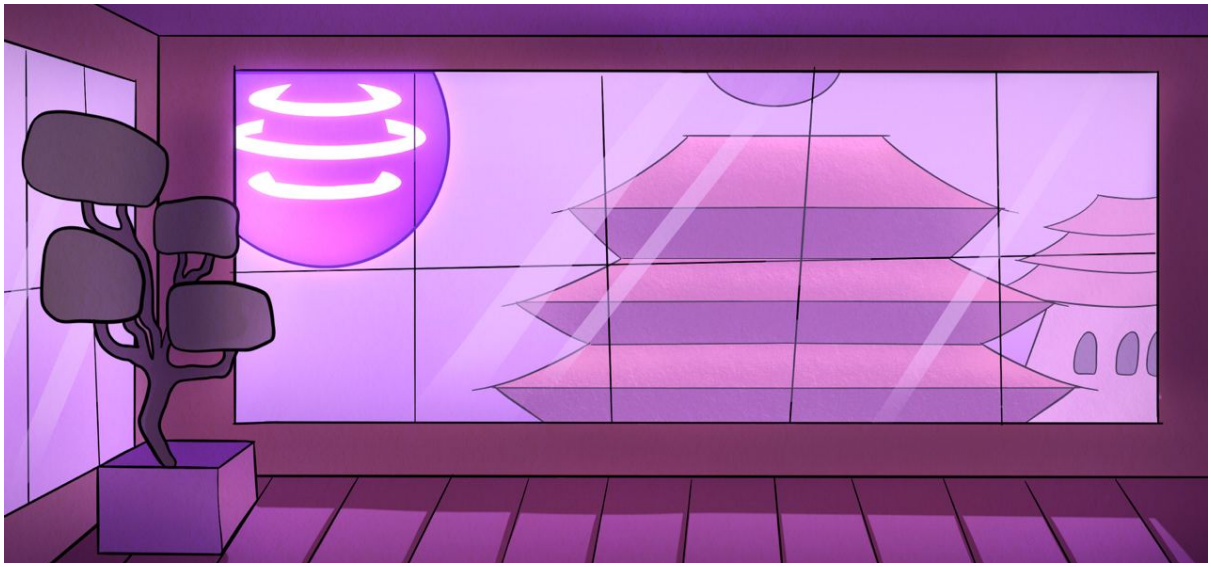

Aura Finance Upgrades: Booster Owner, Pool Manager and Reward Stash

by *Hephyrius.eth*

All notes are based on internal PR 6 and PR 26 respectively.

1. Introduction

Aura finance contributors have been informed of a number of vulnerabilities that may be exploitable by privileged actors. Contributors have worked on mitigating the long-tail risk posed by these vectors. Doing so by reducing the range of functions that privileged entities may call. This report reviews the changes proposed and evaluates the impact on the deployment of the protocol.

1. Introduction
2. Issues
    2.1 Deposit Freezing
    2.2 Deposit Theft
3. Proposed Changes
4. `PoolManagerV4`
5. `BoosterOwnerSecondary`
6. `StashToken`
7. `ExtraRewardStashV3`
8. Patch Review

8.1 Deposit Freezing
8.2 Deposit Theft
8.3 On-chain Execution
9. Overview

2. Issues

Please note that the issues outlined below require a contrived sequence of governance transactions to be executed and therefore are of low to medium risk. Governance processes would need to be hijacked in order for any real damage to occur.

2.1 Deposit Freezing

A scenario exists within the deployment of the protocol where an overflow in the number of tokens received by the `ExtraRewardStashV3` contract may lead to deposits being frozen causing `BaseRewardPool` to become inoperable/bricked.

If a malicious token is added to a reward pool it may be possible to brick deposits by causing an overflow in the number of tokens transferred. such as when an address receives more than `type(uint).max`

2.2 Deposit Theft

A scenario exists where user deposits may be stolen during the `Booster` shutdown process and are reliant on the attacker having owner and operator privileges for `Booster`, `BoosterOwner`, `PoolManagerSecondary` and. This theft requires a specific sequence of steps to take place in order to pull off successfully.

1. Trigger `shutdownPool` on a target pool using the `PoolManagerSecondary`
2. Add a *malicious token* and *malicious gauge* using `forceAddPool` on `PoolManagerSecondary`
3. Re-add the pool that was shut down using `addPool` on the `PoolManagerSecondary`, this has the effect of placing the victim pool after the malicious one.
4. `deposit` BPT tokens of the victim pool into the `Booster` .
5. Trigger `shutdownSystem` on `PoolManagerSecondary` and then trigger `queueForceShutdown` on `BoosterOwner`
6. Trigger `forceShutdownSystem` on `BoosterOwner`
7. Re-enter using a transfer hook in the malicious token

8. During re-entry set `VoterProxy` operator to the malicious token and execute a transfer of all target BPT tokens held by the `VoterProxy` before they are transferred to the `Booster`
9. Reset the operator of `VoterProxy` to `Booster` Before re-entry ends
10. At this point, the attacker has Aura gauge tokens for the newly added malicious pool and the BPT tokens of the victim pool are sitting in the `Booster`. The attacker may now trigger `withdrawall` on the `Booster` using the malicious gauge tokens to withdraw the victim's BPT tokens.

Note; In order to steal all of the tokens in the BPT, the attacker needs to deposit an equivalent number of BPT in step 4. For instance, if a pool has 10 BPT, an attacker must deposit 10 BPT to take all of the funds. The attacker essentially receives the equivalent number of BPT tokens from the victim pool as they deposited in their malicious pool.

3. Proposed Changes

The Upgrade proposed by Aura contributors has a few different strands;

1. Swap `PoolManagerV3` contract for `PoolManagerV4` Contract
2. Make `BoosterOwnerSecondary` the owner of `BoosterOwner`
3. Introduction of `StashToken`
4. Swap existing `ExtraRewardStashV3` for modified `ExtraRewardsStashV3` contract

The `BoosterOwnerSecondary` and `StashToken` contracts are new contracts that are being introduced, The `PoolManagerV4` and `ExtraRewardsStashV3` contracts are modifications of existing contracts.

4. `PoolManagerV4`

The `PoolManagerV4` contract is intended to become the operator of the pool management system. This is done by the multi-sig executing the `setOperator` function of `PoolManagerSecondary` which has the effect of replacing the current operator, `PoolManagerV3`, with `PoolManagerV4`.

The `PoolManagerV4` contract is largely the same as `PoolManagerV3` with a few notable modifications;

1. The removal of the global variable `gaugeController`. This variable is a public variable that is not used directly within the functions of the `V3` contract or any of

its callers.

2. Removal of `forceAddPool` function. The removal of this function reduces the likelihood that the operator can maliciously update and add pools when the system is shut down.
3. Introduction of the `shutdownSystem` which allows the contract to trigger `shutdownSystem` on `PoolManagerSecondary`. Triggering this function permanently prevents new pools from being added.

The adoption of the `PoolManagerV4` as the operator of the `PoolManagerSecondary` contract occurs in conjunction with the sealing of `PoolManagerSecondary`. Ownership of `PoolManagerSecondary` is passed from the protocol multi-sig to `PoolManagerV4`.

This means that privileged functions that are guarded by the `onlyOwner` modifier cannot be called unless they are explicitly defined in `V4`. There is only one instance where this occurs, and this is the `shutdownSystem` function which triggers the `onlyOwner` guarded `shutdownSystem` function of `PoolManagerSecondary`

This means that the ownership of `PoolManagerSecondary` cannot be transferred as `setOwner` does not have any calls from functions within `V4` , effectively making `V4` the operator permanently.

This also means that the operations such as `forceAddPool` on `PoolManagerSecondary` can no longer be called, which helps in patching the vulnerability presented in 2.2.

5. `BoosterOwnerSecondary`

The `BoosterOwner` is the owner of the `Booster` and is able to trigger functions that are gated by `onlyOwner`. The `BoosterOwnerSecondary` intends to become the owner of the `BoosterOwner` contract and it will be the contract that triggers `onlyOwner` on `BoosterOwner` and by proxy, `Booster`. This new contract effectively seals the ownership and access to certain functionality on both contracts. The owner of `BoosterOwnerSecondary` is intended to be the multi-sig.

As functionality becomes gated, the `operator` of `VoterProxy` may no longer be transferred back and forth as seen in the shutdown theft vulnerability, which helps in patching the vulnerability presented in 2.2.

The `StashToken` contract has the following functionality;

1. The contract contains two modifiers that gate function access;

a. `onlyOwner` allows only the `owner` to call a function

b. `onlyManager` allows only the `manager` to call a function

2. The `manager` role has a single purpose, this is to set the `feeTokenVerifier` value via the `setFeeTokenVerifier` function. The `feeTokenVerifier` is currently not deployed and is intended to be managed by balancer.

3. The `owner` may the `manager` using `setManager`. If the current `manager` is `address(0)` then it cannot be set to a different address. This effectively seals the `feeTokenVerifier`

4. Ownership transfer of the contract is handled by a two-step process;

   a. the current `owner` triggers `transferOwnership` to set the `pendingowner`

   b. the `pendingowner` triggers `acceptOwnership` to become the `owner`

5. The transfer of ownership of the `BoosterOwner` contract from the multi-sig to `BoosterOwnerSecondary` is handled by `acceptOwnershipBoosterOwner` . Ownership of `BoosterOwner` can no longer be transferred once ownership has been accepted as `BoosterOwnerSecondary` lacks the logic to transfer to another address.

6. `BoosterOwnerSecondary` has a number of pass-through functions that map 1:1 to the equivalent function on `BoosterOwner`. The ones listed below have no additional logic beyond the `onlyOwner` modifier;

   `setArbitrator` , `updateFeeInfo` , `setFeeManager` , `setVoteDelegate` , `shutdownSystem` , `queueForceShutdown` , `forceShutdownSystem` , `setRescueTokenDistribution` and `setStashRewardHook`

7. `BoosterOwnerSecondary` has a set of pass-through functions that contain additional logic. All of which are accessible by only the `owner`. These are;

   a. `sealStashImplementation` - Allows the `owner` to seal access to `setStashFactoryImplementation` preventing access to the same function on `BoosterOwner`. This action cannot be undone.

   b. `setFeeInfo` - Checks whether the `feeToken` is verified on the `feeTokenVerifier` contract. This is skipped If the address of the `feeTokenVerifier` is 0.

   c. `setStashFactoryImplementation` - This function will revert if `sealStashImplementation` is true, which occurs when `setSealStashImplementation` has been triggered.

   d. `setRescueTokenReward` - Reverts if the `poolId` is lower than the `oldPidCheckpoint`. Passes stash address retrieved from the `PoolInfo` function

of the `Booster`

e. `setStashTokenIsValid` - sets a `StashToken` as active by calling `setIsValid` on it. This is called from the `BoosterOwner` using its `execute`. The `BoosterOwner` is the only address that can call `setIsValid`
Function reverts if the `execute` call is unsuccessful.

f. `execute` - This function can arbitrarily call functions from the `BoosterOwner` to other contracts. This is done by calling `execute` on `BoosterOwner`. The only exception is that `Booster` cannot be called.
This function reverts when the following function selectors are invoked;

i. `setFeeInfo(address, address)`

ii. `setFactories(address, address, address)`

iii. `setImplementation(address, address, address)`

iv. `setExtraReward(address)`

6. `StashToken`

The `StashToken` is a newly introduced contract that acts as a reward token intermediate. It is a non-standard and non-transferable `ERC20` token that acts as a wrapper. Additional rewards that are not BAL that have been accumulated by the pools are held by a `StashToken`. The stash token deals with fund freezing identified in 2.1

The `StashToken` contract has the following functionality;

1. `ERC20`-related metadata; `totalSupply`, `name`, `symbol`
2. An `init` function for setting operation variables; `operator`, `rewardPool` and `baseToken`.
3. A maximum total supply constant of `1e38`
4. a `mint` function that transfers the `baseToken` to the contract from the related `ExtraRewardStashV3` checking that the `totalSupply` after transferring `baseToken` does not exceed the maximum. This function is guarded so only the `ExtraRewardStashV3` related to the token may trigger it.
5. A `transfer` function that transfers the `baseToken` to the claimant address when the `getReward` has triggered from the `rewardPool` . This function can only be called by the `rewardPool`. This function has the effect of reducing `totalSupply`.

6. A `setIsValid` function that sets the bool value `isValid`. The `BoosterOwner` can only call this

Each `StashToken` used is deployed as a clone of the `StashToken` implementation. As such, they are proxy contracts. The implementation contract correctly deals with `init` calls to the implementation contract.

7. `ExtraRewardStashV3`

The `ExtraRewardStashV3` contract is modified to incorporate the `StashToken`. The incorporation of the `StashToken` has the effect of ensuring all extra rewards are wrapped. This ensures that withdrawal freezing is unlikely due to the protections introduced by `StashToken`.

The following modifications have been made to `ExtraRewardStashV3`;

1. `Import` of `Clones` library from OpenZeppelin contracts.
2. Added a `stashTokenImplementation` variable. This is used during cloning.
3. Added `stashToken` to `TokenInfo` struct
4. Deploy a `StashToken` corresponding to `stashTokenImplementation` during contract initialization
5. The `CreateTokenRewards` call in `setToken` now uses the cloned `StashToken` address, instead of the token address.
6. The `StashToken` are deployed as a `clone` in the `setToken` function. The `clone` is also initialized and added to the `tokenInfo` struct.
7. The `processStash` function has an additional check that ensures that the `StashToken` is marked as valid, skipping if the `BoosterOwner` has marked the `StashToken` as invalid.
8. The `processStash` function approves the `StashToken` to transfer its entire balance of tokens. This is required for the `mint` call to `StashToken` that follows. This logic replaces the existing `transfer`

8. Patch Review

8.1 Deposit Freezing

The introduction of the `StashToken` and its incorporation into the `ExtraRewardStashV3` are an effective means of preventing the `RewardPool` from becoming inoperable. The

added checks related to token balances, as well as wrapping rewards into an intermediate ERC20-like token means that bricking is mitigated.

Further, the inclusion of the `isValid` bool adds an additional axis of control. This allows the protocol to effectively turn off specific reward tokens if they become malicious or are no longer needed.

8.2 Deposit Theft

The theft of user deposits during the shutdown process is hinged on the transferability of privileged roles across the `Booster` and `PoolManagerSecondary` contracts in combination with accessing `owner` / `operator` -only functions.

The proposed upgrade seals off the ability to transfer ownership of these key contracts to a malicious third party. Sealing the contracts so that the owner is effectively immutable. This cuts off the ability of an actor to pull off the attack demonstrated. Contributors went one step further by reducing the surface of privileged functions that these top-level ownership contracts may call.

8.3 On-chain Execution

A proposal has been made within the Aura Snapshot under AIP-20. This proposal would seek to execute the upgrades. Voting on this proposal ends February 13 at 14:00 UTC. If the proposal passes, it is expected that the upgrade will occur.

The upgrade has not been executed at the time of writing. However, we may assess its execution based on the proposed governance actions using tenderly and the sequence of actions provided to us in the snapshot proposal.

The proposed transaction has the following sequence of execution;

1. Calling `setStashFactoryImplementation` on the `BoosterOwner` contract to set the v3 factory to the modified `ExtraRewardStashV3` contract.
2. Call `transferOwnership` on the `BoosterOwner` contract to set the `BoosterOwnerSecondary` contract as the new `pendingowner`
3. Call `acceptOwnershipBoosterOwner` on the `BoosterOwnerSecondary` contract to set the current pending owner(which is `BoosterOwnerSecondary`) as the new `owner` of `BoosterOwner`
4. Call `setOperator` on the `PoolManagerSecondaryProxy` contract setting the `PoolManagerV4` contract as the new `operator`

5. call `setOwner` on the `PoolManagerSecondaryProxy` contract setting the `PoolManagerV4` contract as the new `owner`

9. Overview

The upgraded contracts and their operational roles within the protocol, as proposed by the Aura Finance contributors, successfully reduce the longtail risk posed to the protocol by actors with elevated permissions. Doing so by sealing away contracts and functions that may be abused, whilst introducing a number of new robust checks.

Overall, this reviewer is satisfied that both vulnerabilities 2.1 and 2.2 have been successfully addressed and that the proposed upgrades do not adversely impact the execution and continued operation of the protocol.