturnZeroAssets.selector;
    }
```

Developer Response:

Fixed at commit `bec7f26`.

Disclaimer

This report does not endorse or critique any specific project or team. It does not assess the economic value or viability of any product or asset developed by parties engaging Enigma Dark for security assessments. We do not provide warranties regarding the bug-free nature of analyzed technology or make judgments on its business model, proprietors, or legal compliance.

This report is not intended for investment decisions or project participation guidance. Enigma Dark aims to improve code quality and mitigate risks associated with blockchain technology and cryptographic tokens through rigorous assessments.

Blockchain technology and cryptographic assets inherently involve significant risks. Each entity is responsible for conducting their own due diligence and maintaining security measures. Our assessments aim to reduce vulnerabilities but do not guarantee the security or functionality of the technologies analyzed.

This security engagement does not guarantee against a hack. It is a review of the codebase during a specific period of time. Enigma Dark makes no warranties regarding the security of the code and does not warrant that the code is free from defects. By deploying or using the code, the project and users of the contracts agree to use the code at their own risk. Any modifications to the code will require a new security review.