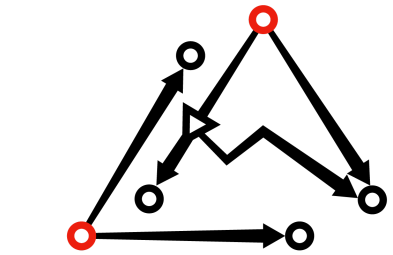

Least Authority

PRIVACY MATTERS

Bridge Smart Contracts

Security Audit Report

Ava Labs

Final Audit Report: 7 July 2023

Table of Contents

Overview

Background

Ava Labs has requested that Least Authority perform a security audit of their Bridge Smart Contracts.

Project Dates

- **April 3 - May 8, 2023:** Code Review (Completed)
- **May 10, 2023:** Delivery of Initial Audit Report *(Completed)*
- **May 29, 2023:** Delivery of Updated Initial Audit Report *(Completed)*
- **July 6, 2023:** Verification Review *(Completed)*
- **July 7, 2023:** Delivery of Final Audit Report *(Completed)*

Review Team

- Alicia Blackett, Security Researcher and Engineer
- Mukesh Jaiswal, Security Researcher and Engineer
- Ahmad Jawid Jamiulahmadi, Security Researcher and Engineer
- DK, Security Researcher and Engineer
- Xenofon Mitakidis, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Bridge Smart Contracts followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repositories are considered in scope for the review:

- Teleporter:
  https://github.com/ava-labs/teleporter
- EVM Relayer:
  https://github.com/ava-labs/awm-relayer
- Avalanchego:
  https://github.com/ava-labs/avalanchego/tree/master/vms/platformvm/warp
- Subnet-EVM PR:
  https://github.com/ava-labs/subnet-evm/pull/586

Specifically, we examined the Git revisions for our initial review:

- Teleporter: `a92507128e321243721d9b4cb8f5b0cf3d91cee4`
- AVM Relayer: `eab4112914de96d1d58911a7925c161286b5c5d0`
- Avalanchego: `d0a55ba57a5d8ebaa0717a812994f4a3df49c9d8`
- Subnet-EVM: `21c1fb1fe45861a3a4a2323f228dbc6ea0d37d70`
- Subnet-EVM PR: `ef4fb8d2897045cc532873739942d16a37f70425`

For the verification, we examined the Git revisions:

- Teleporter: `0f76bf51d02027a3139680a880a0d4ddff782ec1`
- AVM Relayer: `e646a953cc79f749ee8eb843bfa88185483018d1`

For the review, these repositories were cloned for use during the audit and for reference in this report:

- Teleporter: https://github.com/LeastAuthority/AvaLabs_Teleporter
- EVM Relayer: https://github.com/LeastAuthority/AvaLabs_awm_Relayer
- Avalanchego : https://github.com/LeastAuthority/AvaLabs_Avalanchego/tree/master/vms/platformvm/warp
- Subnet-EVM PR: https://github.com/LeastAuthority/AvaLabs-Subnets/pull/6

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third-party code, unless specifically mentioned as in scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- Avalanche Warp Messaging: https://github.com/ava-labs/subnet-evm/blob/warp-e2e/precompile/contracts/warp/README.md
- Teleporter Implementation Document.pdf *(shared with Least Authority via Slack on 30 March 2023)*

In addition, this audit report references the following documents:

- Type assertions: https://go.dev/ref/spec#Type_assertions
- SetFinalizer mechanism: https://pkg.go.dev/runtime#SetFinalizer
- NatSpec Format: https://docs.soliditylang.org/en/v0.8.19/natspec-format.html
- Solidity Style Guide: https://docs.soliditylang.org/en/v0.8.19/style-guide.html

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Common and case-specific implementation errors;
- Adversarial actions and other attacks on the bridge;
- Attacks that impact funds, such as the draining or manipulation of funds;
- Mismanagement of funds via transactions;
- Denial of Service (DoS) attacks and security exploits that would impact or disrupt execution of the bridge;
- Vulnerabilities within individual components and whether the interaction between the components is secure;
- Exposure of any critical information during interactions with any external libraries;
- Proper management of encryption and signing keys;
- Protection against malicious attacks and other methods of exploitation;
- Data privacy, data leaking, and information integrity;
- Inappropriate permissions and excess authority;

- Vulnerabilities in the solidity contract, including, but not limited to, re-entrancy attacks, access control, etc.;
- Opportunities for optimizations, such as increasing gas efficiency in the solidity contracts and following the latest solidity best practices;
- Possible attack vectors in the Teleporter contract due to the reordering of messages or subsequent attempts to retry messages;
- Usage of Peer-to-Peer (P2P) networking interface and DoS vectors in the AWM relayer; and
- Whether EVM nonce-management is currently in accordance with best practices.

Findings

General Comments

Our team performed a security audit of the Ava Labs Teleporter system, which is intended to facilitate efficient cross-chain transfer of messages and assets. The system is composed of several components, including a relayer, which is a node responsible for passing information across subnets, implemented in Go, an Ethereum smart contract suite implemented in Solidity, and the Avalanche Subnet EVM Warp Precompile, which is a primitive that allows communication between custom subnets.

In addition to investigating the areas of concern listed above, our team performed a comprehensive review of the design and implementation of Teleporter, its components, and the modifications introduced to the Subnet EVM, including how the transaction access list is handled, the mechanism allowing subnet-to-subnet communication, and changes in total intrinsic gas calculations. We found that the system is well-designed and well-implemented. However, we identified some areas of improvement.

System Design

In our review of the design of the system, our team identified issues, primarily in the design of the smart contract component, that could lead to unexpected behavior. We found that the design of the messaging system could have unintended effects on the balance of the relayer incentive mechanism (Issue A). This issue can be exacerbated by functionality allowing copies of the same receipt to be enqueued (Issue B). We recommend exploring solutions other than a queue for recording outstanding receipts.

We found that tokens with particular characteristics could lead to incorrect token accounting in the bridge smart contracts (Issue E). Additionally, the smart contracts require optional functionality in tokens to be implemented, and tokens that do not implement this functionality cannot use the bridge (Issue F).

Our team identified areas of improvement in the design of the smart contract suite, including the implementation of an emergency pause functionality (Suggestion 10) as well as an upgrade mechanism for the bridge token (Suggestion 11).

We found that security has been taken into consideration in the design of the relayer. However, we found that insufficient clearing of sensitive data is performed in the implementation of the relayer, and we recommend this be improved (Suggestion 4).

Code Quality

Our team performed a manual code review of the components in scope and found that the relayer/Teleporter code is well-written and organized. However, we identified opportunities for improving the overall security and quality of the relayer implementation (Suggestion 1, Suggestion 2, Suggestion 3).

In our review of the solidity codebase, our team identified an implementation Issue whereby the gas fee calculation is not implemented completely, causing a delay in user transactions during high network activity periods (Issue D). Furthermore, we found that the quality of the code can be improved by adhering to solidity best practices (Suggestion 6) and style guidelines (Suggestion 8). Our team also found gas optimizations that will improve gas efficiency and readability for the system and its users (Suggestion 9, Suggestion 5), if implemented.

Tests

Our team found that sufficient tests are implemented for the smart contracts suite. However, the test coverage for the relayer was found to be insufficient, with no test coverage for `relayer.go` and very limited coverage for the `utils` and `config` files. We recommend expanding the test suite for the relayer to cover all success, failure, and edge cases (Suggestion 12).

The Warp precompile tests contain unresolved TODOs, which we recommend completing.

Documentation

The project documentation provided for this review is generally sufficient and accurately describes the system and its components.

Code Comments

The relayer implementation, the smart contracts, and the Warp Precompile are all well-commented. However, we recommend adhering to the NatSpec guidelines for code comments in Solidity codebases (Suggestion 7).

Scope

The scope of this review was generally sufficient and included all security-critical components. We recommend that the implementation of the BLS signatures be included in future security audits of the system.

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Issue A: Unbalanced Activities Between Two Chains Can Grow the Queue of Outstanding Receipts and Reduce Relayer Incentive	Partially Resolved
Issue B: Duplicate Receipts Can Be Re-Enqued in the retryReceipt Function	Resolved
Issue C: Missing Zero Address Check	Resolved
Issue D: MinerTip Is Not Included in the Calculation of MaxFeeCap	Partially Resolved
Issue E: ERC20 Token Supporting “Fee on Transfer” Can Lead To Error in Accounting of Relayer Reward	Resolved
Issue F: submitCreateBridgeToken Will Revert for the Tokens That Do Not	Unresolved

Implement the Optional Function of the ERC20 Token Standard	
Suggestion 1: Check That Type Assertion Succeeds	Resolved
Suggestion 2: Implement a Mechanism Allowing the Subscriber To Control Goroutines	Resolved
Suggestion 3: Return nil Instead of &TeleporterRelayer{}	Resolved
Suggestion 4: Clear Private Keys From Memory	Partially Resolved
Suggestion 5: Prevent Ineffective Use of require Statement	Resolved
Suggestion 6: Define the Visibility of State Variables Explicitly	Resolved
Suggestion 7: Follow NatSpec Format	Unresolved
Suggestion 8: Follow Solidity Style Guide	Unresolved
Suggestion 9: Implement Gas Optimizations	Partially Resolved
Suggestion 10: Add Emergency Shutdown Feature in Bridge Smart Contracts	Unresolved
Suggestion 11: Implement the Upgrade Mechanism for the Bridge Token	Unresolved
Suggestion 12: Improve Test Coverage for Relayer	Unresolved
Suggestion 13: Restrict Payload Size of Warp Precompile	Unresolved

Issue A: Unbalanced Activities Between Two Chains Can Grow the Queue of Outstanding Receipts and Reduce Relayer Incentive

Synopsis

Given two subnets A and B, it is possible for messages to be sent only in one direction from the origin subnet A to the destination subnet B (A → B). This may result in subnet B being unable to send back enough receipts for subnet A. In this case, relayers have to send no-op messages back to subnet A to be able to receive their fee rewards. However, in every attempt, a maximum of five receipts can be sent from the destination subnet to the origin subnet in a batch. This would reduce the relayers' incentive to send no-op messages because their receipts may be at the end of the queue.

Relayers may become unwilling to spend gas on no-op messages that will not unlock their rewards. Even if a relayer has receipts at the start of the queue, they may not have enough incentive to spend gas on a no-op message that unlocks the reward for only one receipt.

Impact

In general, this Issue may reduce the relayers' incentive to send messages from the source chain to the destination chain unless they are offered fees high enough to account for the reward being locked for a prolonged period.

Activities occurring from the source chain to the destination chain may be obstructed due to a large number of outstanding receipts being queued on the destination chain since relayers know that their rewards will not be easily redeemable if they relay messages to a crowded destination subnet. This could also result in relayer fees becoming considerably high.

Preconditions

This Issue is possible in pairs of subnets where messages are sent in one direction most of the time.

Mitigation

We recommend giving the relayer the option to send back their own receipts with a new opcode. We also recommend exploring solutions other than a queue for recording outstanding receipts.

Status

The Ava Labs team has allowed relayers to optionally send back a batch, consisting of a maximum of five receipts, to the source chain using the `retryReceipts` function. However, the team has not removed the related receipts from the outstanding receipts due to the additional complexity and gas costs incurred when tracking the receipts that are in the queue and removing them from arbitrary positions from within the queue. Consequently, the congestion on the queue would still occur.

Verification

Partially Resolved.

Issue B: Duplicate Receipts Can Be Re-Enqueued in the retryReceipt Function

Location

`contracts/src/Teleporter/TeleporterMessenger.sol#L598-L622`

Synopsis

The `retryReceipt` function can add duplicate receipts to the list of outstanding receipts queue, unnecessarily growing the size of the queue with duplicate values. Since adding duplicate receipts to the queue is a relatively inexpensive process, an attacker could try to spam the queue with duplicate values, making it more difficult to send receipts to the origin chain.

Impact

Relayer rewards may not be processed in a timely manner, reducing their incentive to relay messages to the particular chain.

Preconditions

This Issue is possible if the same receipts are added to the queue.

Mitigation

We recommend checking if a particular receipt is already in the queue by adding an appropriate function in the `ReceiptQueue` smart contract. We also recommend exploring solutions other than a queue for recording outstanding receipts.

Status

The Ava Labs team has updated the mechanism such that at present, for a list of receipts, a no-op teleporter message is sent back to the origin chain, removing the necessity for re-enqueueing dropped receipts.

Verification

Resolved.

Issue C: Missing Zero Address Check

Location

contracts/src/CrossChainApplications/ERC20Bridge/ERC20Bridge.sol#L286

contracts/src/CrossChainApplications/ERC20Bridge/ERC20Bridge.sol#L471

contracts/src/CrossChainApplications/ERC20Bridge/ERC20Bridge.sol#L502

contracts/src/CrossChainApplications/ERC20Bridge/ERC20Bridge.sol#L284

contracts/src/CrossChainApplications/ERC20Bridge/ERC20Bridge.sol#L500

Synopsis

There is no zero address check validating the correctness of security-critical referenced address inputs passed to external functions, and thus preventing the loss of funds.

Impact

Passing a zero address to the aforementioned smart contract functions, without any input validation, might result in the loss of funds sent to addresses that can be set as zero addresses.

Remediation

We recommend checking the referenced parameters against zero address.

Status

The Ava Labs team has added zero address checks as suggested.

Verification

Resolved.

Issue D: MinerTip Is Not Included in the Calculation of MaxFeeCap

Location

vms/evm/destination_client.go#L104

Synopsis

`MaxFeeCap` is calculated from the `baseFee` and the `minerTip`. However, while constructing the transaction, it only includes the `baseFee`, due to which the `effectiveTip` will always be zero.

Impact

During heavy congestion, the transaction with a zero `effectiveTip` will take longer to be confirmed because the validators will prioritize the transaction that has the highest effective tip.

Technical Details

The `GasTipCap` is commented while constructing the transaction, so its value will be zero by default. Note that: `EffectiveTip = Min(MaxFeeCap - BaseFee, GasTipCap)`. Therefore, since `GasTipCap` is always zero, the `effectiveTip` will be zero.

Remediation

We recommend setting the `GasTip` value while constructing the transaction to allow miners to earn an effective tip. We recommend referring to ether.js and web3.py for appropriate values (1-2.5 gwei).

Status

The Ava Labs team has allowed `GasTtiIpCap` value to be added when the transaction is constructed, due to which the `effectiveTip` will not be zero by default. However, `MaxFeeCap` still does not include the `minerTip` in the calculation.

Verification

Partially Resolved.

Issue E: ERC20 Token Supporting "Fee On Transfer" Can Lead To Error in Accounting of Relayer Reward

Location

`contracts/src/CrosschainApplication/ERC20Bridge/ERC20Bridge.sol#l359`

Synopsis

In the case of an ERC20 token that features "fee on transfer," or the deflationary token "burn on transfer," the amount received in a transfer would be less than the amount sent.

Impact

The bridge smart contracts will receive an amount that is less than the expected amount, which will affect the reward calculation for the relayer. Since the bridge smart contracts do not approve the actual amount received by the user, this could result in the case where a relayer would be unable to redeem their full reward.

Preconditions

This Issue is possible if bridged tokens support "burn on transfer" or "fee on transfer" functionality.

Remediation

In order to obtain the actual amount received by the contract, we recommend tracking the balance of tokens before and after the transfer of tokens. For example, in the contract test, we recommend implementing the following steps:

```
   function _transfer(uint256 amount) public returns(uint256){

          uint256 balanceBefore = IERC20(token).balanceOf(address(this));

          IERC20Token(token).SafetransferFrom(msg.sender, address(this),
amount);

          uint256 balanceAfter = IERC20(token).balanceOf(address(this));
```

```
        require(balanceAfter >= balanceBefore);

        return  balanceAfter - balanceBefore;
}
```

Status

The Ava Labs team has implemented the method supporting “burn on transfer” or “fee on transfer” functionality.

Verification

Resolved.

Issue F: submitCreateBridgeToken Will Revert for the Tokens That Do Not Implement the Optional Function of the ERC20 Token Standard

Location

`contracts/src/CrossChainApplications/ERC20Bridge/ERC20Bridge.sol#L393`

Synopsis

The functions `ERC20(_token).decimals`, `ERC20(_token).name`, and `ERC20(_token).symbol` are optional functions in the EIP20 standard. The token can still be ERC20 compliant without the implementation of these three functions (e.g. Maker).

Impact

When a user creates a new bridge token through `submitCreateBridgeToken`, if the native token does not implement the optional ERC20 function, the function call will revert.

Remediation

At the contract level, the token can be checked through the `staticCall` function, as follows:

```
(bool result, bytes memory decimal) =
token.staticcall(abi.encodeWithSIgnature(“decimals()”))
```

We recommend that the Ava Labs team verify the Boolean value and check the status of the call.. If the call is successful, the return data must be decoded.

Status

The Ava Labs team stated that they do not plan on supporting tokens that do not implement names, symbols, and decimals and prefer not to introduce additional complexity in checks for minor gas optimizations.

Verification

Unresolved.

Suggestions

Suggestion 1: Check That Type Assertion Succeeds

Location

`teleporter_relayer/relayer.go`

Synopsis

The code in the function `isValidSignatureResponse` does not check that type assertion holds. In the case that type assertion is `false`, a run-time panic can occur.

Mitigation

We recommend that a check be added to verify that type assertion holds.

Status

The Ava Labs team has implemented the suggested mitigation.

Verification

Resolved.

Suggestion 2: Implement a Mechanism Allowing the Subscriber To Control Goroutines

Location

`vms/evm/subscriber.go#L70`

`vms/evm/subscriber.go#L129`

Synopsis

The `subscriber` type uses goroutines ineffectively, which could lead to goroutine leakage or blocking.

For example:

1. `for loop` is used without a `select` mechanism. If the `evmLog` channel is closed, and a subscriber goroutine is running, the goroutine may be blocked;
2. A message is written to a `log` channel without a `select` mechanism. If the receiving goroutine cannot read from the channel, the goroutine will be blocked;
3. A goroutine is created without a description that explicitly specifies when and how it is going to be stopped; and
4. in the `subscriber` type, the context package or stop channels are not used.

Mitigation

We recommend implementing a mechanism that allows users to stop the created goroutines and to be protected against blocking.

Status

The Ava Labs team has implemented the standard “range-close” mitigation as recommended.

Verification

Resolved.

Suggestion 3: Return nil Instead of &TeleporterRelayer{}

Location

teleporter_relayer/relayer.go#L73

teleporter_relayer/relayer.go#L87

teleporter_relayer/relayer.go#L113

Synopsis

`&TeleporterRelayer{}` is returned instead of `nil`. This practice is not idiomatic construction, and it is not consistent with the pointer semantic and error handling paradigm in Go.

Mitigation

We recommend returning `nil` instead of `&TeleporterRelayer{}`.

Status

The Ava Labs team has implemented the suggested mitigation.

Verification

Resolved.

Suggestion 4: Clear Private Keys From Memory

Location

vms/platformvm/warp/signer.go#L36

vms/evm/destination_client.go#L32

config/config.go#L47

Synopsis

Private keys are not cleared from memory after usage.

Mitigation

Private keys should be cleared from memory as long as the Go mechanism allows it. We recommend using the `SetFinalizer` mechanism in Go to clear the memory, despite the fact that it does not guarantee that the secret values will be removed effectively from the memory.

Status

The Ava Labs team has started clearing private keys from memory after usage.

Verification

Partially Resolved.

Suggestion 5: Prevent Ineffective Use of require Statement

Location

`contracts/src/Teleporter/ReceiptQueue.sol#L23`

Synopsis

Any state variable defined in solidity can be read using the `getStorageAt` function, even if it is declared `private` or `internal`. Therefore, the `require` statement referenced above is unnecessary since anyone can calculate the size of the queue.

Mitigation

We recommend removing the `require` statement to save gas.

Status

The Ava Labs team has removed the `require` statement.

Verification

Resolved.

Suggestion 6: Define the Visibility of State Variables Explicitly

Location

Example (Non-Exhaustive):

`contracts/src/Teleporter/ReceiptQueue.sol#L13-L16`

`contracts/src/CrossChainApplications/ERC20Bridge/ERC20Bridge.sol#L44`

`contracts/src/Teleporter/TeleporterMessenger.sol#L28-L56`

Synopsis

State variables defined in the contracts do not explicitly mark the visibility. It is considered best practice to explicitly mark the visibility of state variables and functions to prevent incorrect assumptions about who can call the function and access the variable.

Mitigation

We recommend setting the visibility of state variables explicitly to improve readability.

Status

The Ava Labs team has explicitly defined the visibility of state variables.

Verification

Resolved.

Suggestion 7: Follow NatSpec Format

Synopsis

Although the smart contract inline documentation is comprehensive and detailed, it does not follow a specific format, which inhibits code maintenance and readability. The NatSpec format is the industry standard for Solidity.

Mitigation

We recommend adhering to the NatSpec guidelines specified in Solidity's documentation.

Status

The Ava Labs team stated that the suggested mitigation has been added to their backlog. Hence, this suggestion remains unresolved at the time of verification.

Verification

Unresolved.

Suggestion 8: Follow Solidity Style Guide

Synopsis

The order of functions in the smart contracts is not in accordance with Solidity's style guide, which inhibits the readability of the smart contracts.

Mitigation

We recommend adhering to the Solidity style guide in the development of Solidity smart contracts.

Status

The Ava Labs team stated that the suggested mitigation has been added to their backlog. Hence, this suggestion remains unresolved at the time of verification.

Verification

Unresolved.

Suggestion 9: Implement Gas Optimizations

Location

Use custom errors (non-exhaustive examples):

`contracts/src/Teleporter/ReceiptQueue.sol#L28`

`contracts/src/Teleporter/ReceiptQueue.sol#L37`

`contracts/src/Teleporter/ReceiptQueue.sol#L38`

`contracts/src/Teleporter/TeleporterMessenger.sol#L144`

`contracts/src/Teleporter/TeleporterMessenger.sol#L218`

Cache the array length outside a loop:

contracts/src/Teleporter/TeleporterMessenger.sol#L512

Consider using the prefix increment expression if the return value is not needed:

contracts/src/Teleporter/TeleporterMessenger.sol#L127

contracts/src/Teleporter/TeleporterMessenger.sol#L335

contracts/src/Teleporter/TeleporterMessenger.sol#L512

Declare constructors as payable (non-exhaustive examples):

contracts/src/Teleporter/TeleporterMessenger.sol#L71

contracts/src/Teleporter/ReceiptQueue.sol#L18

contracts/src/CrossChainApplications/ERC20Bridge/BridgeToken.sol#L15

contracts/src/CrossChainApplications/ERC20Bridge/ERC20Bridge.sol#L51

Remove redundant checks:

contracts/src/CrossChainApplications/ERC20Bridge/ERC20Bridge.sol#L74-L78

contracts/src/CrossChainApplications/ERC20Bridge/ERC20Bridge.sol#L179-L184

Consider using nested statements that is more gas-efficient:

contracts/src/CrossChainApplications/ERC20Bridge/ERC20Bridge.sol#L523

Synopsis

There are many gas-related inefficiencies throughout the codebase where improvements can be made to decrease gas consumption.

Mitigation

We recommend performing the aforementioned changes while considering the optimal balance between code readability and gas optimization.

Status

Gas optimization is always a tradeoff between readability, simplicity, and efficiency.

The Ava Labs team has implemented major improvements to decrease gas consumption by using postfix increment expressions, avoiding nested `IF` statements, etc. However, custom errors have not been implemented.

Verification

Partially Resolved.

Suggestion 10: Add Emergency Shutdown Feature in Bridge Smart Contracts

Synopsis

Currently, the bridge smart contracts do not support emergency shutdown. As a result, if a vulnerability is discovered, the system remains compromised and attacks exploiting the vulnerability (for example, to drain the bridge assets) cannot be stopped until an upgrade is performed successfully.

Mitigation

An emergency shutdown feature can provide upgrade safety during the deployment of the smart contracts, in case of a hack.

A quorum of the relayers can be used in order to vote for the activation of an emergency shutdown. This will allow the relayers to determine whether the current issue is a threat to the bridge smart contracts, and pause it for a temporary period, until an upgrade is completed. This mitigation may cause a delay in activation.

The mitigation can also be achieved through a Multi-Sig contract, which has the ability to activate the pausable function in the contract.

Status

The Ava Labs team stated that the suggested mitigation has been added to their backlog. Hence, this suggestion remains unresolved at the time of verification.

Verification

Unresolved.

Suggestion 11: Implement the Upgrade Mechanism for the Bridge Token

Synopsis

The upgrade mechanism provides the ability to fix a bug or a vulnerability in case of a hack. In addition, the mechanism makes it possible to add new functionality to – and remove functionality from – an existing contract. The Avalanche Bridge Token does not implement this upgradable feature.

Mitigation

We recommend implementing a proxy pattern, such as Universal Upgradable Proxy Pattern, Transparent Proxy, etc.

Status

The Ava Labs team stated that the suggested mitigation has been added to their backlog. Hence, this suggestion remains unresolved at the time of verification.

Verification

Unresolved.

Suggestion 12: Improve Test Coverage for Relayer

Synopsis

There is no test coverage for the `relayer.go` file, and very limited coverage for the `utils` (20%) and the `config` (13.5%) files. In addition, our team was unable to run the `contract_message` tests. A robust

test suite helps verify that components are implemented correctly, identifies errors and unintended behavior, and aids in reasoning about the security characteristics of the system.

Mitigation

We recommend further increasing test coverage such that it includes all parts of the codebase.

Status

The Ava Labs team stated that the suggested mitigation has been added to their backlog. Hence, this suggestion remains unresolved at the time of verification.

Verification

Unresolved.

Suggestion 13: Restrict Payload Size of Warp Precompile

Synopsis

Since there is no restriction on the payload in the messages passed to a Warp precompile to be signed and sent, the gas limit could be exceeded and the transaction could be reverted, without an explicit error being emitted.

Mitigation

We recommend adding a check to limit the payload in the messages passed to the Warp precompile.

Status

Our team was unable to confirm that the suggested mitigation has been resolved as of the time of the verification.

Verification

Unresolved.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, zero-knowledge protocols, and consensus protocols. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. We are an international team that believes we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. As we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even before having verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate and comprehensive mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.