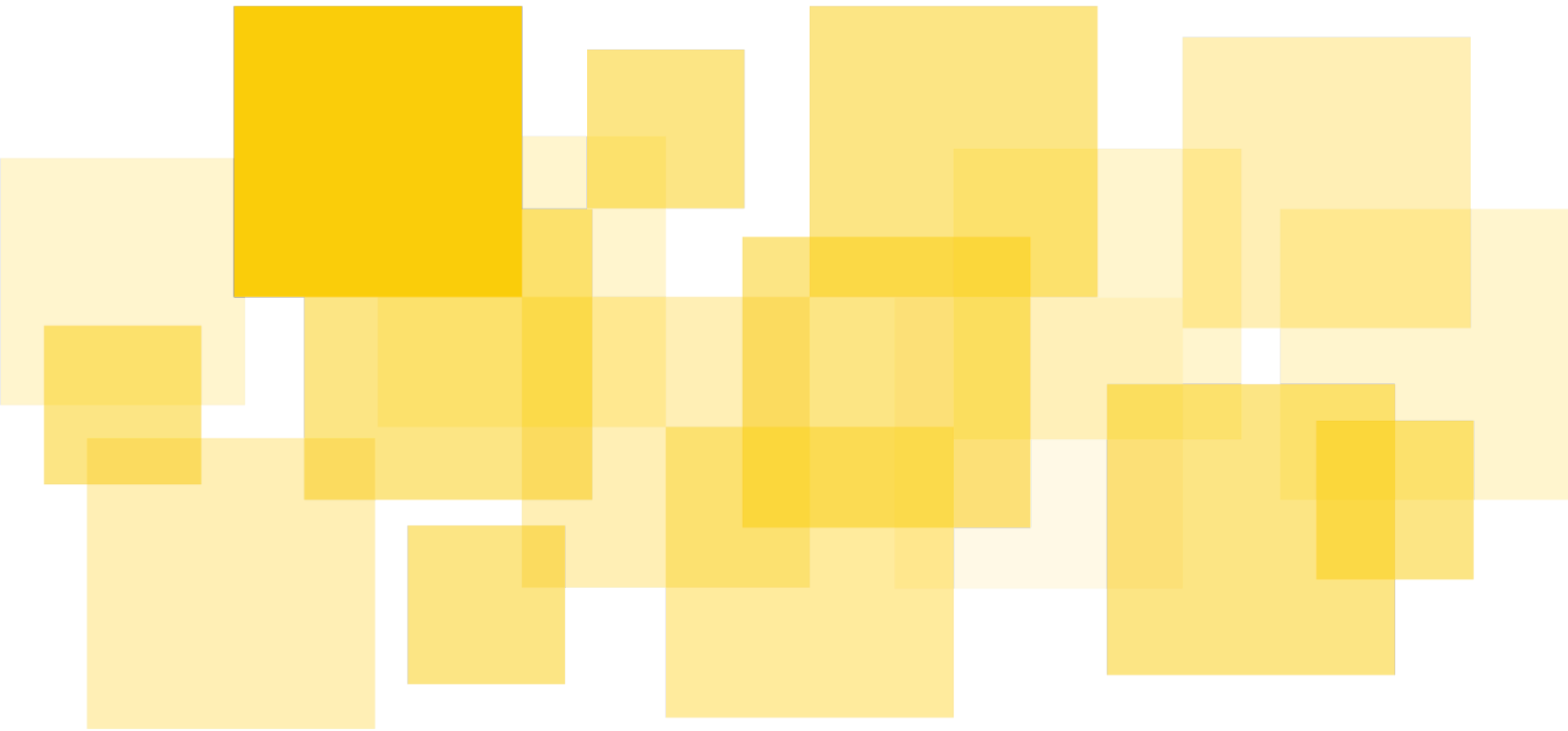
Audit Report

Synonym

Delivered: 2023-22-12



Prepared for Synonym by Runtime Verification, Inc.

Table of Contents

Summary

Runtime Verification, Inc. has audited the smart contract source code of the Synonym project. The review was conducted from 11-06-2023 to 12-22-2023.

Synonym engaged Runtime Verification in checking the security of their cross-chain borrow and lend protocol implemented over an Hub-Spoke model. Users in the protocol have vaults that keep the accounting of the deposited-borrowed assets so far, and accrue interest for the collateral that is deposited and borrowed.

The issues which have been identified can be found in the sections titled Findings and Informative Findings. Issues addressed by the client are identified accordingly with the relevant fixed commit provided.

Scope

The audited smart contracts are:

- contracts
  - InterestRateCalculator
    - BaseInterestRate.sol
    - LinearInterestRate.sol
    - PiecewiseInterestRate.sol
  - lendingHub
    - AssetRegistry.sol
    - Hub.sol
    - HubHelperViews.sol
    - HubInterestUtilities.sol
    - HubPriceUtilities.sol
    - HubState.sol
  - lendingSpoke
    - Spoke.sol
    - SpokeGetters.sol
    - SpokeState.sol
    - SpokeUtilities.sol
  - liquidationCalculator
    - LiquidationCalculator.sol
  - Wormhole
    - TokenReceiverWithCCTP.sol
  - HubSpokeEvents.sol
  - HubSpokeStructs.sol
- interfaces
  - IAssetRegistry.sol
  - IERC20decimals.sol
  - IHub.sol

    - `IHubPriceUtilities.sol`
    - `InterestRateCalculator.sol`
    - `ILiquidationCalculator.sol`
- `libraries`
    - `Disclaimer.sol`
    - `Interest.sol`
- `token`
    - `DelegateAddress.sol`
    - `rCT.sol`
    - `RewardsDistributor.sol`
    - `SYNO.sol`
    - `TokenConverter.sol`
    - `tSYNO.sol`
    - `vlSyno.sol`

The audit has focused on the above smart contracts, and has assumed correctness of the libraries and external contracts they make use of. The libraries are widely used and assumed secure and functionally correct.

The review encompassed the `SynonymFinance/smart-contracts` private code repository. The code was frozen for review at commit `fa7ae618a777c9ccad21ffb821702bf28082f5f4` there has been a number of updates with major changes to the codebase that included fixes for the findings during the audit; these updates were reviewed as well.

The review is limited in scope to consider only contract code. Off-chain and client-side portions of the codebase are *not* in the scope of this engagement. .

Assumptions

The audit assumes that all addresses assigned a role must be trusted for as long as they hold that role. Apart from the deployer that is responsible to create and deploy the contracts, an `owner` role (see OpenZeppelin's access control) is present for some of the contracts in the protocol.

The admin addresses of the respective contracts need to be absolutely trusted. We assume that the deployers and the governance take relevant steps to ensure that the state of the deployed contracts remains correct. In addition to setting the correct state, it is also contingent upon governance to maintain a reasonable state. This includes only accepting trustworthy tokens and setting protocol parameters honestly.

Note that the assumptions roughly assume "honesty and competence". However, we will rely less on competence, and point out wherever possible how the contracts could better ensure that unintended mistakes cannot happen.

Methodology

Although the manual code review cannot guarantee to find all possible security vulnerabilities as mentioned in Disclaimer, we have used the following approaches to make our audit as thorough as possible. First, we rigorously reasoned about the business logic of the contract, validating security-critical properties to ensure the absence of loopholes in the business logic and/or inconsistency between the logic and the implementation. Second, we carefully checked if the code is vulnerable to known security issues and attack vectors. Finally, we periodically met with the Synonym team to provide feedback and suggested development practices and design improvements.

This report describes the **intended** behavior of the contracts under review, and then outlines issues we have found, both in the intended behavior and in the ways the code differs from it. We also point out lesser concerns, deviations from best practice and any other weaknesses we encounter. Finally, we also give an overview of the important security properties we proved during the course of the review.

Disclaimer

This report does not constitute legal or investment advice. The preparers of this report present it as an informational exercise documenting the due diligence involved in the secure development of the target contract only, and make no material claims or guarantees concerning the contract's operation post-deployment. The preparers of this report assume no liability for any and all potential consequences of the deployment or use of this contract.

Smart contracts are still a nascent software arena, and their deployment and public offering carries substantial risk. This report makes no claims that its analysis is fully comprehensive, and recommends always seeking multiple opinions and audits.

This report is also not comprehensive in scope, excluding a number of components critical to the correct operation of this system.

The possibility of human error in the manual review process is very real, and we recommend seeking multiple independent opinions on any claims which impact a large quantity of funds.

Synonym: Protocol Overview

Synonym uses an Hub and Spoke model to provide cross-chain lending and borrowing to its users. During its operation protocol uses Wormhole to provide cross-chain transfers and Pyth oracle to retrieve asset prices. Users of the system provide assets in a Spoke deployed in the source chain to be used as collateral. Collateral assets are then transferred to the Hub in Hub's chain and accounted for by the "vault" of the user. Users may also borrow from the protocol depending on the status of their vaults; if their vaults have a healthy balance in terms of deposited and borrowed assets users can borrow from a completely different chain.

Protocol offers depositing collateral and withdrawing collateral as well as borrowing and repaying debt. Each operation can either be done with cross-chain tokens or with the native tokens of the Spoke's chain. Additionally, protocol also allows users to liquidate other users if the other user's borrowed assets are worth more than their deposited assets.

Protocol may also accrue interest for the deposited and borrowed assets. Deposited collaterals accrue interest over time, letting the user withdraw more assets than the deposited amount for positive interest rates. On the other hand, borrowed assets accrue interest in a complementary way. For positive values, the amount needed to be repaid grows according to the interest rate. Basis interest rate model used by the protocol takes into consideration the amount of deposited and borrowed assets for the protocol as well as the deposited and borrowed assets per vault. Protocol offers a linear interest rate model by default but a piecewise model is also implemented for future use.

Protocol also keeps a collateralization ratio for each vault in order to check the health of vaults. This ratio is employed to provide leeway for the fluctuations in the asset prices and the total amount of borrows of the vault with respect to the total amount of collateral deposited to the vault. If this ratio goes over a certain threshold the vault is considered unhealthy and deemed "underwater" restricting borrowing of additional assets by the vault and even making it possible to liquidate the vault.

Additional in-depth documentation of the protocol is provided by the Synonym in their documentation web-site.

Findings

Even though the standard procedure is to only audit a frozen code commit; for this audit, it was possible to review the changes that made the old code obsolete to ensure the code deployed in mainnet is one closer to the one audited.

Not only all the commits related with addressing the findings were reviewed, but also commits related with major refactoring changes and adding new functionalities were reviewed.

The last commit subject of review can be reached by the commit id [c84ae1d84ec703a465e225757637f43abc5af37d]

A01: Hub - transferring ownership makes it not possible to register a new Spoke

[Severity: Medium | Difficulty: High | Category: Functionality]

The function `registerSpoke` has the modifier `onlyOwner` and calls `setRegisteredSender`, from `WormholeRelayerSDK`. The `registrationOwner` of the `Base` contract in `WormholeRelayerSDK` is the deployer of the `Hub` - see `__Base_init` function. Only the `registrationOwner` of the `Base` contract can call `setRegisteredSender`, but there is no function that allows to change the `registrationOwner` of the `Base` contract. Therefore, if the ownership of the `Hub` changes, the new `owner` cannot register a new spoke, since the `registrationOwner` of the `Base` contract remains the first `owner` of the `Hub`.

Scenario

1. Address `A` deploys the `Hub` and becomes the `owner` of the `Hub` and the `registrationOwner`;
2. Address `A` calls `transferOwnership(B)`, making address `B` the `owner` of the `Hub`;
3. Neither address `A` or address `B` can call the `registerSpoke` function, since the `registerSpoke` requires that `onlyOwner` (address `B`) can call it, and the `setRegisteredSender` requires that only `registrationOwner` (address `A`) can call it.

Recommendation

Override the function `transferOwnership` to also change the `registrationOwner` of the `Base` contract to the new `owner`.

Status

Addressed in commit f9b34ab56d6621cf991c93e6e44ebc74548597ff

A02: Hub - no upper limit on the values for `liquidationFee`

[Severity: Low | Difficulty: High | Category: Input Validation]

There is no upper limit on the value being assigned to the `liquidationFee` in the `initialize` and `setLiquidationFee` functions.

```
function setLiquidationFee(uint256 _liquidationFee) external onlyOwner {
        _state.liquidationFee = _liquidationFee;
        emit SetLiquidationFee(_liquidationFee);
    }
```

Scenario

1. The `liquidationFee` is set to a value greater than `collateralizationRatioPrecision`
2. The `liquidation` function will always revert:

```
uint256 liquidationFee = getLiquidationFee();
uint256 precision = getCollateralizationRatioPrecision();
for (i = 0; i < input.assetReceiptAddresses.length;) {
    uint256 feePortion = (input.assetReceiptAmounts[i] * liquidationFee) /
precision;
    uint256 amountToTransfer = input.assetReceiptAmounts[i] - feePortion;
```

Recommendation

Check that the value being assigned to the `liquidationFee` is less or equal to the `collateralizationRatioPrecision`. Ideally set a reasonable upper limit for the `liquidationFee` variable.

Status

Addressed in commit 8418bbd8d2b4888d4a7cfbbd17f16b7dc6cde60d and in commit [eb07314ce8412e5ae04a5b4c7eb8b847e7ce25db](eb07314ce8412e5ae04a5b4c7eb8b847e7ce25db)

A03: `BaseInterestRate` - `reserveFactor` does not have upper limit

[Severity: Low | Difficulty: High | Category: Input Validation]

There is no upper limit on the value being assigned to the `reserveFactor` variable in the `BaseInterestRate` contract.

```
constructor(uint256 _reserveFactor, uint256 _reservePrecision) Ownable(msg.sender)
{
      reserveFactor = _reserveFactor;
      reservePrecision = _reservePrecision;
}

function setReserveFactorAndPrecision(uint256 _reserveFactor, uint256
_reservePrecision) external onlyOwner {
      reserveFactor = _reserveFactor;
      reservePrecision = _reservePrecision;
}
```

Scenario

1. The `reserveFactor` is set to a value greater than `reservePrecision`;
2. The function `Hub::getCurrentAccrualIndices` will always revert:

```
(uint256 interestFactor, uint256 reserveFactor, uint256 reservePrecision) =
assetCalculator.computeSourceInterestFactor(secondsElapsed, deposited, borrowed,
getInterestAccrualIndexPrecision());
            accrualIndices.borrowed += interestFactor;
            accrualIndices.deposited +=
                (interestFactor * (reservePrecision - reserveFactor) * borrowed) /
reservePrecision / deposited;
```

Recommendation

Check that the value being assigned to the `reserveFactor` is less or equal to the `reservePrecision`. Ideally set a reasonable upper limit for that variable.

Status

Addressed in commit [e3138d3efd4234094327c2d35ed7a0363eb40036]

A04: `LiquidationCalculator::calculateNotionalRepaid` and `LiquidationCalculator::calculateNotionalReceived` denormalize amount inputs

[Severity: Medium | Difficulty: Low | Category: Implementation Flaw]

In the `LiquidationCalculator` contract, in order to calculate the notional repaid and received, the functions `calculateNotionalRepaid` and `calculateNotionalReceived` call the `calculateNotionals` function, which denormalize the amount received as parameter. However, the argument should not be denormalized, since the inputs for the `liquidation` function are not normalized.

```
function calculateNotionals(
        address asset,
        VaultAmount memory vaultAmount
    ) public view returns (VaultAmount memory) {
        AssetInfo memory assetInfo = getAssetInfo(asset);
        VaultAmount memory denormalized = denormalizeVault(asset, vaultAmount);
        (,uint256 priceCollateral, uint256 priceDebt,) = getPrices(asset);
        uint256 expVal = 10 ** (getMaxDecimals() - assetInfo.decimals);

        return VaultAmount(
            denormalized.deposited * priceCollateral * expVal,
            denormalized.borrowed * priceDebt * expVal
        );
}
```

Recommendation

Refactor the `calculateNotionals` function to only denormalize the `vaultAmount` received as parameter in specific cases, perhaps with a flag set to `true` when denormalize is needed and set to `false` when denormalize should be skipped.

Status

Addressed in commit 8418bbd8d2b4888d4a7cfbbd17f16b7dc6cde60d

A05: Hub - no check that `depositNative` is allowed in `userActions`

[Severity: Low | Difficulty: Low | Category: Protocol Invariants]

In `Hub::userActions`, for every action, there is a check to ensure that the action is allowed to be performed, except for the `depositNative` action. This allows a user to `depositNative` even if the conditions to do so are not met.

Recommendation

Add a check that `depositNative` is allowed.

Status

Addressed in commit 44304c783a06d84ac6feaf48d2b3cc87d004f969.

A06: `HubHelperViews::calculateMaxWithdrawableAndBorrowableAmounts` denormalize amount inputs

[Severity: Low | Difficulty: Low | Category: Implementation Flaw]

The function `HubHelperViews::calculateMaxWithdrawableAndBorrowableAmounts` call the `calculateNotionals` function, which denormalize the amount received as parameter. However, the inputs for the `calculateMaxWithdrawableAndBorrowableAmounts` are already denormalized, therefore the argument should not be denormalized again.

Recommendation

Do not denormalize the `vaultAmount` received as a parameter in the function `calculateNotionals`. See A04.

Status

Addressed in commit 8418bbd8d2b4888d4a7cfbbd17f16b7dc6cde60d

A07: `Spoke::registrationOwner` can change the `hubContractAddress`

[Severity: High | Difficulty: High | Category: Security]

The `registrationOwner` of the `Spoke` can call `setRegisteredSender` and change the registered sender of the hub chain - other than the `hubContractAddress` - or register a new sender for another chain.

This makes it possible for the `Spoke` to process messages from a different chain and from a different contract address than the `Hub` in the `receivePayloadAndTokens` and `receivePayloadAndUSDC` functions.

Recommendation

If this behavior should not be allowed then make the `Spoke` contract `Ownable`, and after deployment change the `registrationOwner` to `address(0)`, so it is not possible to call the `setRegisteredSender` anymore.

Status

Addressed in [43e0b37d05b6a5ee8c313438111b93be7351f029]

A08: `Spoke` - wrong refund chain parameter

[Severity: Medium | Difficulty: Low | Category: Implementation Flaw]

The refund chain parameter in `_doAction`, `_sendTokenBridgeMessage`, `_sendTokenBridgeMessageNative` and `_sendUSDCWithPayload` is the `hubChainId()`. However, the refund chain should be the `chainId()` of the `Spoke`.

Recommendation

In the above-mentioned functions replace the refund chain parameter with `chainId()`.

Status

Addressed in commit [61749859153e18e6266d8c1b1e4d6a791688db1f]

A09: `Spoke::depositColatteral` and `repay` do not check that `msg.value` is enough to pay the relayer

[Severity: Medium | Difficulty: Low | Category: Security]

In the `Spoke`, the functions `depositCollateral` and `repay` do not have the following check:

```
uint256 totalCost = getDeliveryCostRoundtrip(costForReturnDelivery, true);
require(msg.value >= totalCost, "Spoke::depositCollateral:Insufficient value
sent");
```

The consequence of this is that, if the `Spoke` holds any eth, the transaction does not revert but it will the `Spoke` to pay the relayer, since the functions `_sendTokenBridgeMessage` and `_sendUSDCWithPayload` call `sendTokenWithPayloadToEvm` and `sendUSDCWithPayloadToEvm`, respectively. These functions calculate the cost necessary to relay the message and provide that cost as the `msg.value` to the `wormholeRelayer`, regardless that value was provided in the `msg.value` or not.

```
(uint256 cost,) = wormholeRelayer.quoteEVMDeliveryPrice(targetChain,
receiverValue, gasLimit);
return wormholeRelayer.sendVaasToEvm{value: cost}(...)
```

Recommendation

Add the above mentioned check to the `Spoke` functions `depositCollateral` and `repay`.

Status

Addressed in commit [43e0b37d05b6a5ee8c313438111b93be7351f029]

A10: `Hub` may be left without an `owner` losing important functionality

[Severity: High | Difficulty: High | Category: Functionality]

The `owner` can call `renounceOwnership` or can call `transferOwnership` to an invalid address, which leaves the contract without an `owner`, thus removing any functionality that is available only to the `owner`.

```
function renounceOwnership() public virtual onlyOwner {
        _transferOwnership(address(0));
}
```

Scenario

Scenario 1: The `owner` calls the `renounceOwnership` function, which can only be called by the current `owner`. As the contract no longer has an `owner`, it is impossible to call the `onlyOwner` functions anymore.

Scenario 2: The `owner` calls the `transferOwnership` function, which can only be called by the current `owner`, giving as a parameter an invalid address. As the contract no longer has an `owner`, it is impossible to call the `onlyOwner` functions anymore.

Recommendation

Although we assume the `owner` is honest and competent, we want to clarify that this undesired behavior is possible. The `renounceOwnership` should be overriden and always revert to avoid this scenario. The `transferOwnership` should be a two-step process where the new `owner` has to accept the ownership (see Ownable2Step from OpenZeppelin).

Status

The `renounceOwnership` was addressed in commit f9b34ab56d6621cf991c93e6e44ebc74548597ff. Implementation of `transferOwnership` via a two step process was decided to be considered at a later stage by the Synonym team.

A11: Hub - does not check `msg.value` to refund tokens

[Severity: High | Difficulty: Medium | Category: Security]

The function `Hub::handleTokenTransfer` checks if the protocol should transfer or refund tokens to the user. If the `shouldTransfer` flag is `true`, then the `valid` flag is `true,` which implies that the `msg.value` is enough to pay the token transfer to the user. However, when the `shouldRefund` flag is `true,` then the valid flag is `false,` and there is no check that ensures that the `msg.value` is enough to pay the refund to the user. This means that the cost to refund users for an invalid `Deposit` or `Repay` might be paid by the `Hub` reserves, potentially leading to protocol losses.

Scenario

1. A malicious user calls numerous times an invalid `Repay` on the `Spoke`, giving `0` as the parameter `costForReturnDelivery`
2. When the `Hub` is processing the `payload,` it does not check that the `msg.value` - which is `0` in this case - is enough to refund tokens.
3. When the `Hub` sends the tokens back to the user, the `cost` paid to the relayer will be supported by the `Hub`.
4. The malicious user can call the invalid `Repay` action enough times to drain all the eth from the Hub.

Recommendation

Add a check that ensures that the msg.value is enough to pay the refund to the user. For instance:

```
if (shouldTransfer || shouldRefund) {
    if (msg.value < getCostForReturnDelivery(sourceChain)) {
        revert InsufficientMsgValue();
    }
}
```

Status

Addressed in commit [43e0b37d05b6a5ee8c313438111b93be7351f029]

A12: Hub - refund tokens may revert

[Severity: High | Difficulty: Low | Category: Security]

The function `Hub::handleTokenTransfer` has the following `if` condition, where the `else` branch is reached for a valid `Deposit` or `Repay`, so the protocol refunds the user the cost for the round trip.

```
if (shouldTransfer || shouldRefund) {
    ...
} else {
    wormholeRelayer.sendToEvm{value: msg.value}(
        ...
        msg.value - getCostForRefundDelivery(sourceChain), // additional value sent
in Hub chain native currency
        ...
    );
}
```

However, if `msg.value` is less than `getCostForRefundDelivery` then the function reverts, and neither the user get its tokens back neither the hub state gets updated with his action.

Scenario

1. A user makes a valid `Deposit` on the `Spoke` but provides as the parameter a value less than `Hub::getCostForRefundDelivery(sourceChain).`
2. The `Hub` processes the `payload,` and when refunding the user, since the `msg.value` is less than `getCostForRefundDelivery(sourceChain),` the function reverts.
3. The user is not refunded with the value of his deposit and since the function reverts his deposit will not be stored in the hub.

Recommendation

Add a check that ensures that the msg.value is enough to pay the refund to the user. For instance:

```
} else {
    if (msg.value >= getCostForRefundDelivery(sourceChain)) {
        wormholeRelayer.sendToEvm{value: msg.value}(...)
    }
}
```

Status

Addressed in commit [43e0b37d05b6a5ee8c313438111b93be7351f029]

A13: Possibility of integer overflow during downcasting

[Severity: Medium | Difficulty: Medium | Category: Overflow]

Following check in the `HubPriceUtilities::getOraclePrices` function negates and downcasts an integer value provided by the oracle.

```
uint256 oraclePrecision = uint256(10 ** uint64(uint32(-oraclePrice.expo)));
```

However, this value is not checked against being positive, which will result in an unchecked overflow when casting.

Recommendation

Check if the `oraclePrice.expo` is negative.

Status

Addressed in commit [d8ef980a92bddc448c2d6a0fe90ca545378981fe]

A14: Transfer to address(0) for burning tokens

[Severity: Low | Difficulty: Low | Category: Implementation Flaw]

Following call in the `TokenConverter::_convert` function makes a `safeTransferFrom` call that uses `address(0)` as destination to burn tokens. This call will revert for ERC20 tokens.

```
newo.safeTransferFrom(recipient, address(0), amount);
```

Recommendation

A proper burn method should be used to burn tokens.

Status

Addressed in commit [43e0b37d05b6a5ee8c313438111b93be7351f029]

A15: `totalSupply` not updated while burning tSYNO

[Severity: Low | Difficulty: Low | Category: Implementation Flaw]

While unstaking tSYNO the unstaked amount of tokens are burned from the unstaker's account. However, tSYNO's total supply variable is not decreased, causing an error in the token accounting.

Recommendation

The `totalSupply` value should also be decreased by the unstaked amount.

Status

Addressed in commit [43e0b37d05b6a5ee8c313438111b93be7351f029]

A16: Unstaking vlSYNO does not follow checks-effects-interactions

[Severity: High | Difficulty: High | Category: Best Practices]

`vlSYNO`'s `unstake` function performs accounting (token burning) operations after it makes a transfer to an external contract. In case of a re-entrancy attack, this situation may result in unsafe operations.

Recommendation

Move the following transfer to the end of the function.

```
IERC20(poolToken).transfer(msg.sender, _stake.amount);
```

Status

Addressed in commit [43e0b37d05b6a5ee8c313438111b93be7351f029]

A17: It is possible to set unreasonable staking periods

[Severity: Low | Difficulty: High | Category: Input Validation]

In `tSYNO` contract, the only sanity check being performed for `stakingPeriodStart` and `stakingPeriodLength` values except being `0` (Some functions also provide proper values when `stakingPeriodStart` is before `block.timestamp`). Protocol allows setting the staking period start to a far future date or setting -practically- an infinite staking period.

Recommendation

Apply range checks for both of these variables.

Status

Not addressed as these values are assumed to be set carefully by the owner during the initialization of the contract.

Informative Findings

B01: Hub - redundant decimals check

[Severity: - | Difficulty: - | Category: Gas Optimization]

In `Hub.sol` contract, the function `registerAsset` checks that the `decimals` of the registered asset are not greater than `PROTOCOL_MAX_DECIMALS`. However, the same check is done in the function `registerAssetInfo`, which is called by the `registerAsset` function.

```
if(decimals > PROTOCOL_MAX_DECIMALS) revert TooManyDecimalsInAnAsset();
```

Recommendation

One of the checks can be suppressed.

Status

Addressed in commit [3ea31a7148530f0fae9438ac5c8219601eef6b48](3ea31a7148530f0fae9438ac5c8219601eef6b48).

B02: Hub - redundant check when updating vault state

[Severity: - | Difficulty: - | Category: Gas Optimization]

In `Hub.sol` contract, the function `userActions` uses the return value of `allowedToRepay` function to check if a call to `_updateVaultAmounts` is required. This check is redundant since the function immediately reverts when this value is `false`. The variable `completed` is not used anywhere else in the function making the related check unnecessary.

```
function userActions(Action action, address asset, uint256 amount) public payable
                                                                    whenNotPaused {

…
   } else if (action == Action.Repay || action == Action.RepayNative) {
      (completed,) = allowedToRepay(msg.sender, asset, amount);
      if (!completed) {
         revert RepayNotAllowed();
      }
…
   if (completed) {
      _updateVaultAmounts(action, msg.sender, asset, amount);
   }
```

Recommendation

The check before the `_updateVaultAmounts` call can be removed.

Status

Addressed in commit [3ea31a7148530f0fae9438ac5c8219601eef6b48](3ea31a7148530f0fae9438ac5c8219601eef6b48).

B03: Hub - compute values only when needed

[Severity: - | Difficulty: - | Category: Gas Optimization]

The functions `checkAllowedToWithdraw` and `checkAllowedToBorrow` compute the values of `vaultDepositedValue` and `vaultBorrowedValue`. However, those values are only used if the two functions called after the computation - `checkVaultHasAssets` and `checkProtocolGloballyHasAssets` return `success == true`.

```
(uint256 vaultDepositedValue, uint256 vaultBorrowedValue) =
getVaultEffectiveNotionals(vaultOwner, true);
```

Recommendation

Compute the values only if they are needed.

Status

`checkAllowedToWithdraw` Addressed in commit [0fc0727e84aa2b3404a864600828481b0c979492](0fc0727e84aa2b3404a864600828481b0c979492)

`checkAllowedToBorrow  has not been addressed yet`

B04: Hub - use defined functions to improve readability

[Severity: - | Difficulty: - | Category: Refactoring]

The functions `getUserBalance` and `getGlobalBalance` return the denormalized balance of some asset of some user or of the protocol, respectively. Since the function, denormalizeVault is already defined, it can be used to improve readability and reduce lines of code.

Recommendation

Refactor the denormalizeVault to also receive the interest accrual indices as parameter. Use the `denormalizeVault` function to calculate the denormalized amounts to be returned.

```
function getUserBalance(address vaultOwner, address assetAddress)
        public
        view
        returns (HubSpokeStructs.VaultAmount memory)
    {
        HubSpokeStructs.VaultAmount memory normalized =
hub.getVaultAmounts(vaultOwner, assetAddress);
        HubSpokeStructs.AccrualIndices memory interestAccrualIndex =
hub.getCurrentAccrualIndices(assetAddress);
        return denormalizeVault(assetAddress, normalized, interestAccrualIndex);
    }
```

```
function getGlobalBalance(address assetAddress) public view
        returns (HubSpokeStructs.VaultAmount memory)
    {
        HubSpokeStructs.VaultAmount memory normalized =
hub.getGlobalAmounts(assetAddress);
        HubSpokeStructs.AccrualIndices memory interestAccrualIndex =
hub.getCurrentAccrualIndices(assetAddress);
        return denormalizeVault(assetAddress, normalized, interestAccrualIndex);
    }
```

Status

Addressed in commit 8418bbd8d2b4888d4a7cfbbd17f16b7dc6cde60d

B05: Hub - unused `spokeContracts` state variable

[Severity: - | Difficulty: - | Category: Refactoring]

The `Hub` state variable `spokeContracts` is not being used anywhere in the code. It should be used when registering a new `spoke` in the `Hub`, but the update to the variable is not being done. However, this variable is redundant since the list of the spokes registered can be accessed in `registeresSenders` in the `WormholeRelayerSDK`.

```
    // allowlist for spoke contracts
    mapping(uint16 => address) spokeContracts;
```

Recommendation

Delete the unused variable.

Status

Addressed in commit [d9ad36c5c86643cfde1e9086a56d74ba419442ad]

B06: Spoke - Simpler validity checks

[Severity: - | Difficulty: - | Category: Code understandability]

In `Spoke` contract, the definition of `withCCTP` variable can be moved towards the beginning of the function to make logical checks such as the following, in a more clean way.

```
if (asset != address(0) && (asset != USDC || !_state.isUsingCCTP))
```

Recommendation

Move definition of `withCCTP` variable to the beginning.

Status

Addressed in commit cb7278db22618b3281d9210d0d1b2aefe09baec1

B07: `LiquidationCalculator` - `getVaultEffectiveNotionals` called inside a loop

[Severity: - | Difficulty: - | Category: Gas Optimization]

The function `checkAllowedToLiquidate` calls the function `checkLiquidationPortion` inside a loop iterating over the assets given in the input. The function `checkLiquidationPortion` computes `getVaultEffectiveNotionals(vault, false)` which does not depend on the asset address.

Recommendation

It would be more efficient if `getVaultEffectiveNotionals(vault, false)` was computed outside of the loop just once, and given as a parameter to the `checkLiquidationPortion` function.

Status

Addressed in commit [4e5100bb7735f3ca7d99447c5401730a0e157b96]

B08: It is possible to stake 0 amount for vlSYNO

[Severity: - | Difficulty: - | Category: Input Validation]

`vlSYNO`'s `stake` function does not check if 0 amount is being staked.

Recommendation

Staking 0 amount should be prevented.

Status

Addressed in commit [43e0b37d05b6a5ee8c313438111b93be7351f029]

B09: Missing address(0) check for vault address in liquidation input validation

[Severity: - | Difficulty: - | Category: Input Validation]

`LiquidationCalculator::checkLiquidationInputsValid` function does not check if `address(0)` is provided as a vault address.

Recommendation

`address(0)` check should be performed.

Status

Addressed in commit [43e0b37d05b6a5ee8c313438111b93be7351f029]

B10: It is possible to restake an already unstaked vlSYNO

[Severity: - | Difficulty: - | Category: Protocol Invariants]

`vlSYNO`'s `restake` function is aimed to extend the staking period of a staked amount. The `unstake` function in token's contract simply deletes the related stake from a mapping that keeps track of the staked amounts and related information via indexes assigned by the token contract. However, the deletion resets the values in the related entries so it is still possible to access the keys in the mapping. Re-staking function uses the reset values instead of the originals.

Recommendation

Protocol aims only to extend the time for current stakes, so the staking of the unstaked tokens should be disabled. For this purpose re-staking should check if the entry to be re-staked is in reset state (such as the `amount` being 0).

Status

Addressed in commit [43e0b37d05b6a5ee8c313438111b93be7351f029]