

Produced for



by



# Contents

<b>1</b>	<b>Executive Summary</b>	<b>3</b>
<b>2</b>	<b>Assessment Overview</b>	<b>5</b>
<b>3</b>	<b>System Overview</b>	<b>6</b>
<b>4</b>	<b>Limitations and use of report</b>	<b>9</b>
<b>5</b>	<b>Terminology</b>	<b>10</b>
<b>6</b>	<b>Findings</b>	<b>11</b>
<b>7</b>	<b>Notes</b>	<b>14</b>

# 1 Executive Summary

Dear all,

Thank you for trusting us to help Aave with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of the BridgeExecutor contracts on Arbitrum/Optimism according to [Scope](#) to support you in forming an opinion on their security risks.

The smart contracts implement the Executor for Governance actions on Arbitrum/Optimism, hence they bear a very privileged role within the Aave contracts on the network.

The most critical subjects covered in our audit are functional correctness and security of the queue / execution mechanism. The issues reported as part of the holistic assessment of the smart contracts security might affect the secure operation, depending on the behavior of the trusted roles.

In summary and under the assumption the trusted roles act correctly as expected, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

# 1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

-Severity Findings	0
-Severity Findings	0
-Severity Findings	0
-Severity Findings	4
•	4



## 2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

### 2.1 Scope

The assessment was performed on the source code files inside the Aave Governance Cross-Chain Bridges repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	30 May 2022	303bd3e16c8d3465325389a72564126c3bacc4f9	Initial Version
2	22 June 2022	fd3a14b8e95a85df54d83cca6386f46b1082ba4d	Version 2
3	20 July 2022	8fa25b0080dd3dcc2390313631aea6796a12c9d8	AddressAliasHelper update

For the solidity smart contracts, the compiler version 0.8.10 was chosen.

The contracts in scope for this review were:

ArbitrumBridgeExecutor - to be deployed on ArbitrumOne

OptimismBridgeExecutor - to be deployed on Optimism Mainnet

The following files were in scope:

OptimismBridgeExecutor.sol, ArbitrumBridgeExecutor.sol, BridgeExecutorBase.sol, L2BridgeExecutor.sol and interfaces used by these contracts.

EVM equivalence, with the differences described in the Arbitrum/Optimism documentation respectively was assumed for the execution environment of the smart contracts under review.

#### 2.1.1 Excluded from scope

The messenger contracts and the message passing mechanism from L1 to L2 provided by the L2 solution was not in scope of this review and expected to work correctly as documented.

All files in subfolder `dependencies` were not part of the review.

All files related to PolygonBridgeExecutor were not in the scope of this review.

## 3 System Overview

The smart contracts reviewed implement the executor contracts for the Aave governance on L2, namely Optimism and Arbitrum. A similar executor is already live on the Polygon network.

Aave governance takes place on L1 Ethereum. Generally speaking, proposals are calls, which are, if accepted, queued to be executed by the executor contract. This executor contract e.g., bears privileged roles in the Aave system and hence can update system parameters.

The BridgeExecutor on L2 will be the executor for governance actions which have been received from L1 Ethereum. Note that the L2 executor contract implementation is similar, but not identical to the existing implementation of the executor on L1 Ethereum.

To execute an action on L2, the proposal on L1 is a call to the messenger contract on L1 of the L2 solution, the message contains the details for the action to be queued for execution in the BridgeExecutor contract on L2. The message is relayed when it is executed through the executor contract on Ethereum L1, this adds the proposal to the queue of the executor contract on L2 where it can be executed.

For Arbitrum the proposal will be a call to the following function in the Arbitrum Inbox contract on Ethereum:

```
/**
 * @notice Put a message in the L2 inbox that can be reexecuted for some fixed amount of time if it reverts
 * @dev all msg.value will be deposited to callValueRefundAddress on L2
 * @param destAddr destination L2 contract address
 * @param l2CallValue call value for retryable L2 message
 * @param maxSubmissionCost Max gas deducted from user's L2 balance to cover base submission fee
 * @param excessFeeRefundAddress maxgas x gasprice - execution cost gets credited here on L2 balance
 * @param callValueRefundAddress l2Callvalue gets credited here on L2 if retryable txn times out or gets cancelled
 * @param maxGas Max gas deducted from user's L2 balance to cover L2 execution
 * @param gasPriceBid price bid for L2 execution
 * @param data ABI encoded data of L2 message
 * @return unique id for retryable transaction (keccak256(requestID, uint(0) )
 */
function createRetryableTicket(
    address destAddr,
    uint256 l2CallValue,
    uint256 maxSubmissionCost,
    address excessFeeRefundAddress,
    address callValueRefundAddress,
    uint256 maxGas,
    uint256 gasPriceBid,
    bytes calldata data
) external payable returns (uint256)
```

For Optimism the proposal will be a call to the following function in the OVM L1 Cross Domain Messenger contract on Ethereum:

```
/**
 * Sends a cross domain message to the target messenger.
 * @param _target Target contract address.
 * @param _message Message to send to the target.
 * @param _gasLimit Gas limit for the provided message.
 */
function sendMessage(
    address _target,
    bytes memory _message,
    uint32 _gasLimit
) public
```

The destAddr (Arbitrum) / \_target (Optimism) will be the respective BridgeExecutor contract. The data (Arbitrum) / \_message (Optimism) will be the encoded data for the call to queue(targets, values, signatures, calldatas, withDelegatecalls). Note all the data should be correctly encoded by the Abi Encoder v2.

The messages relay between L1 and L2 works slightly differently for Optimism vs Arbitrum:

For Optimism:

The message is executed automatically on L2 using the specified gas limit. The cost of such an L1 to L2 transaction is the gas costs on L1 Ethereum. The first 1.92 million gas on L2 are free, a higher gas limit incurs a cost in the form of an increased gas consumption of the L1 message. No fee is paid on L2.

For Arbitrum:

`DepositValue` is credited to the sender's account on L2. A new ticket is created which includes collection of the submission fee and the callvalue from the senders L2 account balance. This fails when the L2 account balance is less than `MaxSubmissionCost` + `Callvalue`.

**Hence on Arbitrum the L2 address of the `_ethereumGovernanceExecutor` must hold or receive some balance.**

The retryable ticket is automatically executed. If this fails, the retryable ticket is placed in the retry buffer where it can be executed by anyone within a certain timeframe. The caller pays for the gas and can set the gas limit.

**If an L1 transaction underpays for a ticket's base submission fee, the ticket creation on L2 simply fails.**

Note that callvalue must be zero for messages to `queue()` since this function is not payable.

## 3.1 Contracts

The top level contracts, `OptimismBridgeExecutor` and `ArbitrumBridgeExecutor` implement the L2 solution specific modifier used to restrict access to the `queue` function to the L1 Executor only.

The core functionality is implemented in the `L2BridgeExecutor` or `BridgeExecutorBase` contract respectively.

Actions sets which contain one or multiple sequential actions (calls) are received from L1: The L1 Executor calls the messenger contract of the L2 solution on L1, the message is relayed and eventually on L2 the `queue()` function is invoked and the actions set is queued.

A queued actions set can be executed by anyone after it has been queued for a minimum time defined by the `delay` until the `grace` period has elapsed. Actions sets may include actions (calls) with non zero `msg.value`. Either the caller of `execute()` provides these funds or the `BridgeExecutor` must hold sufficient funds in order to execute these calls. These funds cannot be transferred alongside the actions set from L1 to L2, these funds must be transferred to the `BridgeExecutor` separately.

When executing an actions set, all calls must be successful or the execution reverts.

An actions set can be in one of the four states:

- `queued`: Queued. May or may not be ready for execution depending on whether the `delay` has already elapsed since the actions set has been queued or not.
- `expired`: The actions set has not been executed within the grace period.
- `cancelled`: The actions set has been cancelled by the `guardian` before it has been executed
- `executed`: The actions set has been executed.

The following state changing functionality is exposed:

- `queue()`:

```
function queue(  
    address[] memory targets,  
    uint256[] memory values,  
    string[] memory signatures,  
    uint gasLimit,  
    uint delay,  
    uint grace,  
    bool payable)  
public returns (uint256)
```

```
bytes[] memory calldatas,  
bool[] memory withDelegatecalls  
)
```

An action consists of the target to call, the callvalue, the signature (A string with the function name & parameter, used to derive the function selector), and the calldata. The signature may be empty. Furthermore, it can be specified if the call should be executed as a delegatecall in the context of the BridgeExecutor. This function can only be invoked by the respective messenger contract and the sender on L1 must be the `_ethereumGovernanceExecutor`.

- `execute()`: Function which can be called by anyone to execute a pending actions set up for execution.
- `cancel()`: The guardian can cancel any queued actions set which has not been executed, expired, or cancelled yet.
- `receiveFunds()` Function allowing the contract to receive the native currency, similar to the default `receive()` of solidity.

The internal state variables of the BridgeExecutor (`_miniumDelay`, `_maximumDelay`, `_delay`, `_gracePeriod`, `_guardian`, `_ethereumGovernanceExecutor`) can be updated through actions calling the respective functions of this contract which can only be accessed via a call originating from this contract itself.

## 3.2 Trust Model & Roles

**Aave Governance:** Fully trusted to act honestly and correctly at all times. Notably, this includes that the governance does not call `execute` with another actions set in one actions set, otherwise the atomicity of the actions set could be broken.

**Guardian:** Trusted role, can cancel queued action sets which have not been executed and expired yet.

**Executor:** Untrusted, anyone can trigger the execution of pending action sets ready for execution.



## 4 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

## 5 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High			
Medium			
Low			

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

## 6 Findings

In this section, we describe our findings. The findings are split into these different categories:

- : Related to vulnerabilities that could be exploited by malicious actors
- : Architectural shortcomings and design inefficiencies

Below we provide a numerical overview of the identified findings, split up by their severity.

-Severity Findings	0
-Severity Findings	0
-Severity Findings	0
-Severity Findings	4

- [Compromised Guardian Can Block the Executor](#)
- [Dangerous Delegatecalls](#)
- [Function Signature as String, Unicode Charset](#)
- [Potential Reentrancy on execute](#)

### 6.1 Compromised Guardian Can Block the Executor

The guardian role has the privilege to cancel queued actions sets before they are executed. Updating the guardian address can only be done through an action. If the guardian account is compromised, it can always cancel an actions set which tries to update the guardian role to a new address. Effectively, once the guardian address is compromised it can block the Executor indefinitely.

---

#### Risk accepted:

Aave replied:

The guardian address is designed to be a multisig or governance executor (never an EOA) so having a compromised guardian is unlikely to happen.

### 6.2 Dangerous Delegatecalls

Actions sets may include actions (calls) to be executed as `DELEGATECALL` in the context of the BridgeExecutor. While this allows to aggregate multiple calls governed by code which can adapt to on chain state, should the called contract write to storage, this would write to the storage of the BridgeExecutor. Hence variables of the contract may be overwritten. This can result in the internal

variables being changed without respecting the restrictions enforced in their setter function, e.g. `updateGracePeriod()` and the check for the minimum grace period.

Besides, a `Delegatecall` is also able to modify/delete existing actions sets, insert arbitrary new actions sets or manipulate entries in `_queuedActions`. The governance must be aware of this danger.

Untrusted code must never be called with `DELEGATECALL`.

Furthermore, note that the following corner case exists: An action consisting of a call to the `BridgeExecutor`'s `executeDelegateCall` function technically allows the governance to execute a call as a `Delegatecall` (with the risks mentioned above) despite the flag `withDelegatecalls` being set to `false`.

---

### Risk accepted:

Aave replied:

The Executor contract assumes that any set of transactions that are queued by a successful proposal is legit. Thus, there are no bad actions that the contract can execute since the proposal passes multiple checks by the community, devs, white hats, auditors, etc.

The `executeDelegateCall` function is designed to be used for executing payload contracts, where a set of actions are described (instead of having multiple encoded calldatas). The governance should check that the delegate call execution does not update or alter any executor contract's state variable.

Apart from that, the correct way of doing a delegate call is through the action set and the `execute` function, instead of calling directly to `executeDelegateCall`. The community would detect and raise this concern if applicable.

## 6.3 Function Signature as String, Unicode Charset

An action may contain the function signature as a string. This allows to display the function to be called in a human readable way. However, this can be dangerous as strings support the unicode charset and many lookalike characters of different alphabets exist in this charset. Hence users might be tricked to approve an action which seemingly contains the intended function call, but actually results in a different function selector. Given a function selector consists of 4 bytes only, it might be feasible to find such a collision.

For more insights into lookalike characters, please refer to: <https://util.unicode.org/UnicodeJsps/confusables.jsp?a=setReserveActive>

---

### Risk accepted:

Aave replied:

Governance should assess, test and simulate each proposal, checking the outcome of its changes without trusting string function signatures. Having the function signature human-readable is not a way of validating the legitimacy of proposals by any means.

## 6.4 Potential Reentrancy on `execute`

The function `execute` is not protected against reentrancy. While the governance is trusted to not create actions sets which reenter into `execute()` and start executing another actions set, generally speaking an action may reach untrusted third-party code. This untrusted code may reenter the BridgeExecutor. This would break the atomicity of sets of actions and may result in unexpected executions and states.

---

### **Risk accepted:**

Aave replied:

The community and governance decide if an action set should execute any other action set of the same executor. The community should assess every governance proposal carefully.

# 7 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

## 7.1 Execution Order of Queued Actions Sets

Multiple queued actions sets which are ready for execution may be executed in an arbitrary order. All actions which depend on a particular execution order must be placed within the same actions set where the order of execution is defined. If multiple actions sets exist at the same time, they must be independent of each other.

Moreover, updating critical system variables in one actions set might change the behavior of other actions sets. For example, assume `_delay` is set to one day at the beginning. Actions set A is queued in the BridgeExecutor, A updates `_delay` to one second. On L1, the governance decides on actions set B. Governance should be careful, depending on whether A is executed before/after actions set B is queued on L2, a different `_delay` is applied before B can be executed.

## 7.2 Impact of Rollback, Finality of Actions

Especially regarding finality, L2 solutions based on optimistic rollups behave differently than L1 Ethereum.

Should a rollback happen due to the discovery of an incorrect tx, this tx and all subsequent tx have to be reexecuted. This impacts the timestamp of the transaction. Most of the times incorrect transaction results are detected immediately and the rollback happens immediately, however in a worst-case scenario the rollback might happen just before the end of the fault proof period. In such cases the timestamp of the transaction changes significantly.

The BridgeExecutor heavily relies on the timestamps, e.g. to determine whether an action can be executed or if it already expired. Similarly the execution time is calculated based on the timestamp when `queue()` is executed. After a rollback the timestamps may have shifted and e.g. a previously executed actions set can no longer be executed as it has expired.

Furthermore, the order of transactions after a rollback is not guaranteed, there may be a change of sequence between a transaction to `execute()` or `cancel()` a pending actions set.

Validating all transactions may help to detect incorrect transactions early, however in a worst-case scenario (e.g. a bug in the validator software) may not detect such a wrong transaction and an unexpected fraud proof may be submitted resulting in a rollback.

The governance needs to be careful about finality on L2. Overall L2 solutions are still considered as experimental, interactions must be done with care.

Note that at the time of this review Optimism has not yet implemented fraud proofs while in Arbitrum only whitelisted addresses can create a challenge.

## 7.3 Potentially Resurrected ActionsSet

An actions set expires if the `_gracePeriod` has elapsed since the `executionTime`. However, an expired actions set may resurrect if the `_gracePeriod` is extended by the governance later in the future. Resulting an expired actions set might be executable again.

It should be carefully thought about if this `suspended` state should be allowed, especially as the `guardian` can only cancel queued actions set which have not yet expired.