Superform core
Security Review

Review by:
Cergyk, Security Researcher

Akshay srivastav, Associate Security Researcher
Windhustler, Associate Security Researcher

September 17, 2024

Contents

1 Introduction

1.1 Disclaimer

A security review is 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 the review 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 a security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.2 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.2.1 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 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

Superform is a non-custodial yield marketplace. It allows other DeFi protocols to permissionlessly list yield opportunities and users to then access them from any EVM chain.

From Jun 30th to Jul 7th the researchers conducted a review of superform-core on commit hash feb9cdcf. The team identified a total of **17** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 2
- Medium Risk: 4
- Low Risk: 6
- Gas Optimizations: 1
- Informational: 4

The current report corresponds to a preliminary review.

3 Findings

3.1 High Risk

3.1.1 XChain deposits with long `payloadBody` can be bricked and unreplayable on destination chain

Severity: High Risk

Context: BaseStateRegistry.sol#L125

Description: When a payload is received via AMB on destination chain, the content of the whole `payloadBody` is written to storage, which is very gas costly since it is a variable length data structure (`bytes`) and is only constrained by the message size limits of the AMB.

A quick calculation, using a 20M block gas limit and 20,000 gas units per word write yields that a `payloadBody` of 32,000 bytes cannot be supported, and that's ignoring any other costs of the transaction.

> Please note that `Scroll` which is intended to be supported in the future by superform has a 10M block size limit

A good on-chain example showcasing this phenomenon is the following transaction, inducing an approx. 300,000 gas cost for a payload size of only 19 words.

Recommendation: Please consider only committing the proof of the `payloadBody` on chain, and pass it in calldata during `updateXPayload` and `processPayload`

Superform: This is going to be acknowledged as we already addressed it in creating a vault limit per transaction (currently set to 5).

3.1.2 `ERC5115Form::_directDepositIntoVault` rewards can be stolen when reward token is also input token

Severity: High Risk

Context: ERC5115Form.sol#L48

Description: When a StandardYield/ERC5115 contract accepts an input token which is also a reward, the rewards denominated in that token can be stolen from `ERC5115Form`. The attacker can use a cross-contract reentrancy to inflate his direct deposit during a same chain swap (using 1inch for example) by the amount of unclaimed rewards.

Some prerequisites are needed for this to happen: the ERC5115toERC4626Wrapper contract would need to wrap a SY which accepts a reward token as an input token (example here: PendleCurveUsdd3CrvSY.sol#L129)

- Setup
  - A ERC5115Form is created, with a `ERC5115toERC4626Wrapper` vault which has `PendleCurveUsdd3CrvSY.sol` as underlying vault.
    - tokenIn => USDD
    - tokenOut => USDD
- Scenario
  - Alice uses a direct deposit by specifying:
    - `liqData.token` => USDC
    - `liqData.bridgeId` => 1inch Aggregator
    - `liqData.txData` => swap using custom Alice controlled `executor`

During the local swap, the executor controlled by Alice correctly swaps `USDC` to `USDD` but also calls on `PendleCurveUsdd3CrvSY::claimRewards` for the ERC5115Form address (since that action is permissionless).

As a result the direct deposit amount of Alice will be inflated by the pending rewards for the Form denominated in USDD, and Alice would steal those rewards from the protocol.

Recommendation: Please ensure claiming of rewards from underlying form before processing withdrawal, for these balances to be accounted for when processing deposit swaps.

Superform: Fixed in commit a72084c3.

3.2 Medium Risk

3.2.1 `multiWithdrawals` which has first leg consuming all forwarded gas, will be bricked with high probability

Severity: Medium Risk

Context: CoreStateRegistry.sol#L727-L757

Description: The maximum number of vaults from which can be withdrawn simultaneously during a multiWithdrawal is 5 to prevent the transaction to run into an out of gas exception and be unable to execute.

However when the first vault liquidity data is applied, if it consumes all available gas, the subsequent liquidity data processing will reduce the gas available for `_processAck` to zero with high probability and make the transaction revert. As a result the user making the withdrawal will not be able to recover neither his funds neither his `SuperPosition`.

- CoreStateRegistry.sol#L727-L757:

```
for (uint256 i; i < len; ++i) {
    // @dev validates if superformId exists on factory
    if (!ISuperformFactory(superformFactory).isSuperform(multiVaultData.superformIds[i])) {
        revert Error.SUPERFORM_ID_NONEXISTENT();
    }

    /// @dev Store destination payloadId_ & index in extraFormData (tbd: 1-step flow doesnt need this)
    try IBaseForm(_getSuperform(multiVaultData.superformIds[i])).xChainWithdrawFromVault(
        InitSingleVaultData({
            payloadId: multiVaultData.payloadId,
            superformId: multiVaultData.superformIds[i],
            amount: multiVaultData.amounts[i],
            outputAmount: multiVaultData.outputAmounts[i],
            maxSlippage: multiVaultData.maxSlippages[i],
            liqData: multiVaultData.liqData[i],
            hasDstSwap: false,
            retain4626: false,
            receiverAddress: multiVaultData.receiverAddress,
            extraFormData: abi.encode(payloadId_, i)
        }),
        srcSender_,
        srcChainId_
    ) {
        /// @dev marks the indexes that don't require a callback re-mint of shares (successful
        /// withdraws)
        multiVaultData.amounts[i] = 0;
    } catch {
        /// @dev detect if there is at least one failed withdraw
        errors = true;
    }
}
```

- The first call to `IBaseForm.xChainWithdrawFromVault` would consume `63/64` (due to EIP-150) of `CoreStateRegistry` call context gas.
- The second call would consume `63/64` of the remaining `1/64`, since these 2 calls are in the same call frame.
- The third call would again consume `63/64` of remaining and so on.

  Please note that second and third call need not necessarily be malicious, since 1/64 of gas left is already a relatively small value and can legitimately end up used by a normal swap

As a result one can expect a factor of (1/64)**5 be applied on the gas available to `_processAck` in the worst case. This means approx `1e-9`, which means there is virtually no gas left to execute `_processAck` and the transaction reverts with OOG.

Please note as a concrete example, that a malicious executor can be passed during a 1inch aggregator swap to consume all forwarded gas

Recommendation: Please consider using an external call to call the loop for iterating over the liquidity data (which means only 63/64 gas will be forwarded for this, leaving 1/64 to call `_processAck`).

Superform: This will be acknowledged as we cannot perform changes to Core State Registry. Also, we can run simulations to try to prevent this on the app side.

3.2.2 `ERC5115Form::withdraw`/`deposit` strict checks can cause stETH (and other rebasing tokens) to revert

Severity: Medium Risk

Context: ERC5115Form.sol#L314, ERC5115Form.sol#L566

Description: Due to a rounding issue with share calculation during the transfer of stEth tokens (see lido-dao/issues/442), the balance difference of destination address before/after transfer of `amount` is not exactly `amount` but can be off by at most 2 wei.

The logic inside xChain deposits (direct deposits should not be affected) and withdrawals may revert in the case the amount transferred is not exact.

- xChain deposit. Logic inside `_xChainDepositIntoVault` (ERC5115Form.sol#L314-L320):

```
/// @dev pulling from sender, to auto-send tokens back in case of failed deposits / reverts
IERC20(vaultTokenIn).safeTransferFrom(msg.sender, address(this), singleVaultData_.amount);

/// @dev allowance is modified inside of the IERC20.transferFrom() call
IERC20(vaultTokenIn).safeIncreaseAllowance(vaultLoc, singleVaultData_.amount);

/// @dev deposit vaultTokenIn for shares and add extra validation check to ensure intended ERC5115
↪  behavior
shares = _depositAndValidate(singleVaultData_, singleVaultData_.amount);
```

  `singleVaultData_.amount` may not be available to deposit, after the transfer from.

- Withdrawal (ERC5115Form.sol#L566):

```
if (
    (assetsBalanceAfter - assetsBalanceBefore != assets)
        || (
            ENTIRE_SLIPPAGE * assets
                < singleVaultData_.outputAmount * (ENTIRE_SLIPPAGE - singleVaultData_.maxSlippage)
        )
) {
    revert Error.VAULT_IMPLEMENTATION_FAILED();
}
```

The strict equality check will revert if the exact amount is not transferred.

Recommendation:

1. Deposit case: Please consider using balance difference around transfer, as done during `directDeposit`.
2. Withdrawal case: Please consider using a tolerance constant EPSILON (can be set to 10 wei for example), and tolerate that difference from exact amount.

Superform: Deposit case fixed in commit 02ee7426; withdrawal case fixed in commit 6527830b.

3.2.3 `DeBridgeValidator::validateTxData` missing destination chain equals liquidity destination chain check

Severity: Medium Risk

Context: DeBridgeForwarderValidator.sol#L95, DeBridgeValidator.sol#L64

Description: In the case of a cross-chain deposit, an arbitrary message is sent through the AMB alongside liquidity and to the same chain. This is checked in some of the `IBridgeValidator` implementations but not all, and could cause lost funds if there is a user mistake setting these destination chains to be different.

Recommendation: Add a check that `args_.dstChainId == args.liqDstChainId` in the deposit case in `SocketValidator.sol` and `DeBridgeValidator.sol`:

- `DebridgeValidator.sol#L72`:

```
  if (args_.deposit) {
      if (args_.srcChainId == args_.dstChainId) {
          revert Error.INVALID_ACTION();
      }

+     if (args_.dstChainId != args_.liqDstChainId) {
+         revert Error.INVALID_ACTION();
+     }
```

Superform: Fixed in commit 2766fac0.

3.2.4 `DeBridgeValidator::validateTxData` wrong destination chain for Debridge/DLN order rescuer

Severity: Medium Risk

Context: DeBridgeForwarderValidator.sol#L83, DeBridgeValidator.sol#L58

Description: When liquidity is bridged via DLN, an order is created on destination chain. If the order is never taken or if it needs to be cancelled for other reasons, a superform controlled address (role `CORE_STATE_REGISTRY_RESCUER_ROLE`) is set to be able to cancel the order.

In `DeBridgeValidator::validateTxData` it is enforced for the order `orderAuthorityAddressDst` to be the `CORE_STATE_REGISTRY_RESCUER_ROLE` but on the wrong chain (see DeBridgeValidator.sol#L58).

Indeed in the case of withdrawals, `dstChainId`, which is the chain on which the `ack` must be received, can be different from `liqDstChainId` which is the chain on which the order is created.

In that case funds may end up unavailable since nobody has the authority to cancel the order on the destination chain.

Recommendation: Change `dstChainId` to `liqDstChainId` (in `DeBridgeValidator` and `DebridgeForwarderValidator`):

```
  if (
-     superRegistry.getAddressByChainId(keccak256("CORE_STATE_REGISTRY_RESCUER_ROLE"), args_.dstChainId)
+     superRegistry.getAddressByChainId(keccak256("CORE_STATE_REGISTRY_RESCUER_ROLE"), args_.liqDstChainId)
      != _castToAddress(deBridgeQuote.orderAuthorityAddressDst)
  ) revert DeBridgeError.INVALID_DEBRIDGE_AUTHORITY();
```

Superform: Fixed in commit a554840b.

3.3 Low Risk

3.3.1 `ERC5115To4626WrapperFactory::createSuperformForWrapper` does not update wrapper metadata with superform implementation

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: `ERC5115To4626WrapperFactory` enables to create a `ERC5115To4626Wrapper` and attach a superform to it later. In that case the metadata mapping in the factory has a field `formImplementation` which should hold the address of the implementation used to instantiate the Form.

However it is not checked when calling `ERC5115To4626WrapperFactory::createSuperformForWrapper` that the wrapper does not have a form associated yet, and the metadata is not updated.

Recommendation: Please check that no superform is yet associated with the wrapper, and update metadata:

```
  function createSuperformForWrapper(
      bytes32 wrapperKey,
      uint32 formImplementationId_
  )
      external
      override
      returns (uint256 superformId, address superform)
  {
      WrapperMetadata memory metadata = wrappers[wrapperKey];

      if (metadata.wrapper == address(0)) revert WRAPPER_DOES_NOT_EXIST();
      ISuperformFactory sf = ISuperformFactory(superRegistry.getAddress(keccak256("SUPERFORM_FACTORY")));


      address formImplementation = sf.getFormImplementation(formImplementationId_);
      if (formImplementation == address(0)) revert Error.FORM_DOES_NOT_EXIST();

+     if (metadata.formImplementation != address(0)) revert WRAPPER_ALREADY_HAS_FORM();
+     metadata.formImplementation = formImplementation;

      (superformId, superform) = sf.createSuperform(formImplementationId_, metadata.wrapper);
      emit CreatedSuperformForExistingWrapper(metadata.wrapper, wrapperKey, superformId, superform);
  }
```

Superform: Fixed in commit a1d8f7c8.

3.3.2 `dispatchPayload` can revert due to the exact `msg.value` check

Severity: Low Risk

Context: LayerzeroV2Implementation.sol#L320

Description: To dispatch the payload through LayerZeroV2 the caller needs to estimate the native token amount to cover all the fees. It's done by calling the `estimateFees` function.

The issue is if between the time `estimateFees` was called off-chain and the transaction is executed the native fee changes the `dispatchPayload` call will revert. The difference between the quote and the actual fees can occur as the price conversion between the chain changes, which is fairly often (see PriceFeed.sol#L15).

Recommendation: The `endpoint.send(...)` function has a refund parameter so it would be better to check:

```
if (msg.value < _fee.nativeFee || _fee.lzTokenFee != 0) revert INVALID_MSG_FEE();
```

Superform: Fixed in commit 7f4f1521.

3.3.3 `DeBridgeValidator::validateTxData` centralization risk, rescuer can steal user funds by patching order

Severity: Low Risk

Context: DeBridgeForwarderValidator.sol#L84, DeBridgeValidator.sol#L59

Description: When creating a DLN order on the destination chain when bridging liquidity with DeBridge, `orderAuthorityDst` is granted to the address which has the `CORE_REGISTRY_RESCUER_ROLE` on the destination chain. However this address would have two capacities:

- Cancel order, and enable sender to be refunded on source chain (harmless).
- Patch order, reducing the price of sent assets (can be used to rug user).

Recommendation: Please consider creating a contract which would be able to cancel the order on the destination chain, but not be able to patch it.

Superform: This is a valid issue we will be addressing.

3.3.4 `ERC5115Form::_depositAndValidate` redundant and inconsistent slippage check

Severity: Low Risk

Context: ERC5115Form.sol#L520-L531

Description: Two different slippage checks are implemented to prevent a user to lose funds when depositing to a ERC5115 vault.

First one can be seen in ERC5115Form.sol#L520 and is checking that the number of shares returned verifies:

```
shares >= singleVaultData_.outputAmount
```

and second one ERC5115Form.sol#L525-L528 checks:

```
shares >= singleVaultData_.outputAmount*(ENTIRE_SLIPPAGE - singleVaultData_.maxSlippage)
```

As we can see these checks have inconsistent values, and for example may surprise the user if he expects a deposit returning more than `singleVaultData_.outputAmount*(ENTIRE_SLIPPAGE - singleVaultData_.maxSlippage)` to succeed.

Recommendation: Please consider changing the call to deposit so both checks are consistent:

```
- shares = v.deposit(sharesReceiver, assetDifference_, singleVaultData_.outputAmount);
+ uint outputAmountMin = singleVaultData_.outputAmount*(ENTIRE_SLIPPAGE - singleVaultData_.maxSlippage);
+ shares = v.deposit(sharesReceiver, assetDifference_, outputAmountMin);
```

Superform: Fixed in commit 22a6bed2.

3.3.5 `ERC5115Form::_directDepositIntoVault` `vars.chainId` is never initialized

Severity: Low Risk

Context: ERC5115Form.sol#L249

Description: IBridgeValidator implementations use 3 chain id values to determine that an action is valid: `srcChainId`, `dstChainId`, `liqDstChainId`. In the case of a direct deposit, all of these three values must be equal to current chain.

However in `ERC5115Form`, initializing of the variable for the three values `vars.chainId` is omitted.

In practice, the validator only compare these values between them (equal or not equal), so this has no impact for current validators. If a new validator is to be introduced which has a custom behavior on some chains, this would cause problems.

Recommendation: Initialize this value as is done currently in ERC4626FormImplementation.sol

Superform: Fixed in commit e8227b71.

3.3.6 `DeBridgeValidator::validateTxData` Forcing control of same addr on src and dest breaks smart wallets compatibility

Severity: Low Risk

Context: DeBridgeForwarderValidator.sol#L78, DeBridgeValidator.sol#L53

Description: In the withdrawal case of `DebridgeValidator::validateTxData`, it is enforced that `args_.receiverAddress` is the receiver of funds on the destination chain, but also `args_.receiverAddress` is set to be the authority to patch the order on the source chain. This assumes that the user is able to control the same address on the source and destination chain, which is generally right for EOAs but not for contract wallets.

Recommendation: Enable to choose a different address to have authority over the source chain order.

Superform: Acknowledged. This will prevent smart wallets from using DeBridge; however, as this information/ability to add a third wallet is not factored in the `CoreStateRegistry`/`SuperformRouter`, we cannot implement such checks.

3.4 Gas Optimization

3.4.1 State variables can be marked as `immutable`

Severity: Gas Optimization

Context: AxelarImplementation.sol#L33, LayerzeroV2Implementation.sol#L39

Description: The `AxelarImplementation::gateway` and `LayerzeroV2Implementation::endpoint` state variables can only be set once by protocol admin. As values of these variables are known at the time of contract deployment, they can be made `immutables`.

Recommendation: Consider marking the variables as `immutable`.

Superform: Acknowledged. The endpoint and gateway of future deployment chains could be different from current deployments, leading to the inability to deploy to the same address using `CREATE2` as the byte-encoded constructor argument changed. Provided that we decide to keep it consistent with other implementations by just setting the super registry in the constructor.

3.5 Informational

3.5.1 Remove `isComposeMsgSender` function

Severity: Informational

Context: LayerzeroV2Implementation.sol#L264-L277

Description: `LayerzeroV2Implementation` doesn't implement any of the composing functionality available with LzV2. It does not include the `lzCompose` interface nor can it send a composed message by calling `sendCompose`.

Recommendation: Remove the `isComposeMsgSender` function as it's not used for anything.

Superform: Fixed in commit 4a53b97b.

3.5.2 `DeBridgeValidator::validateTx` and `DeBridgeForwarderValidator::validateTx` share order creation validation logic

Severity: Informational

Context: DeBridgeForwarderValidator.sol#L66

Description: The order creation on DLN has almost the same constraint in both cases, and this logic could be factored into a common library to reduce error surface.

Recommendation: Factor the order creation logic into a common util

Superform: Acknowledged. The suggested changes would only address a small portion of the common logic, specifically four validation checks. More importantly, the `DeBridgeForwarderValidator` decodes the

data into a custom `DecodedQuote` struct, fundamentally different from how the DeBridgeValidator handles the data. Provided that we decided not to implement the recommendation as mentioned earlier.

3.5.3 Missing non-existent `authorizedImpl` validation in `AxelarImplementation.dispatchPayload`

Severity: Informational

Context: AxelarImplementation.sol#L181

Description: In `AxelarImplementation.dispatchPayload` function the address of `AxelarImplementation` contract for destination chain is read from `authorizedImpl` storage state. This value can be an empty value in case the `setReceiver` function is not yet called by the admin for a particular destination chain.

Recommendation: Consider validating that the `axelerDstImpl` address value is non-zero.

Superform: Fixed in commit 2caeec09.

3.5.4 Unused code

Severity: Informational

Context: ERC5115To4626Wrapper.sol#L43

Description: ERC5115To4626Wrapper.sol#L43 contains the unused constant `NATIVE`.

Recommendation: Consider removing the unused constant.

Superform: Fixed in commit 3fefe13a.