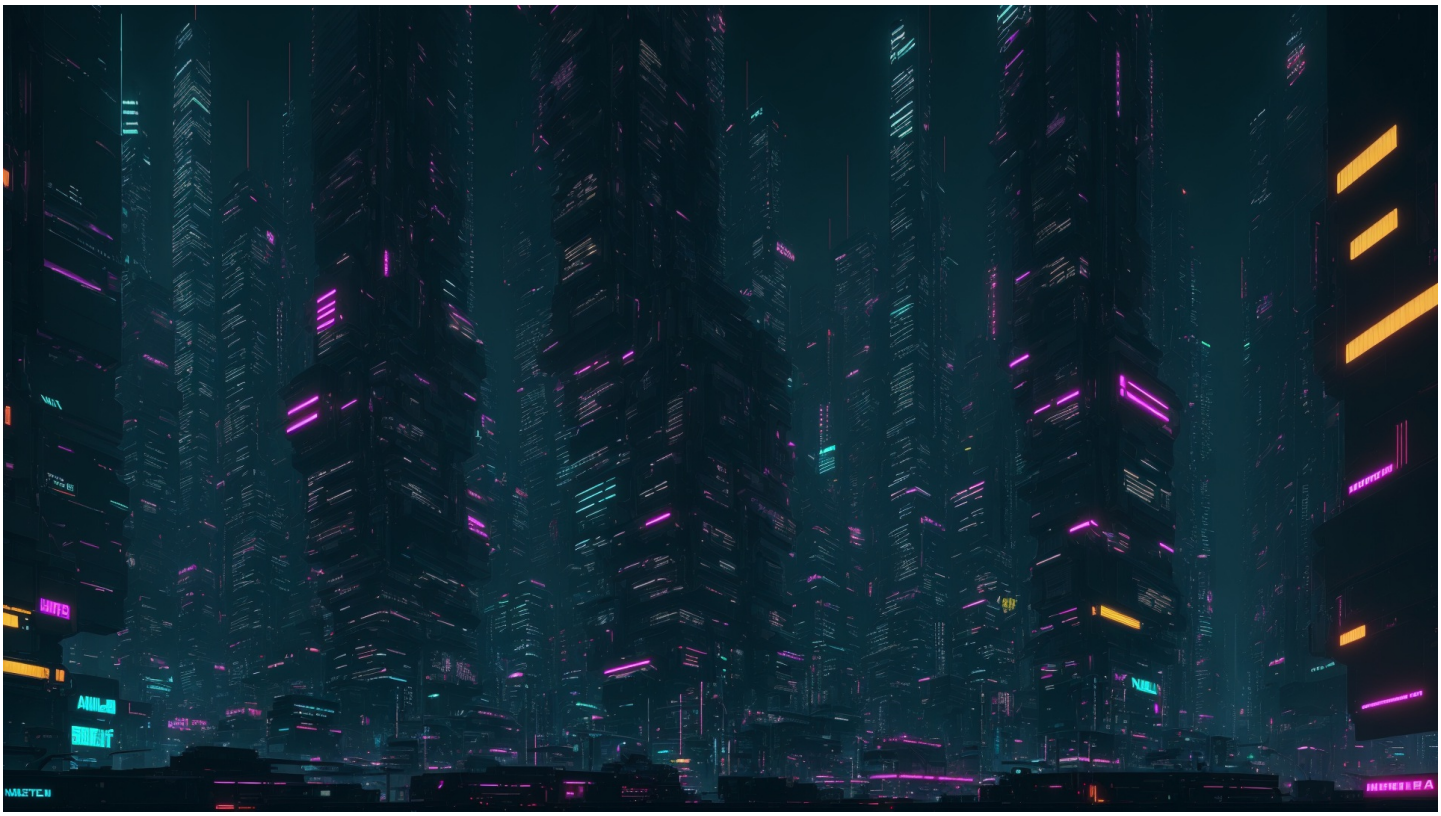
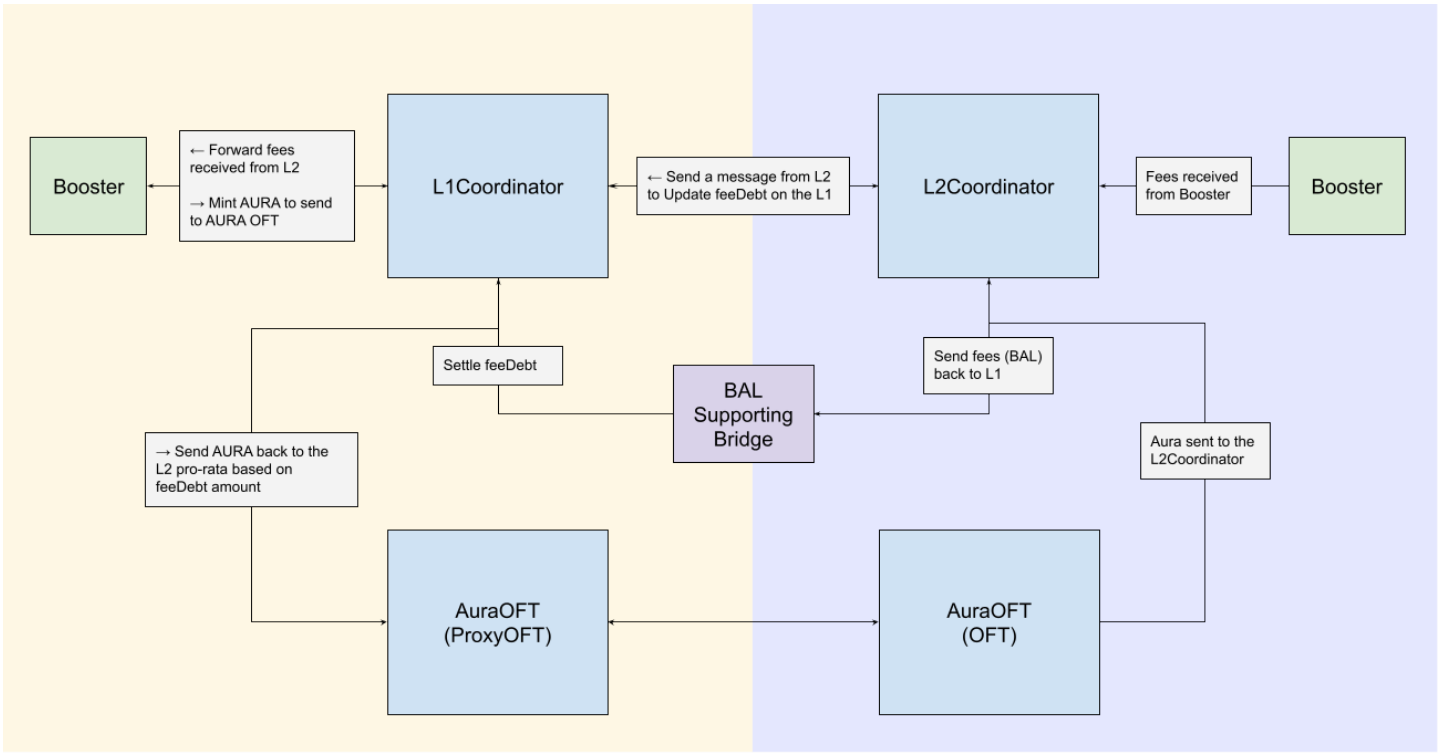


Aura Sidechain Review

Aura Finance is expanding to become a cross-chain protocol. Deploying to chains beyond Ethereum, with the intent to deploy to wherever the balancer is deployed and emitting rewards. As part of this expansion, development contributors have created a cross-chain variant of the protocol to deploy to the new chains. The developed contracts include a mix of entirely new contracts and variations of existing contracts. This report aims to review these contracts thoroughly and how they interface with the existing aura contracts.



Source: 0xFry

Subsections:

Convex Platform

The convex platform contracts developed for the sidechain are ‘lite’ variations of those used on Ethereum. In practice this means that contracts have functionality changed or removed, multiple contracts are merged together or contracts are not utilised on the sidechain. This section of the report reviews this aspect of the contracts

Contracts:

VoterProxyLite

PoolManagerLite

BoosterOwnerLite

BoosterLite

BaseRewardPool4626

VoterProxyLite

The review of this contract is based on and

Overview

On Ethereum, the voter proxy is responsible for maintaining the VE lock on the underlying platform and depositing tokens into the platform. The sidechain lite version does not need locks as the boost provided by the lock on Ethereum is mapped to the sidechain addresses.

Differences to Canonical

As the Voter Proxy Lite does not need to manage the Vote Escrow lock on Balancer, a large part of the original functionality is removed. This is largely related to locking, voting and claiming fees.

A full diff can be found here

1. Introduction of an Initializer
2. Removal of vote-related logic
3. Removal of vote escrow lock management features such as `increaseTime` and `increaseAmount`
4. Removal of fee logic

Issues

Nothing of note

Function Access

The table below provides a high-level view of function access for the smart contract.

Function Name	Access Modifier	Modifiers
`constructor`	public	None
`initialize`	external	Only `owner` can call this
`getName`	external	pure
`setOwner`	external	Only `owner` can call this
`setRewardDeposit`	external	Only `owner` can call this
`setSystemConfig`	external	Only `owner` can call this
`setOperator`	external	Only `owner` can call this
`setStashAccess`	external	Only `operator` can call this
`deposit`	external	Only `operator` can call this
`withdraw(1 arg)`	external	Only the `withdrawer` can call this
`withdraw(3 args)`	public	Only `operator` can call this
`_withdrawSome`	internal	-
`withdrawAll`	external	Only `operator` can call this
`claimCrv`	external	Only `operator` can call this
`claimRewards`	external	Only `operator` can call this
`balanceOfPool`	public	view
`execute`	external	Only `operator` can call this

BaseRewardPool4626

The review of this contract is based on and

Overview

The base reward pool is an ERC4626-compliant vault contract that handles user deposits of balancer BPT tokens. The canonical version of this contract does not allow the token to be transferred, this is because the ERC4626 defines token transfers as an optional feature and not a requirement. In order to allow for composability on sidechains transferability of the token has been enabled.

This means that the `transfer`, `transferFrom` and `_transfer` functions of the contract have been modified from the previous implementation to facilitate transferability. A diff of these changes can be found here.

Issues

Issues from previous commits have been resolved. Nothing of note as of the review commit.

BoosterLite

The review of this contract is based on and

Overview

The booster contract is essential in coordinating canonical deployment. It keeps track of pool info, and user deposits and handles rewards. The sidechain lite version of this contract plays a broadly similar role with a few notable changes to remove or amend functionality.

Notable Changes from Canonical

A full diff can be found here

1. Introduction of L2Coordinator logic
2. Introduction of an Initializer
3. Removal of vote-related logic
4. Removal of fee logic such as `earmarkFees`
5. Removal of delegate logic and L2 logic such as `distributeL2Fees`

Issues

Nothing of note

Function Access

The table below provides a high-level view of function access for the smart contract.

Function Name	Access Modifiers	Modifiers
`constructor`	public	-
`initialize`	external	only the `boosterOwner` may call this
`setOwner`	external	only the `boosterOwner` may call this
`setFeeManager`	external	only the `boosterOwner` may call this
`setPoolManager`	external	only pool manager can call this
`setFactories`	external	only the `boosterOwner` may call this
`setRewardContracts`	external	only the `boosterOwner` may call this
`setFees`	external	`nonReentrant` only fee manager can call this
`setTreasury`	external	only fee manager can call this
`poolLength`	external	view
`addPool`	external	only the pool manager can call this Can only be called if booster is not shutdown
`shutdownPool`	external	`nonReentrant` only pool manager can call this
`shutdownSystem`	external	only the `boosterOwner` may call this
`deposit`	public	`nonReentrant` only if booster is not shutdown, pool is not shutdown and gauge is not `address(0)`
`depositAll`	external	-
`_withdraw`	internal	`nonReentrant`
`withdraw`	public	-
`withdrawAll`	public	-
`withdrawTo`	external	only reward contract may call

Function Name	Access Modifiers	Modifiers
`claimRewards`	external	only the stash of the pool id can call
`setGaugeRedirect`	external	only the stash of the pool id can call
`_earmarkRewards`	internal	only when not shutdown
`earmarkRewards`	external	`nonReentrant` payable only when not shutdown
`rewardClaimed`	external	only the reward contract for the pool id can call

BoosterOwnerLite

The review of this contract is based on and

Overview

The booster owner contract on the canonical deployment wraps sensitive features of the booster with additional safeguards. The canonical chain has two layers of safeguarding, the `boosterOwner` and the `boosterOwnerSecondary`. The sidechain version of the `boosterOwner` merges the functionality of both these contracts into a single lite contract.

Issues

Nothing of note

Function Access

The table below provides a high-level view of function access for the smart contract.

Function Name	Access Modifiers	Modifiers
`constructor`	public	-
`transferOwnership`	external	`onlyOwner`
`acceptOwnership`	external	-
`sealOwnership`	external	`onlyOwner`
`setBoosterOwner`	external	`onlyOwner`
`setFactories`	external	`onlyOwner`
`setFeeManager`	external	`onlyOwner`
`shutdownSystem`	external	`onlyOwner`
`queueForceShutdown`	external	`onlyOwner`
`forceShutdownSystem`	external	`onlyOwner`
`execute`	external	`onlyOwner`
`setRescueTokenDistribution`	external	`onlyOwner`
`setRescueTokenReward`	external	`onlyOwner`
`setStashExtraReward`	external	`onlyOwner`
`setStashRewardHook`	external	`onlyOwner`
`setStashFactoryImplementation`	external	`onlyOwner`

PoolManagerLite

The review of this contract is based on and

Overview

The sidechain version of the Pool Manager serves as a management tool for handling pools in a Booster lite contract. The main responsibilities of this contract include adding new pools, shutting down individual pools, and if required, shutting down the whole pool management system. The functionality of the Pool Manager lite is an amalgamation of the multiple layers of pool manager proxies of the canonical deployment, where the functionality of multiple contracts has been merged into one.

Issues

Issues from previous commits have been resolved. Nothing of note as of the review commit.

Function Access

The table below provides a high-level view of function access for the smart contract.

Function Name	Access Modifier	Modifiers
`constructor`	public	None
`setOperator`	external	Only `Operator` may call
`setProtectPool`	external	Only `Operator` may call
`addPool (1 arg)`	external	None
`addPool (2 args)`	external	None
`_addPool`	internal	If protected add pool flag is true then only `operator` may call. Cannot be called if system is shutdown.
`shutdownPool`	external	Only `Operator` may call Cannot be called if system is already shutdown
`shutdownSystem`	external	Only `Operator` may call

Aura Contracts and LayerZero

The largest introduction of smart contracts comes from the introduction of LayerZero-related contracts. These contracts facilitate the cross-chain operation of the protocol. This section of the report explores these new contracts.

Create2Factory

PauseGuardian

CrossChainConfig

CrossChainMessages

PausableOFT

AuraOFT

PausableProxyOFT

AuraProxyOFT

AuraBalOFT

AuraBalProxyOFT

L2Coordinator

L1Coordinator

L2Coordinator

The review of this contract is based on and

Overview

The L2Coordinator is a sidechain-specific contract that coordinates between the canonical and sidechain deployments. Its primary roles within the deployment are; to notify the L1 of BAL fees that are to be sent to the contract, track AURA sent to the sidechain, act as a reservoir of AURA for the booster to send rewards from and keep track of the 'mint rate', which is the number of tokens to award per BAL token received as fees.

Issues

Severity	Issue	Fix	Code
Informational	If a buffer of AURA is given to the coordinator while `accBALRewards` is low, an abnormally high mint rate will occur. This will over-reward users.	Avoid triggering the fees callback prematurely whilst the sidechain L2Coordinator is in it's infancy. In other words, only provide AURA for fees that have actually been captured by the sidechain rather than anticipated rewards.	-
Informational	when the `L2Coordinator` is warming up / newly deployed, the `mintRate()` may be volatile.	-	-

Function Access

The table below provides a high-level view of function access for the smart contract.

Function Name	**Access Modifier**	**Modifiers**
`constructor`	public	-
`mintRate`	public	view
`initialize`	external	`onlyOwner`
`setBridgeDelegate`	external	`onlyOwner`
`setConfig`	external	`onlyOwner`
`mint`	external	-
`queueNewRewards`	external	payable
`_nonblockingLzReceive`	internal	virtual

PausableOFT

The review of this contract is based on and

Overview

The PausableOFT is a LayerZero OFT contract where the Pause Guardian has been introduced. The pause guardian is introduced via the `whenNotPaused` modifier on the `sendFrom` function. All other OFT behaviour is left untouched.

Issues

Nothing of note

Function Access

The table below provides a high-level view of function access for the smart contract.

Function Name	Access Modifier	Modifiers
`sendFrom`	public	payable `whenNotPaused`

CrossChainMessages

The review of this contract is based on and

Introduction

The cross-chain messages library is used in encoding and decoding messages sent through LayerZero. The library is used extensively by Coordinators and OFT contracts.

Issues

Nothing of note

Function Access

The table below provides a high-level view of function access for the smart contract.

Function Name	Access Modifier	Modifiers
`isCustomMessage`	internal	pure
`encodeLock`	internal	pure
`encodeFees`	internal	pure
`encodeFeesCallback`	internal	pure
`decodeFeesCallback`	internal	pure
`decodeFees`	internal	pure
`decodeLock`	internal	pure

L1Coordinator

The review of this contract is based on and

Overview

The L1 coordinator is a crucial aspect of the new sidechain system. The coordinator keeps track of BAL received by a sidechain and the corresponding AURA that has been given in exchange for these fees. The L1 Coordinator also handles incoming BAL from the various delegate contracts. Both pulling in BAL from these contracts and distributing it to the booster via the `distributeL2Fees.`

Issues

Nothing of note

Function Access

The table below provides a high-level view of function access for the smart contract.

Function Name	Access Modifier	Modifiers
`constructor`	public	-
`setConfig`	external	`onlyOwner`
`setBridgeDelegate`	external	`onlyOwner`
`setL2Coordinator`	external	`onlyOwner`
`setDistributor`	external	`onlyOwner`
`_notifyFees`	internal	-
`distributeAura`	external	`onlyDistributor` payable
`_distributeAura`	internal	-
`settleFeeDebt`	external	-
`_nonblockingLzReceive`	internal	virtual
`receive`	external	payable

Create2Factory

The review of this contract is based on and

Overview

The create2factory is a contract that allows Aura to deploy contracts to the same addresses on multiple chains as this will reduce the operational complexity of the protocol whilst improving the overall user and developer experience. The contract facilitates this by the use of the Create2 EVM opcode to precompute deployment addresses.

The factory also allows additional callback logic to be executed post-contract deployment, this means that once a contract is deployed, it can be invoked. This is a useful feature as it allows functionality such as initializers and ownership transferrers to occur on contract creation.

Create2 functionality is provided by OpenZeppelin contracts.

Issues

Severity	Issue	Fix	Code
Informational	The contract has a `receive()` but does not have payable functions or handle the chain's native token intentionally.	1) Remove Receive **or** 2) Allow `_execute` to forward `msg.value`	Link
Informational	The contract has no means of calling `_execute(address _to, bytes calldata _data)` directly. This means that ownership or other permissions for a contract cannot be transferred if a deployer forgets to provide callbacks.	Introduce an execute function that allows deployers to arbitrary calls. Example: `function execute(address _to, bytes calldata _data) external onlyDeployer returns (bool, bytes memory) { (bool success, bytes memory result) = _execute(_to, _data); return (success, result); }`	-

Function Access

The table below provides a high-level view of function access for the smart contract.

Function Name	Access Modifier	Modifiers
`updateDeployer`	external	`onlyOwner`
`deploy`	external	`onlyDeployer`
`_execute`	private	-
`computeAddress`	external view	-
`receive`	external	payable

CrossChainConfig

The review of this contract is based on and

Overview

The cross-chain config contract provides adapter parameters for cross-chain operations and is utilised as a building block by other contracts, including Coordinators and OFTs. A key feature of the contract is the `configs` mapping. This allows contracts to store chain ***and*** function-specific adapter configurations.

Issues

Nothing of note

Function Access

The table below provides a high-level view of function access for the smart contract.

Function Name	Access Modifier	Modifiers
`setConfig`	external	virtual
`_setConfig`	internal	-

AuraOFT

The review of this contract is based on and

Overview

The AuraOFT is the sidechain version of the Aura token. This is what end users will receive as rewards from the protocol on side chains. The oft functionality allows users to lock directly from the sidechain without having to first bridge the token back to the canonical deployment on Ethereum.

Issues

Severity	Issue	Fix	Code	Resolved
Low	`lock()` only works for `msg.sender` as the lock recipient. There may be situations where we may want to lock to a different address instead of `msg.sender`	Adjust the function to take an address for the locker on Ethereum	Link	

Resolved in commit:

Function Access

The table below provides a high-level view of function access for the smart contract.

Function Name	Access Modifier	Modifiers
`constructor`	public	-
`initialize`	external	`onlyOwner`
`setConfig`	external	`onlyOwner`
`lock`	external	payable

AuraProxyOFT

The review of this contract is based on and

Overview

The AuraProxyOFT is the contract that wraps Aura and provides it with LayerZero OFT functionality, as such It is intended that this OFT is deployed on Ethereum. The AuraProxyOFT contract is a PausableProxyOFT with additional functionality to enable cross-chain locking of Aura. In a scenario where the locking contract is shut down, the contract sends tokens to the user instead. The contract utilises the Cross Chain Message library for decoding lock messages.

Issues

Nothing of note

Function Access

The table below provides a high level view of function access for the smart contract.

Function Name	Access Modifier	Modifiers
`constructor`	public	-
`_lockFor`	internal	-
`_nonblockingLzReceive`	internal	virtual

AuraBalOFT

The review of this contract is based on and

Overview

The AurabalOFT is a contract representing Aurabal on sidechains. It is a PausableOFT with no additional features of note, other than an initialize function.

Issues

Nothing of note

Function Access

The table below provides a high-level view of function access for the smart contract.

Function Name	Access Modifier	Modifiers
constructor	public	-
initialize	external	`onlyOwner`

AuraBalProxyOFT

The review of this contract is based on and

Overview

The AuraBalProxyOFT is the contract that wraps Aurabal and provides it with LayerZero OFT functionality. Aurabal that is held by this contract is deposited into an Aurabal auto-compounder. This leads to a steady increase in Aurabal on sidechains in a leveraged fashion, as Aurabal staking rewards are compounded to increase the total Aurabal held by the contract. The contract inherits behaviours from the Pausable Proxy OFT and Crosschain config contracts.

Issues

Nothing of note

Function Access

The table below provides a high-level view of function access for the smart contract.

Function Name	Access Modifier	Modifiers
`constructor`	public	-
`setConfig`	external	`onlyOwner`
`setRewardReceiver`	external	`onlyOwner`
`updateAuthorizedHarvesters`	external	`onlyOwner`
`setOFT`	external	`onlyOwner`
`setHarvestSrcChainIds`	external	`onlyOwner`
`circulatingSupply`	public	view
`_debitFrom`	internal	-
`_creditTo`	internal	-
`harvest`	external	-
`processClaimable`	external	payable
`vaultExecute`	external	`onlyOwner`
`rescue`	external	-
`_withdraw`	internal	-
`_stakeAll`	internal	-
`_processHarvestableTokens`	internal	-

PausableProxyOFT

The review of this contract is based on and

Overview

The pausable proxy OFT is an extension of the LayerZero Proxy OFT that utilises the Pause Guardian. In scenarios where the pause guardian has been triggered, transactions are added to a queue of pending transactions, which it will be processed after a set wait period.

This extension of the OFT also implements a net in-outflow limit to the OFT. These limits are applied to epochs, where one epoch is a week. The intention of this limit is to prevent risk exposure in a worst-case scenario on side chains.

Issues

Severity	Issue	Fix	Code
Low-Medium	A malicious `sudo`/`admin` user is able to mint infinite tokens from a limited number of tokens on sidechains. This can be done by looping between calling `rescue()` and `sendFrom()`	Add a modifier to ensure `pause` has been triggered before `rescue` can be called	Link

Function Access

The table below provides a high-level view of function access for the smart contract.

Function Name	Access Modifier	Modifiers
`constructor`	public	-
`setQueueDelay`	external	`onlyOwner`
`setInflowLimit`	external	`onlyOwner`
`getCurrentEpoch`	external	view
`sendFrom`	public	payable, `whenNotPaused`
`_sendAck`	internal	-
`processQueued`	external	`whenNotPaused`
`rescue`	external	-
`_getCurrentEpoch`	internal	view
`_getNetInflow`	internal	pure

PauseGuardian

The review of this contract is based on and

Overview

A pause guardian is a contract that provides pause functionality to contracts. The contract introduces the `guardian` role. This role has the power to pause and unpause the contract. The contract is utilised as a building block by various contracts within the suite, including the pausableOFT and the pausableProxyOFT. Pause functionality is provided by OpenZeppelin contracts.

Issues

Nothing of note

Function Access

The table below provides a high level view of function access for the smart contract.

Function Name	Access Modifier	Modifiers
`_initializePauseGuardian`	internal	-
`pause`	external	`onlyGuardian`
`unpause`	external	`onlyGuardian`

Chain Considerations and Bridging

Each chain that aura deploys a sidechain instance to has its own security considerations that need to be accounted for thoroughly. These include but are not limited to; the consensus mechanism, standards variations, EVM variations, and native bridging.

Additionally, BAL tokens on sidechains are deployed using different methods. Each deployment method requires a different means of bridging back to Ethereum in order for fees to be accrued by the canonical protocol. This section explores these aspects of the deployments.

Bridge Delegate Receiver

Bridge Delegate Sender

Gnosis

Arbitrum

Gnosis

The BAL token that Balancer distributes on the gnosis chain is minted by interacting with the Omnibridge on Ethereum.

The bridge uses affirmation and signature requests using a 4-of-7 schema. This means that 4 of the 7 validators of the bridge must sign a request before any tokens are sent on either end of the bridge.

Bridge Delegate

For information about the bridge delegate for the gnosis chain please review:

Gnosis Bridge Delegate

ERC677 Fallback

Every token created by the Omnibridge on the Gnosis chain has an inbuilt receive hook that needs to be accounted for when operating on the chain. When the BAL token is sent, the token looks for the hook, and if it exists control of execution is passed to the received. Forks of Ethereum protocols, deployed on gnosis, such as Hundred Finance and Agave, have fallen victim to attackers utilising this fallback. Caution is advised.

Bridging From Ethereum

Bridging a native Ethereum token, like BAL, from Ethereum to Gnosis requires a `relayTokens` transaction to be sent via the Token Mediator. This transaction emits an affirmation request event, within this event is a data field, which validators affirm on the Gnosis chain, this triggers tokens to be minted and sent to the receiver defined in the original request. Currently, BAL emitted as LP rewards on the Gnosis chain has been distributed to the chain using the L2 Gauge Checkpointer which requests the affirmation. Previously, this was handled by a multi-sig.

Bridging From Gnosis

Bridging the BAL token back to Ethereum utilises a similar but more involved process. To initiate the bridging back to Ethereum, a request for signature needs to be emitted. This is done by triggering the `transferAndCall(address to, uint value, bytes data)` function of the BAL token, where `to` is the token mediator, `value` is the number of tokens to send, and `data` is a packed encoding of the receiver address on Ethereum.

This action will trigger a request for signature to be emitted, this event has a data field which contains the information that validators will be signing. Once signed signatures are committed to the Arbitrary Message Bridge contract on the Gnosis chain.

Once enough signatures have been committed, the requester is able to execute the token transfer on Ethereum by triggering the `safeExecuteSignaturesWithAutoGasLimit` function of the Arbitrary Message Bridge contract on Ethereum. This function requires the data field emitted by the request for the signature event and signatures that were committed to the contract on the gnosis side.

Gnosis Bridge Delegate

The review of this contract is based on and

Overview

The gnosis bridge delegate sender is a contract that uses the Omnibridge to bridge BAL tokens back to Ethereum using the `transferAndCall` function of the gnosis BAL token. It inherits from the Bridge Delegate Sender.

Issues

Nothing of note

Function Access

The table below provides a high-level view of function access for the smart contract.

Function Name	Access Modifiers	Modifiers
`send`	external	`onlyOwner`

Bridge Delegate Sender

The review of this contract is based on and

Overview

A bridge delegate sender is a contract used as a building block for chain-specific senders. Sender contracts interact with the chain-specific bridge to bridge BAL tokens back to Ethereum. The tokens are sent to the bridge delegate receiver for the specific sidechain.

Issues

Nothing of note

Function Access

The table below provides a high-level view of function access for the smart contract.

Function Name	Access Modifiers	Modifiers
`setL1Receiver`	external	`onlyOwner`
`setL2Coordinator`	external	`onlyOwner`
`send`	external	virtual

Arbitrum Bridge Delegate

The review of this contract is based on https://github.com/aurafinance/aura-contracts/pull/202/commits/e5c193604fa5779df2567ca9ff90f52a8979d73a and

Overview

The Arbitrum bridge delegate sender is a contract that uses the Arbitrum gateway router to bridge BAL tokens back to Ethereum using the `outboundTransfer` function. It inherits from the Bridge Delegate Sender.

Issues

Nothing of note

Function Access

The table below provides a high-level view of function access for the smart contract.

Function Name	Access Modifiers	Modifiers
`send`	external	onlyOwner

Arbitrum

The BAL token that Balancer distributes on gnosis chain is minted by interacting with the Gateway router on Ethereum. The bridge uses an inbox-outbox system for sending messages between Ethereum and Arbitrum, where tickets are used to claim token deployments.

Bridge Delegate

For information about the bridge delegate for Arbitrum please see:

Arbitrum Bridge Delegate

Bridging From Ethereum

Bridging Bal from Ethereum to Arbitrum requires a `outboundTransfer` transaction to be sent via the Gateway router. This transaction will create a retriable ticket which can be used to claim tokens on Arbitrum. Commonly, the aliased Aliased L1 ERC20 Gateway on arbitrum will submit the ticket and finalize the transaction for the bridger. Finalising the transaction causes the token to be minted for the receive. Tokens sent to Arbitrum can usually be claimed in under an hour. Currently BAL emitted as LP rewards on Arbitrum is distributed using the L2 Gauge Checkpointer.

Bridging From Arbitrum

Bridging BAL tokens back to Ethereum requires a `outboundTransfer` to be sent using the L2 gateway router. This action burns the tokens being bridged from the callers balance. Arbitrum has a 7 day dispute window, this means that bridging to ethereum requires a 7 day wait before tokens can be received. Once the dispute period is over the tokens can be bridged using the `executeTransaction` method of the outbox contracts. This `executeTransaction` call needs to be done by the bridge and is not done by the arbitrum foundation.

Bridge Delegate Receiver

The review of this contract is based on and

Overview

The Bridge delegate receiver is a simple contract that receives the BAL that has been accumulated as fees on a sidechain. The intention is for there to be one receiver per sidechain. Fees received by this contract are pulled by the L1Coordinator.

Issues

Nothing of note

Function Access

The table below provides a high-level view of function access for the smart contract.

Function Name	Access Modifiers	Modifiers
`constructor`	public	
`settleFeeDebt`	external	`onlyOwner`

Additional Issues

Impact	Issue	Action	Code
Informational	Difference between LayerZero OFT library interface and the one used by the sidechain contracts	No action required	
Informational	A race condition occurs when the Autocompounder compounds. A user might be able to deposit on the side chain before compounded tokens are accounted for, such that they can gain a profitable sandwich on the accounting bridge transaction.	A higher withdraw fee can counter larger compounds	-