Security Audit

December 2023

Superform

Audited by
Ethan Bennett

Contents

About Darklinear Solutions

Darklinear Solutions provides unrivaled security for blockchain applications, from the bytecode to the browser. With years of experience in smart contract development and traditional software engineering, we find the bugs that others miss. 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 competitive audit for Superform 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

In addition to the two listed findings below, two low severity vulnerabilities were submitted. However, since Cantina only published the high and medium severity issues, the low severity findings have also been omitted from this report.

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

Findings

M-1: 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.

M-2: `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`.