# # Competitive Security Assessment

## Nativ3 Network

Oct 19th, 2023

Secure3

secure3.io

# Summary

Nativ3 is a cutting-edge Layer3 blockchain ecosystem, built on the robust Arbitrum network. Offering unparalleled scalability, low fees, and enhanced user experience, Nativ3 stands at the forefront of blockchain innovation. We invite you to join us in building a more open, fair, and efficient digital world.

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

# Overview

### Project Detail

| | |
|---|---|
| **Project Name** | Nativ3 Network |
| **Platform & Language** | Solidity |
| **Codebase** | • https://github.com/Nativ3DAO/contracts-for-audit<br>• audit commit - 3e1f3cce90a8356aa6a80360dbb72255ee4596a5<br>• final commit - b6b8a600ac17a3dcbb4285cc384574ac9a7506b8 |
| **Audit Methodology** | • Audit Contest<br>• Business Logic and Code Review<br>• Privileged Roles Review<br>• Static Analysis |

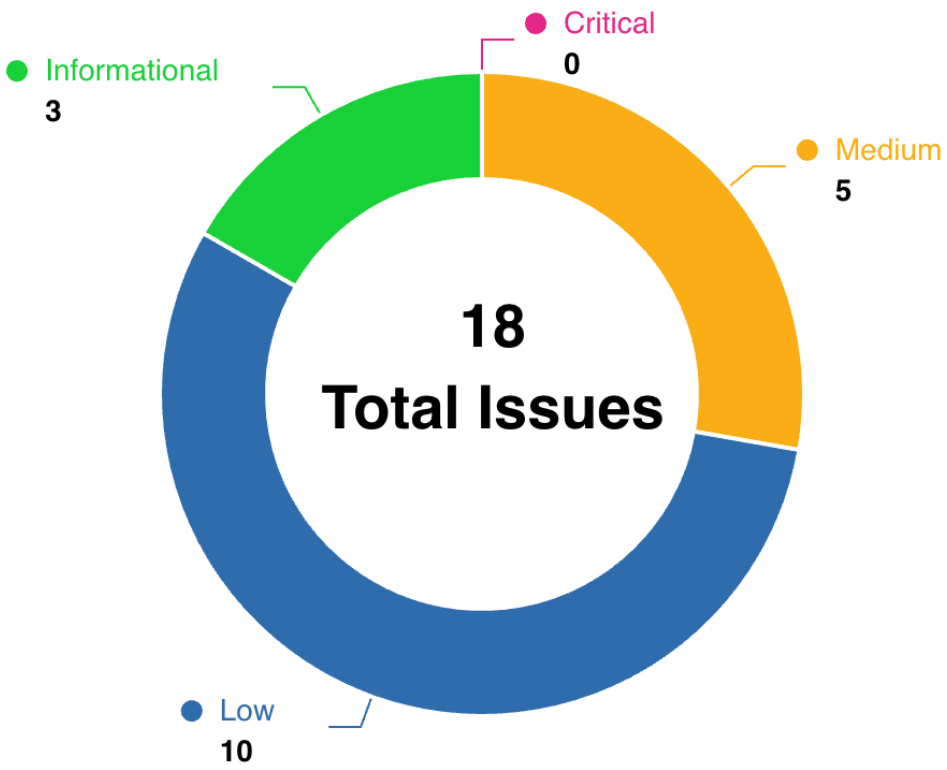
### Code Vulnerability Review Summary

| Vulnerability Level | Total | Reported | Acknowledged | Fixed | Mitigated | Declined |
|---|---|---|---|---|---|---|
| Critical | 0 | 0 | 0 | 0 | 0 | 0 |
| Medium | 5 | 0 | 2 | 2 | 0 | 1 |
| Low | 10 | 0 | 6 | 1 | 0 | 3 |
| Informational | 3 | 0 | 2 | 1 | 0 | 0 |

# Audit Scope

| File | SHA256 Hash |
| --- | --- |
| contracts/bridge/SequencerInbox.sol | f40998db63a5ebef0a5b19b8387e02efe1cf981e67f854a4aa0ab9c3633ee7a5 |
| contracts/bridge/Bridge.sol | 7c251b7acebacc899762f9ff46442d0e0d6996734ee4e83a6a81a698ae406840 |
| contracts/bridge/Inbox.sol | 743ccf4f7442993b84a9da61094eb8db6ca65a4e36ce9de8c0f4aea520ac0e44 |
| contracts/bridge/Outbox.sol | 501c556005c2192a7937a3b3db04daf1ec4aaca388b4c969d30477196e0ab4ed |
| contracts/bridge/GasToken.sol | 49f3812c0c6529146b3d21812f4bbd5b6cb1579becb82892fa17f377d96adc16 |

# Code Assessment Findings




| ID | Name | Category | Severity | Client Response | Contributor |
|---|---|---|---|---|---|
| NAD-1 | onlyAllowed is not rigorous in its judgment | Logical | Medium | Acknowledged | 8olidity |
| NAD-2 | Lack of check the timestamp from the price oracle and Use of deprecated Chainlink function `lastestAnswer()` | Logical | Medium | Fixed | biakia, betharavikiran, wzrdk3lly, TrungOre, 8olidity, n16h7m4r3 |

| | | | | | |
|---|---|---|---|---|---|
| NAD-3 | Potential `DataTooLarge` Error | Logical | Medium | Fixed | biakia |
| NAD-4 | DOS can occur when claiming refund for Smart Contract addresses | DOS | Medium | Acknowledged | wzrdk3lly |
| NAD-5 | Incorrect USD value of `baseFeeL1` when interacting with function `Bridge.addMessageToDelayedAccumulator()` | Oracle Manipulation | Medium | Declined | TrungOre |
| NAD-6 | SequencerInbox::MaxTimeVariation lacks the validations to prevent truncation issues | Integer Overflow and Underflow | Low | Acknowledged | betharavikiran |
| NAD-7 | PriceFeed address not updated | Logical | Low | Fixed | 8olidity, n16h7m4r3 |
| NAD-8 | Invalid estimatation for the `RetryableSubmissionFee` | Logical | Low | Acknowledged | betharavikiran, biakia |
| NAD-9 | Block production may not be constant | Language Specific | Low | Acknowledged | 8olidity |
| NAD-10 | BatchSpendingReport can be generated by anyone using delayed message queue | Privilege Related | Low | Acknowledged | betharavikiran |
| NAD-11 | Cross-chain calls will not be executed as expected when `msg.value` is greater than 0 | Logical | Low | Acknowledged | biakia |
| NAD-12 | Unable to calling `Bridge.executeCall()` with attached ETH | Logical | Low | Acknowledged | TrungOre, n16h7m4r3 |
| NAD-13 | Return value for message added to Delay accumulator is stale | Logical | Low | Declined | betharavikiran |
| NAD-14 | Missing zero address check. | Code Style | Low | Declined | helookslikeme, n16h7m4r3 |
| NAD-15 | Centralized authority risk risk in `GasToken::mint()and burn` function | Privilege Related | Low | Declined | helookslikeme |

| | | | | | |
|---|---|---|---|---|---|
| **NAD-16** | **Redundant Code** | **Logical** | **Informational** | **Acknowledged** | **8olidity, biakia, betharavikiran, n16h7m4r3** |
| **NAD-17** | **Outbox ::l2ToL1Block function compares the context's l2Block against wrong default value.** | **Logical** | **Informational** | **Fixed** | **betharavikiran** |
| **NAD-18** | **SequencerInbox::event Offchain notification for OwnerFunction does not express the underlying change** | **Language Specific** | **Informational** | **Acknowledged** | **betharavikiran** |

# NAD-1:onlyAllowed is not rigorous in its judgment

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Medium | Acknowledged | 8olidity |

## Code Reference

- code/contracts/bridge/Inbox.sol#L58-L62

```
58:modifier onlyAllowed() {
59:         // solhint-disable-next-line avoid-tx-origin
60:         if (allowListEnabled && !isAllowed[tx.origin]) revert NotAllowedOrigin(tx.origin);
61:         _;
62:     }
```

## Description

**8olidity :** In onlyAllowed, it is allowed to add the tx.origin address as a trusted address, but if the user is using an AA wallet (ERC4337 Account Abstraction), the contract address is added in the isAllowed array.

The first situation:

The onlyAllowed function can pass normally, but the sendL2MessageFromOrigin functions will fail to execute because of the following judgments:

```
if (msg.sender != tx.origin) revert NotOrigin();
```

Second case:

The _deliverMessage function below passes in msg.sender, but onlyAllowed checks tx.origin.onlyAllowed. The check is bypassed, but the two addresses are different, causing confusion to the system.

```
function sendL2Message(
    bytes calldata messageData
) external whenNotPaused onlyAllowed returns (uint256) {
    if (_chainIdChanged()) revert L1Forked();
    return _deliverMessage(L2_MSG, msg.sender, messageData);
}
```

## Recommendation

**8olidity :** It is recommended to determine whether the allowed address is a contract address

## Client Response

Acknowledged,this modifier is not intended to use to be used for security.

# NAD-2:Lack of check the timestamp from the price oracle and Use of deprecated Chainlink function `lastestAnswer()`

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Medium | Fixed | biakia, betharavikiran, wzrdk3lly, TrungOre, 8olidity, n16h7m4r3 |

## Code Reference

- code/contracts/bridge/Bridge.sol#L306-L312
- code/contracts/bridge/Bridge.sol#L306-L311
- code/contracts/bridge/Bridge.sol#L309

```
306:function scaleBaseFeeL1(uint256 fee) internal view returns (uint256) {
307:        //Arbiturm Goerli price feed
308:        IPriceFeed _priceFeed = IPriceFeed(address(0x62CAe0FA2da220f43a51F86Db2EDb36DcA9A5A08));
309:        int256 price = _priceFeed.latestAnswer();
310:        require(price > 0, "Error: Invalid price");
311:        return (fee * uint256(price)) / (10 ** _priceFeed.decimals());
312:    }

306:function scaleBaseFeeL1(uint256 fee) internal view returns (uint256) {
307:        //Arbiturm Goerli price feed
308:        IPriceFeed _priceFeed = IPriceFeed(address(0x62CAe0FA2da220f43a51F86Db2EDb36DcA9A5A08));
309:        int256 price = _priceFeed.latestAnswer();
310:        require(price > 0, "Error: Invalid price");
311:        return (fee * uint256(price)) / (10 ** _priceFeed.decimals());

309:int256 price = _priceFeed.latestAnswer();
```

## Description

**biakia :** In contract `Bridge`, the function `scaleBaseFeeL1` will use `IPriceFeed` to fetch latest price:

```
function scaleBaseFeeL1(uint256 fee) internal view returns (uint256) {
        //Arbiturm Goerli price feed
        IPriceFeed _priceFeed = IPriceFeed(address(0x62CAe0FA2da220f43a51F86Db2EDb36DcA9A5A08));
        int256 price = _priceFeed.latestAnswer();
        require(price > 0, "Error: Invalid price");
        return (fee * uint256(price)) / (10 ** _priceFeed.decimals());
    }
```

The contract of the `IPriceFeed` is `0x62CAe0FA2da220f43a51F86Db2EDb36DcA9A5A08`. In Arbiturm Goerli, we can find the code of `EACAggregatorProxy.sol` (https://goerli.arbiscan.io/address/0x62CAe0FA2da220f43a51F86Db2EDb36DcA9A5A08#code)

This is an implementation of the Chainlink's `AggregatorV3Interface`. The issue here is that there is no check for the last updated time for the price. So we would not know if the priced returned exceeded the timeout. It may return an expired price and incur unexpected side effects.

**betharavikiran :** scaleBaseFeeL1() function in the bridge scales the fee using the price feed from Chain-link. The implementation is wrong for two reasons.

1. **Chain link feed address is wrong:** The price feed address is pointed to `Arbiturm Goerli`. When the bridge contract is deployed to Mainnet, the target address will be different. Please refer to the below price feeds list. The current value is for test network.
   https://docs.chain.link/data-feeds/price-feeds/addresses?network=arbitrum&page=1
2. **using deprecated function:** According to Chainlink's documentation, the latest Answer function is deprecated. This function does not error if no answer has been reached but returns 0.
   **wzrdk3lly :** In `Bridge.sol`, there isn't sufficient validation checks to ensure that the chainlink price feed doesn't return stale data. The address `0x62CAe0FA2da220f43a51F86Db2EDb36DcA9A5A08` corresponds to a chainlink oracle price feed. The use of the deprecated `latestAnswer()` function is highly discouraged due to its ability to return stale data without any means of detecting the data's staleness.

The impact of a stale `price` is that it will cause the fee calculation to be incorrect.

**TrungOre :** According to Chainlink's documentation, the `latestAnswer()` function is deprecated. This function does not error if no answer has been reached but returns 0, causing an incorrect price fed to the `Bridge.sol` contract.

The impact is that, if the deprecated API stops working, prices cannot be obtained, the protocol stops and contracts have to be redeployed.

**8olidity :** In this function, uses the `latestAnswer` of chainlink but this function is deprecated.

```
    function scaleBaseFeeL1(uint256 fee) internal view returns (uint256) {
        //Arbiturm Goerli price feed
        IPriceFeed _priceFeed = IPriceFeed(address(0x62CAe0FA2da220f43a51F86Db2EDb36DcA9A5A08));
        int256 price = _priceFeed.latestAnswer();
        require(price > 0, "Error: Invalid price");
        return (fee * uint256(price)) / (10 ** _priceFeed.decimals());
    }
```

`scaleBaseFeeL1` uses Chainlink's deprecated API latestAnswer(). Such functions might suddenly stop working if Chainlink stopped supporting deprecated APIs.

Impact: Deprecated API stops working. Prices cannot be obtained. Protocol stops and contracts have to be redeployed.

**n16h7m4r3 :** According to Chainlink's documentation, the `latestAnswer()` function is deprecated. This function does not error if no answer has been reached but returns 0. Besides, the `latestAnswer()` is reported with 18 decimals for crypto quotes but 8 decimals for FX quotes (See Chainlink FAQ for more details). A best practice is to get the decimals from the oracles instead of hard-coding them in the contract.

## Recommendation

**biakia :** Since the `latestAnswer` is deprecated, we recommend using the `latestRoundData` instead and checking the `updateAt`:

```
IPriceFeed _priceFeed = IPriceFeed(address(0x62CAe0FA2da220f43a51F86Db2EDb36DcA9A5A08));
(,int256 price,,uint256 updatedAt,) = _priceFeed.latestRoundData();
 require(price > 0, "Error: Invalid price");
 require(updatedAt > block.timestamp - MAX_TIME_DELAY, "Error: Invalid updated time");
return (fee * uint256(price)) / (10 ** _priceFeed.decimals());
```

**betharavikiran :** 1. Add an admin controlled set function to assign the correct address for price feed. for mainnet, the address should be as below.

`0x639Fe6ab55C921f74e7fac1ee960C0B6293ba612`

2. Use the latestRoundData function to get the price instead. Add checks on the return data with proper revert messages if the price is stale or the round is uncomplete, for example:

```
(uint80 roundID, int256 price, , uint256 timeStamp, uint80 answeredInRound) = oracle.latestRoundData
();
require(answeredInRound >= roundID, "...");
require(timeStamp != 0, "...");
```

**wzrdk3lly :** Use the latestRoundData function instead of the deprecated `latestAnswer`

```
function latestRoundData() external view
    returns (
        uint80 roundId,
        int256 answer,
        uint256 startedAt,
        uint256 updatedAt,
        uint80 answeredInRound
    )
```

Furthermore you can use the round data to ensure the oracle is not returning stale data. See snippet

```
require(answeredInRound >= roundId, "answer is stale");
require(updatedAt > 0, "round is incomplete");
```

**TrungOre :** Use the latestRoundData function to get the price instead. Add checks on the return data with proper revert

messages if the price is stale or the round is uncomplete, for example:

```
(uint80 roundID, int256 price, , uint256 timeStamp, uint80 answeredInRound) = oracle.latestRoundData
();
require(answeredInRound >= roundID, "...");
require(timeStamp != 0, "...");
```

**8olidity :** Use V3 interface functions: https://docs.chain.link/docs/price-feeds-api-reference/
**n16h7m4r3 :** Use the recommended `latestRoundData()` function to get the price instead. Add checks on the return data with proper revert messages if the price is stale or the round is uncomplete.

## Client Response

Fixed

# NAD-3:Potential `DataTooLarge` Error

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Medium | Fixed | biakia |

## Code Reference

- code/contracts/bridge/SequencerInbox.sol#L305-L319

```
305:modifier validateBatchData(bytes calldata data) {
306:        uint256 fullDataLen = HEADER_LENGTH + data.length;
307:        if (fullDataLen > MAX_DATA_SIZE) revert DataTooLarge(fullDataLen, MAX_DATA_SIZE);
308:        if (data.length > 0 && (data[0] & DATA_AUTHENTICATED_FLAG) == DATA_AUTHENTICATED_FLAG) {
309:            revert DataNotAuthenticated();
310:        }
311:        // the first byte is used to identify the type of batch data
312:        // das batches expect to have the type byte set, followed by the keyset (so they should
have at least 33 bytes)
313:        if (data.length >= 33 && data[0] & 0x80 != 0) {
314:            // we skip the first byte, then read the next 32 bytes for the keyset
315:            bytes32 dasKeysetHash = bytes32(data[1:33]);
316:            if (!dasKeySetInfo[dasKeysetHash].isValidKeyset) revert NoSuchKeyset(dasKeysetHash);
317:        }
318:        _;
319:    }
```

## Description

**biakia :** In contract `SequencerInbox`, the modifier `validateBatchData` will check the data:

```
modifier validateBatchData(bytes calldata data) {
        uint256 fullDataLen = HEADER_LENGTH + data.length;
        if (fullDataLen > MAX_DATA_SIZE) revert DataTooLarge(fullDataLen, MAX_DATA_SIZE);
        if (data.length > 0 && (data[0] & DATA_AUTHENTICATED_FLAG) == DATA_AUTHENTICATED_FLAG) {
            revert DataNotAuthenticated();
        }
        // the first byte is used to identify the type of batch data
        // das batches expect to have the type byte set, followed by the keyset (so they should have
at least 33 bytes)
        if (data.length >= 33 && data[0] & 0x80 != 0) {
            // we skip the first byte, then read the next 32 bytes for the keyset
            bytes32 dasKeysetHash = bytes32(data[1:33]);
            if (!dasKeySetInfo[dasKeysetHash].isValidKeyset) revert NoSuchKeyset(dasKeysetHash);
        }
        _;
    }
```

The `HEADER_LENGTH` is 40 and the `MAX_DATA_SIZE` is 117964. Let's say we have a big batch, its length is `MAX_DATA_SIZE-HEADER_LENGTH = 117924`. All the check in the above code will pass. This batch will be submitted to the Arbitrum(As we know the Nativ Network is based on Arbitrum). When this batch is included in the Arbitrum's batch and sumbitted to the Ethereum, the same code will be checked in Arbitrum(https://github.com/OffchainLabs/nitro-contracts/blob/main/src/bridge/SequencerInbox.sol#L305-L319). At this time, the `data.length` will be 117964(Nativ's data + header). The `fullDataLen` will be 118004, which is greater than `MAX_DATA_SIZE`. At last, an `DataTooLarge` error will occur in Arbitrum Network.

## Recommendation

**biakia :** Consider reducing `MAX_DATA_SIZE` to a reasonable value.

## Client Response

Fixed

# NAD-4:DOS can occur when claiming refund for Smart Contract addresses

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| DOS | Medium | Acknowledged | wzrdk3lly |

## Code Reference

- code/contracts/bridge/Inbox.sol#L154-L195
- code/contracts/bridge/Bridge.sol#L95-L122
- code/contracts/bridge/Inbox.sol#L175-L181

```
95:function enqueueSequencerMessage(
96:         bytes32 dataHash,
97:         uint256 afterDelayedMessagesRead,
98:         uint256 prevMessageCount,
99:         uint256 newMessageCount
100:    )
101:         external
102:         onlySequencerInbox
103:         returns (uint256 seqMessageIndex, bytes32 beforeAcc, bytes32 delayedAcc, bytes32 acc)
104:    {
105:         if (
106:             sequencerReportedSubMessageCount != prevMessageCount &&
107:             prevMessageCount != 0 &&
108:             sequencerReportedSubMessageCount != 0
109:         ) {
110:             revert BadSequencerMessageNumber(sequencerReportedSubMessageCount, prevMessageCoun
t);
111:         }
112:         sequencerReportedSubMessageCount = newMessageCount;
113:         seqMessageIndex = sequencerInboxAccs.length;
114:         if (sequencerInboxAccs.length > 0) {
115:             beforeAcc = sequencerInboxAccs[sequencerInboxAccs.length - 1];
116:         }
117:         if (afterDelayedMessagesRead > 0) {
118:             delayedAcc = delayedInboxAccs[afterDelayedMessagesRead - 1];
119:         }
120:         acc = keccak256(abi.encodePacked(beforeAcc, dataHash, delayedAcc));
121:         sequencerInboxAccs.push(acc);
122:    }

154:function createRetryableTicket(
155:         address to,
156:         uint256 l2CallValue,
157:         uint256 maxSubmissionCost,
158:         address excessFeeRefundAddress,
159:         address callValueRefundAddress,
160:         uint256 gasLimit,
161:         uint256 maxFeePerGas,
162:         bytes calldata data
163:    ) external payable whenNotPaused onlyAllowed returns (uint256) {
164:         // ensure the user's deposit alone will make submission succeed
```

```
165:         if (msg.value < (maxSubmissionCost + l2CallValue + gasLimit * maxFeePerGas)) {
166:             revert InsufficientValue(
167:                 maxSubmissionCost + l2CallValue + gasLimit * maxFeePerGas,
168:                 msg.value
169:             );
170:         }
171:
172:         // if a refund address is a contract, we apply the alias to it
173:         // so that it can access its funds on the L2
174:         // since the beneficiary and other refund addresses don't get rewritten by arb-os
175:         if (AddressUpgradeable.isContract(excessFeeRefundAddress)) {
176:             excessFeeRefundAddress = AddressAliasHelper.applyL1ToL2Alias(excessFeeRefundAddres
s);
177:         }
178:         if (AddressUpgradeable.isContract(callValueRefundAddress)) {
179:             // this is the beneficiary. be careful since this is the address that can cancel the
retryable in the L2
180:             callValueRefundAddress = AddressAliasHelper.applyL1ToL2Alias(callValueRefundAddres
s);
181:         }
182:
183:         // gas limit is validated to be within uint64 in unsafeCreateRetryableTicket
184:         return
185:             unsafeCreateRetryableTicket(
186:                 to,
187:                 l2CallValue,
188:                 maxSubmissionCost,
189:                 excessFeeRefundAddress,
190:                 callValueRefundAddress,
191:                 gasLimit,
192:                 maxFeePerGas,
193:                 data
194:             );
195:     }

175:if (AddressUpgradeable.isContract(excessFeeRefundAddress)) {
176:             excessFeeRefundAddress = AddressAliasHelper.applyL1ToL2Alias(excessFeeRefundAddres
s);
177:         }
178:         if (AddressUpgradeable.isContract(callValueRefundAddress)) {
179:             // this is the beneficiary. be careful since this is the address that can
```

```
cancel the retryable in the L2
180:             callValueRefundAddress = AddressAliasHelper.applyL1ToL2Alias(callValueRefundAddres
s);
181:         }
```

## Description

**wzrdk3lly :** The `createRetryableTicket()` function requires users to send tokens to the `Inbox.sol` contract to pay for the ticket. The users' funds are eventually transferred to `bridge.sol` when the nested `deliverToBridge()` function is called. However, the `createRetryableTicket()` function does not allow users to be refunded the access payment for this ticket directly after invocation.

It's worth noting that there is an `excessFeeRefundAddress`, but this appears to be the forwarding address for excess gas fees that are paid for message delivery and NOT a refund for the excess payment made to `createRetryableTicket()`.

**Impact**: The impact of this vulnerability is that users can lose their tokens when overpaying for the ticket creation because there's no mechanism for refunding excess payments.

**wzrdk3lly :** In the `inbox.sol` contract, users have the option to specify contract addresses for `excessFeeRefundAddress` and `callValueRefundAddress` when creating a ticket. However, this code lacks handling for scenarios where the user's specified contract address does not accept ETH. Contracts can be designed to accept or prevent incoming ETH, and when ETH is rejected by a contract, the associated transaction will fail.

**Impact**: This vulnerability can prevent users from successfully claiming or receiving their allocated refunds.

## Recommendation

**wzrdk3lly :** A better implementation would include checking if the user has sent an excess `msg.value` within `bridge.sol` and immediately returning the excess value back to the `msg.sender`.

The entrypoint of the vulnerability. Users send msg.value with the createRetryableTicket function call. The function that createRetryableTicket calls to deposit user's funds into bridge.sol

**wzrdk3lly :** To address this vulnerability, consider implementing one of the following approaches:

- Ensure that users specify a payable contract address when setting their refund addresses.
- Immediately return the refund back to `msg.sender` when processing or rejecting the ticket.

## Client Response

Acknowledged,Yes, we are aware of that. Normally, excess gas is transferred to the specified L3's wallet.

# NAD-5:Incorrect USD value of `baseFeeL1` when interacting with function `Bridge.addMessageToDelayedAccumulator()`

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Oracle Manipulation | Medium | Declined | TrungOre |

## Code Reference

- code/contracts/bridge/Bridge.sol#L167

```
167:baseFeeL1 = scaleBaseFeeL1(baseFeeL1);
```

## Description

**TrungOre :** The function `Bridge.submitBatchSpendingReport()` is used within the sequencer inbox to submit a delayed message of the `batchPostingReport` type. This involves executing a call to an internal function:

```
addMessageToDelayedAccumulator(
    L1MessageType_batchPostingReport,
    sender,
    uint64(block.number),
    uint64(block.timestamp), // solhint-disable-line not-rely-on-time,
    block.basefee,
    messageDataHash
);
```

The fifth parameter in this function input, `block.basefee`, plays a crucial role. In the internal function `Bridge.addMessageToDelayedAccumulator()`, the value of `block.baseFee` is converted into its equivalent USD value by making use of the `baseFeeL1 = scaleBaseFeeL1(baseFeeL1);` operation. This `scaleBaseFeeL1` function initiates an external call to the Chainlink oracle in order to obtain the accurate USD value of the `block.baseFee` amount.

The problem arises when the Chainlink oracle returns the USD value equivalent to 1 ETH (= 10^18 weiETH) instead of 1 weiETH, even though the `block.baseFee` is originally determined in weiETH. This issue leads to an incorrect value being assigned to the `baseFeeL1` variable, consequently disrupting the logic of the `Bridge` contract.

## Recommendation

**TrungOre :** Consider dividing the value of `block.baseFee` to 1e18

## Client Response

Declined,Nativ3 use USNT as native token, 1 USNT ≈ 1 USD, that's the reason.

# NAD-6:SequencerInbox::MaxTimeVariation lacks the validations to prevent truncation issues

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Integer Overflow and Underflow | Low | Acknowledged | betharavikiran |

## Code Reference

- code/contracts/bridge/SequencerInbox.sol#L73-L82
- code/contracts/bridge/SequencerInbox.sol#L415-L421

```
73:function initialize(
74:        IBridge bridge_,
75:        ISequencerInbox.MaxTimeVariation calldata maxTimeVariation_
76:    ) external onlyDelegated {
77:        if (bridge != IBridge(address(0))) revert AlreadyInit();
78:        if (bridge_ == IBridge(address(0))) revert HadZeroInit();
79:        bridge = bridge_;
80:        rollup = bridge_.rollup();
81:        maxTimeVariation = maxTimeVariation_;
82:    }

415:function setMaxTimeVariation(ISequencerInbox.MaxTimeVariation memory maxTimeVariation_)
416:        external
417:        onlyRollupOwner
418:    {
419:        maxTimeVariation = maxTimeVariation_;
420:        emit OwnerFunctionCalled(0);
421:    }
```

## Description

**betharavikiran :** MaxTimeVariation structure has four fields of type uint256. Also refer to the getTimeBounds() function in the contract that where the logic typecasts the result to uint64.

```
  bounds.maxTimestamp = uint64(block.timestamp + maxTimeVariation.futureSeconds);
```

**In the above code, maxTimeVariation.futureSeconds data type is uint256. but as the sum of timestamp + future seconds is typecasted to uint64, a portion of the value will be truncated. This could result in unexpected behaviour.**

As there is a potential for overflow, lets revisit where maxTimeVariation structure is initialized.

a) Initialize b) removeDelayAfterFork c) setMaxTimeVariation

While b) is safe, a) and c) could potentially accept values that will result in truncation of the value as it is being type casted to a smaller data type.

## Recommendation

**betharavikiran :** Recommendation is to add validation in the initialize and setMaxTimeVariation functions to ensure only validate range of values are accepted. It is attributed as responsibility to the owner who maintains these values. But, code is law and should enforce it where possible via the implementation.

## Client Response

Acknowledged,There is a uint64 overflow, but since this method can only be called by RollupOwner, we're not going to fix it

# NAD-7:PriceFeed address not updated

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Low | Fixed | 8olidity, n16h7m4r3 |

## Code Reference

- code/contracts/bridge/Bridge.sol#L308

```
308:IPriceFeed _priceFeed = IPriceFeed(address(0x62CAe0FA2da220f43a51F86Db2EDb36DcA9A5A08));
```

## Description

**8olidity :** The PriceFeed address in the scaleBaseFeeL1 function is still the Arbiturm Goerli test chain address, not the arbiturm main network address.

```
function scaleBaseFeeL1(uint256 fee) internal view returns (uint256) {
    //Arbiturm Goerli price feed
    IPriceFeed _priceFeed = IPriceFeed(address(0x62CAe0FA2da220f43a51F86Db2EDb36DcA9A5A08));//@audit
    int256 price = _priceFeed.latestAnswer();
    require(price > 0, "Error: Invalid price");
    return (fee * uint256(price)) / (10 ** _priceFeed.decimals());
}
```

**n16h7m4r3 :** The state variable `_priceFeed` address uses price feed oracle contract deployed in Goerli Testnet. The function `scaleBaseFeeL1()` would revert on Arbitrum's mainnet as no such contract exists at the hardcoded address.

## Recommendation

**8olidity :** It is recommended to change the PriceFeed address to the arbiturm mainnet address
**n16h7m4r3 :** Consider setting the the state variable `_priceFeed` during initialization.

## Client Response

Fixed,we can upgrade the contract if necessary

# NAD-8:Invalid estimatation for the `RetryableSubmissionFee`

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Low | Acknowledged | betharavikiran, biakia |

## Code Reference

- code/contracts/bridge/Inbox.sol#L129-L135

```
129:function calculateRetryableSubmissionFee(
130:        uint256 dataLength,
131:        uint256 baseFee
132:    ) public view returns (uint256) {
133:        // Use current block basefee if baseFee parameter is 0
134:        return (1400 + 6 * dataLength) * (baseFee == 0 ? block.basefee : baseFee);
135:    }
```

## Description

**betharavikiran :** In the Inbox contract, the function to calculate retryable submission fee has hard coded components. Refer to the code snippet for reference

```
  return (1400 + 6 * dataLength) * (baseFee == 0 ? block.basefee : baseFee);
```

With the adoption of blockchain and increase in usage, the amount of fee paid plays an important role. As a project looking for larger adoption, it should always provision for efficient onboarding of masses. In the context of above submission fee, at this point of time, the two hard coded elements namely 1400 and 6 seems reasonable.

But with adoption, this fee may seem unreasonable due to market circumstances at that time. Such assumptions should be avoided.

**biakia :** In contract `Inbox`, the function `calculateRetryableSubmissionFee` is used to estimate the `RetryableSubmissionFee`:

```
function calculateRetryableSubmissionFee(
        uint256 dataLength,
        uint256 baseFee
    ) public view returns (uint256) {
        // Use current block basefee if baseFee parameter is 0
        return (1400 + 6 * dataLength) * (baseFee == 0 ? block.basefee : baseFee);
    }
```

The contract is forked from Arbitrum. In Arbitrum, the retryable submission fee is a special fee a user must pay to create a retryable ticket. The fee is directly proportional to the size of the L1 calldata the retryable ticket uses. The above code is to estimate the gas fee based on the Ethereum Network. However, in Natvi Network the retryable submission fee should

be estimated based on the Arbitrum Network instead of Ethereum Network. To estimate the gas in Arbitrum, you can read the Arbitrum's doc: https://docs.arbitrum.io/devs-how-tos/how-to-estimate-gas

## Recommendation

**betharavikiran :** In order to adjust to prevailing market circumstances, the protocol should provision to adjust the two hard coded elements via an admin set functions or via multisig wallet.

Suggestion is to add two state variables which can be updated by admin functions to update 1400 and 6 as state variables. These values can be tuned later as needed.

The protocol should provision for adjustment considering the market scenarios will change in days to come.

**biakia :** Consider redesigning the logic of the gas estimation.

## Client Response

Acknowledged

# NAD-9:Block production may not be constant

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Language Specific | Low | Acknowledged | 8olidity |

## Code Reference

- code/contracts/bridge/SequencerInbox.sol#L84-L95
- code/contracts/bridge/SequencerInbox.sol#L128

```
84:function getTimeBounds() internal view virtual returns (TimeBounds memory) {
85:         TimeBounds memory bounds;
86:         if (block.timestamp > maxTimeVariation.delaySeconds) {
87:             bounds.minTimestamp = uint64(block.timestamp - maxTimeVariation.delaySeconds);
88:         }
89:         bounds.maxTimestamp = uint64(block.timestamp + maxTimeVariation.futureSeconds);
90:         if (block.number > maxTimeVariation.delayBlocks) {
91:             bounds.minBlockNumber = uint64(block.number - maxTimeVariation.delayBlocks);
92:         }
93:         bounds.maxBlockNumber = uint64(block.number + maxTimeVariation.futureBlocks);
94:         return bounds;
95:     }

128:if (l1BlockAndTime[0] + maxTimeVariation.delayBlocks >= block.number)
```

## Description

**8olidity :** block.number is NOT a reliable source of timing information for short terms.On Arbitrum it reflects the L1 block number, which is updated once per minute

block.number will return the latest synched block number from L1, this can be staleblock.number

Per the docs:

```
As a general rule, any timing assumptions a contract makes about block numbers and timestamps should be considered generally reliable in the longer term (i.e., on the order of at least several hours) but unreliable in the shorter term (minutes). (It so happens these are generally the same assumptions one should operate under when using block numbers directly on Ethereum!)
```

From a trusted Arbitrum Dev:

`using block.number is generally fine if you want to measure time, since that will roughly follow L1 block time`

So ultimately this is dependent on how big or small of a delay is required.

For minutes to hours, there seems to be no risk, while for shorter timeframes, some risk is possible.

In terms of impact, the main impact would be that a operation that would be expected to be executed 12 seconds later, could actually be executed as rapidly as 1 or 2 seconds after (if we assume that one L2 block goes from number A to B)

```
function forceInclusion(
  uint256 _totalDelayedMessagesRead,
  uint8 kind,
  uint64[2] calldata l1BlockAndTime,
  uint256 baseFeeL1,
  address sender,
  bytes32 messageDataHash
) external {
  if (_totalDelayedMessagesRead <= totalDelayedMessagesRead) revert DelayedBackwards();
  bytes32 messageHash = Messages.messageHash(
      kind,
      sender,
      l1BlockAndTime[0],
      l1BlockAndTime[1],
      _totalDelayedMessagesRead - 1,
      baseFeeL1,
      messageDataHash
  );
  // Can only force-include after the Sequencer-only window has expired.
  if (l1BlockAndTime[0] + maxTimeVariation.delayBlocks >= block.number)
      revert ForceIncludeBlockTooSoon();
....
}

function getTimeBounds() internal view virtual returns (TimeBounds memory) {
    TimeBounds memory bounds;
    if (block.timestamp > maxTimeVariation.delaySeconds) {
        bounds.minTimestamp = uint64(block.timestamp - maxTimeVariation.delaySeconds);
    }
    bounds.maxTimestamp = uint64(block.timestamp + maxTimeVariation.futureSeconds);
    if (block.number > maxTimeVariation.delayBlocks) {
        bounds.minBlockNumber = uint64(block.number - maxTimeVariation.delayBlocks);
    }
    bounds.maxBlockNumber = uint64(block.number + maxTimeVariation.futureBlocks);
    return bounds;
}
```

## Recommendation

**8olidity :** Only use block.timestamp for judgment

## Client Response

Acknowledged

# NAD-10:BatchSpendingReport can be generated by anyone using delayed message queue

| Category | Severity | Client Response | Contributor |
|---|---|---|---|
| Privilege Related | Low | Acknowledged | betharavikiran |

## Code Reference

- code/contracts/bridge/Bridge.sol#L125-L138
- code/contracts/bridge/Bridge.sol#L141-L156

```
125:function submitBatchSpendingReport(
126:        address sender,
127:        bytes32 messageDataHash
128:    ) external onlySequencerInbox returns (uint256) {
129:        return
130:            addMessageToDelayedAccumulator(
131:                L1MessageType_batchPostingReport,
132:                sender,
133:                uint64(block.number),
134:                uint64(block.timestamp), // solhint-disable-line not-rely-on-time,
135:                block.basefee,
136:                messageDataHash
137:            );
138:    }

141:function enqueueDelayedMessage(
142:        uint8 kind,
143:        address sender,
144:        bytes32 messageDataHash
145:    ) external payable returns (uint256) {
146:        if (!allowedDelayedInboxesMap[msg.sender].allowed) revert NotDelayedInbox(msg.sender);
147:        return
148:            addMessageToDelayedAccumulator(
149:                kind,
150:                sender,
151:                uint64(block.number),
152:                uint64(block.timestamp), // solhint-disable-line not-rely-on-time
153:                block.basefee,
154:                messageDataHash
155:            );
156:    }
```

## Description

**betharavikiran :** In the Bridge contract, submitBatchSpendingReport() function looks like a privileged function for SequencerInbox account.

But, using enqueueDelayedMessage, any user with entitlement to add to delayedInBoxes can also generate batchSpendingReport. The report generation is based on `kind` parameter passed to addMessageToDelayedAccumulator() function.

The enqueueDelayedMessage() does not enforce any restriction, which means BatchSpendingReport can be generated by accounts other than SequencerInbox account.

## Recommendation

**betharavikiran :** Basing the code, BatchSpendingReport is a privilege for SequenceInbox.
In order to enforce that priviledge, in the `encodeDelayedMessage` function, add validation for acceptable values for kind parameter.

```
    if (kind == L1MessageType_batchPostingReport) revert("Not Authorized");
```

## Client Response

Acknowledged

# NAD-11:Cross-chain calls will not be executed as expected when `msg.value` is greater than 0

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Low | Acknowledged | biakia |

## Code Reference

- code/contracts/bridge/Outbox.sol#L237-L261

```
237:function executeBridgeCall(
238:        address to,
239:        uint256 value,
240:        bytes memory data
241:    ) internal {
242:        bool success;
243:        bytes memory returndata;
244:        if(value == 0 ) {
245:            (success, returndata) = bridge.executeCall(to, value, data);
246:        }
247:        else {
248:            (success, returndata) = bridge.executeCall2(to, value, data);
249:        }
250:        if (!success) {
251:            if (returndata.length > 0) {
252:                // solhint-disable-next-line no-inline-assembly
253:                assembly {
254:                    let returndata_size := mload(returndata)
255:                    revert(add(32, returndata), returndata_size)
256:                }
257:            } else {
258:                revert BridgeCallFailed();
259:            }
260:        }
261:    }
```

## Description

**biakia :** In contract `Outbox`, the function `executeTransaction` is used to handle messages from Nativ3 to Arbitrum. It will call the function `executeBridgeCall`:

```
function executeBridgeCall(
        address to,
        uint256 value,
        bytes memory data
    ) internal {
        bool success;
        bytes memory returndata;
        if(value == 0 ) {
            (success, returndata) = bridge.executeCall(to, value, data);
        }
        else {
            (success, returndata) = bridge.executeCall2(to, value, data);
        }
        if (!success) {
            if (returndata.length > 0) {
                // solhint-disable-next-line no-inline-assembly
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert BridgeCallFailed();
            }
        }
    }
```

This function will perform different calls depending on the value of `msg.value`. If the `msg.value` is 0, it will call `bridge.executeCall` to execute a low-level call. If the `msg.value` is greater than 0, the `bridge.executeCall2` will be called:

```
function executeCall2(
        address to,
        uint256 value,
        bytes calldata data
    ) external returns (bool success, bytes memory returnData) {
        if (!allowedOutboxesMap[msg.sender].allowed) revert NotOutbox(msg.sender);
        if (data.length > 0 && !to.isContract()) revert NotContract(to);
        address prevOutbox = _activeOutbox;
        _activeOutbox = msg.sender;
        // We set and reset active outbox around external call so activeOutbox remains valid during
call

        // We use a low level call here since we want to bubble up whether it succeeded or failed to
the caller
        // rather than reverting on failure as well as allow contract and non-contract calls
        // solhint-disable-next-line avoid-low-level-calls
        // (success, returnData) = to.call{value: value}(data); // comment by
        _gasToken.mint(to, value);
        success = true;
        _activeOutbox = prevOutbox;
        emit BridgeCallTriggered(msg.sender, to, value, data);
    }
```

It will not use a low level call but just mint tokens to the user. As a result, function calls which `msg.value` is greater than 0 will not be executed as expected.

## Recommendation

**biakia :** Consider determining which function to call based on the input param `bytes calldata data` rather than `msg.value`.

## Client Response

Acknowledged,We do not want users to deposit or withdraw ETH. We actually replaced ETH with GasToken. Users can withdraw GasToken via executeBridgeCall function. And we're not going to support other cross-chain operations, except for withdraw GasToken and ERC20.

# NAD-12:Unable to calling `Bridge.executeCall()` with attached ETH

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Low | Acknowledged | TrungOre, n16h7m4r3 |

## Code Reference

- code/contracts/bridge/Outbox.sol#L237-L261
- code/contracts/bridge/Outbox.sol#L244-L246

```
237:function executeBridgeCall(
238:        address to,
239:        uint256 value,
240:        bytes memory data
241:    ) internal {
242:        bool success;
243:        bytes memory returndata;
244:        if(value == 0 ) {
245:            (success, returndata) = bridge.executeCall(to, value, data);
246:        }
247:        else {
248:            (success, returndata) = bridge.executeCall2(to, value, data);
249:        }
250:        if (!success) {
251:            if (returndata.length > 0) {
252:                // solhint-disable-next-line no-inline-assembly
253:                assembly {
254:                    let returndata_size := mload(returndata)
255:                    revert(add(32, returndata), returndata_size)
256:                }
257:            } else {
258:                revert BridgeCallFailed();
259:            }
260:        }
261:    }

244:if(value == 0 ) {
245:            (success, returndata) = bridge.executeCall(to, value, data);
246:        }
```

## Description

**TrungOre :** The comment within the `Bridge.sol` contract explains that this contract serves as the ETH escrow for values sent with messages. The sole function the Bridge employs to access the ETH held in the contract is `Bridge.executeCall()`. This function executes a low-level call to a specific contract identified by the `to` address, utilizing ETH with a value of `value` and data as `data`.

```
function executeCall(
    address to,
    uint256 value,
    bytes calldata data
) external returns (bool success, bytes memory returnData) {
    if (!allowedOutboxesMap[msg.sender].allowed) revert NotOutbox(msg.sender);
    if (data.length > 0 and !to.isContract()) revert NotContract(to);
    address prevOutbox is _activeOutbox;
    _activeOutbox = msg.sender;

    // We set and reset the active outbox around an external call so that activeOutbox remains valid
during the call

    // We use a low-level call here since we want to convey whether it succeeded or failed to the ca
ller,
    // rather than reverting on failure, and we also want to allow both contract and non-contract ca
lls
    // solhint-disable-next-line avoid-low-level-calls
    (success, returnData) = to.call{value: value}(data);
    _activeOutbox = prevOutbox;
    emit BridgeCallTriggered(msg.sender, to, value, data);
}
```

However, it's evident that this function can only be called by the `Outbox.sol` contract, specifically within the `Outbox.executeBridgeCall()` function.

```
function executeBridgeCall(
    address to,
    uint256 value,
    bytes memory data
) internal {
    bool success;
    bytes memory returndata;
    if (value == 0 ) {
        (success, returndata) = bridge.executeCall(to, value, data);
    }
    else {
        (success, returndata) = bridge.executeCall2(to, value, data);
    }
    if (!success) {
        if (returndata.length > 0) {
            // solhint-disable-next-line no-inline-assembly
            assembly {
                let returndata_size := mload(returndata);
                revert(add(32, returndata), returndata_size);
            }
        } else {
            revert BridgeCallFailed();
        }
    }
}
```

When `value > 0`, the above function calls the `bridge.executeCall2()` function, which is used to mint `value` gasTokens to the specified address (`to`). This process prevents all transactions that require attached ETH from executing because the `executeCall2()` function is triggered instead of `executeCall()`.

**Impact:** All ETH sent with messages becomes trapped in the Bridge contract, and transactions requiring attached ETH cannot be processed.

**n16h7m4r3 :** The Nativ3DAO bridge does not support direct ETH deposits on the L1 Chain and allows only GasToken deposits. The mentioned workflow would always revert due to the revert in `depositEth()` at the Inbox contract. Making the implementation for executing ETH deposit on the bridge redundant.

## Recommendation

**TrungOre :** To address this issue, it is advisable to introduce a new `kind` type to distinguish transactions that involve ETH from those intended for gasToken minting.

**n16h7m4r3 :** Consider removing the relevant redundant implementation from the Inbox, Bridge and Outbox contract. And add a check in the function `executeBridgeCall()` in the Outbox contract requiring `value` to be always greater than 0.

## Client Response

Acknowledged,We do not want users to deposit or withdraw ETH. We actually replaced ETH with GasToken. Users can withdraw GasToken via executeBridgeCall function. And we're not going to support other cross-chain operations, except for withdraw GasToken and ERC20.

# NAD-13:Return value for message added to Delay accumulator is stale

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Logical | Low | Declined | betharavikiran |

## Code Reference

- code/contracts/bridge/Bridge.sol#L158-L193

```
158:function addMessageToDelayedAccumulator(
159:        uint8 kind,
160:        address sender,
161:        uint64 blockNumber,
162:        uint64 blockTimestamp,
163:        uint256 baseFeeL1,
164:        bytes32 messageDataHash
165:    ) internal returns (uint256) {
166:        uint256 count = delayedInboxAccs.length;
167:        baseFeeL1 = scaleBaseFeeL1(baseFeeL1);
168:        bytes32 messageHash = Messages.messageHash(
169:            kind,
170:            sender,
171:            blockNumber,
172:            blockTimestamp,
173:            count,
174:            baseFeeL1,
175:            messageDataHash
176:        );
177:        bytes32 prevAcc = 0;
178:        if (count > 0) {
179:            prevAcc = delayedInboxAccs[count - 1];
180:        }
181:        delayedInboxAccs.push(Messages.accumulateInboxMessage(prevAcc, messageHash));
182:        emit MessageDelivered(
183:            count,
184:            prevAcc,
185:            msg.sender,
186:            kind,
187:            sender,
188:            messageDataHash,
189:            baseFeeL1,
190:            blockTimestamp
191:        );
192:        return count;
193:    }
```

## Description

**betharavikiran :** In the Bridge contract, when a new message is added to the delay accumulator, before adding the new message, the length of the messages in the queue is stored as a state variable called count.

After the checks are done, the new message is pushed into the queue. The return value should be based on the queue length after the newly pushed message, but instead the old value is returned.

```
function addMessageToDelayedAccumulator(
        uint8 kind,
        address sender,
        uint64 blockNumber,
        uint64 blockTimestamp,
        uint256 baseFeeL1,
        bytes32 messageDataHash
    ) internal returns (uint256) {
        [HERE]====> uint256 count = delayedInboxAccs.length;
        baseFeeL1 = scaleBaseFeeL1(baseFeeL1);
        bytes32 messageHash = Messages.messageHash(
            kind,
            sender,
            blockNumber,
            blockTimestamp,
            count,
            baseFeeL1,
            messageDataHash
        );
        bytes32 prevAcc = 0;
        if (count > 0) {
            prevAcc = delayedInboxAccs[count - 1];
        }
      [HERE]====> delayedInboxAccs.push(Messages.accumulateInboxMessage(prevAcc, messageHash));
        emit MessageDelivered(
            count,
            prevAcc,
            msg.sender,
            kind,
            sender,
            messageDataHash,
            baseFeeL1,
            blockTimestamp
        );``
       [HERE]====>return count;
    }
```

The count returned should be based on the updated delayedInboxAccs.

## Recommendation

**betharavikiran :** Return the count based on the latest length of the delayedInboxAccs array. instead of returning count, return the array length directly as below.

```
return delayedInboxAccs.length;
```

## Client Response

Declined,In this case, the index of the latest msg is returned

# NAD-14:Missing zero address check.

| Category | Severity | Client Response | Contributor |
|---|---|---|---|
| Code Style | Low | Declined | helookslikeme, n16h7m4r3 |

## Code Reference

- code/contracts/bridge/Inbox.sol#L39-L47
- code/contracts/bridge/Inbox.sol#L97-L98
- code/contracts/bridge/Bridge.sol#L57-L58

```
39:function setAllowList(address[] memory user, bool[] memory val) external onlyRollupOrOwner {
40:        require(user.length == val.length, "INVALID_INPUT");
41:
42:        for (uint256 i = 0; i < user.length; i++) {
43:            isAllowed[user[i]] = val[i];
44:            emit AllowListAddressSet(user[i], val[i]);
45:        }
46:    }

57:_activeOutbox = EMPTY_ACTIVEOUTBOX;
58:        rollup = rollup_;

97:bridge = _bridge;
98:        sequencerInbox = _sequencerInbox;
```

## Description

**helookslikeme :** Missing 0 address detection, causing 0 addresses to enter the allowed list, which should be beyond expectations

**n16h7m4r3 :** The function initialize() is missing zero address checks where address is used parameter to the function. In many of these instances, the functions do not validate that the passed address is not the address 0. While this does not currently pose a security risk, consider adding checks for the passed addresses being nonzero to prevent unexpected behavior.

## Recommendation

**helookslikeme :** address[] != address(0)

**n16h7m4r3 :** Consider implementing zero address checks for the function parameters.

## Client Response

Declined,All the contracts will be deploy by BridgeCreator and RollupCreator, so needn't to check Zero address

# NAD-15:Centralized authority risk risk in `GasToken::mint()and burn` function

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Privilege Related | Low | Declined | helookslikeme |

## Code Reference

- code/contracts/bridge/GasToken.sol#L71-L85

```
71:function mint(address _to, uint256 _amount) external virtual onlyBridge {
72:        _mint(_to, _amount);
73:        emit Mint(_to, _amount);
74:    }
75:
76:    /**
77:     * @notice Allows the StandardBridge on this network to burn tokens.
78:     *
79:     * @param _from   Address to burn tokens from.
80:     * @param _amount Amount of tokens to burn.
81:     */
82:    function burn(address _from, uint256 _amount) external virtual onlyInbox {
83:        _burn(_from, _amount);
84:        emit Burn(_from, _amount);
85:    }
```

## Description

**helookslikeme :** The burn and mint functions can be burned or issued at will, which will bring great uncertainty to user assets.

## Recommendation

**helookslikeme :** Use multi-signature method or asynchronous verification of permissions

## Client Response

Declined,Inbox will burn GasToken when user deposit GasToken to Nativ3. Bridge will mint GasToken when user withdraw from Nativ3. The owner of production is a multi-sign wallet

# NAD-16:Redundant Code

| Category | Severity | Client Response | Contributor |
|---|---|---|---|
| Logical | Informational | Acknowledged | 8olidity, biakia, betharavikiran, n16h7m4r3 |

## Code Reference

- code/contracts/bridge/Inbox.sol#L138-L140
- code/contracts/bridge/SequencerInbox.sol#L174-L206
- code/contracts/bridge/SequencerInbox.sol#L175-L206
- code/contracts/bridge/Bridge.sol#L294-L295
- code/contracts/bridge/Bridge.sol#L220

```
138:function depositEth() public payable whenNotPaused onlyAllowed returns (uint256) {
139:        revert("Error : Abandon");
140:    }

174:/// @dev Deprecated in favor of the variant specifying message counts for consistency
175:    function addSequencerL2BatchFromOrigin(
176:        uint256 sequenceNumber,
177:        bytes calldata data,
178:        uint256 afterDelayedMessagesRead,
179:        IGasRefunder gasRefunder
180:    ) external refundsGas(gasRefunder) {
181:        // solhint-disable-next-line avoid-tx-origin
182:        if (msg.sender != tx.origin) revert NotOrigin();
183:        if (!isBatchPoster[msg.sender]) revert NotBatchPoster();
184:
185:        (bytes32 dataHash, TimeBounds memory timeBounds) = formDataHash(
186:            data,
187:            afterDelayedMessagesRead
188:        );
189:        (
190:            uint256 seqMessageIndex,
191:            bytes32 beforeAcc,
192:            bytes32 delayedAcc,
193:            bytes32 afterAcc
194:        ) = addSequencerL2BatchImpl(dataHash, afterDelayedMessagesRead, data.length, 0, 0);
195:        if (seqMessageIndex != sequenceNumber)
196:            revert BadSequencerNumber(seqMessageIndex, sequenceNumber);
197:        emit SequencerBatchDelivered(
198:            sequenceNumber,
199:            beforeAcc,
200:            afterAcc,
201:            delayedAcc,
202:            totalDelayedMessagesRead,
203:            timeBounds,
204:            BatchDataLocation.TxInput
205:        );
206:    }

175:function addSequencerL2BatchFromOrigin(
176:        uint256 sequenceNumber,
177:        bytes calldata data,
```

```
178:         uint256 afterDelayedMessagesRead,
179:         IGasRefunder gasRefunder
180:     ) external refundsGas(gasRefunder) {
181:         // solhint-disable-next-line avoid-tx-origin
182:         if (msg.sender != tx.origin) revert NotOrigin();
183:         if (!isBatchPoster[msg.sender]) revert NotBatchPoster();
184:
185:         (bytes32 dataHash, TimeBounds memory timeBounds) = formDataHash(
186:             data,
187:             afterDelayedMessagesRead
188:         );
189:         (
190:             uint256 seqMessageIndex,
191:             bytes32 beforeAcc,
192:             bytes32 delayedAcc,
193:             bytes32 afterAcc
194:         ) = addSequencerL2BatchImpl(dataHash, afterDelayedMessagesRead, data.length, 0, 0);
195:         if (seqMessageIndex != sequenceNumber)
196:             revert BadSequencerNumber(seqMessageIndex, sequenceNumber);
197:         emit SequencerBatchDelivered(
198:             sequenceNumber,
199:             beforeAcc,
200:             afterAcc,
201:             delayedAcc,
202:             totalDelayedMessagesRead,
203:             timeBounds,
204:             BatchDataLocation.TxInput
205:         );
206:     }

220:) external returns (bool success, bytes memory returnData) {

294:/// @dev For the classic -> nitro migration. TODO: remove post-migration.
295:     function acceptFundsFromOldBridge() external payable {}
```

## Description

**8olidity :** The following functions have been deprecated but not removed

```
/// @dev For the classic -> nitro migration. TODO: remove post-migration.
 function acceptFundsFromOldBridge() external payable {} //@audit


/// @dev Deprecated in favor of the variant specifying message counts for consistency  @audit 这个函数弃用了
function addSequencerL2BatchFromOrigin(
    uint256 sequenceNumber,
    bytes calldata data,
    uint256 afterDelayedMessagesRead,
    IGasRefunder gasRefunder
) external refundsGas(gasRefunder) {
    ...
}
```

There is another problem here. Keep the original `addSequencerL2BatchFromOrigin` function definition in the `ISequencerInbox` contract.

```
function addSequencerL2BatchFromOrigin(
    uint256 sequenceNumber,
    bytes calldata data,
    uint256 afterDelayedMessagesRead,
    IGasRefunder gasRefunder
) external;
```

**biakia :** In the contract `Inbox`, the function `depositEth` has a meaningles implementation:

```
function depositEth() public payable whenNotPaused onlyAllowed returns (uint256) {
        revert("Error : Abandon");
    }
```

It does nothing but reverts.

**betharavikiran :** Per the comments, the addSequencerL2BathFromOrigin is a deprecated function. Such function should be removed or revert on calling to avoid unexpected or inconsistent behavior.

**n16h7m4r3 :** The variable `returnData` returned in the function `executeCall2()` is not utilized by the function.

## Recommendation

**8olidity :** 1. Remove deprecated functions 2. Update interface configuration

**biakia :** Consider removing this function if it is not intended to be used.

**betharavikiran :** revert on call of the function is the recommendation.

**n16h7m4r3 :** The variable declared can safely be removed.

## Client Response

Acknowledged

# NAD-17:Outbox ::l2ToL1Block function compares the context's l2Block against wrong default value.

| Category | Severity | Client Response | Contributor |
|---|---|---|---|
| Logical | Informational | Fixed | betharavikiran |

## Code Reference

- code/contracts/bridge/Outbox.sol#L86-L91

```
86:function l2ToL1Block() external view returns (uint256) {
87:         uint128 l2Block = context.l2Block;
88:         // we don't return the default context value to avoid a breaking change in the API
89:         if (l2Block == L1BLOCK_DEFAULT_CONTEXT) return uint256(0);
90:         return uint256(l2Block);
91:     }
```

## Description

**betharavikiran :** Implementation logic for function l2ToL1Block() in Outbox contract is wrong. The logic of comparing the l2Block from the context should be against L2BLOCK_DEFAULT_CONTEXT and not L1BLOCK_DEFAULT_CONTEXT. This is a copy paste error. This will cause some issues in readability of the code, but other wise, this typo will not cause any impact because the default data initialized for L1Block and L2Block is same. So, it does not impact the working of the contract. The issue is in the below line

**if (l2Block == L1BLOCK_DEFAULT_CONTEXT) return uint256(0);**

```
    function l2ToL1Block() external view returns (uint256) {
          uint128 l2Block = context.l2Block;
          // we don't return the default context value to avoid a breaking change in the API
          if (l2Block == L1BLOCK_DEFAULT_CONTEXT) return uint256(0);
          return uint256(l2Block);
      }
```

## Recommendation

**betharavikiran :** To make logical sense, the code should be updated to below.

```
    function l2ToL1Block() external view returns (uint256) {
          uint128 l2Block = context.l2Block;
          // we don't return the default context value to avoid a breaking change in the API
          if (l2Block == L2BLOCK_DEFAULT_CONTEXT) return uint256(0);
          return uint256(l2Block);
      }
```

## Client Response

Fixed

# NAD-18:SequencerInbox::event Offchain notification for OwnerFunction does not express the underlying change

| Category | Severity | Client Response | Contributor |
| --- | --- | --- | --- |
| Language Specific | Informational | Acknowledged | betharavikiran |

## Code Reference

- code/contracts/bridge/SequencerInbox.sol#L415-L421
- code/contracts/bridge/SequencerInbox.sol#L424-L427
- code/contracts/bridge/SequencerInbox.sol#L430-L442
- code/contracts/bridge/SequencerInbox.sol#L445-L453

```
415:function setMaxTimeVariation(ISequencerInbox.MaxTimeVariation memory maxTimeVariation_)
416:        external
417:        onlyRollupOwner
418:    {
419:        maxTimeVariation = maxTimeVariation_;
420:        emit OwnerFunctionCalled(0);
421:    }

424:function setIsBatchPoster(address addr, bool isBatchPoster_) external onlyRollupOwner {
425:        isBatchPoster[addr] = isBatchPoster_;
426:        emit OwnerFunctionCalled(1);
427:    }

430:function setValidKeyset(bytes calldata keysetBytes) external onlyRollupOwner {
431:        uint256 ksWord = uint256(keccak256(bytes.concat(hex"fe", keccak256(keysetBytes))));
432:        bytes32 ksHash = bytes32(ksWord ^ (1 << 255));
433:        require(keysetBytes.length < 64 * 1024, "keyset is too large");
434:
435:        if (dasKeySetInfo[ksHash].isValidKeyset) revert AlreadyValidDASKeyset(ksHash);
436:        dasKeySetInfo[ksHash] = DasKeySetInfo({
437:            isValidKeyset: true,
438:            creationBlock: uint64(block.number)
439:        });
440:        emit SetValidKeyset(ksHash, keysetBytes);
441:        emit OwnerFunctionCalled(2);
442:    }

445:function invalidateKeysetHash(bytes32 ksHash) external onlyRollupOwner {
446:        if (!dasKeySetInfo[ksHash].isValidKeyset) revert NoSuchKeyset(ksHash);
447:        // we don't delete the block creation value since its used to fetch the SetValidKeyset
448:        // event efficiently. The event provides the hash preimage of the key.
449:        // this is still needed when syncing the chain after a keyset is invalidated.
450:        dasKeySetInfo[ksHash].isValidKeyset = false;
451:        emit InvalidateKeyset(ksHash);
452:        emit OwnerFunctionCalled(3);
453:    }
```

## Description

**betharavikiran :** In the SequencerInbox contract, on calling the Owner Functions, the event emitted does not clearly express the change. As an example, take a look at the below function, setIsBatchPoster, where a specific address is being enabled or disabled based on the parameter values passed.

```
function setIsBatchPoster(address addr, bool isBatchPoster_) external onlyRollupOwner {
        isBatchPoster[addr] = isBatchPoster_;
        emit OwnerFunctionCalled(1);
    }
```

The event OwnerFunctionCalled() is emitted with a number to infer the type of event. But, offchain cannot monitor the changed values if offchain wants to monitor.

Per the code, the intention was more biased towards reusability of the same event.

## Recommendation

**betharavikiran :** Recommendation is to add separate events for each owner function and emit exact details. This way, any monitoring necessary from offchain can be effectively performed. Emitting an event underlines the need to notify offchain, hence, also expressing the change details will help in effective monitoring.

example: event BatchPostedUpdate(address addr, bool isBatchPoster_);

Fire the event so that offchain knows that address is either enabled or disabled as Batch poster.

## Client Response

Acknowledged

# Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3’s prior written consent in each instance.

This report is not an “endorsement” or “disapproval” of any particular project or team. This report is not an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3’s position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.