Tribe Turbo Audit by Team2 (takeda)

Overall Critical/High risk findings

TT-001: `TurboSafe.sweep` allow to sweep directly `underlying` token from a Safe

Tools/Techniques: Manual
Difficulty+Impact: High

Details

Usually a Safe user would be able to withdraw underlying token form the Safe only via the ERC4626 `withdraw` function that would also burn the corrisponding `share` (`tsUnderlying`).

The sweep method is not checking that the `token` is different form the Safe's `asset` to prevent sweeping of deposited `asset` that will not burn the corrisponding vault's shares.

I can imagine that for any reason the same `underlying` token could be sent to the `Safe` itself so the user should be able to `sweep` that excess of `underlying` that the Safe have.

The user should be able to transfer at max `asset.balanceOf(safe) - assetsOf(user)`. The problem with that is if more than one user can manage the same `Safe`

Mitigation

Add an additional check to prevent the user to `sweep` the `underlying` instead of `withdraw` it from the vault.

Manage possible `sweepable` `underlying` that have been sent to the `Vault` for any reason (excess of `underlying` from `safe`'s `vault`)

TT-002: because of `TurboSafe.sweep` exploit, an insolvent DAO could sweep the `underlying` preventing `gib`

Tools/Techniques: Manual
Difficulty+Impact: High

Details

Because of the exploit

> *`sweep` allow to sweep directly `underlying` token from a Safe*

this means that after the DAO has sweeped the whole collateral deposited in the `Safe` the `Tribe` DAO will not be able to `gib` in case of an insolvent `Safe`.

This is possible because at the end of the `gib` the `Gibber` is trying to do

`asset.safeTransfer(to, underlyingAmount);` but there are no more `asset` to transfer and the transaction will revert.

Mitigation

Fix the `sweep` exploit allowing the DAO to sweep the `underlying` (collateral) deposited, it would also be good to revoke any auth from the DAO to access the `Safe`.

TT-003: after `TurboSafe.gib` insolvent DAO will still be able to `slurp`

Tools/Techniques: Manual
Difficulty+Impact: High

Details

After the debt has been repayed by Tribe the DAO (that was insolvent) is still be able to `slurp` interested accrued by the vaults.

Because part/all their collateral has been transferred to Tribe (to repay the debpt) the interested accrued by the vaults should be sent to the Tribe as a re-payment for the debt.

Mitigation

After `gib` update the `Clerk` `getCustomFeePercentageForSafe[safe]` for the insolvent `Safe` to `100%` and remove DAO (insolvent) from the `Safe` auth

TT-004: after `TurboSafe.gib` the DAO is still able to `less`, stealing funds (debt already repaid by Tribe) from vaults

Tools/Techniques: Manual
Difficulty+Impact: High

Details

The `gib` operation is repaying debt and transferring the collateral to an address (Tribe DAO?).
After the `gib` all `tsUnderlying`, `Safe` accounting, `Master` accounting and `Safe` boost to `vaults` are still unchanged.

The insolvent DAO can still call `less` with the max feiAmount withdrawable from each vault and steal funds from those vaults with a combo of less + sweep.

Calling `less` will also break the `vault`/`underlying` accounting sync between Safe and Master.

Let's see a scenario: DAO is insolvent and the Tribe DAO call `Gibber.impoundAll` that mint new Fei, repay debt and call `Safe.gib` redeeming `underlyingTurboCToken` and transferring `underlying` (asset) to a desired address.

At this point the insolvent DAO can in a single transaction:

- iterate over all boosted vaults and call `less(vault[index], maxFeiAmountFromVaultOfIndex)`

Inside the `less` function

1. the `getTotalFeiBoostedForVault[vault]` is updated
2. `totalFeiBoosted` is updated
3. fei are withdrawn from the vault and sent to the Safe
4. feiDebt will be 0
5. becasue `feiAmount > feiDebt` -> `feiAmount=feiDebt=0`
6. because now feiAmount = 0 no fei will be used to repay the borrow (because all the debt has been already paid by `gib`)
7. all the fei withdrawn from the vaults are now on the `Safe`
8. `master.onSafeLess` is called but feiAmount is 0 (so the Safe accounting has been updated but Master accounting will not because we pass 0 as feiAmount)

After all the vaults have been drained the DAO will call `sweep(myHotWallet, feiAddress, totalFeiDrainedFromVaults)` and steal all the Fei that were in the vaults

Mitigation

Afrer the `gib` revoke all the auth access to the `Safe` for the insolvent DAO.

I don't have atm a valid solution to correctly update `Master` and `Safe` accounting and burn `fei` that are in the `vaults`.

TT-005: `TurboSafe.gib` is not handling tsUnderlying

Tools/Techniques: Manual
Difficulty+Impact: High

Details

While it's true that after repaying the debt the `Gibber` seize the `Safe`'s `underlying` (`asset`), the corresponding amount of `tsUnderlying` is still in the wallet of the debtor and that could have value in the market (it could have a 1:1 value to the underlying). It could be possible that it can be sold in the market even if in reality it has no value because it would be not possible to redeem it for `underlying` given that the `Safe` has not more underlying to exchange with the `tsUnderlying`.

Mitigation

A possible solution would be to make a custom ERC4626 where the Tribe DAO can force burn the corrisponding seized tsTokens?

TT-006: after `TurboSafe.gib` protocol accounting are not correctly updated

Tools/Techniques: Manual
Difficulty+Impact: High

Details

While `boost`, `less` and `slurp` are correctly updating the internal `Safe` accounting variables and `Master` account values after the `Gib` operations those values are still not updated.

While it's true that in the `gib` operations, all `fei` are mintied freshly new to repay the debt, the borrowed `fei` are still in the `Safe`'s vault (that are not tracked directly in the `Safe`.

Not updating those informations prevent the shared state variables in `Master` to prevent new `boosting` for the the same `vaults` and `underlyings` used by the `Safe` that had the debt.

Mitigation

I don't think that with the current implementation there's an easy way to solve this issue. Probably the best thing to do is to track which `vaults` the `Safe` has `boosted` and for how many `fei` and "force" update `Safe` and `Master` variables burning those `fei` (that should be burned because minted during `gib`)

TurboMaster.sol

TT-007: change `TurboMaster` `underlying` nomenclature to `asset` following `TurboSafe` `underlying` issue (TT-011)

Tools/Techniques: Manual
Difficulty+Impact: Informational

Details

For the same reason `TurboMaster.sol` should change the nomenclature of `underlying` to `asset`

Mitigation

Change `underlying` to `asset` in all the code/natspec.

TT-008: `getAllSafes` is also returning the invalid (first one) safe

Tools/Techniques: Manual
Difficulty+Impact: Low risk

Details

The first Safe inside the `safes` array is added at `construction` time to prevent it by using the ID 0. If someone try to use the first safe (see all the `require` in the code) the functions will always revert.

The `getAllSafes` function does not explicity suggest to the extrnal contract/webapp that is using the function that the first safe of the array should not be used because alwyas invalid.

Not knowing this, they could interact with the contract in the wrong way using the invalid safe as input.

Mitigation

Update the `natspec` doc to document this behaviour or just return the safe array without the first element (invalid pool).

TT-009: `setGibber`, `setClerk` and `setBooster` should be under timelock

Tools/Techniques: Manual
Difficulty+Impact: Low risk

Details

Those functions that replace core modules should be placed under governance timelock because they are chaning core functionalities that will influence how the whole protocol works

Mitigation

Put those functions under governance timelock

TT-010: updating the `booster` wihtout also migrating previous `booster` limit for vaults/underlying could break future `boost`

Tools/Techniques: Manual
Difficulty+Impact: Low risk

Details

Be aware that when a `booster` get replaced it could revert future `Safe` `boost` if values from the old `booster` are not migrated to the new one.

Mitigation

When deploying a new `booster` remember to make sure that the new booster logic (that could no rely on the same limits/logic of the old one) should not break future `boost` of current enabled `Safe`

TurboSafe.sol

TT-011: change `underlying` nomenclature to `asset` following `ERC4626` standard

Tools/Techniques: Manual
Difficulty+Impact: Informational

Details

`TurboSafe` is extending `ERC4626`. Inside of `ERC4626` the asset that is managed by the Vault is called `asset`. The TurboSafe should inherit also this standard nomenclature and change `underlyingTurboCToken` to `assetTurboCToken` in all the code/natspec.

Mitigation

Change `underlyingTurboCToken` to `assetTurboCToken` in all the code/natspec.

TT-012: on `constructor` pass both `fei` and `pool`

Tools/Techniques: Manual
Difficulty+Impact: Gas Optimisation

Details

Avoid making 2 external call to the master, given that the TurboSafe must be created by a `TurboMaster`. Both `fei` and `pool` can be passed down from the `constructor`

Mitigation

Make the `SafeMaster` pass both `fei` and `pool` when creating a new `TurboSafe`

TT-013: comment and code changes in slurp function

Tools/Technique: Manual
Difficulty+Impact: Informational

Details

Some comments are incomplete or confusing, here are some proposotions to update those

Mitigation

Comments
Compute what percentage of the interest earned will go back to the **Master**
Cannot overflow because the total cannot be **greater than totalFeiBoosted**
Code
On line 244, `uint256 protocolFeePercent =`

`master.clerk().getFeePercentageForSafe(this, asset);`, should `this` be replaced by `address(this)`?

TurboBooster.sol

TT-015: update frozen to true as default value

Tools/Technique: Manual
Difficulty+Impact: Minor

Details

`Frozen` is false by default, which may lead to unwanted boosting of a Vault if the operator is not done setting it or wants to check values before approving

Mitigation

Set frozen to true and update it to false when the initial configuration of the booster is done

TurboGibber.sol

TT-016: gibber needs to be approved to mint fei

Tools/Technique: Manual
Difficulty+Impact: Minor

Details

More of a reminder for future gibber deployments and it depends on how Fei handles approval for minting

Mitigation

Documentation for future maintainers