Aave

Aave CrossChain Infrastructure Contract Review

Version: 3.2

August, 2023

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the Aave smart contracts. The review focused solely on the security aspects of the Solidity implementation of the contract, though general recommendations and informational comments are also provided.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

Document Structure

The first section provides an overview of the functionality of the Aave smart contracts contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the Aave smart contracts.

Overview

The Aave CrossChain Infrastructure is the new cross-chain communication layer for Aave. It is the system responsible for the communication across different networks for Aave Governance V3. It enables the communication using different bridges that allow the system to receive and forward messages to and from different chains.

The CrossChain Infrastructure has also an emergency mode. If the cross chain communication and execution infrastructure breaks, this mode is triggered and gives the permissions to the *Guardian* to replace bridges and change other configurations.

Security Assessment Summary

This review was conducted on the files hosted on the Aave CrossChain Infrastructure repository and were assessed at commit a85bc4c, then again at commit 759e28a.

Specifically, the files in scope are as follows:

- `BaseCrossChainController.sol`
- `CrossChainController.sol`
- `CrossChainControllerWithEmergencyMode.sol`
- `CrossChainForwarder.sol`
- `CrossChainReceiver.sol`
- `EmergencyConsumer.sol`
- `EmergencyRegistry.sol`
- `ChainIds.sol`
- `EncodingUtils.sol`
- `Erros.sol`
- `SameChainAdapter.sol`
- `BaseAdapter.sol`

Interfaces of the previous contract are included in the scope too.

Note: the OpenZeppelin and Solidity Utils libraries and dependencies were excluded from the scope of this assessment.

The manual code review section of the report is focused on identifying any and all issues/vulnerabilities associated with the business logic implementation of the contracts. This includes their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout). Additionally, the manual review process focused on all known Solidity anti-patterns and attack vectors. These include, but are not limited to, the following vectors: re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers. For a more thorough, but non-exhaustive list of examined vectors, see [1, 2].

To support this review, the testing team used the following automated testing tools:

- Mythril: `https://github.com/ConsenSys/mythril`
- Slither: `https://github.com/trailofbits/slither`
- Surya: `https://github.com/ConsenSys/surya`

Output for these automated tools is available upon request.

Findings Summary

The testing team identified a total of 5 issues during this assessment. Categorised by their severity:

- High: 1 issue.
- Low: 1 issue.
- Informational: 3 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Aave smart contracts. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as “informational”.

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
CCI-01	The `SameChainAdapter` Cannot Forward Messages	High	Resolved
CCI-02	`forwardMessage()` Should Not Be Called Directly	Low	Closed
CCI-03	Discrepancy Between Documentation & Implementation	Informational	Resolved
CCI-04	Upgradeable Contracts Must Disable Initializers In The Implementation Contracts	Informational	Closed
CCI-05	Miscellaneous General Comments	Informational	Resolved

CCI-01	The `SameChainAdapter` Cannot Forward Messages		
Asset	`adapters/sameChain/SameChainAdapter.sol` & `CrossChainForwarder.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: Medium	Likelihood: High

Description

Due to the mismatch between the bytes parameter `message` of the function `SameChainAdapter.forwarMessage()` and the input `encodedTransaction.data` used to call the bridge adapter function `forwardMessage()` in `CrossChainForwarder._bridgeTransaction()`, the `SameChainAdapter` cannot forward messages to the intended destination.

In fact, in `SameChainAdapter.forwarMessage()`, the `message` parameter, which is the 4th parameter of this function, is decoded in line [**25-27**] decoded to `(uint256, address, address, bytes)`.

```
17  function forwardMessage(
      address,
19    uint256,
      uint256 destinationChainId,
21    bytes calldata message
    ) external returns (address, uint256) {
23    require(destinationChainId == getChainID(), Errors.DESTINATION_CHAIN_NOT_SAME_AS_CURRENT_CHAIN);

25    (, address msgOrigin, address msgDestination, bytes memory decodedMessage) = abi.decode(
        message,
27      (uint256, address, address, bytes)
      );
```

However, in the function `CrossChainForwarder._bridgeTransaction()`, the 4th parameter that is used to make a delegatecall to the function `forwardMessage` of the `currentBridgeAdapater` is `encodedTransaction.data`.

```
265 (bool success, bytes memory returnData) = bridgeAdapters[i]
      .currentChainBridgeAdapter
267   .delegatecall(
        abi.encodeWithSelector(
269       IBaseAdapter.forwardMessage.selector,
          bridgeAdapters[i].destinationBridgeAdapter,
271       gasLimit,
          destinationChainId,
273       encodedTransaction.data
        )
275   );
```

`encodeTransaction.data` is `abi.encode(Transaction)` which would be :

- `0x20:` nonce
- `0x40:` encodedEnvelope pointer
- `0x60:` encodedEnvelope length
- `0x80:` encodedEnvelope - nonce
- `0x0100:` encodedEnvelope - origin
- `...`

As a result, the `encodeTransaction.data` doesn't match with the expected bytes parameter in `SameChainAdapter.forwarMessage()`, and the delegatecall to this function in

`CrossChainForwarder._bridgeTransaction()` will always revert, hence the `SameChainAdapter` cannot forward messages.

Recommendations

Add the necessary decoding function in `SameChainAdapter.forwarMessage()` so that the message can be decoded properly.

Resolution

This issue has been fixed in commit 6b9fddb by using the `decode()` function from the library `TransactionUtils` to properly decode the `encodeTransaction.data` input sent in `CrossChainForwarder._bridgeTransaction()` and then get the envelope from the decoded transaction using the function `TransactionUtils.getEnvelope()`.

CCI-02	`forwardMessage()` Should Not Be Called Directly		
Asset	`adapters/*`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Medium

Description

For each adapter that implements `IBaseAdapter` it is possible to call `forwardMessage()` directly and interact with the associated bridge.

The protocol is designed for the `CrossChainController` to make a `delegatecall` to `forwardMessage()`, where it will transfer the required fee tokens from the `CrossChainController` to the associated bridge.

There is no access control on `forwardMessage()`, hence an attacker may interact with the implementation contract directly to send messages via the bridge.

For example, `HyperLaneAdapter` will create a cross chain message and send `quotedPayment` amount of native tokens to the gas payment control when `forwardMessage()` is called directly.

```
function forwardMessage(
    address receiver,
    uint256 destinationGasLimit,
    uint256 destinationChainId,
    bytes calldata message
    ) external returns (address, uint256) {
    uint32 nativeChainId = SafeCast.toUint32(infraToNativeChainId(destinationChainId));
    require(nativeChainId != uint32(0), Errors.DESTINATION_CHAIN_ID_NOT_SUPPORTED);
    require(receiver != address(0), Errors.RECEIVER_NOT_SET);

    bytes32 messageId = HL_MAIL_BOX.dispatch(
        nativeChainId,
        TypeCasts.addressToBytes32(receiver),
        message
    );

    // Get the required payment from the IGP.
    uint256 quotedPayment = IGP.quoteGasPayment(nativeChainId, destinationGasLimit);

    require(quotedPayment <= address(this).balance, Errors.NOT_ENOUGH_VALUE_TO_PAY_BRIDGE_FEES);

    // Pay from the contract's balance
    IGP.payForGas{value: quotedPayment}( //@audit native tokens transferred to payment
        messageId, // The ID of the message that was just dispatched
        nativeChainId, // The destination domain of the message
        destinationGasLimit,
        address(this) // refunds go to msg.sender, who paid the msg.value
    );

    return (address(HL_MAIL_BOX), uint256(messageId));
    }
```

Similar issues are present in `LayerZeroAdapter` and `CCIPAdapter`, except `CCIPAdapter` will transfer $LINK$ tokens rather than native tokens.

This issue is rated as low impact since the implementation contracts of `IBaseAdapter` are not intended to maintain token balances. However, it is still possible for funds to be transferred into this contract either directly or via a bridge.

Note that although these contracts do not have a `receive()` or `payable` function it is still possible to transfer native tokens into the contract via miner rewards or `SELFDESTRUCT`.

Furthermore, since these transactions will be `call` rather than `delegatecall`, therefore the bridge origin address will be the `IBaseAdapter` rather than `CrossChainController`. Thus, it is not possible to use this method to forge messages from the `CrossChainController`.

Recommendations

Consider adding access control to the functions which should only be triggered via `delegatecall`:

- `HyperLaneAdapter.forwardMessage()`
- `LayerZeroAdapter.forwardMessage()`
- `CCIPAdapter.forwardMessage()`
- `CCIPAdapter.setupPayments()`

A check to ensure the `CrossChainController` is making a `delegatecall` can be implemented as follows:

```
require(address(this) == address(CROSS_CHAIN_CONTROLLER), "not cross chain controller delegatecall");
```

Resolution

The development team has clarified that the proposed access control will not add any safeguard to the project because the origin checks are done in the destination.

CCI-03	Discrepancy Between Documentation & Implementation
Asset	`contracts/*`
Status	**Resolved:** See Resolution
Rating	Informational

Description

There is are discrepancies between the documentation in the README.md and the implementation of the project.

In fact, it is mentioned in the `README.md` under the section *CrossChain Contracts* that the `CrossChainForwarder` and `CrossChainReceiver` contracts are `abstract`, while in the implementation these contracts are not abstract.

Added to that, the sub-section *Permissions* of the section *Aave Governance Emergency Mode* indicates that the call to `set Emergency` can be done by anyone, but, in the implementation the function `setEmergency()` has the `onlyOwner` modifier.

Recommendations

Ensure that the documentation matches with the implementation of the project.

Resolution

This issue has been fixed in PR #86.

CCI-04	Upgradeable Contracts Must Disable Initializers In The Implementation Contracts
Asset	`CrossChainController.sol`
Status	**Closed:** See Resolution
Rating	Informational

Description

Quoting OpenZeppelin:

> Avoid leaving a contract uninitialized.
>
> An uninitialized contract can be taken over by an attacker. This applies to both a proxy and its implementation contract, which may impact the proxy. To prevent the implementation contract from being used, you should invoke the `_disableInitializers` function in the constructor to automatically lock it when it is deployed.

If the `CrossChainController` function was to remain uninitialized and a malicious user were able to set a bridge adapter to a contract which self-destructs, the `delegatecall` in `_bridgeTransaction()` would cause the proxy contract to self-destruct. It would not be possible to recover the proxy.

Recommendations

Add the following code to the `CrossChainController` contract:

```
constructor(
  address clEmergencyOracle
)
  CrossChainReceiver(new ConfirmationInput[](0), new ReceiverBridgeAdapterConfigInput[](0))
  CrossChainForwarder(new ForwarderBridgeAdapterConfigInput[](0), new address[](0))
  EmergencyConsumer(clEmergencyOracle)
{
  _disableInitializers();
}
```

Resolution

The development team has clarified that there is no need to implement the proposed recommendation as they used a custom version of the transparent proxy which already has the `_disableInitializers()` method called on the constructor.

CCI-05	Miscellaneous General Comments
Asset	`contracts/*`
Status	**Resolved:** See Resolution
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

1. **Missing interface inheritance.**
   - The contract `CrossChainController` inherits from the contract `EmergencyConsumer` while the interface `ICrossChainController` does not inherit from the interface `IEmergencyConsumer`.
   - The contract `CrossChainReceiver` inherits from the contract `OwnableWithGuardian` while the interface `ICrossChainReceiver` does not inherit from the interface `IOwnableWithGuardian`.
   - The contract `CrossChainForwarder` inherits from the contract `OwnableWithGuardian` while the interface `ICrossChainForwarder` does not inherit from the interface `IOwnableWithGuardian`.
2. **Misplaced comments.**
   - The comments in line [**53-56**] in the `CrossChainController.initialize()` function are misplaced because as it is indicated in the comments that *"by strictly following this update order we can ensure that the state after the emergency is solved"*, while the function here is `initialize()` and it doesn't have a relation with the emergency mode.
   - In `EncodingUtils.sol`, the *@notice* in line [**123**] of the `decode()` function in the `TransactionUtils` library does not match with the function as this comment indicates that *method that encodes a Transaction and generates its id* which is not correct.
   - In `EncodingUtils.sol`, the *@return* in line [**125**] of the `decode()` function in the `TransactionUtils` library does not match with the function as this comment indicates that the return value is an *object containing the encoded transaction and the transaction id* which is not correct because this function returns the object `Transaction` which contains the nonce of the transaction and the associated encoded envelope.
   - In `crosschain-infra/adapters/hyperLane/HyperLaneAdapter.sol` line [**76**] the refunded address is not `msg.sender` which does not match the comment *refunds go to msg.sender, who paid the msg.value*.
3. **Typos and grammar**
   - `CrossChainForwarder.sol` line [**280**] "doesnt" should be "doesn't".
   - `CrossChainForwarder.sol` line [**320**] "dont" should be "don't".
   - `ICrossChainForwarder.sol` line [**202**] "adateprs" should be "adapters".
   - `ICrossChainForwarder.sol` line [**117**] "ore" should be "or".
   - `CrossChainReceiver.sol` line [**165**] "did't" should be "didn't".
   - `EncodingUtils.sol` line [**132**] "an transaction id" should be "a transaction id".
4. **Unused errors.**
   The following errors are not used and may be safely deleted.

- `INCORRECT_ORIGIN_CHAIN_ID`
- `BRIDGE_ADAPTER_NOT_SET_FOR_ANY_CHAIN`
- `ENVELOPE_DELIVERY_FAILED`
- `SYSTEM_NEEDS_AT_LEAST_ONE_CONFIRMATION`
- `INVALID_CONFIRMATIONS_FOR_ALLOWED_ADAPTERS`

5. **Avoid using `call` and `delegatecall` on a single contract.**

   The `IBaseAdapter` contracts have functions which should only be made via `delegatecall` such as `forwardMessage()`. Additionally, they have functions which should only be made via `call` such as `ccipReceive()` and `handle()`.

   Using contracts for both `calls` and `delegatecalls` may result in reading or modifying storage with incorrect slots. Currently, the functions which use `delegatecall` do not read from or write to any state, thus are not vulnerable to this class of bug.

   It is advised to separate the logic of adapters into two contracts, one for `call` and one for `delegatecall`.

6. **Add storage gaps to proxied contracts.**

   The contract `CrossChainController` is intended to be a proxy contract. Adding new storage variables to inherited contracts would not be possible under the current setup as it would modify the slot ordering for other contracts.

   Consider adding an empty array of storage slots which may be decremented if new variables are required e.g. `uint256[50] __gap;`.

   These may be added to each of the following inherited contracts:

   - `CrossChainForwarder`
   - `CrossChainReceiver`
   - `EmergencyConsumer`

7. **Additional issues in comments.**

   - `MetisAdapter.sol` line [**16**] "note that this adapter inherits from Optimism adapter and overrides only supported chain" is not true as it also overrides `forwardMessage()`.
   - `ArbAdapter.sol` line [**17**] "note that this adapter is can only be used for ..." should remove "is".

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The development team have acknowledged these findings, addressing them where appropriate as follows:

1.
   - A new interface `ICrossChainControllerL2` has been added that inherits from the interface `IEmergencyConsumer` in commit 7df3edb
   - As `OwnableWithGuardian` doesn't have an interface, the development has stated that it is fine that `ICrossChainReceiver` does not inherit from an interface associated to `OwnableWithGuardian`

- As `OwnableWithGuardian` doesn't have an interface, the development has stated that it is fine that `ICrossChainForwarder` does not inherit from an interface associated to `OwnableWithGuardian`

2. Resolved in PR #119.
3. Resolved in PR #119.
4. Resolved in PR #119.
5. The development team have stated that it is by design to not separate the logic of adapters in order to reduce the project complexity and not having many contracts.
6. A storage gap has been added to the `CrossChainForwarder` and `CrossChainReceiver` contracts in PR #119. However, the development team have decided to not add a storage gap to the `EmergencyConsumer` contract as they believe that this contract will not be updated with extra storage.
7. Resolved in PR #126.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are provided alongside this document. The `brownie` framework was used to perform these tests and the output is given below.

```
test_constructor                                          PASSED  [2%]
test_get_trusted_remote_by_chain_id                       PASSED  [4%]
test_register_received_message                            PASSED  [6%]
test_register_received_message_delegatecall               PASSED  [8%]
test_basic                                                PASSED  [10%]
test_constructor                                          PASSED  [12%]
test_initialize                                           PASSED  [14%]
test_forward_message                                      PASSED  [16%]
test_forward_message_no_bridge_adapter                    PASSED  [18%]
test_forward_message_wrong_caller                         PASSED  [20%]
test_retry_envelope                                       PASSED  [22%]
test_retry_envelope_non_registered_envelope               PASSED  [24%]
test_retry_envelope_no_bridge                             PASSED  [26%]
test_retry_transaction                                    PASSED  [28%]
test_retry_transaction_no_bridge                          PASSED  [30%]
test_retry_transaction_not_prev_forwarded_transaction     PASSED  [32%]
test_retry_transaction_use_the_same_bridge_twice          PASSED  [34%]
test_approve_senders                                      PASSED  [36%]
test_remove_senders                                       PASSED  [38%]
test_enable_bridge_adapters                               PASSED  [40%]
test_enable_bridge_adapters_zero_address                  PASSED  [42%]
test_disable_bridge_adapters                              PASSED  [44%]
test_receive_cross_chain_message_case_1                   PASSED  [46%]
test_receive_cross_chain_message_case_2                   PASSED  [48%]
test_receive_cross_chain_message_case_3                   PASSED  [51%]
test_deliver_envelope                                     PASSED  [53%]
test_deliver_envelope_invalid_state                       PASSED  [55%]
test_allow_receiver_bridge_adapters                       PASSED  [57%]
test_allow_receiver_bridge_adapters_same_adapter_same_chain  PASSED  [59%]
test_allow_receiver_bridge_adapters_zero_address_bridge   PASSED  [61%]
test_disable_receiver_bridge_adapters                     PASSED  [63%]
test_update_confirmations                                 PASSED  [65%]
test_update_confirmations_incorrect_nb_confirmation       PASSED  [67%]
test_update_messages_validity_timestamp                   PASSED  [69%]
test_update_messages_validity_timestamp_invalid_timestamp  PASSED  [71%]
test_update_cl_emergency_oracle                           PASSED  [73%]
test_update_cl_emergency_oracle_not_owner                 PASSED  [75%]
test_update_cl_emergency_oracle_zero_address              PASSED  [77%]
test_solve_emergency                                      PASSED  [79%]
test_solve_emergency_not_in_emergency                     PASSED  [81%]
test_emergency_token_transfer                             PASSED  [83%]
test_emergency_ether_transfer                             PASSED  [85%]
test_set_emergency                                        PASSED  [87%]
test_set_emergency_same_chain                             PASSED  [89%]
test_set_emergency_wrong_caller                           PASSED  [91%]
test_forward_message                                      PASSED  [93%]
test_native_to_infra_chain_id                             PASSED  [95%]
test_infra_to_native_chain_id                             PASSED  [97%]
test_get_trusted_remote_by_chain_id                       PASSED  [100%]
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

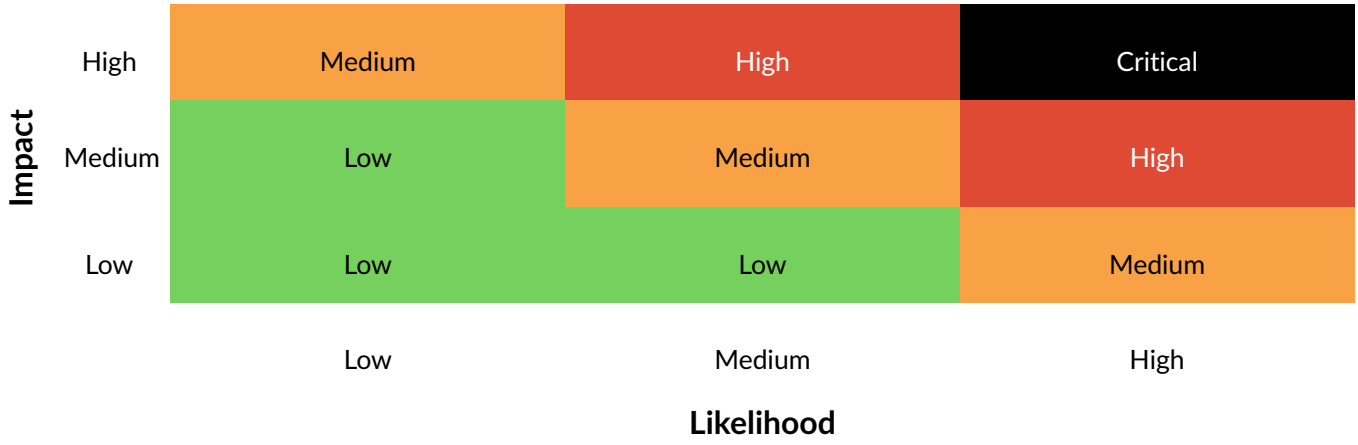

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Sigma Prime. Solidity Security. Blog, 2018, Available: https://blog.sigmaprime.io/solidity-security.html. [Accessed 2018].

[2] NCC Group. DASP - Top 10. Website, 2018, Available: http://www.dasp.co/. [Accessed 2018].