CANTINA

Silo Finance Dynamic Kink Model

Security Review

Cantina Managed review by:

Rikard Hjort, Lead Security Researcher
Kankodu, Security Researcher

October 3, 2025

Contents

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings are a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

The Silo Protocol is a non-custodial lending primitive that creates programmable risk-isolated markets known as silos.

From Sep 23rd to Sep 30th the Cantina team conducted a review of silo-contracts-v2 on commit hash 499f0ce6. The team identified a total of **4** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	1	1	0
Medium Risk	1	1	0
Low Risk	0	0	0
Gas Optimizations	0	0	0
Informational	2	2	0
Total	**4**	**4**	**0**

The Cantina Managed team reviewed Silo Finance's `silo-contracts-v2` holistically on commit hash f12498e3 and concluded that all findings were addressed and no new vulnerabilities were identified.

3 Findings

3.1 High Risk

3.1.1 Missing access control on `getCompoundInterestRateAndUpdate()`

Severity: High Risk

Context: DynamicKinkModel.sol#L126

Description: As per the documentation, `getCompoundInterestRateAndUpdate()` should only be callable by its Silo. However, this is not enforced. In addition, the function accepts any debt/collateral parameters given by the caller. The function updates the `k` value. In effect, this means that anyone can modify the `k` value arbitrarily. The Silo will still update it according to current utilization, but the dynamic model is based on moving the `k` value up and down from its current value to optimize utilization, and with the current bug, the initial `k` is almost infinitely controllable by an attacker.

Impact Explanation: The interest rate is a central part of the correct functioning of the protocol and the main feature of the dynamic kink model, which is the scope of this audit. This bug undermines the model and gives anyone the ability to move the derivative for interest rates according to their own best interest. `k` can be set to `0` (by giving invalid inputs, e.g. ones that don't fit in a `uint128`) or any value between `kmin` and `kmax`.

Likelihood Explanation: Anyone can call the function at any time with no privileges.

Proof of Concept: The following test can be used to experiment with possible manipulations:

```
function test_getCompoundInterestRateAndUpdate_nonSilo(uint ca, uint da, uint irt) public {
    irm = DynamicKinkModel(
        address(
            FACTORY.create(_defaultConfig(), _defaultImmutableArgs(), address(this), address(this), bytes32(0))
        )
    );

    vm.warp(667222222);

    (IDynamicKinkModel.ModelState memory stateBefore,,) = irm.getModelStateAndConfig({_usePending: false});

    // Putdome: Set `k` to 0: give any parameter higher than int256.max
    // Outcome: Set `k` to `kmin`: valid params except timestamp > block.timestamp (causes revert, try-catch sets
    ↪  to kmin)
    // Outcome: Set `k` to `kmax`: set a _detAssets > _collateralAssets
    address randomUser = makeAddr("RandomUser");
    vm.prank(randomUser);
    irm.getCompoundInterestRateAndUpdate({
        _collateralAssets: ca,
        _debtAssets: da,
        _interestRateTimestamp: irt
    });
    (IDynamicKinkModel.ModelState memory stateMiddle,,) = irm.getModelStateAndConfig({_usePending: false});

    // Silo calling
    irm.getCompoundInterestRateAndUpdate({
        _collateralAssets: 44500000000000000000000000,
        _debtAssets: 17924514248008922230752344385138042 26,
        _interestRateTimestamp: block.timestamp - 10000
    });

    (IDynamicKinkModel.ModelState memory stateAfter,,) = irm.getModelStateAndConfig({_usePending: false});

    console2.log("k before %s", stateBefore.k);
    console2.log("k middle %s", stateMiddle.k);
    console2.log("k after  %s", stateAfter.k);
    // Use any asserts here to check that the desired manipulation has been met.
    // assertFalse(3170980000 == stateAfter.k || 1585490000 == stateAfter.k, "should be max k");
    assertEq(stateAfter.silo, address(this), "silo should be the same");
}
```

Recommendation: Only allow the Silo to call the update function. This can be done by inserting a check at the beginning of the function:

```
  function getCompoundInterestRateAndUpdate(
      uint256 _collateralAssets,
```

```
      uint256 _debtAssets,
      uint256 _interestRateTimestamp
  )
      external
      virtual
      returns (uint256 rcomp)
  {
+     require(msg.sender == modelState.silo, InvalidSilo());
      (rcomp, modelState.k) = _getCompoundInterestRate(CompoundInterestRateArgs({
```

Alternatively, since `_getCompoundInterestRate()` checks that the `.silo` argument corresponds to `modelState.silo` with `require(_silo == state.silo, InvalidSilo())`, passing `msg.sender` as `silo` would also solve the issue:

```
  function getCompoundInterestRateAndUpdate(
      uint256 _collateralAssets,
      uint256 _debtAssets,
      uint256 _interestRateTimestamp
  )
      external
      virtual
      returns (uint256 rcomp)
  {
      (rcomp, modelState.k) = _getCompoundInterestRate(CompoundInterestRateArgs({
-         silo: modelState.silo,
+         silo: msg.sender,
          collateralAssets: _collateralAssets,
          debtAssets: _debtAssets,
          interestRateTimestamp: _interestRateTimestamp,
          blockTimestamp: block.timestamp,
          usePending: false
      }));
  }
```

Silo Finance: Fixed in commit 3608d589.

Cantina Managed: Fix verified.

3.2 Medium Risk

3.2.1 New configuration immediately updates `k` without regard to timelock

Severity: Medium Risk

Context: DynamicKinkModel.sol#L307, DynamicKinkModel.sol#L370, DynamicKinkModel.sol#L404-L405, DynamicKinkModel.sol#L416

Description: When updating a configuration, the owner can specify a timelock, indicating the time when the new configuration should go into effect. However, while setting the *pending* update, the *current* slope is updated to `kmin` of the new configuration. This means that `k` can be changed prematurely, which will immediately start impacting interest rate calculations.

Proof of Concept: `k` has become high due to heavy utilization. The owner sets a new config that is supposed to activate in one week -- say for argument's sake that the config is largely identical and only updates `u1`. For whatever reason, the owner cancel the update. This should be a no-op, but `k` has now been reset to `kmin` and the interest rate will now be perpetually lower than the interest rate model is designed for.

Recommendation: There are a few ways to address this:

Use a two-step config update model where the new pending model needs to be activated through a transaction (this can be done by anyone, not just owner) when it is ready, and otherwise skip using `activateConfigAt` to figure out which config is active. When activating the new config, update the `k` value.

Make the `k` pending, and use a pending `k` the same way you use a pending config, and take pains to ensure the correct `k` is being used at all times.

Silo Finance: Fixed in commit 2e1fb286.

Cantina Managed: Fix verified.

3.3 Informational

3.3.1 Unnecessary type casts

Severity: Informational

Context: DynamicKinkModelFactory.sol#L81, DynamicKinkModelFactory.sol#L155-L159, DynamicKinkModel.sol#L46-L47

Description: Internally to the factory, the `IRM` is a dynamic kink model and can be treated as such everywhere. If `create()` must return an `IInterestRateModel` then that cast can be performed at when returning, keeping the abstraction clean:

```
  diff --git a/silo-core/contracts/interestRateModel/kink/DynamicKinkModelFactory.sol
  ↪  b/silo-core/contracts/interestRateModel/kink/DynamicKinkModelFactory.sol
  index 6aba9de0..4cdea429 100644
-  -- a/silo-core/contracts/interestRateModel/kink/DynamicKinkModelFactory.sol
+  ++ b/silo-core/contracts/interestRateModel/kink/DynamicKinkModelFactory.sol
  @@ -43,7 +43,7 @@ contract DynamicKinkModelFactory is Create2Factory, IDynamicKinkModelFactory {
           virtual
           returns (IInterestRateModel irm)
       {
-          return _create(_config, _immutableArgs, _initialOwner, _silo, _externalSalt);
+          return IInterestRateModel(address(_create(_config, _immutableArgs, _initialOwner, _silo,
↪  _externalSalt)));
       }

       /// @inheritdoc IDynamicKinkModelFactory
  @@ -146,17 +146,17 @@ contract DynamicKinkModelFactory is Create2Factory, IDynamicKinkModelFactory {
       )
           internal
           virtual
-          returns (IInterestRateModel irm)
+          returns (DynamicKinkModel irm)
       {
           IRM.verifyConfig(_config);

           bytes32 salt = _salt(_externalSalt);

-          irm = IInterestRateModel(Clones.cloneDeterministic(address(IRM), salt));
-          IDynamicKinkModel(address(irm)).initialize(_config, _immutableArgs, _initialOwner, _silo);
+          irm = DynamicKinkModel(Clones.cloneDeterministic(address(IRM), salt));
+          irm.initialize(_config, _immutableArgs, _initialOwner, _silo);

           createdByFactory[address(irm)] = true;
-          emit NewDynamicKinkModel(IDynamicKinkModel(address(irm)));
+          emit NewDynamicKinkModel(irm);
       }

       function _castConfig(IDynamicKinkModel.UserFriendlyConfig calldata _default)
```

Silo Finance: Fixed in commit 6faf93c7.

Cantina Managed: Fix verified.

3.3.2 Returning `(0, 0)` is an imperfect form of error reporting

Severity: Informational

Context: DynamicKinkModel.sol#L409-L411

Description: In cases where the arguments to `_getCompoundInterestRate()` would overflow when cast from `uint256` to `int256`, the function returns `(0, 0)` to represent `(rcomp, k)`. Both these variables can get the value 0 under normal circumstances, hence this is not a foolproof way to report the error. As seen in the finding "Missing access control on `getCompoundInterestRateAndUpdate()`" this results in an invariant-breaking state update. An external caller could only get `rcomp` through `getCompoundInterestRate()` or `getPendingCompoundInterestRate()`. Since 0 a fairly normal value for `rcomp` there is no clear way for the caller to observe the overflow happening. Hence, it is on the caller to be aware of the exact implementation of the function and pre-emptively check for these overflows.

Recommendation: Return an error flag, or offer a reverting version of the external functions.

Status: This is a known issue and is implemented this way for compatibility with existing interfaces, and thus will not be fixed. Instead the returned value will be `(0, kmin)`, a valid return value.

Silo Finance: Fixed in commit f12498e3.

Cantina Managed: Fix verified.