Superform

Security Review

Ethan Bennett Darklinear Solutions

Contents

About Darklinear Solutions

Darklinear Solutions leverages years of expertise in smart contract and software engineering to provide unrivaled security reviews for blockchain applications. Learn more at darklinear.com.

Introduction

Superform is cross-chain marketplace for yield. This review consists of issues discovered during the course of Cantina's Superform contest in December 2023. It does not represent a full and exhaustive audit of the protocol.

The findings described below are classified according to Cantina's standards.

Findings Summary

Classification	Finding
Medium risk (M-1)	Insufficient validation of `txData` risks loss of funds
Medium risk (M-2)	`v.deposit` and `v.redeem` can silently fail
Low risk (L-1)	Upgradable vault admin can attack Superform users by changing the implementation of any vault underlying an existing superform
Informational (I-1)	Type `uint8` for `bridgeId` may be too low

Findings

Insufficient validation of `txData` risks loss of funds

Severity: Medium risk

Context: DataTypes.sol, BridgeValidator.sol#L49, BaseRouterImplementation.sol

Description: When a user initiates a deposit or withdrawal, they pass in two amounts: one is explicitly defined in the `SFData`, and another is nested deeper, encoded in the `liqRequest.txData`. While the former is used to interact with the vault underlying a superform, all subsequent logic (for example, sending those funds along to a different chain) references the latter.

This is useful in accounting for slippage (and maybe underlying fee-on-transfer tokens), but the lack of any validation of the `liqRequest.txData` `amount` in the deposit or withdrawal flows also makes it prone to costly mistakes.

There are some cases when a discrepency would eventually be caught in `_updateTxData`, but its validation does not happen early enough to prevent such a mistake for many users.

Note that confusion here is even more likely considering there is also a third, unrelated amount in `liqRequest`: `nativeAmount`, which is used as the `msg.value` sent with the `txData`. The cases when an accidental `0` in the encoded `txData` would not be caught are exactly when this `nativeAmount` should be `0`.

Proof of concept: This test illustrates a scenario where a user attempts to withdraw 100 Dai using a superform, but accidentally inputs a `0` in the encoded `liqRequest.txData`. The transaction succeeds, and the user burns all their superpositions, but the amount sent along to the bridge is `0`. The user will receive nothing, and they will not be able to reattempt the withdrawal.

Recommendation: The slippage check between the two values that occurs in `_updateTxData` should happen earlier, ideally in the router, so that it can catch problematic variance between the two in all cases.

`v.deposit` and `v.redeem` can silently fail

Severity: Medium risk

Context: ERC4626FormImplementation.sol, ERC4626TimelockForm.sol

Description: When a user calls any variation of deposit or withdraw from the SuperformRouter, the contracts will eventually deposit or redeem the specified amount of tokens from the superform's underlying vault. When the form implementation calls `v.redeem`, it checks that the number returned from the external call is greater than (or equal to) the amount included in the `txData` object supplied by the user — the problem is, it does not actually check that any tokens were received by the superform. For `v.deposit`, there is no validation of `dstAmount` of any kind.

Since it is not safe to assume anything about how an external vault is designed, it is possible that a flawed implementation could silently fail but still return a valid integer. A malicious vault could also intentionally return a value without sending any tokens back.

This is an issue even for cross-chain withdrawals that validate messages from multiple AMBs, because the `dstValue` returned from deposit or redeem is never referenced again in either case (except for the above-mentioned check in the withdrawal flow).

Proof of concept: This test simulates a vault that silently fails to send its underlying assets back to the user calling `redeem`, despite returning an integer that implies it succeeded. Since the test user calls `singleDirectSingleVaultWithdraw`, they should receive Dai directly from the vault contract upon redemption. They do not receive the Dai, but they do burn their superpositions, permanently losing access to the funds they were attempting to withdraw.

Recommendation: Each form implementation should check that the receiver address has received the expected amount of the expected token — either the asset or the vault share token, depending on the context — after executing `v.deposit` or `v.redeem`.

Upgradable vault admin can attack Superform users by changing the implementation of any vault underlying an existing superform

Severity: Low risk

Context: BaseForm.sol, SuperformRouter.sol

Description: Because there is no explicit handling of upgradable vaults in any of the existing form implementations, any vault could update its logic to harm Superform users at any time.

Although this should be a consideration for any user of an ERC4626 vault whether or not the Superform protocol is involved, Superform will connect users with risky vaults and abstract the implementations away from them, making it far easier for users to call updated contracts without realizing it. Consequently, it would make sense for Superform to prevent this kind of attack when it can.

Proof of concept: This test illustrates a scenario where a user successfully deposits into a vault by way of its superform. The vault admin then updates the vault's implementation, so when the user tries to make another deposit, their Dai is now sent straight to the attacker — despite calling the same superform in exactly the same way.

Note that this is not meant to be a realistic design for an upgradable vault, but to simulate a subset of its behavior from the perspective of Superform users. Likewise, a malicious vault implementation could execute higher-impact attacks, such as, for example, draining the entire contract after letting deposits accrue. This is, however, an immediately demonstrable impact in the context of a unit test.

Recommendation: Proxy detection may be difficult on-chain, but it is a perfect role for an additional keeper. Assuming such off-chain keepers exist, the contracts would benefit from:

- A `bool` for each superform specifying whether or not the underlying vault is upgradable (or is a proxy)
  - This can be set by the proxy keeper
- The ability to pause individual superforms
  - If centralization risk is a concern, this functionality could be restricted to superforms flagged by the proxy keeper
- An adapter for superforms flagged by the proxy keeper that checks for implementation changes in the underlying vault before depositing or withdrawing
  - If it finds changes, it could temporarily pause the superform
  - The protocol for unpausing depends on how opinionated the design aims to be: it could require admin review, it could require that a quorum of unique addresses request unpausing, or it could even bypass pausing entirely and simply trigger a warning for users who interact with the vault
  - See MakerDAO's GemJoin6, a similar adapter for upgradable ERC20 tokens
- A check for the proxy `bool` when a specific superform is first referenced in the router, and a call to the proxy adapter if necessary, before continuing with deposit or withdrawal logic

---

Type `uint8` for `bridgeId` may be too low

Severity: Informational

Description: Assuming that the Superform protocol is meant to operate forever once it has been deployed, it seems plausible that it could one day exceed the maximum of 250 supported bridges (implicitly enforced by storing `bridgeId` as a `uint8`). There is no mechanism for reusing a deprecated `bridgeId`, so the protocol would not necessarily need to support 250 bridges concurrently at any single time to hit this limit.

It is difficult to predict the rate of growth of EVM-based layer two networks, but it would be reasonable to account for the possibility that it may become very high in the coming years. With this growth would come the death of some bridges, the birth of many new ones, and potentially unexpected permutations, like chain-specific or even application-specific bridges.

Recommendation: Considering the low cost of increasing the storage capacity of this variable contrasted with the high impact to the protocol if this limit is ever reached, it is likely worth making `bridgeId` a larger `uint`.