Three Sigma Labs

Code Audit

Fuji Finance Cross-chain money market aggregator

Disclaimer

Code Audit

Fuji Finance Cross-chain money market aggregator

Disclaimer

The ensuing audit offers no assertions or assurances about the code's security. It cannot be deemed an adequate judgment of the contract's correctness on its own. The authors of this audit present it solely as an informational exercise, reporting the thorough research involved in the secure development of the intended contracts, and make no material claims or guarantees regarding the contract's post-deployment operation. The authors of this report disclaim all liability for all kinds of potential consequences of the contract's deployment or use. Due to the possibility of human error occurring during the code’s manual review process, we advise the client team to commission several independent audits in addition to a public bug bounty program.

Table of Contents

Code Audit

Fuji Finance Cross-chain money market aggregator

Table Of Contents

Summary

Code Audit

Fuji Finance Cross-chain money market aggregator

Summary

Three Sigma Labs audited Fuji Finance in a 6 person week engagement. The audit was conducted from 05-06-2023 to 05-25-2023.

Protocol Description

Fuji is a protocol that aggregates money markets in the decentralized finance (DeFi) space. In the background, Fuji routes users' loan requests to the liquidity sources with the best rates and refinances the open positions. It relies on external secure bridges (such as Connext) to enable unlimited cross-chain composability.

Fuji users benefit by obtaining lowest borrowing rates, and constant automated refinancing to the lowest rate once market conditions change. It enables users to provide collateral on chain A and get their loans disbursed on chain B by narrowing down the whole operation to a single transaction. Fuji analyzes thousands of lend and borrow terms across multiple protocols and chains to provide users with the best rates.

Fuji’s goal is to build a cross-chain loan aggregator and debt management platform that identifies the best terms on different lending protocols and reduces the friction of managing debt on multiple chains.

Scope

Code Audit

Fuji Finance Cross-chain money market aggregator

Scope

```
src/
├── abstracts
│   ├── BaseFlasher.sol
│   ├── BaseRouter.sol
│   ├── BaseVault.sol
│   ├── EIP712.sol
│   └── PausableVault.sol
├── helpers
├── access
│   ├── CoreRoles.sol
│   └── SystemAccessControl.sol
├── interfaces
├── libraries
├── mocks
├── providers
├── routers
│   ├── ConnextHandler.sol
│   ├── ConnextRouter.sol
├── swappers
└── vaults
    ├── VaultPermissions.sol
    ├── borrowing
    │   └── BorrowingVault.sol
    └── yield
        └── YieldVault.sol
```

Assumptions

The scope of the audit was carefully defined to include the contracts at the lowest level of the inheritance hierarchy, as these are the ones that will be deployed to the mainnet. The only external libraries used in the implementation of these contracts were ones trusted by the community (i.e. OpenZeppelin) - these libraries have already been battle-tested by multiple protocols, guaranteeing a high level of security.

Methodology

Code Audit

Fuji Finance Cross-chain money market aggregator

Methodology

To begin, we reasoned meticulously about the contract's business logic, checking security-critical features to ensure that there were no gaps in the business logic and/or inconsistencies between the aforementioned logic and the implementation. Second, we thoroughly examined the code for known security flaws and attack vectors. Finally, we discussed the most catastrophic situations with the team and reasoned backwards to ensure they are not reachable in any unintentional form.

Taxonomy

In this audit we report our findings using as a guideline Immunefi's vulnerability taxonomy, which can be found at immunefi.com/severity-updated/. The final classification takes into account the severity, according to the previous link, and likelihood of the exploit. The following table summarizes the general expected classification according to severity and likelihood; however, each issue will be evaluated on a case-by-case basis and may not strictly follow it.

Severity / Likelihood	LOW	MEDIUM	HIGH
NONE	None		
LOW	Low		
MEDIUM	Low	Medium	Medium
HIGH	Medium	High	High
CRITICAL	High	Critical	Critical

Project Dashboard

Code Audit

Fuji Finance Cross-chain money market aggregator

Project Dashboard

Application Summary

Name	Fuji Finance
Commit	42e9d36
Language	Solidity
Platform	Ethereum

Engagement Summary

Timeline	05-06-2023 to 25-06-2023
Nº of Auditors	2
Review Time	6 person weeks

Vulnerability Summary

Issue Classification	Found	Addressed	Acknowledged
Critical	2	2	0
High	7	4	3
Medium	10	6	4
Low	8	4	4
None	19	18	1

Category Breakdown

Suggestion	15
Documentation	1
Bug	17
Optimization	11
Good Code Practices	3

Code Maturity Evaluation

Code Audit

Fuji Finance Cross-chain money market aggregator

Code Maturity Evaluation

Code Maturity Evaluation Guidelines

Category	Evaluation
Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system.
Arithmetic	The proper use of mathematical operations and semantics.
Centralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Code Stability	The extent to which the code was altered during the audit.
Upgradeability	The presence of parameterizations of the system that allow modifications after deployment.
Function Composition	The functions are generally small and have clear purposes.
Front-Running	The system’s resistance to front-running attacks.
Monitoring	All operations that change the state of the system emit events, making it simple to monitor the state of the system. These events need to be correctly emitted.
Specification	The presence of comprehensive and readable codebase documentation.
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage.

Code Maturity Evaluation Results

Category	Evaluation
Access Controls	Satisfactory. The codebase has a strong access control mechanism.
Arithmetic	Satisfactory. The codebase uses Solidity version >0.8.0 as well as takes the correct measures in rounding the results of arithmetic operations.
Centralization	Moderate. The owner has some privileges over the protocol.
Code Stability	Satisfactory. The code was stable during the audit.
Upgradeability	Weak. The contracts are not upgradeable.
Function Composition	Satisfactory. There is little duplicated logic and functions have a clear purpose.
Front-Running	Moderate. There are a few front-running opportunities.
Monitoring	Moderate. Some events are emitted, but not all state changing operations are covered with events.
Specification	Satisfactory. There is a comprehensive and readable documentation.
Testing and Verification	Satisfactory. There is an adequate testing suite with unit, integration, functional and fuzz testing.

Findings

Code Audit

Fuji Finance Cross-chain money market aggregator

Findings

3S-FUJI-C01

In **BaseRouter**, the beneficiary isn't checked when starting a **flashloan** action and it replaces the previous beneficiary

Id	3S-FUJI-C01
Classification	Critical
Severity	Critical
Likelihood	High
Category	Bug
Relevant Links	abstracts/BaseRouter.sol#L292 POC/ POCAttackerChangesBeneficiary.t.sol #L111

Description

When doing a flashloan action in the **BaseRouter.sol** the beneficiary isn't compared with the beneficiary of the previous action, and the variable gets replaced with the beneficiary for the **flashloan**, this can lead to a beneficiary change, which could lead to stolen assets.

For example, using **3S-FUJI-M10**, it is possible to withdraw for another user, use a **flashloan** to change the beneficiary to the attacker's address and use the funds withdrawn with **3S-FUJI-M10** however the attacker wishes.

Recommendation

Add a function to check the beneficiary instead of replacing.

POC

https://github.com/threesigmaxyz/fuji-issues-external/blob/master/test/POC/POCAttackerChangesBeneficiary.t.sol#L111

Status

Addressed here: Fujicracy/fuji-v2#615

3S-FUJI-C02

Wrong **tokensToCheck** logic in **BaseRouter** enables attackers to steal funds

Id	3S-FUJI-C02
Classification	Critical
Severity	Critical
Likelihood	High
Category	Bug
Relevant Links	abstracts/BaseRouter.sol#L636-L640 POC/ POCWrongTokensToCheckLogic.t.sol# L80

Description

The **BaseRouter** implements a mechanism to prevent tokens from being stuck that checks the balance of the router before and after the actions and ensures it remains equal. Problem is, the **_addTokenToList(...)** function has a bug such that only the last token interacted with is added to the array at the first element (0). The **_isInTokenList(...)** function returns index 0 if the token is not in the array, so the position **uint256 position = index == 0 ? index : index + 1;** will always be 0.

Attackers can use this to steal other users by waiting for withdraw approvals for the **BaseRouter** of other users and then use these approvals to withdraw to the **BaseRouter**, followed by a deposit action in another vault to remove the asset of the withdraw from the **tokensToCheck** array. The asset is then held by the **BaseRouter**, which can be retrieved in a following transaction by the attacker by

1) depositing the funds to the vault, where the receiver and owner are the BaseRouter.

2) withdrawing these funds to the attacker, where the receiver is the attacker and the owner the **BaseRouter**.

3) depositing another asset to another vault, removing the first asset from the **tokensToCheck** array.

Recommendation

Fix the implementation of the **`_addTokenToList(...)`** function by keeping a memory variable with the number of tokens to check. The position to add in **`_addTokenToList(...)`** should be the number of tokens to check + 1, if the token is not in the array.

POC

https://github.com/threesigmaxyz/fuji-issues-external/blob/master/test/POC/POCWrongTokensToCheckLogic.t.sol#L80

Status

Addressed here: Fujicracy/fuji-v2#622

3S-FUJI-H01

Changing providers might lead to lost assets in **BorrowingVault** and **YieldVault**

Id	3S-FUJI-H01
Classification	High
Severity	Critical
Likelihood	Low
Category	Bug
Relevant Links	borrowing/BorrowingVault.sol#L943 yield/YieldVault.sol#L276

Description

`totalAssets(...)` and **`totalDebt(...)`** are fetched from the set of providers, which can be changed. Thus, if providers change, but debt or assets are deposited/borrow from the old providers, **`totalAssets(...)`** and/or **`totalDebt(...)`** will change.

In the case of **`totalAssets(...)`**, this means that users shares would be worth less, such that existing users would take a loss and new users could profit from depositing at this reduced rate and, after/if the **`totalAssets(...)`** were recovered, take a profit.

In the case of **`totalDebt(...)`**, **`debtShares`** would be worth less, so existing users would make a profit by being able to **`payback`** their debt for less debt. New users would take a loss, following the same reasoning as before.

Recommendation

When removing a provider, check if the debt balance is bigger than 0, in which case, it should be repaid/moved to another provider first.

In the case of the assets, ensuring that the deposit balance of a provider is 0 is vulnerable to a DoS attack, where attackers could deposit on the vault's behalf. Thus, when changing providers, if the deposit balance of an old provider is not 0, these assets could be moved atomically to a new provider.

Status

Acknowledged with the following statement:

The issue identified is acknowledged by the team. However, current **`BorrowingVault`** smart contract bytecode size is a limitation for the proposed change and requires a major refactor that is planned to be introduced at a later stage.

The team recognizes the risk which is only a temporary DOS to the vault's funds when a provider with assets is removed by mistake. The provider that was removed mistakenly can always be reestablished by a subsequent **`timelock`** call.

3S-FUJI-H02

`REBALANCER_ROLE` can drain funds by rebalancing in a loop in the `BorrowingVault`

Id	3S-FUJI-H02
Classification	High
Severity	Critical
Likelihood	Trusted Actor is compromised
Category	Suggestion
Relevant Links	borrowing/BorrowingVault.sol#L791

Description

The rebalancer in the borrowing vault receives a fee for providing **`debtAsset`** to repay the originating provider and send to the destination provider. A malicious provider could use this to profit, for example

max fee = **`10 / 10000`** = **`1 / 1000`** = 0.001

100_000 debt -> 100_100

100_100 deby -> 100200.1

and so on...

For this attack, the attacker only needs to pay a **`flashloan`** fee once (or have the initial funds available).

Recommendation

The likelihood of this attack is reduced because the rebalancer is a privileged role, but maybe adding a timelock to the rebalancing function would be a good measure so that the **`chief`** can disallow a rebalancer if they behave maliciously, before they profit too much from this exploit.

Status

Acknowledged with the following statement:

The issue identified is acknowledged by the Fuji team, however, the team will implement the following security measures at the **`BorrowingVault`** at a later stage. The rationale to delay the fixes is that the team does not consider this issue an imminent user risk considering that only the **`RebalancerManager`** contract should have the **`REBALANCER`** role. In addition the **`BorrowingVault`** bytecode size is near limits and additional logic is not possible until a major refactor. The future measures that will be considered are:

1. Restricting the **`REBALANCER_ROLE`** to only contract implementations. This action will reduce the chance of a compromised **`EOA`** account.
2. Adding a non-reentrant restriction to the **`rebalance(...)`** function. This action will restrict the looping optionality that is described in this issue.

3S-FUJI-H03

`UniswapV2Swapper` uses `block.timestamp` for deadline [Out of scope]

Id	3S-FUJI-H03
Classification	High
Severity	High
Likelihood	High
Category	Bug
Relevant Links	swappers/UniswapV2Swapper.sol#L72

Description

Uniswap sets a deadline to limit arbitrage opportunities if the swap does not get included right away. If the swap specifies a deadline of `block.timestamp`, then the swap transaction can be included in any block. This means that the price can, by then, have changed significantly.

Recommendation

Send a deadline argument.

Status

Acknowledged with the following statement:

The issue identified is acknowledged by the Fuji team. It is worth noting that the bundleInternal(...) currently has some indirect protection against slippage. However, Fuji will proceed to implement a deadline argument in the LiquidationManager.sol and BaseRouter.sol during Q3-2023 while working on general optimizations/improvements.

3S-FUJI-H04

`ConnextHandler` `executeFailedWithUpdatedArgs(...)` reentrancy `allowedCaller` can steal all `ConnextHandler` tokens

Id	3S-FUJI-H04
Classification	High
Severity	Critical
Likelihood	Low
Category	Bug
Relevant Links	routers/ConnextHandler.sol#L221

Description

In `executeFailedWithUpdatedArgs(...)`, the allowedCaller can steal all assets available on the `ConnextHandler` by calling `executeFailedWithUpdatedArgs(...)` again after the `xBundle(...)` call. This can be mitigated also by adding the `nonReentrant` modifier to `xBundle(...)`.

Recommendation

Write `txn.executed = true;` before the try/catch call and rewrite `txn.executed = false` if it fails.

Status

Addressed here: Fujicracy/fuji-v2#621

3S-FUJI-H05

Wrong transformation in function **previewMintDebt(...)**

Id	3S-FUJI-H05
Classification	High
Severity	High
Likelihood	High
Category	Bug
Relevant Links	

Description

In the **BorrowingVault.sol** the function **previewMintDebt(...)** is supposed to take an amount of shares and turn them into an amount of debt. Currently it is taking the shares as if they were debt and turning them to shares.

```
  function previewMintDebt(uint256 shares) public view override
returns (uint256 debt) {
    return _convertDebtToShares(shares, Math.Rounding.Down);
  }
```

Recomendation

Change the function to use **_convertToDebt(...)** instead of **_convertDebtToShares(...)**, as following

```
  function previewMintDebt(uint256 shares) public view override
returns (uint256 debt) {
    return _convertToDebt(shares, Math.Rounding.Down);
  }
```

Status

Addressed here: Fujicracy/fuji-v2#624

3S-FUJI-H06

`safeApprove(...)` reverts if approval is different than 0, use `safeIncreaseAllowance(...)` instead

Id	3S-FUJI-H06
Classification	High
Severity	High
Likelihood	High
Category	Bug
Relevant Links	abstracts/BaseRouter.sol#L554

Description

The **`BaseRouter`** interacts with the vault and some actions require tokens being pulled from the **`BaseRouter`** to the vault, which means that the **`BaseRouter`** must first approve the vault. This approval is using **`SafeERC20 safeApprove(...)`**, which is deprecated and should not be used due to the fact that it reverts if the allowance of pair owner and spender is not 0. This allowance should always be 0 for most tokens and situations when **`safeApprove(...)`** is called, given that the tokens are spent right after the approval is emitted. However, some weird tokens or behaviour could lead to not all tokens being spent, which would freeze the **`BaseRouter`** actions that require approval.

Recommendation

Use **`safeIncreaseAllowance(...)`** instead of **`safeApprove(...)`**.

Status

Addressed here: Fujicracy/fuji-v2#623

3S-FUJI-H07

Attackers can claim deposits to vaults if users specify the router as receiver and don't withdraw shares after

Id	3S-FUJI-H07
Classification	High
Severity	Critical
Likelihood	Low
Category	Bug
Relevant Links	abstracts/BaseRouter.sol#L192 abstracts/BaseRouter.sol#L205 POC/POCAttackerStealsUser.t.sol#L40

Description

The **`BaseRouter`** checks if the initial balance of the tokens interacted with is the same as the end balance. However, for vaults, it only checks the balance of the underlying asset, not the shares themselves. Thus, for example, when a user makes a deposit action and specifies the **`BaseRouter`** as the receiver, but does not do any further action, the shares will be locked in the **`BaseRouter`**. Then, an attacker can call the withdrawal action of the **`BaseRouter`** and redeem these shares for any receiver.

Recommendation

Add to the **`tokensToCheck`** array the address of the vault itself (its shares).

POC

https://github.com/threesigmaxyz/fuji-issues-external/blob/master/test/POC/POCAttackerStealsUser.t.sol#L40

Status

Addressed here: Fujicracy/fuji-v2#622

3S-FUJI-M01

`BorrowingVault`, if the debt/assets ratio falls too much, liquidators could choose to repay debt equal to the assets at a discount

Id	3S-FUJI-M01
Classification	Medium
Severity	High
Likelihood	Low
Category	Suggestion
Relevant Links	borrowing/BorrowingVault.sol#L835

Description

Currently the borrowing vault enables liquidating a user's position 50% if the debt/assets ratio is above **`liqRatio`**, but below **`liqRatio / FULL_LIQUIDATION_THRESHOLD`**. If the ratio debt/assets increases too much, a liquidation might not be profitable for liquidators, because users might not have enough assets. In this scenario, it could be possible to completely close the user position by paying back the debt corresponding to the assets of the user. This will still leave some amount of bad debt, but hopefully much less.

In the current implementation, if a specific collateral asset drops too much, it could happen that many liquidations would not be profitable and a lot of bad debt would appear. This bad debt could be limited with this recommendation. In the end, it could prevent a liquidation in a provider.

Example:

75 debt, 100 assets

asset/debt value falls to 0.74

paying 75 debt, would give the liquidator 100 assets, which is equivalent to 74 debt at 0.74 price, **not profitable**.

This would leave **75** bad debt in the protocol

Recommendation

Let the liquidators "close" the position of the user by getting 100 assets for the corresponding **debt** at a discount.

Same Example:

Letting liquidators choose to repay the debt corresponding to the total amount of assets, if these are worth less than the debt, would help keep the protocol healthier.

In this case, let's say a liquidator wants to repay the 100 assets, it would cost **`assets * price * LIQUIDATION_PENALTY = 100 * 0.74 * 0.9 = 66.6`** debt.

This would leave **8.4** bad debt in the protocol.

Note that the current liquidation logic would require skipping the **`maxRedeem(...)`** of the user, if the liquidation is not profitable, **`userAssets < userDebt * price * 10 ** _asset.decimals()`**.

Status

Addressed here: Fujicracy/fuji-v2#636

3S-FUJI-M02

Payback can be **DoSed** in the **BorrowingVault**, may be profitable for liquidators

Id	3S-FUJI-M02
Classification	Medium
Severity	Medium
Likelihood	High
Category	Bug
Relevant Links	borrowing/BorrowingVault.sol#L723 POC/POCPaybackDoSed.t.sol#L27

Description

The borrowing vault enables anyone to **payback** the debt of another user. Thus, attackers can frontrun a user **payback** transaction, **payback** 1 wei and stop users from paying back their debts. This would make sense for liquidators, who are incentivized to do this to liquidate users.

Recommendation

There are 2 alternatives:

- Instead of reverting if the **payback** is bigger than the maximum, limit it to the maximum.

- Only enable the user to **payback** its debt.

POC

https://github.com/threesigmaxyz/fuji-issues-external/blob/master/test/POC/POCPaybackDoSed.t.sol#L27

Status

Addressed here: Fujicracy/fuji-v2#642

3S-FUJI-M03

Approval in **`BaseVault`** reduces over time

Id	3S-FUJI-M03
Classification	Medium
Severity	Medium
Likelihood	High
Category	Bug
Relevant Links	extensions/ERC4626.sol#L50 abstracts/BaseVault.sol#L146 abstracts/BaseVault.sol#L163 abstracts/BaseVault.sol#L180 abstracts/BaseVault.sol#L196 abstracts/BaseVault.sol#L212 borrowing/BorrowingVault.sol#L676 POC/POCFaillingApproval.t.sol#L32

Description

BaseVault stores users allowances as underlying assets allowances. When a user calls, for example, **`approve(...)`**, it converts the shares amount in the argument to a corresponding asset amount. This approach means that, over time, due to the value of shares increasing over time (yield from providers), the same amount of shares will be worth more assets. Thus, if users approve a certain allowance, and then a few seconds later, another user tries to use this same allowance amount, it should revert, as this shares amount now equals more assets.

The BorrowingVault does not have this issue because it never converts between **`debt`** and **`debtShares`** and the allowance is stored in debt. Thus, if a user approves another user for, let's say, 100 debt, the other user can always spend 100 debt.

Recommendation

Instead of converting to the underlying assets, store the allowances in shares. This is how the openzeppelin ERC4626 implementation does it and is more reliable.

POC

https://github.com/threesigmaxyz/fuji-issues-external/blob/master/test/POC/POCFaillingApproval.t.sol#L32

Status

Acknowledged with the following statement:

The issue identified is acknowledged by the Fuji team, however, the team encountered that such change will require major refactors at: smart contracts, sdk, and front-end. Therefore, they intend to address this issue in Q4-2023, to replace storing assets for asset-shares allowance.

3S-FUJI-M04

BaseFlasher transfers tokens and then calls **xBundle(...)**, so the sent tokens can't be returned due to the balance check

Id	3S-FUJI-M04
Classification	Medium
Severity	Medium
Likelihood	High
Category	Bug
Relevant Links	abstracts/BaseFlasher.sol#L102 POC/ POCCantUseFlashloanFunds.t.sol#L97

Description

The **BaseRouter** checks if the balances of the assets remain the same after the **xBundle(...)** execution. The BaseFlasher, **_requestorExecution(...)** is implemented in the following way

```
IERC20(asset).safeTransfer(requestor, amount);
requestor.functionCall(requestorCalldata);
// approve flashloan source address to spend to repay flashloan
IERC20(asset).safeApprove(getFlashloanSourceAddr(asset), amount +
fee);
```

Thus, when **xBundle(...)** is called in **requestor.functionCall(requestorCalldata);**, the initial balance of the **BaseRouter** will take into account the received funds from the **flashloan**, such that it will be impossible to return the funds at the end of the call.

Recommendation

Similarly to **xReceive(...)** of the ConnextRouter, the BaseRouter could implement a similar function. In this case, the caller of, let's say, **flashloanReceive(...)**, could be whitelisted to one of the flashers.

POC

https://github.com/threesigmaxyz/fuji-issues-external/blob/master/test/POC/POCCantUseFlashloanFunds.t.sol#L97

Status

Addressed here: Fujicracy/fuji-v2#622

3S-FUJI-M05

`_crossTransfer(...)` reverts for smart contracts that don't share the same address on different chains

Id	3S-FUJI-M05
Classification	Medium
Severity	Medium
Likelihood	High
Category	Bug
Relevant Links	routers/ConnextRouter.sol#L275

Description

`_crossTransfer(...)` has a beneficiary check such that the receiver of the funds in the destination chain must be the same as the previous beneficiary (most likely the msg.sender or the **`ConnextRouter`**). If the address is a smart contract, the address will probably be different on the different chain, which will make the transaction revert.

Recommendation

Allow users to specify in a mapping the corresponding beneficiary in the destination chain, in a function **`addBeneficiaryToDestDomain(...)`**, which would set **`beneficiary[msg.sender][destDomain] = newBeneficary;`**.

Status

Acknowledged with the following statement:

The issue identified is acknowledged by the Fuji team. Smart contract integrations will require the capability of defining a different address as beneficiary in a different domain. This feature will be added during Q3-2023 while working on general optimizations/improvements on the router contracts.

3S-FUJI-M06

ConnextRouter fails to record failed message if gas sent is not enough

Id	3S-FUJI-M06
Classification	Medium
Severity	High
Likelihood	Low
Category	Bug
Relevant Links	routers/ConnextRouter.sol#L171 POC/POCXReceiveGas.t.sol#L34

Description

The ConnextRouter places the **`xBundleConnext(...)`** call on a **`try`**/**`catch block`** to prevent failed actions to make the transaction revert and the funds being given to **`Connext`**. The **`EVM`** saves 1/64th of gas before an external call to try to finish execution after the external call. However, if the **`xBundleConnext(...)`** transaction inside the **`try`**/**`catch`** fails, the following execution consumes a non significant amount of gas (transferring and recording the failed message), such that 1/64 of the sent **`gas`** could not be enough to finish the execution.

Recommendation

Calculate the minimum amount required of gas to finish execution after the try/catch block and revert if this gas is not available prior to the block with **`gasleft()`**. This ensures that whatever actions users do inside the try/catch block, the message will always be recorded if it fails.

POC

https://github.com/threesigmaxyz/fuji-issues/blob/master/test/POC/POCXReceiveGas.t.sol#L34

Status

Acknowledged with the following statement:

The issue identified is acknowledged by the Fuji team, however, as indicated the likelihood of its occurrence is low. Along with issue **3S-FUJI-N05**, during Q3-2023 we will be working on general optimizations/improvements on the router contracts that will allow a more consistent estimation of gas during a failed transfer and implement fix as suggested.

3S-FUJI-M07

It's impossible to do a **depositETH** action on **xReceive(...)** in **ConnextRouter**

Id	3S-FUJI-M07
Classification	Medium
Severity	Medium
Likelihood	High
Category	Bug
Relevant Links	abstracts/BaseRouter.sol#L299-L301

Description

The **ConnextRouter** allows users to bridge items from one chain to another and perform arbitrary actions there. Currently, the **ConnextRouter** checks, in the **depositETH** action, that the amount is equal to the **msg.value** sent. This means that for **xReceive(...)** calls, which never send native, it's impossible to deposit **ETH**.

Recommendation

The check that the amount is equal to the msg.value is unnecessary, given that it should be impossible to steal an already existing **ETH** from the router due to the balance checks. Additionally, this should be implemented for consistency, because for ERC20 tokens this check does not exist currently; users can specify the router as the sender in deposit actions, effectively not checking if the users sent the tokens.

Thus, remove

```
if (amount != msg.value) {
  revert BaseRouter__bundleInternal_insufficientETH();
}
```

Status

Addressed here: Fujicracy/fuji-v2#650

3S-FUJI-M08

Users can claim tokens in **`ConnextRouter`** by calling **`xReceive(...)`** directly

Id	3S-FUJI-M08
Classification	Medium
Severity	High
Likelihood	Low
Category	Suggestion
Relevant Links	routers/ConnextRouter.sol#L148 abstracts/BaseRouter.sol#L119 routers/ConnextRouter.sol#L148 routers/ConnextRouter.sol#L175

Description

The **`ConnextRouter`** allows anyone to call **`xReceive(...)`** and the amount of assets the user provided is the balance of the contract.

Thus, users can use this functionality to sweep any existing funds into their wallets. In fact, this should also happen when a **`xReceive(...)`** call fails at **`this.xBundleConnext(...)`**, and the balance, including the existing tokens, get attributed to the user.

It should not be a likely event given that there are checks to ensure that the **`ConnextRouter`** does not end up with funds after **`xBundle(...)`** calls. Still, given that the **`BaseRouter`** has a **`sweepTokens(...)`** function, this might lead to a false sense of security.

Recommendation

Only let a permissioned address call **`xReceive(...)`** (should be the **`xConnect`** address).

Additionally, instead of sending the **`balance`** in the **`ConnextRouter`** if **`this.xBundleConnext(actions, args, beforeSlipped)`** fails, send the amount.

Status

Addressed here: Fujicracy/fuji-v2#639

3S-FUJI-M09

Connext delegates can perform important actions, make sure smart contract users implement them

Id	3S-FUJI-M09
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Suggestion
Relevant Links	https://docs.connext.network/developers/guides/handling-failures#high-slippage

Description

Connext implements a mechanism of adjusting/forcing parameters in the bridged assets which can be triggered by a delegate. If the delegate is a **EOA** or wallet, there is no problem, as the delegate can call directly the connext bridge. However, care must be taken if the caller of **xBundle(...)** is a smart contract, in which it may not implement the necessary **Connext** function calls. This can be dangerous if the smart contract user set, for example, a low slippage tolerance and did not implement the function **forceUpdateSlippage(...)** of the connext bridge.

Recommendation

Implement methods in the **Router**/**Handler** to handle these function calls, such as **bumpTransfer(...)** and make the ConnextRouter the delegate. In alternative, make it very clear in the documentation and comments in the interface/code that the user must implement these important **Connext** methods if it is a smart contract.

Status

Addressed here: Fujicracy/fuji-v2#628

3S-FUJI-M10

Withdraw and borrow can be **DoSed** in **BaseRouter**

Id	3S-FUJI-M10
Classification	Medium
Severity	Medium
Likelihood	Medium
Category	Suggestion
Relevant Links	abstracts/BaseRouter.sol#L197C3-L218

Description

Withdraw and borrow actions in the **BaseRouter** require the owner to increase the approval allowances. If the allowance increase and withdrawal action transaction are not done atomically (via a **multicall**, for example), anyone can withdraw 1 token, which will make the real withdraw action revert due to insufficient balance (borrowing is similar).

Recommendation

Similarly to the **deposit(...)** action **_safePullTokenFrom(...)**, only the owner of the withdrawal and borrow actions should be able to call these actions directly. To allow someone withdrawing for a user or cross chain withdrawals, the withdrawal should only be allowed if there was a permit action before (same for borrowing).

Status

Acknowledged with the following statement:

The issue identified is acknowledged by the Fuji team including the severity and potential for griefing attacks possible with user's excess allowance on the router. The team intends to implement the discussed solution: combining **IRouter.Action.PermitBorrow** and **IRouter.Action.PermitWithdraw** followed respectively by **borrow** or **withdraw** operations. Then restrict the "pure" **IRouter.Action.Withdraw**, and **IRouter.Action.Borrow** to ONLY be callable

when **`msg.sender`** is the owner. This change will be implemented among other pending changes in Q3 2023.

3S-FUJI-L01

Handling someone else repaying debt for the **BorrowingVault** could be done differently

Id	3S-FUJI-L01
Classification	Low
Severity	N/A
Likelihood	N/A
Category	Suggestion
Relevant Links	extensions/ERC4626.sol#L232 extensions/ERC4626.sol#L239 borrowing/BorrowingVault.sol#L462-L464 borrowing/BorrowingVault.sol#L975

Description

In **`_convertDebtToShares(...)`**, if **`totalDebt(...)`** is 0, it reverts. This means that, if someone decides to repay the full vault debt, this function would always revert. The current measure taken is pausing the vault and then borrowing a debt amount equal to the debt shares. This leads to a temporary halting of the vault and it may require rebalancing the providers to enable borrowing all the debt from the same provider (if the debt was distributed among many providers).

Recommendation

The Openzeppelin ERC4626 implementation, instead of placing **`totalAssets(...)`** in the denominator (here equivalent to **`totalDebt(...)`**), places **`totalAssets(...) + 1`** (also adds 1 when converting shares to assets). The yield vault could also use this logic, although I don't think it's possible for the **`totalAssets(...)`** to be 0 and the shares different from 0.

This would fix this problem, take a look at the following example

There are 10000 shares and 10000 assets.

User repays 10000 assets (directly at provider(s)), and now the supply is 10000 and totalDebt is 0.

Some user wants to borrow 100 debt:

`shares = debt * supply / (totalAssets + 1)` = **`100 * 10_000 / 1`** = **`1_000_000`**

if this user tries to pay back the debt

`debt = shares * (totalAssets + 1) / supply` = **`1_000_000 * 101 / 1_010_000`** = 100

which adds up correctly

if another user wants to repay their debt, the calculation would be

shares = 100, supply = 10000, assets = 0

`debt = shares * (totalAssets + 1) / supply` = **`100 * 1 / 10000`** = 0

which means their debt would have been paid by the attacker. It would only increase if they borrow more, which would work correctly, as shown in the previous example.

Status

Addressed here: Fujicracy/fuji-v2#640

3S-FUJI-L02

BaseVault is not fully ERC5143 compliant

Id	3S-FUJI-L03
Classification	Low
Severity	N/A
Likelihood	N/A
Category	Bug
Relevant Links	https://eips.ethereum.org/EIPS/eip-20 https://eips.ethereum.org/EIPS/eip-4626

Description

The BaseVault implements ERC5143, an extension of ERC4626. Thus, it should try to match every requirement of the ERC. In this case, when setting a new withdraw allowance, an event **`Approval(address indexed _owner, address indexed _spender, uint256 _value)`** should be triggered.

Additionally, **`allowance`** related functions only let spenders send assets to themselves (unless calling **`withdrawAllowance`** related functions).

Recommendation

- Emit the **`approval`** event when setting a new withdrawal allowance (when calling **`approve(...)`**, **`increaseAllowance(...)`** and **`decreaseAllowance(...)`**).

- Approval related functions should be fully compliant and set the approval of an address to all other addresses. This is important for other dApps to be compatible. Given that borrowing is not part of the standard, the borrow balance could be left as is. However, the withdraw allowance should be turned into the usual allowance mapping of ERC20. Applying the recommended fix to **3S-FUJI-M10** would make this change fully secure.

Status

Addressed here: Fujicracy/fuji-v2#647

3S-FUJI-L03

In **BaseRouter**, **_handleSwapAction(...)**, users shouldn't be allowed to send funds to an allowed flasher

Id	3S-FUJI-L04
Classification	Low
Severity	N/A
Likelihood	N/A
Category	Suggestion
Relevant Links	

Description

_handleSwapAction(...) allows sending funds to the **Flasher**. This should be to pay for a taken **flashloan**. However, if the goal of this exception is only to enable the payment of a **flashloan**, it should be removed and handled differently. This is because it may allow users to send funds to the flasher, without taking a **flashloan**, either maliciously or by mistake.

Recommendation

Create a new action called **repayFlashLoan** and use it to repay the **flashLoan**. This action should revert if the amount being repaid is bigger than the **flashloan** taken.

Status

Acknowledged with the following statement:

The issue identified is acknowledged by the Fuji team, however, as indicated the severity of this issue is low. During Q3-2023 we will be working on general optimizations/improvements on the router contracts.

3S-FUJI-L04

BaseRouter, **_bundleInternal(...) Action.Flashloan** does not check if the selector matches **xBundle(...)**

Id	3S-FUJI-L05
Classification	Low
Severity	N/A
Likelihood	N/A
Category	Bug
Relevant Links	abstracts/BaseRouter.sol#L288-L290

Description

Users make **flashloans** by calling **xBundle(...)** with a **flashloan** action. When the **flashloan** calls the callback of the **Flasher**, in **_requestorExecution(...)** of the **BaseFlasher**, it calls the function selector in the first 4 bytes in the **requestorCalldata**. However, in **_bundleInternal(...)**, the correct selector is not enforced, so users can call any function of the BaseRouter. There does not seem to be any clear path for an exploit, but it's best to check.

Recommendation

Add **if (bytes4(LibBytes.slice(requesterCalldata, 0, 4)) != this.xBundle(...).selector) revert();** or similar.

Status

Acknowledged with the following statement:

The issue identified is acknowledged by the Fuji team, however, as indicated the severity of this issue is low. During Q3-2023 the team will be working on general optimizations/improvements on the router contracts.

3S-FUJI-L05

`_crossTransfer(...)` should revert if users specify `routerByDomain[destDomain]` as destination

Id	3S-FUJI-L06
Classification	Low
Severity	N/A
Likelihood	N/A
Category	Suggestion
Relevant Links	routers/ConnextRouter.sol#L275 routers/ConnextRouter.sol#L324

Description

This scenario only happens if the router in the **`destDomain`** has the same address as the router in the **`destDomain`**. Else, the beneficiary check will fail.

`_crossTransfer(...)` enables users to bridge assets to any destination, without **`calldata`**. If users want to send assets to the router on the other chain, they might select **`_crossTransfer(...)`** instead of **`_crossTransferWithCalldata(...)`**, which would cause them to lose their funds.

Recommendation

Add the check **`if (receiver) == routerByDomain[destDomain]) revert();`** in the **`crossTransfer()`** call.

Status

Addressed here: Fujicracy/fuji-v2#651

3S-FUJI-L06

`executeFailedWithUpdatedArgs(...)` shouldn't be able to change beneficiary

Id	3S-FUJI-L07
Classification	Low
Severity	N/A
Likelihood	N/A
Category	Suggestion
Relevant Links	routers/ConnextHandler.sol#L202

Description

When the **`ConnextRouter`** message fails, it records the message in the **`ConnextHandler`** and sends it the funds.

In the **`ConnextHandler`**, it's possible to change the actions and arguments of a failed tx in **`executeFailedWithUpdatedArgs(...)`**.

Thus, it's possible to change the beneficiary, which would leave room for attacks to steal assets.

Recommendation

Check the beneficiary of the first action to match the previous beneficiary.

Status

Acknowledged with the following statement:

The issue identified is acknowledged by the Fuji team, however, as indicated the severity of this issue is low. The call to execute a failed transfer is permissioned and it's in the team's best interest to resolve failed transfers ethically. During Q3-2023 the team will be working on general optimizations/improvements on the router contracts in which this fix will be implemented.

3S-FUJI-L07

When changing addresses, use 2 step transfer and/or contract size and/or address **`0x0`** checks

Id	3S-FUJI-L08
Classification	Low
Severity	N/A
Likelihood	N/A
Category	Suggestion
Relevant Links	abstracts/BaseRouter.sol#L97-L98 abstracts/BaseRouter.sol#L120 abstracts/BaseRouter.sol#L125 routers/ConnextRouter.sol#L102 borrowing/BorrowingVault.sol#L791

Description

When changing important addresses, it's best to use additional safety measures to prevent wrong addresses from being set. When the address to change is guaranteed to be a smart contract, a **`isContract`** check is very helpful. Another layer of protection is using 2 step address transfer, in which a pending role is set first and only then, from the pending role account, it accepts the new role.

Recommendation

Check every address **`setter`** and use the mentioned patterns.

Status

Acknowledged with the following statement:

The optimization identified is acknowledged by the Fuji team. However, as indicated the issue is of low severity. During Q3 we will be working on optimizations that will improve general patterns within our contracts.

3S-FUJI-L08

`_getBeneficiaryFromCalldata(...)` in `ConnextRouter` should not allow the first action to be `depositETH(...)`

Id	3S-FUJI-ERROR01
Classification	Low
Severity	N/A
Likelihood	N/A
Category	Suggestion
Relevant Links	routers/ConnextRouter.sol#L400

Description

Currently issue **3S-FUJI-M07** means that the deposit action can't be performed on a cross transfer at all.

Still, having fixed it, the first action should not be **`depositETH(...)`**, as there is no ETH to be deposited yet (unless users are using the balance of the **`ConnextRouter`** in the other chain, which should not be intended).

Recommendation

In **`_getBeneficiaryFromCalldata(...)`**, revert if the first action is **`depositETH(...)`**. **`Flashloans`** would not be able to call **`depositETH(...)`** in the first action, which is okay because they send ERC20s anyway.

Status

Addressed here: Fujicracy/fuji-v2#650

3S-FUJI-N01

Borrowing is not vulnerable to an inflation attack, it's unnecessary to borrow when initializing the vault

Id	3S-FUJI-N01
Classification	None
Severity	N/A
Likelihood	N/A
Category	Optimization
Relevant Links	borrowing/BorrowingVault.sol#L171

Description

An inflation attack occurs because attackers manipulate the denominator of the assets calculation by increasing **`totalAssets(...)`**, which when calculating the shares as **`shares = assets * supply / totalAssets(...)`**, would make it round down and the depositing user is stolen. In the case of debt, it's not possible to increase the denominator **`totalDebt(...)`** without borrowing, which increases the shares. In fact, if the **`debtShares`** were rounded down by attackers, it would be beneficial to the user, who would have to pay less debt for the same **`debtShares`**.

Recommendation

Remove the borrow line in the **`initializeVault(...)`**.

Status

Addressed here: Fujicracy/fuji-v2#637

3S-FUJI-N02

`YieldVault` `maxRedeem(...)` unnecessarily converts shares to assets and back to shares again

Id	3S-FUJI-N02
Classification	None
Severity	N/A
Likelihood	N/A
Category	Optimization
Relevant Links	abstracts/BaseVault.sol#L272 abstracts/BaseVault.sol#L280 yield/YieldVault.sol#L211

Description

`maxRedeem(...)` enables users to know how many shares they can redeem. In the case of the borrowing vault, it depends on the amount of debt a user has taken, which is taken into account in **`_computeFreeAssets(...)`**.

In the case of the yield vault, **`maxRedeem(...)`** is the balance of the user. Currently, it converts the balance of the user to assets in **`_computeFreeAssets(...)`**, and then in **`maxRedeem(...)`** back to shares again. This is unnecessary and the code could be refactored to handle these 2 situations.

Recommendation

The **`YieldVault`** could override **`maxRedeem(...)`** and return the balance of the user directly (if not paused).

Status

Addressed here: Fujicracy/fuji-v2#641

3S-FUJI-N03

Throughout code base, implement using **`SafeERC20`** for **`IERC20`** for better readability

Id	3S-FUJI-N03
Classification	None
Severity	N/A
Likelihood	N/A
Category	Suggestion
Relevant Links	utils/SafeERC20.sol#L16 abstracts/BaseVault.sol#L24 abstracts/BaseRouter.sol#L13 borrowing/BorrowingVault.sol#L27 yield/YieldVault.sol#L21 routers/ConnextHandler.sol#L19

Description

When using SafeERC20, it's common to implement with **`using SafeERC20 for IERC20`**. Take a look at the openzeppelin code.

Status

Addressed here: Fujicracy/fuji-v2#648

3S-FUJI-N04

`BaseFlasher` does extra **`abi.encode`** unnecessarily

Id	3S-FUJI-N04
Classification	None
Severity	None
Likelihood	
Category	Optimization
Relevant Links	abstracts/BaseFlasher.sol#L73 abstracts/BaseFlasher.sol#L86

Description

The **`_entrypoint`** in **`BaseFlasher`** is **`keccak256(abi.encode(data))`**

Recommendation

Remove **`abi.encode`** and set it as **`keccak256(data)`**

Status

Addressed here: Fujicracy/fuji-v2#614

3S-FUJI-N05

`ConnextHandler` can store the hash of the failed messages instead

Id	3S-FUJI-N05
Classification	None
Severity	N/A
Likelihood	N/A
Category	Optimization
Relevant Links	routers/ConnextHandler.sol#L202

Description

Storing the failed message itself is very expensive.

Recommendation

Store the hash and then when calling `executeFailedWithUpdatedArgs(...)` send the `txn` as an argument and match it against the stored hash.

Status

Acknowledged with the following statement:

The optimization identified is acknowledged by the Fuji team. During Q3 the team will be working on optimizations that will improve the router contracts.

3S-FUJI-N06

When recording failed transactions in **ConnextHandler**, getting the next **Nonce** involves an unnecessary for loop

Id	3S-FUJI-N06
Classification	None
Severity	N/A
Likelihood	N/A
Category	Optimization
Relevant Links	routers/ConnextHandler.sol#90 routers/ConnextHandler.sol#179

Description

In **ConnextHandler.sol**, in order to record a failed transaction it is necessary to know the next **nonce** in the **mapping**, this involves a **for** cycle with a storage read that expends **gas**. Since it only adds at the end of the list and access is always done using the **nonce**, the variable **_failedTxns** could be changed from a **mapping** of a **mapping** to the **mapping** of an **array**, this way in order to add a transaction **push()** could be used and to be able to know the nonce the **length** of the array would suffice. This way the **for** cycle wouldn't be needed and it would save on **gas**.

Recommendation

New declaration: `mapping(bytes32 => FailedTxn[]) private _failedTxns;`

Fetching the nonce: `uint128 nextNonce = _failedTxns[transferId].length;`

And adding to the list: `_failedTxns[transferId].push(...);`

Status

Addressed here: Fujicracy/fuji-v2#616

3S-FUJI-N07

Mismatching **calldata** in **_crossTransferWithCalldata(...)** and **xReceive(...)** in **ConnextRouter**

Id	3S-FUJI-N07
Classification	None
Severity	N/A
Likelihood	N/A
Category	Bug
Relevant Links	POC/EncodingCalldata.t.sol#L12

Description

Users can bridge assets by inserting an **Action.XTransferWithCall** action, with calldata specified in the **args** field. The calldata in **_crossTransferWithCalldata(...)** and **_getBeneficiaryFromCalldata(...)** is decoded as :

```
(Action[] memory actions, bytes[] memory args,) =
      abi.decode(callData, (Action[], bytes[], uint256));
```

Whereas in **xReceive(...)**, it lacks the last **uint256**:

```
(Action[] memory actions, bytes[] memory args) = abi.decode(callData,
(Action[], bytes[]));
```

Recommendation

Remove the last **uint256** parameter from being decoded, as it is not being used.

POC

https://github.com/threesigmaxyz/fuji-issues-external/blob/master/test/POC/EncodingCalldata.t.sol#L12

Status

Addressed here: Fujicracy/fuji-v2#651

3S-FUJI-N08

In **`ConnextHandler`**, **`executeFailedWithUpdatedArgs(...)`**, the whole **`tx`** is updated on storage if the **`try`** call succeeds

Id	3S-FUJI-N08
Classification	None
Severity	N/A
Likelihood	N/A
Category	Optimization
Relevant Links	routers/ConnextHandler.sol#L222-L223

Description

In the ConnextHandle, **`executeFailedWithUpdatedArgs(...)`**, the failed **`tx`** executed field is set to true. Currently it is loaded in memory and then written to storage completely, which is unnecessary and adds significant gas overhead.

Recommendation

Change the code below **`try connextRouter.xBundle(actions, args)`** to **`_failedTxns[transferId][nonce].executed = true;`** to save on storage writes.

Status

Addressed here: Fujicracy/fuji-v2#657

3S-FUJI-N09

When transferring tokens, if the amount is 0, the transfer should be skipped

Id	3S-FUJI-N09
Classification	None
Severity	N/A
Likelihood	N/A
Category	Optimization
Relevant Links	abstracts/BaseRouter.sol#L120 abstracts/BaseRouter.sol#L523 abstracts/BaseRouter.sol#L542 abstracts/BaseFlasher.sol#L112 abstracts/BaseVault.sol#L537 abstracts/BaseVault.sol#L587 borrowing/BorrowingVault.sol#L775 borrowing/BorrowingVault.sol#L791

Description

Before each transfer call, if the amount is 0, it should be skipped. This is also true for **`SafeERC20.safeTransfer(...)`** and **`SafeERC20.safeTransferFrom(...)`**.

Recommendation

An example would be

```
function _safeTransferETH(address receiver, uint256 amount) internal
{
  if (amount == 0) return;
  (bool success,) = receiver.call{value: amount}(new bytes(0));
  if (!success) revert BaseRouter__safeTransferETH_transferFailed();
}
```

Status

Addressed here: Fujicracy/fuji-v2#652

3S-FUJI-N10

xBundle(...) and **xReceive(...)** should have **nonReentrant** modifiers

Id	3S-FUJI-N10
Classification	None
Severity	N/A
Likelihood	N/A
Category	Suggestion
Relevant Links	facets/BaseConnextFacet.sol#L51 facets/BaseConnextFacet.sol#L51 abstracts/BaseRouter.sol#L102 routers/ConnextRouter.sol#L131

Description

xBundle(...) and **xReceive(...)** make several external calls and check the balances before and after these calls. It's safer to include nonReentrant modifiers to prevent reentrancy.

Recommendation

Check the Connext implementation for an example. Using non zero state variables as mutex saves gas, because the second write is to the same slot as the first write. Add a check such that if **msg.sender** is the flasher, it can reenter.

Status

Fujicracy/fuji-v2#622

3S-FUJI-N11

_to argument missing **0x0** address check in the **ConnextRouter**

Id	3S-FUJI-N11
Classification	None
Severity	N/A
Likelihood	N/A
Category	Suggestion
Relevant Links	routers/ConnextRouter.sol#L275 routers/ConnextRouter.sol#L324

Description

The **ConnextRouter** does not check, in the **_crossTransfer(...)** and **_crossTransferWithCalldata(...)** functions, if the **receiver** and **routerByDomain[destDomain]** equal **address(0)**. This should revert anyway because the Connext contract reverts if **_to** is **address(0)**, but it's safer to make this additional check.

Recommendation

Check if the address is 0.

Status

Addressed here: Fujicracy/fuji-v2#632

3S-FUJI-N12

`_checkNoBalanceChange(...)` cycle should `break` in `BaseRouter`

Id	3S-FUJI-N12
Classification	None
Severity	N/A
Likelihood	N/A
Category	Optimization
Relevant Links	abstracts/BaseRouter.sol#L662

Description

`_checkNoBalanceChange(...)` has a `for` cycle that goes through all tokens. For each token it checks if they are `address(0)`, this is only true if we have arrived at the end of the list of tokens, therefore we can break the cycle to save on gas.

Recomendation

Add a break to the `for` cycle to save gas.

Status

Addressed here: Fujicracy/fuji-v2#627

3S-FUJI-N13

`_handleSwapAction(...)` creates situations where the arguments receiver and sweeper need to be the same address in **`BaseRouter`**

Id	3S-FUJI-N13
Classification	None
Severity	N/A
Likelihood	N/A
Category	Documentation
Relevant Links	abstracts/BaseRouter.sol#L506 abstracts/BaseRouter.sol#L510

Description

In **`_handleSwapAction(...)`** the function checks if both the receiver and sweeper are the beneficiary given certain conditions. Since there can only be one beneficiary there are situations where a user can't name the receiver and sweeper as two different addresses.

Recommendation

This should be explained as expected behavior in the documentation or in comments.

Status

Addressed here: Fujicracy/fuji-v2#626

3S-FUJI-N14

Constants should be placed as constants and not hardcoded for better readability

Id	3S-FUJI-N14
Classification	None
Severity	N/A
Likelihood	N/A
Category	Good Code Practices
Relevant Links	N/A

Description

Hardcoded values are hard to track and constants should be used instead.

Recommendation

For example in **`BorrowingVault`**, **`1e18`** is used several times for precision, which could be set as a constant such as **`uint256 private constant PRECISION_CONSTANT = 1e18`**

Status

Addressed here: Fujicracy/fuji-v2#625

3S-FUJI-N15

_tempTokenToCheck in **BaseRouter** does not need to be a state variable

Id	3S-FUJI-N15
Classification	None
Severity	N/A
Likelihood	N/A
Category	Optimization
Relevant Links	abstracts/BaseRouter.sol#L89 abstracts/BaseRouter.sol#L102 abstracts/BaseRouter.sol#L140 routers/ConnextRouter.sol#L151-L153

Description

_tempTokenToCheck checks that the balance of the **ConnextRouter** does not change after having received tokens in **xReceive(...)** . This variable is unnecessary as the **tokensToCheck** variable of **_bundleInternal(...)** could be created before the **_bundleInternal(...)** call and passed as argument.

Recommendation

In the ConnextRouter, create the **tokensToCheck** variable in the **xReceive(...)** function and pass it as argument to **xBundleConnext(...)**, which should pass it as argument to **_bundleInternal(...)**. In the BaseRouter, create the **tokensToCheck** variable in **xBundle(...)** and pass it as argument to **_bundleInternal(...)**.

Status

Addressed here: Fujicracy/fuji-v2#622

3S-FUJI-N16

Unnecessary extra condition in if statement

Id	3S-FUJI-N16
Classification	None
Severity	N/A
Likelihood	N/A
Category	Optimization
Relevant Links	

Description

On the **`BorrowingVault.sol`** contract in the function **`_withdraw(...)`** the **`if`** statement checks the following conditions:

```
if (totalDebt_ == 0 && supply > 0 && supply > totalDebt_)
```

If the first two conditions are true the third will always be true, therefore there is no need to check it.

Recommendation

Remove the third check in the **`if`** statement.

Status

Addressed here: Fujicracy/fuji-v2#625

3S-FUJI-N17

Useless else statement

Id	3S-FUJI-N17
Classification	None
Severity	N/A
Likelihood	N/A
Category	Good Code Practices
Relevant Links	routers/ConnextRouter.sol#L151-L153 vaults/borrowing/BorrowingVault.sol#L163 vaults/yield/YieldVault.sol#L64 vaults/borrowing/BorrowingVault.sol#L204-L207

Description

If a call reverts in the first **`if`** statement, the following **`else`** is not required.

Recommendation

Replace

```
if (balance < amount) {
  revert ConnextRouter__xReceive_notReceivedAssetBalance();
} else {
  _tempTokenToCheck = Snapshot(asset, balance - amount);
}
```

with

```
if (balance < amount) {
  revert ConnextRouter__xReceive_notReceivedAssetBalance();
}
_tempTokenToCheck = Snapshot(asset, balance - amount);
```

Status

Addressed here: Fujicracy/fuji-v2#625 and Fujicracy/fuji-v2#622

3S-FUJI-N18

Saving parameters in memory without using them spends gas

Id	3S-FUJI-N18
Classification	None
Severity	N/A
Likelihood	N/A
Category	Optimization
Relevant Links	abstracts/BaseVault.sol/#L136 abstracts/BaseVault.sol/#L160 abstracts/BaseVault.sol/#L178 abstracts/BaseVault.sol/#L194

Description

In **`BaseVault.sol`** several functions receive a variable as a parameter then save it in memory and immediately use it to make a call without making any changes. This is most likely done to increase readability however since there is no reason to save it in memory, this is spending unnecessary gas.

```
  function increaseAllowance(address receiver, uint256 shares) public
override returns (bool) {
    address operator = receiver;
    increaseWithdrawAllowance(operator, receiver,
convertToAssets(shares));
    return true;
  }
```

Recommendation

In the mentioned functions in the links, remove the variable in memory being created and pass the variable directly to the call. In the example above, **`address operator`**.

Status

Addressed here: Fujicracy/fuji-v2#625

3S-FUJI-N19

Don't return the same memory variable if your passing it as argument

Id	3S-FUJI-N19
Classification	None
Severity	None
Likelihood	
Category	Good Code Practices
Relevant Links	abstracts/BaseRouter.sol#L192

Description

In BaseRouter, **`_bundleInternal(...)`**, the **`tokensToCheck`** array is updated in **`tokensToCheck = _addTokenToList(token, tokensToCheck);`**. In **`_addTokenToList(...)`**, the token is added to the array if it is not present yet, which modifies the **`tokensToCheck`** in the outer scope of the function; there is no need to return it - when a variable is passed as memory to an internal function, the same memory location is used, solidity does not create a copy.

Recommendation

Change **`_addTokenToList(...)`** to not return anything.

Status

Addressed here: Fujicracy/fuji-v2#622