

Arbitrum Smart Contracts

Date	November 2021
Auditors	Martin Ortner, Dominik Muhs, Heiko Fisch


1 Executive Summary

This report presents the results of our engagement with **Arbitrum** to review **their L1 and L2 smart contract systems**.

The review was conducted over four weeks, from **October 4th, 2021** to **November 5th, 2021**. A total of 50 person-days were spent.

During the first week, the assessment team tried to get an overview of the whole system, starting with an [Arbitrum Deep-Dive](#) to mapping out the code-bases in scope (`arb-bridge-eth` , `arb-bridge-peripherals`). Most of the high-level diagrams were created during this week.

One team member focused on the TokenBridge (peripherals) during the second week, while another started to dive deeper into the Rollup/Nodes/Challenges logic. The client was notified of a potential security issue with the TokenBridge and chose to upgrade their contracts the same week.

After a one-week hiatus, one more person completed the assessment team. While this person ramped up with the architecture, the others combined forces  the Rollup/Nodes/Challenges logic after finishing work on the TokenBridge.

The final week concluded the review of the Rollup/Nodes/Challenges logic.

2 Scope

This assessment heavily prioritized Arbitrum's L1 and L2 smart contracts. Components in the `arch` folder were excluded from scope. Off-chain components have been deemed out-of-scope.

Our review focused on the commit hash `d730bc40932c24e239577d16b1737455ab75ee26`. The list of files in scope can be found in the [Appendix](#).

2.1 Objectives

Together with the Arbitrum team, we identified the following priorities for our review:

1. Identify vulnerabilities and weaknesses relating to the passing of messages between L1 and L2.
2. Review the token bridge and other peripherals to ensure the security of funds locked in escrow.
3. Analyze potential malicious validator behavior and validate that the challenge process cannot be manipulated without punishment.
4. Ensure that the system is implemented consistently with the intended functionality and without unintended edge cases.
5. Identify known vulnerabilities particular to smart contract systems, as outlined in our [Smart Contract Best Practices](#), and the [Smart Contract Weakness Classification Registry](#).

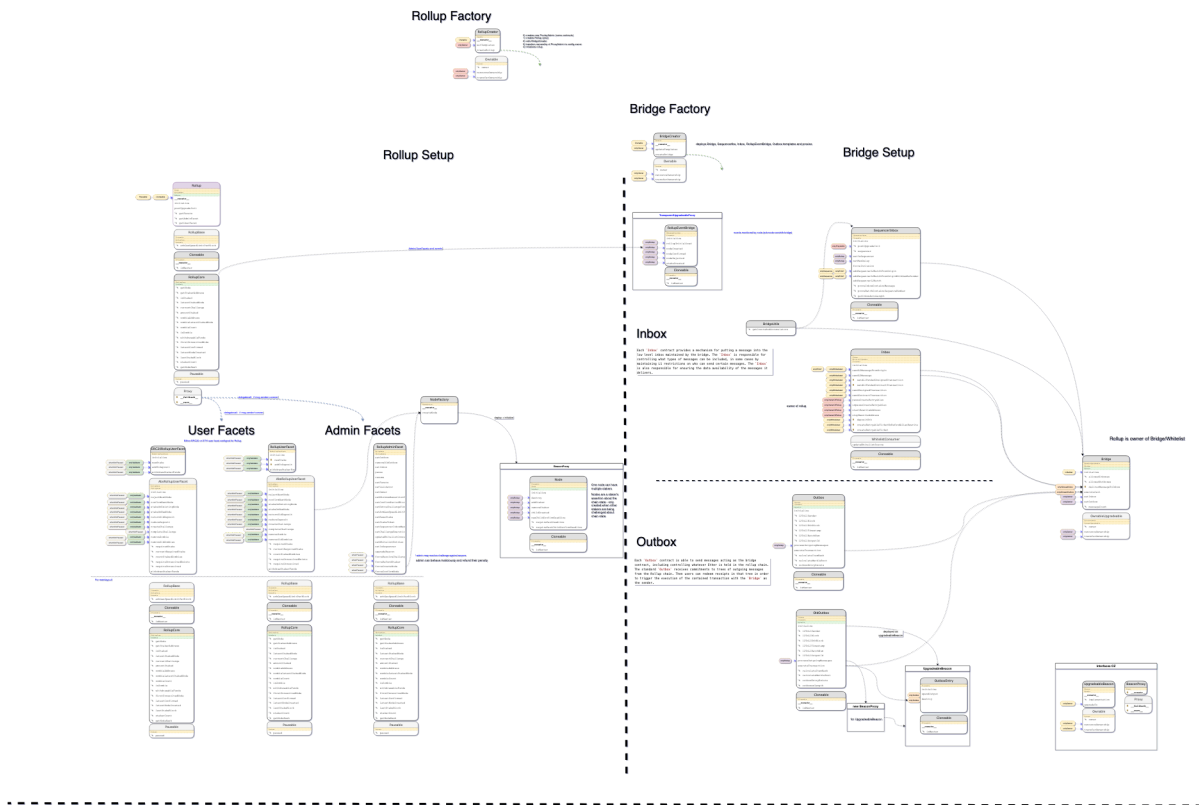
3 System Overview

This section describes the top-level/deployable contracts, their inheritance structure and interfaces, actors, permissions and important contract interactions of the [system under review](#). Please refer to [Section 4 - Security Specification](#) for a security-centric view on the system.



Contracts are depicted as boxes. Public reachable interface methods are outlined as rows in the box. The 🔍 icon indicates that a method is declared as non-state-changing (view/pure) while other methods may change state. A yellow dashed row at the top of the contract shows inherited contracts. A green dashed row at the top of the contract indicates that that contract is used in a usingFor declaration. Modifiers used as ACL are connected as yellow bubbles in front of methods.

3.1 Arbitrum Core Architecture



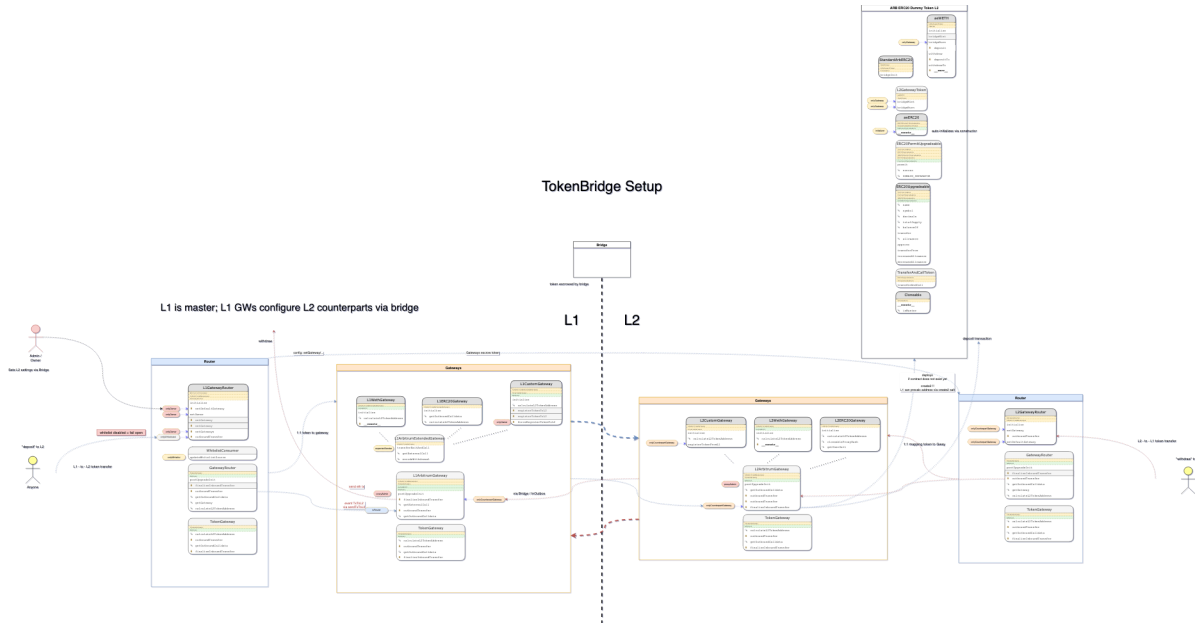
Arbitrum core architecture

The core architecture consists of a Bridge contract, an `Inbox` (L1->L2 messaging), an `Outbox` (L2->L1 messaging; 7-days delay to challenge rollup blocks), a `Rollup` contract, that, depending on whether an **admin** or **user** is calling it, provides different functionality (`AdminFacet` , `UserFacet`). Rollup blocks are submitted to the `Rollup` contract as `Node` contracts that can be challenged. Users can become stakers in the the `Rollup` contract and stake on valid “blocks”/ `Node` contracts.



The system provides various factories. The `RollupCreator` is used to deploy a new `Rollup` setup with RollupFacet implementations, Inboxes/Outboxes, EventBridge. It calls the `BridgeCreator` to deploy the bridge setup and configures the NodeFactory, ChallengeFactory for the contracts. `RollupCreator` is one entrypoint to deploy the system.

3.2 Arbitrum TokenBridge Peripherals

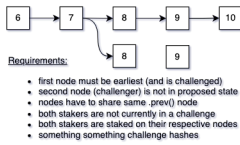


Bridge peripherals architecture

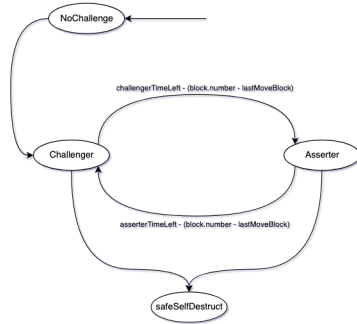
The TokenBridge is an application built on top of the Arbitrum Bridge (Core Logic). It allows users to bridge L1 Tokens to L2 via the generic Arbitrum bridge. An admin can configure tokens and their respective gateways. The setup is based on an `L1Router` that maps token addresses to `L1Gateway` contracts. Users have to go through the `L1Router` in order to bridge L1 tokens to L2. An admin configuring the `L1Router` indirectly also configures the `L2Router` counterpart (via bridge messages). L1 `ETH` is escrowed on the Arbitrum bridge while specific tokens are escrowed in the `L1Gateway` contracts. `WETH` is first unwrapped and then sent to the bridge, escrowed there, and then wrapped on L2 before being made available to users.

3.3 Arbitrum Challenge Details



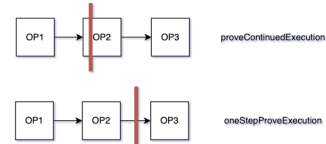


Legal challenge state machine transitions

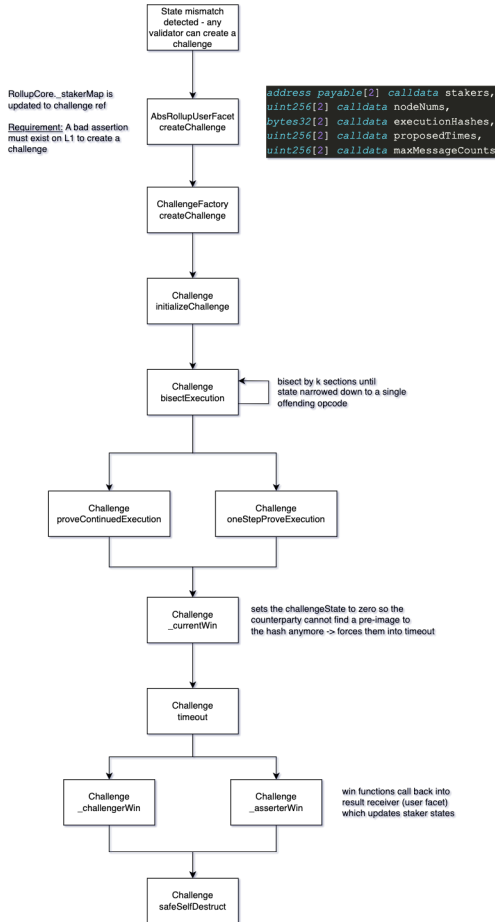


Prove function differences

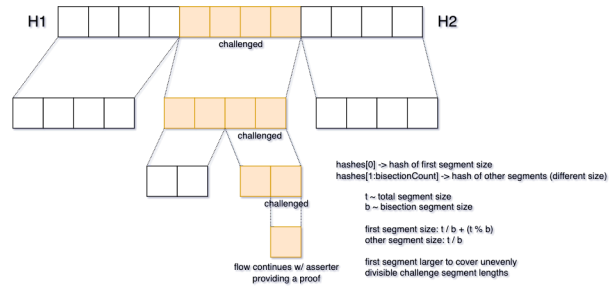
- bisection happens based on gas consumption
- prove function calls depend on where the gas boundary ends up after bisection (either "inside" an opcode or on the boundary to the next one)
- "inside" -> proveContinuedExecution
- boundary -> oneStepProveExecution



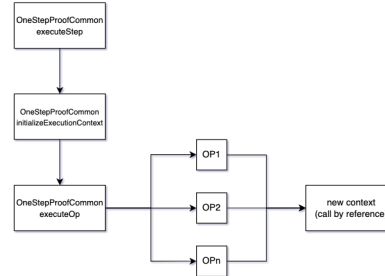
Challenge User Flow



bisectionExecution, n=12, k=3



oneStepProveExecution



General challenge functionality notes

4 Security Specification

This section describes, from a security perspective, the expected behavior of the system under review. It is not a substitute for documentation. The purpose of this section is to identify specific security properties and outline trust assumptions. While the security impact of the trust model notes is limited, they



tain information that should be taken into account for the system's continued security.

4.1 Rollup

Admin

- The rollup admin (a multisig, acc. to the client) is set by setting the `owner` variable on Rollup initialization. The `owner` (and proxy owner) is in a very exposed position (risk) as it can theoretically take over complete control of the `Rollup` contract (upgrade, update settings, and facets), and this `Rollup` contract basically “owns” the bridge and, therefore, indirectly all the escrowed funds.
- The rollup admin controls who can interact with the Rollup user functionalities (`onlyValidator` modifier).
 - It is a closed system right now that is controlled by the rollup admin ((a) via access restrictions configurable by the admin, (b) via pausable, (c) the Rollup contract seems to be upgradeable/proxied).
 - The admin decides who can challenge/confirm/reject block submissions which might undermine trust in the system. e.g., `createChallenge` can only be called by `onlyValidator` . That list is manually maintained by the Rollup Admin, while the initial assumption was that anyone should be able to call out malicious behavior and force a challenge.
- System properties can be changed at any time, with only little input validation via the `RollupAdminFacet` (risk of misconfiguration).
- The admin can interfere with challenges up to a point where they can force a challenge outcome (`forceResolveChallenge`).
- The admin can `forceCreateNode` and `forceConfirmNode` to submit an unchallengeable block.
- The admin can pause the user functionalities. A similar effect might be created by an admin removing everyone from the `onlyValidator` access list.
- The contract can be upgraded due to the use of a proxy pattern. However, an admin can also choose to change the `Admin` and `User` facet while the contract is in use.
 - An admin can basically upgrade the contract to run any code.
 - An admin may interfere with users by unexpectedly upgrading user functionality while they interact with the system. (Consider only



allowing upgrades while the contract is paused.)

- The ownership transfer is one-step which might come with a significant risk of losing access to the contract.
 - If this is misconfigured, the AdminFacet will not be callable anymore.
- System parameter changes are not always sanity checked, increasing the risk for inconsistent configurations.
 - Parameter changes may interfere with users and block them from participating in the system or the system's sub-processes (e.g., challenges).
 - For example, a misconfiguration of `confirmedPeriodBlocks` to zero may allow anyone to initialize the contract again, setting a malicious admin facet, gaining control of the bridge, and/or destroying the Rollup contract.
 - `setStakeToken` bypasses sanity checks `UserFacet.initialize` would otherwise perform, leading to an inconsistent configuration. The stake already provided with the previously configured token will be accounted for in the new token, while the old token amounts will become inaccessible. Changing from an `ERC20` token to `ETH` may reset the contract's initialization state and allow anyone to initialize it again.
 - `setBaseStake` allows setting a staking requirement of zero, which may harm the system's security. Furthermore, the `baseStake` should be multiple times higher than the minimum gas required for one party to resolve a challenge, or else this may open up a griefing vector.

onlyValidator/User

- Can add `newStake` to become **Staker**.
- **Note** Race: might block `confirmNextNode` if stake joins just before the confirm call (might be intentional, griefing). A malicious stake can grief `confirmNextNode` and may grief for `removal` with zero risk by not taking on the latest valid tip. Note that the longer someone delays block confirmation, the higher the stake requirements.
- **Note: Anyone** can remove a new **Staker** immediately by calling `returnOldDeposit` if they don't bundle their call to `newStake` with `stakeOnExisting` / `stakeOnNewNode`.



- **Note:** A zombie cannot be a **Staker** unless they're removed from their **Zombie** status.
- Can add funds to any existing **Staker** (usually staker would do this).
- Can `confirmNextNode` if block deadlines are met and **all** stakers are staked on the next node (everyone agrees).
 - Can be blocked by a single staker not advancing their stake to a new tip.
- Can `rejectNextNode` if block deadlines expired and the node does not point to `lastConfirmed` or no staker is staked.
- Can `returnOldDeposit` on any Staker (griefing: when called on new staker that does not stake on the tip).
- Can `createChallenge` for competing nodes.
 - a challenge contract is created, and challengers are proving their nodes. finally, the challenge contract resolves the challenge by submitting the result via `completeChallenge`.
 - **Note:** It is assumed that this method should not be restricted to **onlyValidator** given that anyone should be able to challenge misbehavior.
- Can `removeZombie`.
 - Removes a given zombie from nodes it is staked on.
- Can `removeOldZombies`.
 - Remove any zombies whose latest stake is earlier than the first unresolved node.
- Can `withdrawStakerFunds`.
 - Stake that can be withdrawn (e.g., because `returnOldDeposit` or `reduceStake` where called) is accounted internally until `withdrawStakerFunds` is called, allowing a staker to pull their stake off the system.

Staker

Note: Stakers should always stake way ahead of `lastConfirmed` node, or else they might open themselves up to an attack where anyone can call `returnOldDeposit` on them if they are only staked on `lastConfirmed`.



Can `stakeOnExistingNode` (supporting a validated rollup block).

- Must stake on all previous nodes (or new staker).
 - Staker can optimize if too far off the tip by removing and re-adding themselves as staker (automatically stakes them on `latestConfirmed`) to then stake at the new validated tip.
- Can `stakeOnNewNode` (submitting a rollup block).
 - Assumes the previous node is `lastStakedNode`.
- Can `reduceStake` to the minimum required stake at that moment.
- Can “block” node confirmation by not staking at the newly agreed tip.
 - May be challenged if they disagree on the new tip.
 - May be removed from stakers via `returnOldDeposit` if they do not advance to the new `tip`.

Challenge contract (called on challenge end)

- Calls `completeChallenge` with the challenge’s final verdict.
 - 1/2 of loser stake is credited to the **winner** (accounted as stake for the winner).
 - 1/2 of loser stake is credited to the **contract owner** (as withdrawable funds).
 - The amount of stake that can be slashed from the loser is capped at the amount the winner has staked (excess is refunded to the loser).
 - Loser is turned into a zombie and loses its **Staker** status.

Zombie

- Cannot participate in the system until they’re removed from the zombie list.

4.2 Token Bridge

Notes:

- All admin functionality takes effect immediately. Users should ensure that the admin is a trustworthy time-locked/Multisig contract. One admin account has far-reaching permissions (consider splitting up responsibilities).



- All contracts are deployed as proxies to an implementation that can be upgraded at any time.
- The router does not escrow funds.
- `L1WETHGateway` unwraps `WETH -> ETH`, bridges `ETH` via the Arbitrum bridge (bridge escrows `ETH`) to L2, `L2WETHGateway` wraps `L2ETH -> L2WETH`. `WETHGateway` does not escrow funds.
- `ERC20Gateway` and `CustomGateway` escrow funds in their contracts.
- Risk of losing funds is concentrated on the bridge and token gateway contracts. If someone takes over/exploits the respective contract, all funds may be lost. There is no concept of a cold/hot escrow with speed breaks to reduce such risks.
- **Warning:** `ERC20Gateway` deploys a generic `L2ERC20` (`StandardArbERC20`) on `L2` that might not match the `L1` characteristics at all. According to the client, it is up to the users to ensure the token is bridgeable to their `L2` implementation. (Note: de/inflationary tokens, fee token, interest accumulating tokens, token with callbacks, e.g., ERC777, esoteric tokens may behave differently on L2 allowing for potential attacks.)
- `L1CustomGateway` admin may misconfigure existing token mapping to block `L2` users from spending their tokens.
- **Anyone** can theoretically call `initialize()` on all the proxy instances and implementations. Proxies, are initialized by the deployment scripts when deployed by a factory. Ensure that off-chain deployment scripts deploy & initialize in one transaction to mitigate front-running. Consider auto-initializing implementations in the constructor. Implementations are often not initialized, which bears some security risks, especially if `delegatecall` or `selfdestruct` are reachable for someone who can initialize the contracts (make sure this is well understood by devs/new hires and verified on deployment).

L1 - Router

The router does not escrow funds.

Admin/OnlyOwner



- Can transfer ownership (insecure one-step transfer; may lose access to contract if configured with a wrong address).
- Can `setDefaultGateway` on L1 and L2 (L1 side configures L2 via bridge message).
 - May configure a zero-address on L2 if `L1GW.counterpartGateway()` returns `address(0)`.
 - Can be overridden at any time. However, the L2 contract will still be accessible as access through the router is not enforced on L2 in contrast to L1.
- Can call `setGateways` on L1, passively configuring matching L2 gateways.
 - May result in a NOP (and a NOP call on L2) if an empty `_token`, `_gateway` array is provided (hiding a pot. erroneous call).
 - Allows overriding `token-address -> gateway handler contract` at any time.
- `proxyAdmin` can call `postUpgradeInit` after upgrade.

Additionally: `onlyWhitelist` can `updateWhitelistSource`. – Note that it is planned to remove the whitelist altogether.

Anyone/OnlyWhitelisted

- Can call `outboundTransfer` to initiate a token transfer from L1 -> L2 via configured Gateways.
 - Token gateway selection is enforced by the L1 token bridge. Direct interaction with Token Gateway is disabled.

NoOne/Disabled

- `setGateway` is currently disabled.
- `Router.finalizeInboundTransfer` is disabled and needs to be called directly on the gateway.

L2 - Router

Admin/OnlyOwner indirectly via onlyCounterpartGateway

- Bridge calls `L2GatewayRouter.setGateway` and `setDefaultGateway` from L1 admin calls. L1 effectively configures `L2GatewayRouter`.



- Note the interface inconsistency between `L1.setGateway` and `L2.setGateway`.

Anyone/OnlyWhitelisted

- Can call `outboundTransfer` to initiate a token transfer from L2 -> L1 via configured Gateways. This is **optional** and users can instead also directly interact with the L2 Token gateway contracts. The router, therefore, has no authority over which L2 gateways are currently usable. According to the client, this ensures users can always spend their L2 tokens back to L1.

NoOne/Disabled

- `Router.finalizeInboundTransfer` is disabled. **Note** the router and gateways should not necessarily share an interface given that they fulfill different purposes and most methods are overridden or functionality disabled.

L1 - Gateway (`WETHGateway` , `ERC20Gateway` , `CustomGateway`)

Admin/onlyOwner

- Can `L1CustomGateway.forceRegisterTokenToL2`.
 - **Note** that an admin can intentionally misconfigure this to block `L2->L1` token bridging.
- `proxyAdmin` can call `postUpgradeInit` after upgrade.

Router

- Can initiate `outboundTransfer`, which sends a bridge message from `L1->L2` to the `L2` counterpart.
 - Note: For `ERC20Gateway`, the first call to bridge a token configures the `L2.StandardArbERC20`. Allowing any `ERC20` token to be used (e.g. `DefaultGateway == ERC20Gateway`) may be dangerous due to `L1 -> L2` token contract mismatches.

onlyCounterpartGateway



Only L2 gateway counterpart (via bridge call `L2->L1`) can `finalizeInboundTransfer` after rollup block confirmation. (Basically initiated by a

validator when confirming a rollup block.)

Anyone

- May be able to register their own custom tokens in the future. This is disabled now, and we would not advise enabling this without moderation as it might be misused for malicious activities.

NoOne/Disabled

- `L1CustomGateway.registerTokenToL2` is disabled.

L2 - Gateway (`WETHGateway` , `ERC20Gateway` , `CustomGateway`)

Admin/onlyOwner

- `proxyAdmin` can call `postUpgradeInit` after upgrade.

onlyCounterpartGateway (Note: either L1 counterpart address directly, or L2 aliased L1 address)

- `L1CustomGateway.registerTokenFromL1` can only be called by an admin on L1 via `L1->L2` bridge call.
- `Gateway.finalizeInboundTransfer` is called via `L1Gateway.outboundTransfer` `L1->L2` bridge call.
 - For `L2ERC20Gateway` , the first call deploys the “dummy” `StandardArbERC20` token.
- Only L2 gateway counterpart (via bridge call `L2->L1`) can `finalizeInboundTransfer` after rollup block confirmation. (Basically initiated by a validator when confirming a rollup block.)

Anyone

- Can call `outboundTransfer` on the gateway directly to bridge tokens from `L2 -> L1` .
 - Even if the `Router` does not allow the token anymore.



Each issue has an assigned severity:

- **Minor** issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- **Medium** issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- **Major** issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- **Critical** issues are directly exploitable security vulnerabilities that need to be fixed.

5.1 Router/Gateway - Reentrancy in inbound/outbound transfer allows arbitrary creation of L2 tokens **Critical**

Description

Inbound/OutboundTransfer is missing protection to prevent the `tokenFrom` account from reentering through a token callback (e.g., ERC777). This reentrancy allows the caller to effectively escrow fewer tokens on L1 than are being minted on L2.

Examples

Let's assume the following scenario:

- We bridge the custom token "TOK"
- `TOK.balanceOf(msg.sender) = 100`
- `TOK.balanceOf(gateway) = 0` (initial balance)



```

L1GwRouter.outboundTransfer(TOK, ..., 50){💰 for L2Gas}
--> //GwRouter.outboundTransfer(...)
  --> Gateway(CustomGateway).outboundTransfer(...){💰 for L2Gas}
    --> //L1ArbitrumGateway.outboundTransfer(...)
    --> //L1ArbitrumGateway.outboundEscrowTransfer(TOK, ..., 50)
      prevBalance = 0 (erc20.balanceOf(this))
      --> tok.transferFrom(from, this, 50)
      //start-----subcall-----
      //(reentering *from*, before balance changes: ERC777)
        ==> calls *from* who calls ==> L1GwRouter.outboundTransfer(TOK, ..., 50)
          --> //L1ArbitrumGateway.outboundEscrowTransfer(TOK, ..., 50)
            prevBalance = 0 (erc20.balanceOf(this))

            --> tok.transferFrom(from, this, 50) //new balanceOf(msg.sender) = 100 - 50
            postBalance = 50 (erc20.balanceOf(this))
            DIFF = 50 - 0 = 50 ==> createOutboundTx(amount=50)

          //end-----subcall-----
          //after subcall/transferFrom: new balanceOf(msg.sender) = 50 - 50 = 0
          postBalance = 100 (erc20.balanceOf(this))
          DIFF = 100 - 0 = 100 ==> createOutboundTx(amount=100)

```

In this scenario, the attacker spent/escrowed 100 L1 TOK to create 150 L2 TOK (ERC777, or any custom token w. callback)

An external call (`ERC777.transferFrom`) allows the `from` address to reenter the `outboundTransfer` function, changing the balance of the contract after `prevBalance` but before `postBalance` is recorded.

code/packages/arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1ArbitrumGateway.sol:L234-L245

```

function outboundEscrowTransfer(
    address _l1Token,
    address _from,
    uint256 _amount
) internal virtual returns (uint256 amountReceived) {
    // this method is virtual since different subclasses can handle escrow differently
    // user funds are escrowed on the gateway using this function
    uint256 prevBalance = IERC20(_l1Token).balanceOf(address(this));
    IERC20(_l1Token).safeTransferFrom(_from, address(this), _amount);
    uint256 postBalance = IERC20(_l1Token).balanceOf(address(this));
    return SafeMath.sub(postBalance, prevBalance);
}

```



Note: This attack can theoretically also happen in `L2CustomGateway` if the custom L2 implements callbacks:

code/packages/arb-bridge-peripherals/contracts/tokenbridge/arbitrum/gateway/L2CustomGateway.sol: L53-L70

```
function outboundEscrowTransfer(
    address _l2Token,
    address _from,
    uint256 _amount
) internal override returns (uint256 amountBurnt) {
    uint256 prevBalance = IERC20(_l2Token).balanceOf(_from);

    // in the custom gateway, we do the same behaviour as the superclass, but account
    // for the balances of tokens to ensure that inflationary / deflationary changes
    // are taken into account
    // we ignore the return value since we actually query the token before and after
    // the amount of tokens that were burnt
    super.outboundEscrowTransfer(_l2Token, _from, _amount);

    uint256 postBalance = IERC20(_l2Token).balanceOf(_from);
    return SafeMath.sub(prevBalance, postBalance);
}
```

Recommendation

Add reentrancy protection (ReentrancyGuard) to both the outbound (escrow) and inbound (release) interaction flows (L1 and L2).

5.2 RollupAdmin - changing stake token may render existing stake inaccessible Major

Description

The Rollup Admin can change the token used for staking in a live deployment of the `Rollup` contract, which will always lead to inconsistencies or tokens stuck in the `Rollup` contract until it is upgraded to support a token change properly.



First, `setStakeToken` does not enforce any sanity checks on the system's state. Examples can include:

- require a `paused` state and no stake tokens currently owned by the system
- parametrization of the `ERC20` or `ETH` user facet
- checks from `initialize` to make sure the system has been appropriately initialized
- future initialization checks to prevent accidental reinitialization

Furthermore, if a user already staked tokens to the `Rollup` contract and an admin changes the staking token, their deposit will become inaccessible while still credited as a stake in the new token. Changes to the staking token can mess up the system's internal accounting.

One would assume that the proper way of changing the staking token would be to first `pause` the contract, then force everyone to withdraw their stake, change the token and `unpause` to let stakers continue with their duties. The problem is that all user methods are `whenNotPaused`. Hence, there is no way to `withdraw/unstake/force-unstake` (via `returnOldDeposit`) existing stake. Effectively, there is no consistent way of changing the staking token unless this is accompanied by a contract upgrade that facilitates that.

If the staking token is changed while users are staked, their old tokens are lost, their current stake will be accounted for in the new token, but they will very likely not be able to get out either old or new stake tokens (unless the contract holds enough new stake tokens).

An admin may call `forceRefundStaker` on the staker address, but this does force them to pull out the tokens, and, hence, the old token may be locked in the contract.

Examples

[code/packages/arb-bridge-eth/contracts/rollup/facets/RollupAdmin.sol:L153-L156](#)



```
function setStakeToken(address newStakeToken) external override {
    stakeToken = newStakeToken;
    emit OwnerFunctionCalled(13);
}
```

code/packages/arb-bridge-eth/contracts/rollup/facets/RollupUser.sol:L645-L657

```
*/
function withdrawStakerFunds(address payable destination)
    external
    override
    onlyValidator
    whenNotPaused
    returns (uint256)
{
    uint256 amount = withdrawFunds(msg.sender);
    // This is safe because it occurs after all checks and effects
    require(IERC20(stakeToken).transfer(destination, amount), "TRANSFER_FAILED");
    return amount;
}
```

Recommendation

This and a variety of other admin methods carry a significant risk of misconfiguration. The contract can be left vulnerable to reinitialization, messing up internal accounting (active stake during token change), or having other unexpected effects on user interaction.

Consider removing this method or providing a concrete process to change the staking token of a live system. Consider requiring the contract to be paused when changing the setting. Force users' stake to become withdrawable when changing the token. Allow users to withdraw tokens they previously provided. Require users to add a new stake in the newly configured token.

5.3 Bisection degree changes can enable malicious challenges Major



Description

In the `Challenge` contract's `bisectExecution` function, it is required that the number of previously obtained chain hashes matches up with the result receiver's target bisection degree. In the system's context, the result receiver is the `RollupUser` facet.

However, an admin can change the facet's bisection degree. Already initiated challenges in an ongoing bisection process will fetch the facet's new bisection degree and always revert as the new degree does not match the pre-existing chain hashes. Both stakers are already marked as challenged in the `RollupUser` facet. They cannot restart the challenge, and the final counterparty to act will lose.

In a scenario where admins announce a delayed upgrade of the bisection degree parameter, an attacker can start a challenge and use their time contingent to stall the challenge's progress until the upgrade becomes effective. As a result, the counterparty will not be able to progress bisection as it always reverts, eventually resulting in a timeout and a potentially legitimate staker getting punished.

Consequentially, an attacker can submit arbitrary state changes and bypass the threat of being challenged - assuming that a parameter change of the bisection degree is incoming.

Examples

code/packages/arb-bridge-eth/contracts/challenge/Challenge.sol:L161-L167

```
uint256 challengeExecutionBisectionDegree = resultReceiver
    .challengeExecutionBisectionDegree();
require(
    _chainHashes.length ==
        bisectionDegree(_challengedSegmentLength, challengeExecutionBisectionDe
    "CUT_COUNT"
);
```

Recommendation



When a new `Challenge` contract is initialized, it should copy the user facet's system parameters into its storage and retrieve them from there for future

reference. Mirrored constant contract state variables will prevent admin upgrades from interfering with ongoing challenges while allowing subsequent challenges to reflect the new parameter set.

5.4 ERC20Gateway - Potential L1-to-L2 ERC20 Token

inconsistencies: decimals Major

Description

When bridging an ERC20 token transfer, the `L1ERC20Gateway` calls out to the respective token to retrieve `name`, `symbol`, and `decimals`. This external call is realized via a low-level `staticcall`, which returns the result as a byte-sequence, or, if the call fails, returns an empty `bytes memory`.

“Non-standard” ERC20 implementations (e.g., no getter for `decimals`) are gracefully accepted and mapped to a `StandardArbErc20Token` on the L2 side. If one of `name`, `symbol`, or `decimals` is not provided, a default ("", "", 18) will be assumed. If the L1 token `decimals` is of a different type, this might result in a broken initialization on L2.

Assuming token properties can be dangerous as behavior between L1 and L2 might be different. For example, if an L1 token has 0 decimals but does not expose the `decimals` getter, the code will assume an 18 decimals on the L2 side. A token transfer of 10.000 (decimals 0 -> 10.000) tokens on L1 will show up as $10.000 * 10^{-18}$ on an L2 UI visualizing the token value.

Examples

- L1 side

code/packages/arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1ERC20Gateway.sol:L62-L73



```

function callStatic(address targetContract, bytes4 targetFunction)
    internal
    view
    returns (bytes memory)
{
    (
        ,
        /* bool success */
        bytes memory res
    ) = targetContract.staticcall(abi.encodeWithSelector(targetFunction));
    return res;
}

```

code/packages/arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1ERC20Gateway.sol:L75-L97

```

function getOutboundCalldata(
    address _token,
    address _from,
    address _to,
    uint256 _amount,
    bytes memory _data
) public view override returns (bytes memory outboundCalldata) {
    // TODO: cheaper to make static calls or save isDeployed to storage?
    bytes memory deployData = abi.encode(
        callStatic(_token, ERC20.name.selector),
        callStatic(_token, ERC20.symbol.selector),
        callStatic(_token, ERC20.decimals.selector)
    );

    outboundCalldata = abi.encodeWithSelector(
        ITokenGateway.finalizeInboundTransfer.selector,
        _token,
        _from,
        _to,
        _amount,
        GatewayMessageHandler.encodeToL2GatewayMsg(deployData, _data)
    );
}

```



L2 side

code/packages/arb-bridge-peripherals/contracts/tokenbridge/arbitrum/StandardArbERC20.sol:L36-L50

```
function bridgeInit(address _l1Address, bytes memory _data) public virtual {
    (bytes memory name_, bytes memory symbol_, bytes memory decimals_) = abi.de
        _data,
        (bytes, bytes, bytes)
    );
    // what if decode reverts? shouldn't as this is encoded by L1 contract

    L2GatewayToken._initialize(
        BytesParserWithDefault.toString(name_, ""),
        BytesParserWithDefault.toString(symbol_, ""),
        BytesParserWithDefault.toUint8(decimals_, 18),
        msg.sender, // _l2Gateway,
        _l1Address // _l1Counterpart
    );
}
```

Note that using the `BytesParserWithDefault` library appears to implement graceful deploy data parsing. The system should avoid using this functionality since it might be misused to create L1-L2 token discrepancies, which an attacker can exploit.

Recommendation

Avoid assuming or falling back to token defaults. The source token should be mimicked as closely as possible on the L2 chain. A token that fails to provide the required information to be bridged to the L2 chain should not be accepted or handled by an L2 “clone” token that replicates the L1 behavior as closely as possible.

To enforce strict source token requirements, one could use the original `IERC20` interface to retrieve the values and fail if the source token cannot provide them.

5.5 RollupAdmin Facet - Bad configuration can enable malicious re-initialization Major



Description

AdminFacet methods are missing basic input validation. For example, an erroneous call resetting `confirmPeriodBlocks` to `0` allows anyone to call `initialize()` again, which could end up in an arbitrary code execution taking over the `Rollup` contract. Note that `Rollup` owns `Bridge`, and `Bridge` stores the system's assets.

Examples

- `setConfirmPeriodBlocks` allows to reset `confirmPeriodBlocks` to `0` which would allow **anyone** to call `initialize()` again performing an arbitrary `delegateCall` via `userFacet.initialize` which allows arbitrary code execution.

code/packages/arb-bridge-eth/contracts/rollup/facets/RollupAdmin.sol:L111-L118

```
/**
 * @notice Set number of blocks until a node is considered confirmed
 * @param newConfirmPeriod new number of blocks
 */
function setConfirmPeriodBlocks(uint256 newConfirmPeriod) external override {
    confirmPeriodBlocks = newConfirmPeriod;
    emit OwnerFunctionCalled(9);
}
```

code/packages/arb-bridge-eth/contracts/rollup/Rollup.sol:L85-L88

```
function isInit() internal view returns (bool) {
    return confirmPeriodBlocks != 0;
}
```

- `setStakedToken` does not take the input validation requirements of UserFacet into account (`stakedToken` must be `0x0` for EthUserFacet, must not be `0x0` for Erc20UserFacet). May allow anyone to call `initialize` on Erc20UserFacet if `stakedToken` is reset to `0x0`.

code/packages/arb-bridge-eth/contracts/rollup/facets/RollupAdmin.sol:L153-



```
function setStakeToken(address newStakeToken) external override {
    stakeToken = newStakeToken;
    emit OwnerFunctionCalled(13);
}
```

Recommendation

Additional `requires` should be added to safeguard against potential misconfiguration that will practically reset the contract's initialization.

5.6 Outbox - unsafe downcast Medium

Description

When `Outbox.executeTransaction` is called, in the context of `Outbox.executeBridgeCall`, certain "execution context variables" are preserved and exposed to a potential callee. However, the preserved data types are smaller than the types of the original value, which may be unsafe as it can lead to an integer type value wrapping (e.g. `uint256 -> uint128`). In such a case, the callee might be presented with inaccurate information about the execution context.

Examples

At least `batchNum` can be an arbitrary `uint256`. Therefore, downcasting it to `uint128` may be unsafe and lead to an integer value wrap.

code/packages/arb-bridge-eth/contracts/bridge/Outbox.sol:L179-L187

```
context = L2ToL1Context({
    sender: l2Sender,
    l2Block: uint128(l2Block),
    l1Block: uint128(l1Block),
    timestamp: uint128(l2Timestamp),
    batchNum: uint128(batchNum),
    outputId: outputId
});
```



code/packages/arb-bridge-eth/contracts/bridge/Outbox.sol:L49-L49


```
mapping(uint256 => OutboxEntry) public outboxEntries;
```

code/packages/arb-bridge-eth/contracts/bridge/Outbox.sol:L115-L119

```
require(data.length == 97, "BAD_LENGTH");
uint256 batchNum = data.toUint(1);
// Ensure no outbox entry already exists w/ batch number
require(!outboxEntryExists(batchNum), "ENTRY_ALREADY_EXISTS");
```

OldOutbox:

code/packages/arb-bridge-eth/contracts/bridge/Old_Outbox/OldOutbox.sol:L164-L169

```
_sender = l2Sender;
_l2Block = uint128(l2Block);
_l1Block = uint128(l1Block);
_timestamp = uint128(l2Timestamp);
```

Recommendation

We recommend using safe casts or preserving the type of the original value.

5.7 Inconsistent bridge security assurances Medium

Description

According to the developer team, the token bridge should give users the assurance that their L2 tokens will always be redeemable to L1. As a result of this requirement, anyone can bypass the L2 router by calling the respective L2 gateway's `outboundTransfer()` function directly.

code/packages/arb-bridge-



peripherals/contracts/tokenbridge/arbitrum/gateway/L2ArbitrumGateway.sol

L134-L174

```

function outboundTransfer(
    address _l1Token,
    address _to,
    uint256 _amount,
    uint256, /* _maxGas */
    uint256, /* _gasPriceBid */
    bytes calldata _data
) public payable virtual override returns (bytes memory res) {
    // This function is set as public and virtual so that subclasses can override
    // it and add custom validation for callers (ie only whitelisted users)

    // the function is marked as payable to conform to the inheritance setup
    // this particular code path shouldn't have a msg.value > 0
    // TODO: remove this invariant for execution markets
    require(msg.value == 0, "NO_VALUE");

    address _from;
    bytes memory _extraData;
    {
        if (isRouter(msg.sender)) {
            (_from, _extraData) = GatewayMessageHandler.parseFromRouterToGateway(msg.sender, _data);
        } else {
            _from = msg.sender;
            _extraData = _data;
        }
    }
    // the inboundEscrowAndCall functionality has been disabled, so no data is allowed
    require(_extraData.length == 0, "EXTRA_DATA_DISABLED");

    uint256 id;
    {
        address l2Token = calculateL2TokenAddress(_l1Token);
        require(l2Token.isContract(), "TOKEN_NOT_DEPLOYED");
        require(IArbToken(l2Token).l1Address() == _l1Token, "NOT_EXPECTED_L1_TOKEN");

        _amount = outboundEscrowTransfer(l2Token, _from, _amount);
        id = triggerWithdrawal(_l1Token, _from, _to, _amount, _extraData);
    }
    return abi.encode(id);
}

```



While this effectively undermines the L2 router's authority over configured/allowed gateways, it prevents a gateway from being rendered

unusable once it's not listed on the L1 router anymore, and thus prevents user tokens on L2 from getting stuck.

While this requirement holds for ERC20 Gateways, it does not for the custom token gateway. An administrator can register custom L1-L2 token pairs using `L2CustomGateway.registerTokenFromL1` OR `L1CustomGateway.forceRegisterTokenToL2`. Both functions effectively accept a list of L1-L2 token address mappings.

This mapping can be overridden by the admin at any time (<https://github.com/ConsenSys/arbitrum-audit-2021-10/issues/18>). If an administrator updates an existing L1-L2 token mapping to one pointing to an invalid L2 address, i.e. address is not a contract, the L2 token will be rendered unbridgeable from L2 to L1 until that (mis-)configuration is reversed.

Examples

L1 -> L2 token registration:

**code/packages/arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1CustomGateway.sol:
L153-L175**



```

function forceRegisterTokenToL2(
    address[] calldata _l1Addresses,
    address[] calldata _l2Addresses,
    uint256 _maxGas,
    uint256 _gasPriceBid,
    uint256 _maxSubmissionCost
) external payable returns (uint256) {
    require(msg.sender == owner, "ONLY_OWNER");
    require(_l1Addresses.length == _l2Addresses.length, "INVALID_LENGTHS");

    for (uint256 i = 0; i < _l1Addresses.length; i++) {
        // here we assume the owner checked both addresses offchain before force
        // require(address(_l1Addresses[i]).isContract(), "MUST_BE_CONTRACT");
        L1ToL2Token[_l1Addresses[i]] = _l2Addresses[i];
        emit TokenSet(_l1Addresses[i], _l2Addresses[i]);
    }

    bytes memory _data = abi.encodeWithSelector(
        L2CustomGateway.registerTokenFromL1.selector,
        _l1Addresses,
        _l2Addresses
    );

```

code/packages/arb-bridge-peripherals/contracts/tokenbridge/arbitrum/gateway/L2CustomGateway.sol: L83-L93

```

function registerTokenFromL1(address[] calldata l1Address, address[] calldata l2Address)
    external
    onlyCounterpartGateway
{
    // we assume both arrays are the same length, safe since its encoded by the L2 oracle
    for (uint256 i = 0; i < l1Address.length; i++) {
        // here we don't check if l2Address is a contract and instead deal with
        // in `handleNoContract` this way we keep the l1 and l2 address oracles
        L1ToL2Token[l1Address[i]] = l2Address[i];
        emit TokenSet(l1Address[i], l2Address[i]);
    }
}

```



If the L2 address is set to a non-contract, the `outboundTransfer` call reverts. As a result, the custom token cannot be bridged to L1 anymore. This can be caused

by an admin updating the mapping at any time:

code/packages/arb-bridge-peripherals/contracts/tokenbridge/arbitrum/gateway/L2ArbitrumGateway.sol:L165-L168

```
address l2Token = calculateL2TokenAddress(_l1Token);
require(l2Token.isContract(), "TOKEN_NOT_DEPLOYED");
require(IArbToken(l2Token).l1Address() == _l1Token, "NOT_EXPECTED_L1_TOKEN");
```

Recommendation

While `ERC20` tokens can always be redeemed to L1, this does not seem to be the case with custom tokens once their internal L1-L2 token mapping is updated by an admin. Consider disallowing updates for existing L1-L2 custom token mappings. The bridge's assurances on tokens transfers and recovery should be clearly communicated to the users.

5.8 Lacking separation of concerns between Router and Gateway functionality Medium

Description

The `GatewayRouter` implements the `TokenGateway` interface. However, the domains of a router and a gateway are different and should not overlap. Consequentially, the `GatewayRouter` should implement a separate router interface with different interface naming, while the gateways adhere to the `TokenGateway` interface.

As the interface concerns are not cleanly separated at the moment, some router functions always revert.

Examples

code/packages/arb-bridge-peripherals/contracts/tokenbridge/libraries/gateway/GatewayRouter.sol:L68-L76



```
function finalizeInboundTransfer(  
    address, /* _token */  
    address, /* _from */  
    address, /* _to */  
    uint256, /* _amount */  
    bytes calldata /* _data */  
) external payable virtual override {  
    revert("ONLY_OUTBOUND_ROUTER");  
}
```

Recommendation

Cleanly separate gateway and router interfaces, and refactor router- and gateway-related functions to avoid unnecessary reverts.

5.9 Unpredictable behavior due to admin front running or general bad timing Medium

Description

In several cases, privileged accounts can update or upgrade things in the system without warning. Unannounced upgrades have the potential to violate the system's security goals.

Specifically, privileged roles could use front-running to make malicious changes just ahead of configuration-changing transactions, or random adverse effects could occur due to the unfortunate timing of changes.

Some instances of this issue are more significant than others, but in general, users of the system should have assurances about the behavior of the action they're about to take.

Examples

- `Rollup` (most settings, significantly changing facets)
- `RollupAdminFacet`
- Upgradeable proxies



`TokenBridge` - `CustomTokenGateway` - token mappings can be overridden/unset/set to invalid L2 addresses at any time by providing an L2

token address of `address(0x0)`, resulting in the L2 token becoming unbridgeable.

Recommendation

The underlying issue is that users of the system can't be sure of a function call's behavior. Due to potential configuration changes, the behavior can change at any time.

Consider giving users a time-locked notice of changes in advance. The first step merely tells users that the system will execute a particular change. The second step commits that change after a reasonable waiting period.

5.10 Missing deployment consistency checks Medium

Description

The deployment scripts contained in `arb-bridge-eth` and `arb-bridge-peripherals` do not perform consistency checks after deployed target contracts. If a sanity check fails on a new deployment, the scripts should cancel the deployment process and raise an error to inform the deployer about potentially malicious front-running behavior.

Recommendation

Especially when non-constructor initialization functions are used in the deployment flow, we recommend performing basic sanity checks to assert that no front-running attacks have been completed between contract deployment and their initialization.

5.11 Missing constructors and manual initialization Medium

Description

Various smart contracts in the system require custom initialization functions to be called. The point in time when these calls happen is up to the deploying address. While some initialization functions have authorization or property-based checks to avoid duplicate or malicious initialization, others do not.



Examples

code/packages/arb-bridge-eth/contracts/challenge/Challenge.sol:L99-L110

```
function initializeChallenge(
    IOneStepProof[] calldata _executors,
    address _resultReceiver,
    bytes32 _executionHash,
    uint256 _maxMessageCount,
    address _asserter,
    address _challenger,
    uint256 _asserterTimeLeft,
    uint256 _challengerTimeLeft,
    ISequencerInbox _sequencerBridge,
    IBridge _delayedBridge
) external override {
```

Recommendation

It is recommended to use constructors wherever possible to initialize contracts during deploy-time immediately. If other initialization functions are used, we recommend a standardized, top-level `initialized` boolean, set to `true` on the first deployment and used to prevent future initialization.

Using constructors and locked-down initialization functions will significantly reduce the probability of developer errors and the possibility of attackers re-initializing vital system components.

5.12 Rollup - A reentrant StakeToken may allow zero-fee flash-loans Minor

Description

The `ERC20RollupUserFacet` allows users to stake and unstake in one transaction. In case a reentrant `StakeToken` is configured (i.e., `ERC-777` or equivalent), a user can take a zero-fee flash loan by performing the following steps:

1. Call `newStake()` and provide the minimum stake required (e.g., 1000 TOK).
2. When `newStake()` pulls in the token via `token.transferFrom()`, an `ERC-777` token executes callback `from.tokensToSend()` before the token is actually transferred.



The new stake, however, is already added to the internal accounting.

3. In the reentrant `from.tokensToSend()` callback, the user recursively calls `Rollup.addToDeposit()` to inflate `Rollup`'s internal accounting even more. No tokens have been transferred yet.
4. Repeat until the desired amount (internal accounting) is reached (max. `token.balanceOf(Rollup)`).
5. In the next `from.tokensToSend()` , first call `Rollup.returnOldDeposit()` to move inflated stake to `withdrawableAmount` and then `withdrawStakerFunds()` to pull tokens to the user's address. (Note: No token has been transferred from user to `Rollup` yet.)
6. `withdrawStakerFunds()` calls `token.transfer()` , which triggers the `recipient.tokensReceived()` callback (ERC-777).
7. In `recipient.tokensReceived()` , perform the flash loan target activity (e.g., DEX operation; bypass stake requirement and confirm block in one tx; ...) and make sure the loaned amount is returned to the user right after. That is required to pay back the loan.
8. Let the recursive calls unwrap with the ERC-777 internal token transfers taking place. This will return the loaned amount to `Rollup` and finally synchronize the external with the internal accounting.

Examples

code/packages/arb-bridge-eth/contracts/rollup/facets/RollupUser.sol:L617-L624

```
function newStake(uint256 tokenAmount) external onlyValidator whenNotPaused {
    _newStake(tokenAmount);
    require(
        IERC20(stakeToken).transferFrom(msg.sender, address(this), tokenAmount)
        "TRANSFER_FAIL"
    );
}
```

code/packages/arb-bridge-eth/contracts/rollup/facets/RollupUser.sol:L629-L640



```

*/
function addToDeposit(address stakerAddress, uint256 tokenAmount)
    external
    onlyValidator
    whenNotPaused
{
    _addToDeposit(stakerAddress, tokenAmount);
    require(
        IERC20(stakeToken).transferFrom(msg.sender, address(this), tokenAmount)
        "TRANSFER_FAIL"
    );
}

```

code/packages/arb-bridge-eth/contracts/rollup/facets/RollupUser.sol:L646-L657

```

function withdrawStakerFunds(address payable destination)
    external
    override
    onlyValidator
    whenNotPaused
    returns (uint256)
{
    uint256 amount = withdrawFunds(msg.sender);
    // This is safe because it occurs after all checks and effects
    require(IERC20(stakeToken).transfer(destination, amount), "TRANSFER_FAILED");
    return amount;
}

```

Recommendation

Add a reentrancy guard to methods potentially handling tokens with callbacks. Alternatively, require a stake to be locked for at least one block.

5.13 Outdated Solidity version Minor

Description

Most of the codebase uses the Solidity version pragma `^0.6.11` and the compiler version 0.6.11. Apart from the fact that there's a v0.6.12 release, the v0.6 series is outdated when writing this report, and there are [known bugs](#).

Recommendation

We recommend an upgrade to the latest release of the v0.8 series of the Solidity compiler. However, this means the entire codebase has to be reviewed and adapted to the newer Solidity version. One example of this is given in [issue 6.11](#).


5.14 (Old)Outbox - handleOutgoingMessage message validation Minor

Description

In `Outbox` and `OutboxEntry.root` of `bytes32(0x0)` indicates that the outbox for a specific `batchNum` is not yet set. While there is no proof that every output of common hash functions is reachable for some input and the Merkle root is based on repeated Keccak operations, a Merkle root of `0x00...00` can theoretically occur. Consequentially, using the zero-value as an indicator that the `OutboxEntry` is unset may become problematic unless it is guaranteed that a root of `bytes32(0)` will not occur.

`OldOutbox` enforces `root` to be non-zero (see `OutboxEntry.initialize`) while new `Outbox` silently accepts it (which theoretically allows the `batchNum` to be overridden again).

From the `OldOutbox` implementation, we assume that `batchNum` is related to the `outboxIndex` in the `outboxes` array as otherwise, the `outboxEntryExists` check would not work. This check, however, is not enforced in `handleOutgoingMessage`, and instead, the `batchNum` is dropped in favor of the `outboxIndex`. The original `batchNum` in both the new `Outbox` and the `OldOutbox` is not enforced to be strictly monotonically increasing. The new `Outbox` indexes are based on the `batchNum` while the `OldOutbox` uses the `outboxIndex` (outboxes array length).

Whether the `batchNum` sequence and Merkle root rules are enforced in the component that generates the message and can be challenged might be safe to omit these checks. In general, however, the stricter the message handling is, the less attack surface is available, which might positively contribute to the security posture of the system (if a message is invalid according to the system's rules,  block is likely malicious, and its output should not be spendable)

Examples

- `Outbox` zero Merkle root

code/packages/arb-bridge-eth/contracts/bridge/Outbox.sol:L112-L131

```
function handleOutgoingMessage(bytes memory data) private {
    // Otherwise we have an unsupported message type and we skip the message
    if (data[0] == MSG_ROOT) {
        require(data.length == 97, "BAD_LENGTH");
        uint256 batchNum = data.toUint(1);
        // Ensure no outbox entry already exists w/ batch number
        require(!outboxEntryExists(batchNum), "ENTRY_ALREADY_EXISTS");

        // This is the total number of msgs included in the root, it can be used
        // detect when all msgs were executed against a root.
        // It currently isn't stored, but instead emitted in an event for utility
        uint256 numInBatch = data.toUint(33);
        bytes32 outputRoot = data.toBytes32(65);

        OutboxEntry memory newOutboxEntry = OutboxEntry(outputRoot);
        outboxEntries[batchNum] = newOutboxEntry;
        // keeping redundant batchnum in event (batchnum and old outboxindex fie.
        emit OutboxEntryCreated(batchNum, batchNum, outputRoot, numInBatch);
    }
}
```

code/packages/arb-bridge-eth/contracts/bridge/Outbox.sol:L272-L274

```
function outboxEntryExists(uint256 batchNum) public view override returns (bool)
    return outboxEntries[batchNum].root != bytes32(0);
}
```

- `OldOutbox` unchecked `batchNum`

code/packages/arb-bridge-eth/contracts/bridge/Old_Outbox/OldOutbox.sol:L106-L119



```
function handleOutgoingMessage(bytes memory data) private {
    // Otherwise we have an unsupported message type and we skip the message
    if (data[0] == MSG_ROOT) {
        require(data.length == 97, "BAD_LENGTH");
        uint256 batchNum = data.toUint(1);
        uint256 numInBatch = data.toUint(33);
        bytes32 outputRoot = data.toBytes32(65);

        address clone = address(new BeaconProxy(address(beacon), ""));
        OutboxEntry(clone).initialize(outputRoot, numInBatch);
        uint256 outboxIndex = outboxes.length;
        outboxes.push(OutboxEntry(clone));
        emit OutboxEntryCreated(batchNum, outboxIndex, outputRoot, numInBatch);
    }
}
```

- duplicate Merkle roots should not be allowed

Recommendation

Require that `outputRoot != bytes32(0)`, both when generating the root and decoding it in `handleOutgoingMessage`. Check `batchNum` to be strictly monotonically increasing, highest seen `batchNum < amountBatchesProcessed`, duplicate `batchNum`'s (`01dOutbox`).

5.15 RollupUser - removeZombie() off-by-one comparison check Minor

Description

The user-provided `zombieNum` argument cannot exceed the value of `zombieCount()` (i.e. `_zombies.length`) as zombies are zero-indexed. Providing a `zombieNum == zombieCount()` will always fail with an assertion consuming all gas in `zombieAddress()` due to an out-of-bounds array access.

Examples

code/packages/arb-bridge-eth/contracts/rollup/facets/RollupUser.sol:L381-L393



```

/**
 * @notice Remove the given zombie from nodes it is staked on, moving backwards 1
 * @param zombieNum Index of the zombie to remove
 * @param maxNodes Maximum number of nodes to remove the zombie from (to limit th
 */
function removeZombie(uint256 zombieNum, uint256 maxNodes)
    external
    onlyValidator
    whenNotPaused
{
    require(zombieNum <= zombieCount(), "NO_SUCH_ZOMBIE");
    address zombieStakerAddress = zombieAddress(zombieNum);
    uint256 latestNodeStaked = zombieLatestStakedNode(zombieNum);

```

Recommendation

Change the comparison operator `<=` to strictly `<`.

5.16 Avoid ineffective calls as they may hide misconfiguration

Minor

Description

Ineffective calls that do not lead to state changes may indicate misconfiguration and should be avoided as such. For example, if an admin calls `L1GatewayRouter.setGateways` but does not provide any gateways, the call returns successfully without changing the system's state. When a cross-chain message is emitted, most likely, no change is performed on the L2 side either. In some cases, events will be emitted that may be picked up by third-party components, which then perform actions on data that did not cause any state changes.

Examples

code/packages/arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1GatewayRouter.sol:L97-L106



```
function _setGateways(
    address[] memory _token,
    address[] memory _gateway,
    uint256 _maxGas,
    uint256 _gasPriceBid,
    uint256 _maxSubmissionCost,
    address _creditBackAddress
) internal returns (uint256) {
    require(_token.length == _gateway.length, "WRONG_LENGTH");
```

code/packages/arb-bridge-eth/contracts/rollup/facets/RollupAdmin.sol:L238-L254

```
function forceResolveChallenge(address[] memory stakerA, address[] memory stakerB)
    external
    override
    whenPaused
{
    require(stakerA.length == stakerB.length, "WRONG_LENGTH");
    for (uint256 i = 0; i < stakerA.length; i++) {
        address chall = inChallenge(stakerA[i], stakerB[i]);

        require(address(0) != chall, "NOT_IN_CHALL");
        clearChallenge(stakerA[i]);
        clearChallenge(stakerB[i]);

        IChallenge(chall).clearChallenge();
    }
    emit OwnerFunctionCalled(21);
}
```

code/packages/arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1CustomGateway.sol:L153-L162



```
function forceRegisterTokenToL2(
    address[] calldata _l1Addresses,
    address[] calldata _l2Addresses,
    uint256 _maxGas,
    uint256 _gasPriceBid,
    uint256 _maxSubmissionCost
) external payable returns (uint256) {
    require(msg.sender == owner, "ONLY_OWNER");
    require(_l1Addresses.length == _l2Addresses.length, "INVALID_LENGTHS");
```

code/packages/arb-bridge-eth/contracts/rollup/facets/RollupAdmin.sol:L84-L86

```
function setValidator(address[] memory _validator, bool[] memory _val) external
    require(_validator.length == _val.length, "WRONG_LENGTH");
```

There might be more similar cases in the codebase.

Recommendation

Avoid ineffective calls unless there is a good reason to allow them, such as NOP's for functions not supposed to revert. An error should be returned for external interfaces consumed by users or administrators when the performed call seems to have no state-changing effect.

For the specific cases listed above, we recommend checking whether

```
_token.length == _gateway.length && token.length > 0 .
```

5.17 L2GatewayRouter - L1-to-L2 interface inconsistency Minor

Description

The `L2GatewayRouter.setGateway` function accepts multiple gateway addresses, while the equivalent `L1GatewayRouter.setGateway` function only accepts a single address.

Examples



code/packages/arb-bridge-peripherals/contracts/tokenbridge/arbitrum/gateway/L2GatewayRouter.sol:L43-L55

```
function setGateway(address[] memory _l1Token, address[] memory _gateway)
    external
    onlyCounterpartGateway
{
    // counterpart gateway (L1 router) should never allow wrong lengths
    assert(_l1Token.length == _gateway.length);

    for (uint256 i = 0; i < _l1Token.length; i++) {
        l1TokenToGateway[_l1Token[i]] = _gateway[i];
        emit GatewaySet(_l1Token[i], _gateway[i]);
    }
}
```

code/packages/arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1GatewayRouter.sol:L153-L223

```
*/
function setGateway(
    address _gateway,
    uint256 _maxGas,
    uint256 _gasPriceBid,
    uint256 _maxSubmissionCost
) external payable returns (uint256) {
    return setGateway(_gateway, _maxGas, _gasPriceBid, _maxSubmissionCost, msg.
}

/**
 * @notice Allows L1 Token contract to trustlessly register its gateway.
 * param _gateway l1 gateway address
 * param _maxGas max gas for L2 retryable execution
 * param _gasPriceBid gas price for L2 retryable ticket
 * param _maxSubmissionCost base submission cost L2 retryable ticket
 * param _creditBackAddress address for crediting back overpayment of _maxSubmiss
 * return Retryable ticket ID
 */
function setGateway(
    address, /* _gateway */
```



```

    uint256, /* _maxGas */
    uint256, /* _gasPriceBid */
    uint256, /* _maxSubmissionCost */
    address /* _creditBackAddress */
) public payable returns (uint256) {
    revert("SELF_REGISTRATION_DISABLED");
    /*
    require(
        ArbitrumEnabledToken(msg.sender).isArbitrumEnabled() == uint8(0xa4b1),
        "NOT_ARB_ENABLED"
    );
    require(!_gateway.isContract(), "NOT_TO_CONTRACT");

    address currGateway = l1TokenToGateway[msg.sender];
    if (currGateway != address(0)) {
        // if gateway is already set, don't allow it to set a different gateway
        require(currGateway == _gateway, "NO_UPDATE_TO_DIFFERENT_ADDR");
    }

    address[] memory _tokenArr = new address[](1);
    _tokenArr[0] = address(msg.sender);

    address[] memory _gatewayArr = new address[](1);
    _gatewayArr[0] = _gateway;

    return
        _setGateways(
            _tokenArr,
            _gatewayArr,
            _maxGas,
            _gasPriceBid,
            _maxSubmissionCost,
            _creditBackAddress
        );
    */
}

function setGateways(
    address[] memory _token,
    address[] memory _gateway,
    uint256 _maxGas,
    uint256 _gasPriceBid,
    uint256 _maxSubmissionCost
) external payable onlyOwner returns (uint256) {
    // it is assumed that token and gateway are both contracts
    // require(_token[i].isContract() && _gateway[i].isContract(), "NOT_CONTRACT
    return

```



```

return
    _setGateways(_token, _gateway, _maxGas, _gasPriceBid, _maxSubmissionCos
}

```

Recommendation

We recommend providing a consistent L1-to-L2 interface to avoid confusion. Additionally, `L2GatewayRouter.setGateway` should be renamed to `setGateways`.

5.18 L2GatewayRouter/L2ArbitrumGateway - onlyCounterpartGateway() should only allow L2 aliased addresses Minor

Description

According to the regular user-data-flow gateways communicate with their counterparts through the bridge. However, the `onlyCounterpartGateway()` access modifier allows both the L1 and the L2 counterpart address. As the transition into the address-mapped L1-L2 integration has been completed, the L1 address check has become redundant and might compromise the system's security properties.

Examples

code/packages/arb-bridge-peripherals/contracts/tokenbridge/arbitrum/gateway/L2GatewayRouter.sol:L31-L39

```

modifier onlyCounterpartGateway() override {
    require(
        msg.sender == counterpartGateway ||
        AddressAliasHelper.undoL1ToL2Alias(msg.sender) == counterpartGateway
        "ONLY_COUNTERPART_GATEWAY"
    );
    _;
}

```



code/packages/arb-bridge-peripherals/contracts/tokenbridge/arbitrum/gateway/L2ArbitrumGateway.sol:L56-L63

```

modifier onlyCounterpartGateway() override {
    require(
        msg.sender == counterpartGateway ||
        AddressAliasHelper.undoL1ToL2Alias(msg.sender) == counterpartGateway
        "ONLY_COUNTERPART_GATEWAY"
    );
    -;
}

```

Recommendation

Remove `msg.sender == counterpartGateway` in favor of the L2 aliased address.

5.19 ERC20Gateway / BeaconProxyFactory - is not a proxy or `initialize` may be unnecessary (pot. front-run) Minor

Description

`BeaconProxyFactory` has an `initialize` function that takes an `erc20Beacon` address as an initialization parameter. That `erc20Beacon` contract (ERC20 token implementation on L2) is usually created before the `BeaconProxyFactory` is deployed. Hence, the `initialize` function may be unnecessary and can be switched out for a simple `constructor` initializing the parameter.

code/packages/arb-bridge-peripherals/contracts/tokenbridge/libraries/ClonableBeaconProxy.sol:L27-L31

```

function initialize(address _beacon) external {
    require(_beacon != address(0), "INVALID_BEACON");
    require(beacon == address(0), "ALREADY_INIT");
    beacon = _beacon;
}

```



code/packages/arb-bridge-peripherals/scripts/deploy_token_bridge_l1.ts:L94-L102

```
const erc20Beacon = await UpgradeableBeacon.deploy(standardArbERC20.address)
await erc20Beacon.deployed()
console.log(`erc20 beacon at ${erc20Beacon.address}`)

const BeaconProxyFactory = (
  await ethers.getContractFactory('BeaconProxyFactory')
).connect(l2Signer)
const beaconProxyFactory = await BeaconProxyFactory.deploy()
await beaconProxyFactory.deployed()
```

Since the create2 factory contract does not appear to be a proxy at all (<https://arbiscan.io/bytecode-decompiler?a=0x3fe38087a94903a9d946fa1915e1772fe611000f>), developers cannot switch it out without invalidating all L2 token addresses.

Recommendation

We recommend checking whether the `BeaconProxyFactory` was meant to be a proxy instead of a concrete deployment, as hinted by the `initialize()` function. If that's not the case, consider removing the `initialize` function in favor of a constructor call setting initial parameters instead of preventing other actions from front-run the `BeaconProxyFactory` deployment/initialization procedure.

This issue is connected to [issue 5.11](#).

5.20 aeWETH - Deviations from WETH9 interface and standard Minor

Description

L2 `aeWETH` to L1 `IWETH9` token interface might be unexpected for consumers expecting `IWETH9` functionality and events.

Examples



- `aeWETH` emits `Transfer` events for `withdraw/deposit` events while `WETH9` emits `Withdrawal` and `Deposit`

src/weth9.sol:L25-L26

```
event Deposit(address indexed dst, uint wad);
event Withdrawal(address indexed src, uint wad);
```

- non-standard interfaces `depositTo` and `withdrawTo` allow to deposit to and withdraw to arbitrary accounts.

code/packages/arb-bridge-peripherals/contracts/tokenbridge/libraries/aeWETH.sol:L58-L67

```
function depositTo(address account) public payable {
    _mint(account, msg.value);
}

function withdrawTo(address account, uint256 amount) public {
    _burn(msg.sender, amount);
    (bool success, ) = account.call{ value: amount }("");
    require(success, "FAIL_TRANSFER");
}
```

Recommendation

Keep the L2 WETH interfaces and events as close as possible to the original L1 WETH contract. Consider restricting `withdrawTo` and `depositTo` internal if they are not meant to be called from an external source.

5.21 L1ArbitrumGateway - Unreachable Code Minor

Description

`require(isRouter(msg.sender), "NOT_FROM_ROUTER");` enforces that `msg.sender` is a router, therefore, the else branch in the following conditional statement is never taken.



Note: the two function calls `isRouter` and `super.isRouter` are equivalent.

Examples

code/packages/arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1ArbitrumGateway.sol:L173-L206

```

/**
 * @notice Deposit ERC20 token from Ethereum into Arbitrum. If L2 side hasn't been
 * @param _l1Token L1 address of ERC20
 * @param _to account to be credited with the tokens in the L2 (can be the user's
 * @param _amount Token Amount
 * @param _maxGas Max gas deducted from user's L2 balance to cover L2 execution
 * @param _gasPriceBid Gas price for L2 execution
 * @param _data encoded data from router and user
 * @return res abi encoded inbox sequence number
 */
// * @param maxSubmissionCost Max gas deducted from user's L2 balance to cover L2
function outboundTransfer(
    address _l1Token,
    address _to,
    uint256 _amount,
    uint256 _maxGas,
    uint256 _gasPriceBid,
    bytes calldata _data
) public payable virtual override returns (bytes memory res) {
    require(isRouter(msg.sender), "NOT_FROM_ROUTER");
    // This function is set as public and virtual so that subclasses can override
    // it and add custom validation for callers (ie only whitelisted users)
    address _from;
    uint256 seqNum;
    bytes memory extraData;
    {
        uint256 _maxSubmissionCost;
        if (super.isRouter(msg.sender)) {
            // router encoded
            (_from, extraData) = GatewayMessageHandler.parseFromRouterToGateway
        } else {
            _from = msg.sender;
            extraData = _data;
        }
    }
}

```



commendation

The relevant code is probably from an earlier iteration and should be removed.

5.22 Uninitialized proxy templates/implementations Minor

Description

Proxy templates/implementations are not always automatically initialized, or ACL controlled. Other participants may be able to claim the contracts even though they are not officially being used directly to mislead others which can cause reputational damage.

Examples

The delayed inbox proxy at `0x4Dbd4fc535Ac27206064B68Ffc827b0A60BAB3f` is initialized, however the implementation contract at `0x048cc108763de75E080Ad717bD284003aa49eA15` is not (`bridge == address(0)`).

Similarly, the L1ERC20Gateway proxy at `0xa3A7B6F88361F48403514059F1F16C8E78d60EeC` is initialized, but its implementation at `0xd710c475216999184DB1737aAd197fc855255AD7` is not (`counterpartGateway == address(0)`).

Recommendation

Add a constructor to relevant contracts that automatically initializes the implementations on deployment. We recommend reviewing all production deployments to ensure the problem does not affect any more components.

5.23 Use the precise interface types instead of `address` where possible Minor

Description

For clarity and to get more out of the Solidity type checker, it's generally preferred to use a specific contract interface type for variables rather than the generic `address`. Consider declaring function arguments with the most specific type available instead of `address`; only downcast to a "less safe" `address` when required.



Examples

code/packages/arb-bridge-peripherals/contracts/tokenbridge/arbitrum/gateway/L2ERC20Gateway.sol:L28-L37

```

address public beaconProxyFactory;

function initialize(
    address _l1Counterpart,
    address _router,
    address _beaconProxyFactory
) public {
    L2ArbitrumGateway._initialize(_l1Counterpart, _router);
    require(_beaconProxyFactory != address(0), "INVALID_BEACON");
    beaconProxyFactory = _beaconProxyFactory;
}

```

code/packages/arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1ArbitrumGateway.sol:L234-L245

```

function outboundEscrowTransfer(
    address _l1Token,
    address _from,
    uint256 _amount
) internal virtual returns (uint256 amountReceived) {
    // this method is virtual since different subclasses can handle escrow differently
    // user funds are escrowed on the gateway using this function
    uint256 prevBalance = IERC20(_l1Token).balanceOf(address(this));
    IERC20(_l1Token).safeTransferFrom(_from, address(this), _amount);
    uint256 postBalance = IERC20(_l1Token).balanceOf(address(this));
    return SafeMath.sub(postBalance, prevBalance);
}

```

code/packages/arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1GatewayRouter.sol:L99-L99

```

address[] memory _gateway,

```



code/packages/arb-bridge-peripherals/contracts/tokenbridge/ethereum/L1ArbitrumMessenger.sol:L83-L86

```
function getBridge(address _inbox) internal view virtual returns (IBridge) {
    return IInbox(_inbox).bridge();
}
```

- double cast of `template` and `template` is `address` instead of best type `Validator` :

code/packages/arb-bridge-eth/contracts/validator/ValidatorWalletCreator.sol:L46-L50

```
function createWallet() external returns (address) {
    ProxyAdmin admin = new ProxyAdmin();
    address proxy = address(
        new TransparentUpgradeableProxy(address(template), address(admin), "")
    );
}
```

Recommendation

Where possible, use a specific contract type instead of an `address` .

5.24 Clean up unused imports Minor

Description

In various places, source units have been imported but not referenced in the respective smart contract.

Examples

code/packages/arb-bridge-eth/contracts/bridge/Bridge.sol:L21-L22

```
import "./Inbox.sol";
import "./Outbox.sol";
```



code/packages/arb-bridge-eth/contracts/bridge/Inbox.sol:L25-L25

```
import "../Messages.sol";
```

code/packages/arb-bridge-eth/contracts/bridge/Outbox.sol:L29-L30

```
import "@openzeppelin/contracts/proxy/BeaconProxy.sol";  
import "@openzeppelin/contracts/proxy/UpgradeableBeacon.sol";
```

code/packages/arb-bridge-eth/contracts/bridge/SequencerInbox.sol:L25-L25

```
import "../rollup/Rollup.sol";
```

code/packages/arb-bridge-peripherals/contracts/tokenbridge/arbitrum/gateway/L2ERC20Gateway.sol:L22-L22

```
import "@openzeppelin/contracts/utils/Create2.sol";
```

code/packages/arb-bridge-eth/contracts/arch/IOneStepProof.sol:L21-L21

```
import "../bridge/interfaces/IBridge.sol";
```

code/packages/arb-bridge-eth/contracts/arch/OneStepProof.sol:L21

```
import "../IOneStepProof.sol";
```

code/packages/arb-bridge-eth/contracts/arch/OneStepProof2.sol:L47-L47

```
import "../IOneStepProof.sol";
```

**code/packages/arb-bridge-eth/contracts/arch/OneStepProof2.sol:L50-L50**

```
import "./Machine.sol";
```

`IERC20` import should be in `RollupUser` :

code/packages/arb-bridge-eth/contracts/rollup/Rollup.sol:L24-L24

```
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
```

Recommendation

We recommend checking all imports and removing all unused and unnecessary imports.

5.25 GasRefunder - gas refund can be maxed-out by zero-padding calldata

Resolution

According to the client, this method is configured to only be callable by their `Sequencer Node (isSequencer && isGasRefunder)`. The main reason to refund gas this way is to diversify risk by not keeping all the funds in the sequencer hot-wallet instead of keeping part of the funds in the gas refunded. The Sequencer initially funds the gas refunded.

Description

`SequencerInbox.addSequencerL2BatchFromOriginWithGasRefunder()` refunds the gas that was spent on calling the function. The refund is calculated as (1) the gas consumed on the execution of the instructions (function bodies), (2) the amount of call data provided, and (3) adding an extra gas margin. The maximum amount of gas refunded (initially) is `2e6 * 120 gwei ^= 240000000000000000 wei ^= 0.24 ETH`, that's around ~ 1000 USD at the time of this report.



code/packages/arb-bridge-eth/contracts/validator/GasRefunder.sol:L81-L89

```
commonParams = CommonParameters({
  maxRefundeeBalance: 0, // no limit
  extraGasMargin: 4000, // 4k gas
  calldataCost: 12, // Between 4 for zero bytes and 16 for non-zero bytes
  maxGasTip: 2 gwei,
  maxGasCost: 120 gwei,
  maxSingleGasUsage: 2e6 // 2 million gas
});
}
```

Calldata is refunded with 12 gas, even though zero-bytes might only consume four gas. Consequently, appended zero-bytes are refunded 3x the caller's actual price on providing them with the transaction. Callers are therefore incentivized to zero-pad their calls to max out the gas refund. The contracts function dispatcher ignores the appended bytes. Hence, they provide no value to the contract call, but they are refunded with 3x the actual price the caller has to pay for providing them. In the worst case, depending on the current ethereum gas price (i.e., current gas price is ≤ 120), this would allow a caller whitelisted for gas refunds to get back up to 0.24 ETH every block while likely only spending a fraction on the actual transaction.

Note that the maximum amount of calldata provided with a transaction is limited by the block gas limit (at 10Mio ~ 625k bytes). Restrictions based on how much was refunded (checking the `refundee.balance`) can easily be bypassed. However, this method is authenticated and only allows configured refundees/sequencers to receive a refund (RollupAdminFacet).

Examples

`code/packages/arb-bridge-eth/contracts/bridge/SequencerInbox.sol:L183-L216`



```

function addSequencerL2BatchFromOriginWithGasRefunder(
    bytes calldata transactions,
    uint256[] calldata lengths,
    uint256[] calldata sectionsMetadata,
    bytes32 afterAcc,
    IGasRefunder gasRefunder
) external {
    // solhint-disable-next-line avoid-tx-origin
    require(msg.sender == tx.origin, "origin only");

    uint256 startGasLeft = gasleft();
    uint256 calldataSize;
    assembly {
        calldataSize := calldatasize()
    }

    uint256 startNum = messageCount;
    bytes32 beforeAcc = addSequencerL2BatchImpl(
        transactions,
        lengths,
        sectionsMetadata,
        afterAcc
    );
    emit SequencerBatchDeliveredFromOrigin(
        startNum,
        beforeAcc,
        messageCount,
        afterAcc,
        inboxAccs.length - 1
    );

    if (gasRefunder != IGasRefunder(0)) {
        gasRefunder.onGasSpent(msg.sender, startGasLeft - gasleft(), calldataSize);
    }
}

```

Recommendation

Avoid refunding based on the provided `calldatasize` as this can easily be manipulated by the caller.

6 Recommendations



6.1 Consider using a two-step ownership transfer

Description

To reduce the risk of accidentally losing access to the ownership/admin roles, we recommend implementing a two-step ownership transfer by first offering the role to a different account and having the grantee accept that role.

code/packages/arb-bridge-eth/contracts/libraries/Whitelist.sol:L56-L59

```
function setOwner(address newOwner) external onlyOwner {
    owner = newOwner;
    emit OwnerUpdated(newOwner);
}
```

code/packages/arb-bridge-eth/contracts/rollup/facets/RollupAdmin.sol:L97-L100

```
function setOwner(address newOwner) external override {
    owner = newOwner;
    emit OwnerFunctionCalled(7);
}
```

code/packages/arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1GatewayRouter.sol:L91-L95

```
function setOwner(address newOwner) external onlyOwner {
    require(newOwner != address(0), "INVALID_OWNER");
    // set newOwner to address(1) to disable owner and keep `initialize` safe
    owner = newOwner;
}
```

6.2 ERC20Gateway - Every bridge token transfer carries L2 token details



Description

Bridge token transfers via the `ERC20Gateway` always carry L2 token deployment details (name, symbol, decimals) in L1-to-L2 messages even though they are only used once on the L2 side (unless that L2 token is self-destructed).

Given that this information is encoded as

`abi.encode(bytes memory, bytes memory, bytes memory)`, this may add significant size to L1-to-L2 messages. Furthermore, the `decimals` on the L1 side are packed as `bytes memory` instead of `uint8` in L1-to-L2 messages (consuming more size).

code/packages/arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1ERC20Gateway.sol:L75-L98

```
function getOutboundCalldata(
    address _token,
    address _from,
    address _to,
    uint256 _amount,
    bytes memory _data
) public view override returns (bytes memory outboundCalldata) {
    // TODO: cheaper to make static calls or save isDeployed to storage?
    bytes memory deployData = abi.encode(
        callStatic(_token, ERC20.name.selector),
        callStatic(_token, ERC20.symbol.selector),
        callStatic(_token, ERC20.decimals.selector)
    );

    outboundCalldata = abi.encodeWithSelector(
        ITokenGateway.finalizeInboundTransfer.selector,
        _token,
        _from,
        _to,
        _amount,
        GatewayMessageHandler.encodeToL2GatewayMsg(deployData, _data)
    );

    return outboundCalldata;
}
```

Consider reducing message size (and therefore gas required) by finding a way to force a deployment/initialization on the first call, only removing the necessity



to provide deploy init data with every call (optimizing for the 99% of calls that do not need that data).

6.3 aeERC20 - Optimization: set `initialized=true` instead of calling the modifier

Description

Directly setting `initialized = true` is more efficient than calling the modifier `initialized` from the constructor, as this would cost at least two state variables and additional computation.

code/packages/arb-bridge-peripherals/contracts/tokenbridge/libraries/aeERC20.sol:L30-L33

```
constructor() public initializer {  
    // this is expected to be used as the logic contract behind a proxy  
    // override the constructor if you don't wish to use the initialize method  
}
```

6.4 Contract file system layout, source-unit structure, naming

Description

The project layout, source-unit structure, and naming conventions diverge from best practices in smart contract development.

Recommendations

We recommend the following actions to clean up the code base and make it more maintainable:

- Developers should keep contract interface declarations separately from contract implementations, e.g., in an `interfaces` subfolder.
- All interfaces and source units should adhere to a strict naming scheme



I<name> :

code/packages/arb-bridge-peripherals/contracts/rpc- utils/RetryableTicketCreator.sol:L5-L5

```
interface RetryableTicketCreator {
```

- If a contract/interface already exists, no sub-interfaces should be derived from it.
- Each source unit should only contain a single contract, e.g., `Rollup.sol`.
- Revise stale contract names that are derived from previous pattern usage. E.g., contracts are prefixed with `Clonable` even though it does not adhere to the `Clonable` pattern used in other places of the codebase.
- Instead of un-descriptive integers, Enums should be used:

code/packages/arb-bridge-eth/contracts/rollup/facets/RollupAdmin.sol:L26-L26

```
emit OwnerFunctionCalled(0);
```

code/packages/arb-bridge- peripherals/contracts/tokenbridge/libraries/ClonableBeaconProxy.sol:L8-L21

```
interface ProxySetter {
    function beacon() external view returns (address);
}

contract ClonableBeaconProxy is BeaconProxy {
    constructor() public BeaconProxy(ProxySetter(msg.sender).beacon(), "") {}
}

contract BeaconProxyFactory is ProxySetter {
    bytes32 public constant cloneableProxyHash = keccak256(type(ClonableBeaconProxy));

    /**
     * @notice utility function used in ClonableBeaconProxy.
     */
}
```



- Contracts from external sources should be moved to a separate directory, e.g. `TransferAndCallToken`.
- Consider grouping public interfaces and internal/private interfaces together. E.g., public interfaces first, private/internal afterward.
- The contract should define modifiers at the top. They should not mix with public/private functions:

code/packages/arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1GatewayRouter.sol:L247-L251

```
modifier onlyCounterpartGateway() override {  
    // don't expect messages from L2 router  
    revert("ONLY_COUNTERPART_GATEWAY");  
    -;  
}
```

6.5 Untangle the complex inheritance structure

Description

The system's core and peripheral contracts have a complex inheritance structure with various functions being overridden and called across the hierarchy. Consequentially, it becomes hard to trace the business logic flow through the system and anticipate side effects and potentially unwanted state changes.

Examples



```
StandardArbERC20
└─ Cloneable
└─ ICloneable
└─ L2GatewayToken
└─ IArbToken
└─ aeERC20
└─ TransferAndCallToken
└─ ITransferAndCall
└─ ERC20PermitUpgradeable
└─ EIP712Upgradeable
└─ IERC20PermitUpgradeable
└─ ERC20Upgradeable
└─ IERC20Upgradeable
└─ ContextUpgradeable
└─ Initializable
```

Recommendation

Consider revising the inheritance structure and separation of the system's smart contracts to reflect the use cases better while maintaining a clean separation of concerns. Not only will a simplified hierarchy reduce the potential for errors later on, but it will also increase maintainability and readability, which can help speed up the onboarding process for new developers.

6.6 Inconsistent proxy initialization naming scheme

Description

`StandardArbERC20.bridgeInit()` is the default `initialization` method for the proxy contract. Consider renaming it to `initialize()` to clearly communicate this fact.

Note that the implementation is protected from initializing deep down in `aeERC20.constructor` via the `initializer` modifier.

code/packages/arb-bridge-peripherals/contracts/tokenbridge/arbitrum/StandardArbERC20.sol:L36-L47



```
function bridgeInit(address _l1Address, bytes memory _data) public virtual {
    (bytes memory name_, bytes memory symbol_, bytes memory decimals_) = abi.decode(
        _data,
        (bytes, bytes, bytes)
    );
    // what if decode reverts? shouldn't as this is encoded by L1 contract

    L2GatewayToken._initialize(
        BytesParserWithDefault.toString(name_, ""),
        BytesParserWithDefault.toString(symbol_, ""),
        BytesParserWithDefault.toUint8(decimals_, 18),
        msg.sender, // _l2Gateway,
    );
}
```

Comparably, `aeWETH` initialization is named `initialize`:

code/packages/arb-bridge-peripherals/contracts/tokenbridge/libraries/aeWETH.sol:L25-L34

```
contract aeWETH is L2GatewayToken, IWETH9 {
    function initialize(
        string memory name_,
        string memory symbol_,
        uint8 decimals_,
        address l2Gateway_,
        address l1Address_
    ) external {
        L2GatewayToken._initialize(name_, symbol_, decimals_, l2Gateway_, l1Address_)
    }
}
```

6.7 TransferAndCallToken - emits two `Transfer()` events

Description

The L2 tokens implementing `TransferAndCallToken` are emitting two `Transfer()` events. First the `ERC20Upgradeable.Transfer(msgSender(), to, value)` and then the `TransferAndCallToken.Transfer(msg.sender, to, value, data)` event. No event is emitted to indicate that the external call has actually been executed.



code/packages/arb-bridge-peripherals/contracts/tokenbridge/libraries/TransferAndCallToken.sol:L19-L30

```
function transferAndCall(  
    address _to,  
    uint256 _value,  
    bytes memory _data  
) public virtual override returns (bool success) {  
    super.transfer(_to, _value);  
    emit Transfer(msg.sender, _to, _value, _data);  
    if (isContract(_to)) {  
        contractFallback(_to, _value, _data);  
    }  
    return true;  
}
```

Recommendation

The TransferAndCallToken standard draft seems to be still under discussion or abandoned. If there are remaining efforts to turn the draft into a standard, we recommend renaming the event to distinguish it easily from the `ERC20.Transfer` event.

<https://github.com/ethereum/EIPs/issues/677>

6.8 Unnecessary reset of `callHookData`

Description

`callHookData` is overridden with `bytes("")` even though it is not used as the call to `getExternalCall` discards the returned `data` value unconditionally.

code/packages/arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1ArbitrumGateway.sol:L112-L120



```

if (callHookData.length != 0) {
    // callHookData should always be 0 since inboundEscrowAndCall is disabled
    callHookData = bytes("");
}

// we ignore the returned data since the callHook feature is now disabled
(_to, ) = getExternalCall(exitNum, _to, callHookData);
inboundEscrowTransfer(_token, _to, _amount);

```

The silent reset of the `callHookData` value has the potential to shadow misconfiguration and potential system errors.

6.9 L1ArbitrumGateway - outboundTransfer Consider checking `_from != address(0x0)`

Description

On the L1 gateway, `_from` addresses are allowed to be zero.

code/packages/arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1ArbitrumGateway.sol:L197-L209

```

bytes memory extraData;
{
    uint256 _maxSubmissionCost;
    if (super.isRouter(msg.sender)) {
        // router encoded
        (_from, extraData) = GatewayMessageHandler.parseFromRouterToGateway(_data);
    } else {
        _from = msg.sender;
        extraData = _data;
    }
    // user encoded
    (_maxSubmissionCost, extraData) = abi.decode(extraData, (uint256, bytes));
    // the inboundEscrowAndCall functionality has been disabled, so no data is a

```



commendation

Consider adding a safety check to ensure that the decoded `_from` address is not `address(0x0)`, which should never happen.

6.10 GatewayRouter - Revert early if gateway is disabled

Description

`GatewayRouter.getGateway` may return `0x0` if the token gateway is disabled. In this case, the method will continue until the implicit `isContract` check by the solidity type system at `ITokenGateway(gateway).outboundTransfer()` fails, wasting gas encoding the gateway message.

Examples

[code/packages/arb-bridge-peripherals/contracts/tokenbridge/libraries/gateway/GatewayRouter.sol:L78-L102](#)




```
function outboundTransfer(  
    address _token,  
    address _to,  
    uint256 _amount,  
    uint256 _maxGas,  
    uint256 _gasPriceBid,  
    bytes calldata _data  
) public payable virtual override returns (bytes memory) {  
    address gateway = getGateway(_token);  
    bytes memory gatewayData = GatewayMessageHandler.encodeFromRouterToGateway(  
        msg.sender,  
        _data  
    );  
  
    emit TransferRouted(_token, msg.sender, _to, gateway);  
    return  
        ITokenGateway(gateway).outboundTransfer{ value: msg.value }(  
            _token,  
            _to,  
            _amount,  
            _maxGas,  
            _gasPriceBid,  
            gatewayData  
        );  
}
```

**code/packages/arb-bridge-
peripherals/contracts/tokenbridge/libraries/gateway/GatewayRouter.sol:L115
-L129**



```

function getGateway(address _token) public view virtual returns (address gateway) {
    gateway = l1TokenToGateway[_token];

    if (gateway == ZERO_ADDR) {
        // if no gateway value set, use default gateway
        gateway = defaultGateway;
    }

    if (gateway == DISABLED || !gateway.isContract()) {
        // not a valid gateway
        return ZERO_ADDR;
    }

    return gateway;
}

```

Recommendation

Consider explicitly requiring that `gateway != ZERO_ADDR` in `outboundTransfer`.

6.11 Informational: L1-L2 address aliasing may break with solidity >= 0.8.x

Description

Due to implicit overflow checks, the L1-L2 and L2-L1 address aliasing will break when upgrading to solidity 0.8.x or later.

code/packages/arb-bridge-eth/contracts/libraries/AddressAliasHelper.sol:L28-L30

```

function applyL1ToL2Alias(address l1Address) internal pure returns (address l2Address) {
    l2Address = address(uint160(l1Address) + offset);
}

```

6.12 Missing NatSpec

Description



Some external functions don't have NatSpec annotations. To quote the [Solidity documentation](#): "It is recommended that Solidity contracts are fully annotated using NatSpec for all public interfaces (everything in the ABI)."

Recommendation

We strongly recommend adding NatSpec annotations to at least every contract and every public or external function. Furthermore, critical internal and private functions should be documented with NatSpec to increase maintainability and reduce the potential for future developer errors. NatSpec documentation should primarily be enforced in the sensitive and complex components, such as proof- and challenge-related smart contracts.

6.13 Separation of concerns in the Challenge.timeout function

Description

The `Challenge.timeout` function currently serves two use cases. On the one hand, it is used to handle the case of a non-responsive counterparty; on the other hand, it is used to determine the winner of a challenge. The winner is selected by setting the contract's `challengeState` to zero, which effectively makes it impossible for the counterparty to respond. Consequentially, the challenge will time out, and a subsequent call to `timeout` will determine the winner.

In the latter case, the contract will emit the respective timeout events (`AsserterTimedOut` or `ChallengerTimedOut`). These events can be misleading in the case of a challenge that has determined a clear protocol violation.

Examples

[code/packages/arb-bridge-eth/contracts/challenge/Challenge.sol:L330-L341](#)



```
function timeout() external override {
    uint256 timeSinceLastMove = block.number.sub(lastMoveBlock);
    require(timeSinceLastMove > currentResponderTimeLeft(), TIMEOUT_DEADLINE);

    if (turn == Turn.Asserter) {
        emit AsserterTimedOut();
        _challengerWin();
    } else {
        emit ChallengerTimedOut();
        _asserterWin();
    }
}
```

Recommendation

We recommend separating the concerns served by the `timeout` functions to clarify whether an actual timeout has occurred or the system has determined a winner. Separate functions will future-proof the code and reduce the potential for developer errors as currently changes to one concern might affect the other and result in unintended side effects.

6.14 Avoid unnecessary intermediary casts

Description

The intermediary cast to `bytes20()` can be omitted.

```
uint256(uint160(address(0x03))) == uint256(uint160(bytes20(address(0x03))))
true
```

Examples

code/packages/arb-bridge-eth/contracts/bridge/Outbox.sol:L253-L254

```
uint256(uint160(bytes20(l2Sender))),
uint256(uint160(bytes20(destAddr))),
```



code/packages/arb-bridge-

eth/contracts/bridge/Old_Outbox/OldOutbox.sol:L236-L237

```
uint256(uint160(bytes20(l2Sender))),  
uint256(uint160(bytes20(destAddr))),
```

code/packages/arb-bridge-eth/contracts/rollup/RollupEventBridge.sol:L64-L64

```
uint256(uint160(bytes20(owner))),
```

code/packages/arb-bridge-eth/contracts/rollup/RollupEventBridge.sol:L88-L88

```
uint256(uint160(bytes20(assertor)))
```

code/packages/arb-bridge-eth/contracts/rollup/RollupEventBridge.sol:L105-L105

```
uint256(uint160(bytes20(staker))),
```

code/packages/arb-bridge-eth/contracts/bridge/Inbox.sol:L301-L302

```
uint256(uint160(bytes20(excessFeeRefundAddress))),  
uint256(uint160(bytes20(callValueRefundAddress))),
```

There might be more occasions of this and similar casting issues.

6.15 Inbox - Input validation

Description

If `Inbox.initialize()` is called with `_bridge = address(0x0)` the contract is effectively left uninitialized and it can be claimed by anyone. The lack of validation during initialization becomes a problem if the `BridgeCreator` is not set up in the `Inbox`. If the function is fed erroneous inputs, then the contract may be left initializable



by anyone. The same issue occurs if the contract is deployed but not initialized in the same transaction.

code/packages/arb-bridge-eth/contracts/bridge/Inbox.sol:L48-L52

```
function initialize(IBridge _bridge, address _whitelist) external {
    require(address(bridge) == address(0), "ALREADY_INIT");
    bridge = _bridge;
    WhitelistConsumer.whitelist = _whitelist;
}
```

Another example:

code/packages/arb-bridge-eth/contracts/bridge/SequencerInbox.sol:L55-L65

```
function initialize(
    IBridge _delayedInbox,
    address _sequencer,
    address _rollup
) external {
    require(address(delayedInbox) == address(0), "ALREADY_INIT");
    delayedInbox = _delayedInbox;
    isSequencer[_sequencer] = true;
    rollup = _rollup;
    // it is assumed that maxDelayBlocks and maxDelaySeconds are set by the rollup
}
```

Recommendation

We recommend checking for misconfiguration, especially when it might be left unnoticed in the worst case. Check for valid inputs and revert otherwise.

6.16 Wrap MerkleLib to avoid risk of incorrect usage

Description

`MerkleLib` is a low-level component used for the generation of Merkle tree roots and the verification of Merkle proofs. It is used, for example, in `Outbox` and `ChallengeLib`. However, correct library usage requires knowledge of its subtleties,

and currently, outside business logic has to perform necessary actions and checks to obtain accurate results.

Examples

For example, an extra round of leaf hashing can be necessary to make it impossible to prove non-leaves. That is implemented in `Outbox` :

`code/packages/arb-bridge-eth/contracts/bridge/Outbox.sol:L204-L205`

```
// Hash the leaf an extra time to prove it's a leaf
bytes32 calcRoot = calculateMerkleRoot(proof, path, item);
```

`code/packages/arb-bridge-eth/contracts/bridge/Outbox.sol:L264-L270`

```
function calculateMerkleRoot(
    bytes32[] memory proof,
    uint256 path,
    bytes32 item
) public pure returns (bytes32) {
    return MerkleLib.calculateRoot(proof, path, keccak256(abi.encodePacked(item)))
}
```

Similarly, intricacies that prevent spending the same output repeatedly, as in the following lines, have to be taken care of in business logic – where they distract and can easily be forgotten:

`code/packages/arb-bridge-eth/contracts/bridge/Outbox.sol:L201-L202`

```
require(proof.length < 256, "PROOF_TOO_LONG");
require(path < 2**proof.length, "PATH_NOT_MINIMAL");
```

Recommendation

We recommend adding a lightweight wrapper, e.g., `internal` methods or small utility methods in the library itself, to take care of these details and ensure



correct usage. Business logic code shouldn't (have to) be concerned with low-level details of the Merkle tree implementation.

6.17 Simplify RollupUser.removeOldZombies

Description

Stakers that have lost a challenge become *Zombies*. The list of zombies is stored in an array where new ones are appended to the end. When a particular zombie (given via its array index `i`) is removed from the list, the `RollupCore.removeZombie` function takes care of this by copying the last element to position `i` and then deleting the last element from the array.

The `RollupUser.removeOldZombies` function is implemented as follows:

code/packages/arb-bridge-eth/contracts/rollup/facets/RollupUser.sol:L409-L425

```
/**
 * @notice Remove any zombies whose latest stake is earlier than the first unresol
 * @param startIndex Index in the zombie list to start removing zombies from (to
 */
function removeOldZombies(uint256 startIndex) public onlyValidator whenNotPaused {
    uint256 currentZombieCount = zombieCount();
    uint256 firstUnresolved = firstUnresolvedNode();
    for (uint256 i = startIndex; i < currentZombieCount; i++) {
        while (zombieLatestStakedNode(i) < firstUnresolved) {
            removeZombie(i);
            currentZombieCount--;
            if (i >= currentZombieCount) {
                return;
            }
        }
    }
}
```

The inner loop (`while`) is necessary because of the mechanism described above: The `i`-th element in the array gets replaced with a different one, so the same index must be rechecked.

Recommendation



The code can be simplified and made more elegant by running back through the array. An inner loop is not necessary in this case.

6.18 No support for staking tokens with missing-return-value bug

Description and Recommendation

Currently, ERC-20 tokens that exhibit the missing-return-value bug can't be utilized as staking tokens. Even if there are no immediate plans to use such a token for staking, it might be advisable to employ OpenZeppelin's `SafeERC20` library regardless. It is reasonably lightweight, and using it means there is one less thing to worry about in selecting staking tokens.

Appendix 1 - Files in Scope

This audit covered the following files:

File	SHA-1 hash
<code>./arb-bridge-eth/contracts/bridge/Bridge.sol</code>	93e2393faf941062adca2720 552189ea7b0b776d
<code>./arb-bridge-eth/contracts/bridge/BridgeUtils.sol</code>	e2965d0678a1aecdf4e57232 8cdd29220512a5ff
<code>./arb-bridge-eth/contracts/bridge/Inbox.sol</code>	b65c9d3c4fa775315df2429a bfcde02cb3a159a9
<code>./arb-bridge-eth/contracts/bridge/interfaces/IBridge.sol</code>	5157793467d1e99559538001 4cd951f9c659e1fd
<code>./arb-bridge-eth/contracts/bridge/interfaces/IInbox.sol</code>	5c027c1de6c0243872bc2f50 0661d5361b30955a
<code>./arb-bridge-eth/contracts/bridge/interfaces/IMessageProvider.sol</code>	a0a1cb67a6cde81872696782 ca44b295ec338cf7
<code>./arb-bridge-eth/contracts/bridge/interfaces/IOutbox.sol</code>	b7951b6cc18a0ede82e9cbf2 823d831c4825be8f
<code>./arb-bridge-eth/contracts/bridge/interfaces/ISequencerInbox.sol</code>	086e40fc050162385e984412 2e2becce951710a7



File	SHA-1 hash
<code>./arb-bridge-eth/contracts/bridge/Messages.sol</code>	<code>b8b738d6eb666463061190c40e2a4c48387b1de3</code>
<code>./arb-bridge-eth/contracts/bridge/Old_Outbox/OldOutbox.sol</code>	<code>c84d1d55a4de52f9810b96a2092059b7f7412db6</code>
<code>./arb-bridge-eth/contracts/bridge/Old_Outbox/OutboxEntry.sol</code>	<code>defa40eefdbcaf820e5f3b8956c65fe7d6114bde</code>
<code>./arb-bridge-eth/contracts/bridge/Outbox.sol</code>	<code>4426e477bc98e5e601c45ed1a0e2a33d42465250</code>
<code>./arb-bridge-eth/contracts/bridge/SequencerInbox.sol</code>	<code>89389a7351677a991b2810c1b28a7f4454628337</code>
<code>./arb-bridge-eth/contracts/challenge/Challenge.sol</code>	<code>9ff8d75c7ae098760da1ff22166fbbc83d37a027</code>
<code>./arb-bridge-eth/contracts/challenge/ChallengeFactory.sol</code>	<code>4c9e42698bdde13b492fef542896c32627384e8e</code>
<code>./arb-bridge-eth/contracts/challenge/ChallengeLib.sol</code>	<code>3eb4e47656541dbd645991212c5e68a3bdc3b090</code>
<code>./arb-bridge-eth/contracts/challenge/IChallenge.sol</code>	<code>162ab8957a2669cfa5960fb7123b441e6fce4695</code>
<code>./arb-bridge-eth/contracts/challenge/IChallengeFactory.sol</code>	<code>8dff43bf6729e7998538b72555de9470c41b6fe0</code>
<code>./arb-bridge-eth/contracts/interfaces/IERC20.sol</code>	<code>b8c66b824249a9b96a99a09e6f8d710024bfb0c5</code>
<code>./arb-bridge-eth/contracts/interfaces/IERC721.sol</code>	<code>b681dfb98e9be01e13db73b2474ec3441b3c8d0c</code>
<code>./arb-bridge-eth/contracts/libraries/AddressAliasHelper.sol</code>	<code>bfba5264dc28571e1410bd96e277a4c161a4cf87</code>
<code>./arb-bridge-eth/contracts/libraries/BytesLib.sol</code>	<code>7accb367a0ba0685684e680a4468016e56c7fa7f</code>
<code>./arb-bridge-eth/contracts/libraries/Cloneable.sol</code>	<code>184733ee25e2015bdbf18e9d9bd691feff2540c6</code>
<code>./arb-bridge-eth/contracts/libraries/DebugPrint.sol</code>	<code>f278c15184cf0734a871e182033076109c52148a</code>
<code>./arb-bridge-eth/contracts/libraries/ICloneable.sol</code>	<code>164fa954ca37c543cb9767a5ec978a721d6979d2</code>



File	SHA-1 hash
<code>./arb-bridge-eth/contracts/libraries/MerkleLib.sol</code>	5283d977e4ce646ae59ff53a95bce5eab0092efb
<code>./arb-bridge-eth/contracts/libraries/Precompiles.sol</code>	b6542c905b4ba1ec64146b0b44e0d9f7006dc019
<code>./arb-bridge-eth/contracts/libraries/ProxyUtil.sol</code>	784a36a01ddb0b6dc0f529eea3c2364c638e4fe
<code>./arb-bridge-eth/contracts/libraries/Whitelist.sol</code>	3d24c1e7c6a94d7cac21421874dfaf36db337ed3
<code>./arb-bridge-eth/contracts/rollup/BridgeCreator.sol</code>	44564170e2ebd3ce3a6f6df2ee4523bf1fa508cb
<code>./arb-bridge-eth/contracts/rollup/facets/IRollupFacets.sol</code>	4d6cebbcb4756f183f78b772316a6bee3552e3be
<code>./arb-bridge-eth/contracts/rollup/facets/RollupAdmin.sol</code>	60c7d222e39de09a8dc1a0fdca08df2e53d4e69c
<code>./arb-bridge-eth/contracts/rollup/facets/RollupUser.sol</code>	45cd2c21314e481a9e4d491c8a5d1bf21cd5e503
<code>./arb-bridge-eth/contracts/rollup/INode.sol</code>	4111917ba6d2a1b0d62391b6748042e894a0c2d2
<code>./arb-bridge-eth/contracts/rollup/INodeFactory.sol</code>	e91b8401316cb0cafa82b8b2389d7f57733a8b8c
<code>./arb-bridge-eth/contracts/rollup/IRollupCore.sol</code>	3550563fd6467f64e3c5c4b0a1084b5c9dd86986
<code>./arb-bridge-eth/contracts/rollup/Node.sol</code>	890c9b1120c8aaff6a357e9ef125bef0a0ac7636
<code>./arb-bridge-eth/contracts/rollup/NodeFactory.sol</code>	0b887736130286f19935f4df6c9170f6bc71421e
<code>./arb-bridge-eth/contracts/rollup/Rollup.sol</code>	c7c2f1fc48764799ffaddee62c1ce33fe82171ff
<code>./arb-bridge-eth/contracts/rollup/RollupCore.sol</code>	41d03f30f2fd0d833d0a47fcce3cb8f05b2b1
<code>./arb-bridge-eth/contracts/rollup/RollupCreator.sol</code>	8590e6b9e85c140ac62640c536a814beabc01fe0
<code>./arb-bridge-eth/contracts/rollup/RollupEventBridge.sol</code>	d952ccbdd586c4f82c1827ffef695bf1c3f28a95



File	SHA-1 hash
<code>./arb-bridge-eth/contracts/rollup/RollupLib.sol</code>	2ed0d83e1abe3ac6cee043d0 b7e5a8657ce37c79
<code>./arb-bridge-eth/contracts/validator/GasRefunder.sol</code>	7944e4371f32fc2db3928f29 66dbbfa43f4e06d6
<code>./arb-bridge-eth/contracts/validator/IGasRefunder.sol</code>	a571f9d4dc8cf0619aa5cd08 58921b49b0f074ab
<code>./arb-bridge-eth/contracts/validator/Validator.sol</code>	e651af45af3c026f55e176aa 5b4efd3767200366
<code>./arb-bridge-eth/contracts/validator/ValidatorUtils.sol</code>	c42acb3c032b28c56d16e37d c99d950459706ce0
<code>./arb-bridge-eth/contracts/validator/ValidatorWalletCreator.sol</code>	88e1a057d19b29da6a2896d3 8286e595b3b286f1
<code>./arb-bridge-peripherals/contracts/rpc- utils/NodeInterface.sol</code>	d96c394907ccfc429b821622 311a764e0c686dcf
<code>./arb-bridge-peripherals/contracts/rpc- utils/RetryableTicketCreator.sol</code>	748f56e1ab2bb151bd189f97 6b0921dade091095
<code>./arb-bridge-peripherals/contracts/tokenbridge/arbitrum/gateway/L2ArbitrumGateway.sol</code>	bf75a595767cb8a9965d835c 264881acd7763bed
<code>./arb-bridge-peripherals/contracts/tokenbridge/arbitrum/gateway/L2CustomGateway.sol</code>	b2fd8683ffdfb14a9ba39fe9 be350642025cf1b2
<code>./arb-bridge-peripherals/contracts/tokenbridge/arbitrum/gateway/L2ERC20Gateway.sol</code>	711d7005ce3584ab2bcd1434 027d5c710a2f2017
<code>./arb-bridge-peripherals/contracts/tokenbridge/arbitrum/gateway/L2GatewayRouter.sol</code>	4c4653650c55ee05b0602e65 e8bf41d3a281249a
<code>./arb-bridge-peripherals/contracts/tokenbridge/arbitrum/gateway/L2WethGateway.sol</code>	7839ab0ddc58df9ba6928fe7 bd41226b3c818bd2
<code>./arb-bridge-peripherals/contracts/tokenbridge/arbitrum/IArbToken.sol</code>	d4b4a121017677644a365291 0de98a13c1857c16
<code>./arb-bridge-peripherals/contracts/tokenbridge/arbitrum/L2ArbitrumMessenger.sol</code>	174aeb41169695e06d940094 d01853089f03c322




File	SHA-1 hash
./arb-bridge-peripherals/contracts/tokenbridge/arbitrum/StandardArbERC20.sol	8439da26324d9992938106a84e0950b71db97a60
./arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1ArbitrumExtendedGateway.sol	050658adc210b92119da8ff61cd01001413a2266
./arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1ArbitrumGateway.sol	61541aff960b601c4ca3f9167419d666dc6d94db
./arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1CustomGateway.sol	77940b0b83d3d390de75cc234bb7949c7bdf50a9
./arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1ERC20Gateway.sol	f51e1a6659fe9363ac0e31b93328f5f076e2017e
./arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1GatewayRouter.sol	cbe8dae31b0fa20253ee8c589b681d458cefc8a4
./arb-bridge-peripherals/contracts/tokenbridge/ethereum/gateway/L1WethGateway.sol	15da456d645429983315c945382577833f1876c3
./arb-bridge-peripherals/contracts/tokenbridge/ethereum/ICustomToken.sol	f7116d31844a88ce20720a2d8871d7fb18c771eb
./arb-bridge-peripherals/contracts/tokenbridge/ethereum/L1ArbitrumMessenger.sol	3c804f79c0eda3dd891ed7acd8ef44efae3e177d
./arb-bridge-peripherals/contracts/tokenbridge/libraries/aeERC20.sol	bf6f376401531087b81162f4ae5ca70668f5f3af
./arb-bridge-peripherals/contracts/tokenbridge/libraries/aeWETH.sol	87b5ee486a16fd07195c04846fa3bd7c51fcb4b4
./arb-bridge-peripherals/contracts/tokenbridge/libraries/BytesParser.sol	4c6d4adacdec602b67e1112010398ece069827d3
./arb-bridge-peripherals/contracts/tokenbridge/libraries/ClonableBeaconProxy.sol	a400f3ffd54b8453c9b2c997dc5006ba3c1e1430



File	SHA-1 hash
<code>./arb-bridge-peripherals/contracts/tokenbridge/libraries/gateway/GatewayMessageHandler.sol</code>	<code>ed6e9dad3b5bd42f0c21e68c85c984dfad2c3430</code>
<code>./arb-bridge-peripherals/contracts/tokenbridge/libraries/gateway/GatewayRouter.sol</code>	<code>af9f416663cf0cedf50d7e207e40cd99db66f33f</code>
<code>./arb-bridge-peripherals/contracts/tokenbridge/libraries/gateway/ICustomGateway.sol</code>	<code>7784fd0f65eedf96f8d633b54ebcb694aa05fcf8</code>
<code>./arb-bridge-peripherals/contracts/tokenbridge/libraries/gateway/ITokenGateway.sol</code>	<code>fb765bfd9b0161acfeadb03c7ac6fe7a04a49c5c</code>
<code>./arb-bridge-peripherals/contracts/tokenbridge/libraries/gateway/TokenGateway.sol</code>	<code>849188128bf0252a505bfb3e4905a95c2dd36467</code>
<code>./arb-bridge-peripherals/contracts/tokenbridge/libraries/ITransferAndCall.sol</code>	<code>164bd08878a2af787278c36ccbf1688f1bc250e</code>
<code>./arb-bridge-peripherals/contracts/tokenbridge/libraries/IWETH9.sol</code>	<code>7e42b516352562dce8c03ff4ae879e153861a32f</code>
<code>./arb-bridge-peripherals/contracts/tokenbridge/libraries/L2GatewayToken.sol</code>	<code>58732dac2ce839694d23e7f1009f5f366f575fdd</code>
<code>./arb-bridge-peripherals/contracts/tokenbridge/libraries/TransferAndCallToken.sol</code>	<code>610eb1ec2532fdea39f1825e2b64150c6338991c</code>

Appendix 2 - Disclosure

ConsenSys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or  and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or

having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

PURPOSE OF REPORTS The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of code and only the code we note as being within the scope of our review within this report. Any Solidity code itself presents unique and unquantifiable risks as the Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond specified code that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. In some instances, we may perform penetration testing or infrastructure assessments depending on the scope of the particular engagement.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

LINKS TO OTHER WEB SITES FROM THIS WEB SITE You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners.



I agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability

to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

TIMELINESS OF CONTENT The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.



Request a Security Review Today

Get in touch with our team to request a quote for a smart contract audit.

[CONTACT US](#)

AUDITS
FUZZING
SCRIBBLE
BLOG
TOOLS
RESEARCH

Subscribe to Our Newsletter

Stay up-to-date on our latest offerings, tools, and the world of blockchain security.



- ABOUT
- CONTACT
- CAREERS
- PRIVACY POLICY

Email*

e-mail address



POWERED BY  CONSENSYS

