Veridise. **Auditing Report**

Hardening Blockchain Security with Formal Methods

FOR

catalyst

Generalised Incentives

Veridise Inc.

January 18, 2024

► **Prepared For:**

Cata Labs
https://catalabs.org/

► **Prepared By:**

Benjamin Sepanski
Nicholas Brown

► **Contact Us:** contact@veridise.com

► **Version History:**

Jan. 18, 2024	V1
Jan. 17, 2024	Initial Draft

Contents

1 Executive Summary

From Jan. 8, 2024 to Jan. 12, 2024, Cata Labs engaged Veridise to review the security of their Generalised Incentives smart contracts*. The review covered an incentive scheme to standardize rewards for an arbitrary message bridge (AMB). Veridise conducted the assessment over 2 person-weeks, with 2 engineers reviewing code over 1 week on commit `bb61af18`. The auditing strategy involved a tool-assisted analysis of the source code performed by Veridise engineers as well as extensive manual auditing.

Code assessment. The Cata Labs developers provided the source code of the Generalised Incentives contracts for review. The project creates a standard interface for cross-chain applications to provide incentives to relayers, with incentives for actions at both the source and destination chains denominated in currency native to the source chain.

The code is original, written by the Generalised Incentives developers. It contains some documentation in the form of an extensive README and documentation comments on functions and storage variables. The developers were also very responsive to questions from the auditing team.

The source code contained a test suite, which the Veridise auditors noted covered an extensive variety of scenarios, including both successful and unsuccessful calls to the protocol.

Summary of issues detected. The audit uncovered 7 issues. The most severe issue identified was 1 medium-severity issue, which allowed a relayer to force application calls to revert via gas denial (V-GNI-VUL-001). Two low-severity findings (V-GNI-VUL-002 and V-GNI-VUL-003) relate to missing validations and non-unique IDs. The remaining 1 warning and 3 informational findings do not pose direct security threats, but provide code recommendations and help to ensure the protocol will remain safe under future updates.

Of the 7 issues, Cata Labs has fixed 6. The remaining 1 issue (V-GNI-VUL-003) was determined to be intended behavior after discussions with Cata Labs. The Veridise auditors have included it in this report to make readers aware of behavior which may at first be unexpected.

Recommendations. The auditors found the Generalised Incentives to be very high quality, well-tested code. The main barrier to entry for any users will be some few behaviors which may be unexpected. For example, "ack"s may be replayed, which must be handled by the application. See V-GNI-VUL-003 for another example. These behaviors are well-documented in the code. We recommend the developers provide a checklist of important security considerations when implementing this interface to ease acceptance of this standard.

* `https://github.com/catalystdao/GeneralisedIncentives`

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Generalised Incentives	`bb61af18`	Solidity	Ethereum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
Jan. 8 - Jan. 12, 2024	Manual & Tools	2	2 person-weeks

Table 2.3: Vulnerability Summary.

Name	Number	Fixed	Acknowledged
Critical-Severity Issues	0	0	0
High-Severity Issues	0	0	0
Medium-Severity Issues	1	1	1
Low-Severity Issues	2	1	1
Warning-Severity Issues	1	1	1
Informational-Severity Issues	3	3	3
TOTAL	7	6	6

Table 2.4: Category Breakdown.

Name	Number
Data Validation	3
Maintainability	3
Denial of Service	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of Generalised Incentives's smart contracts. In our audit, we sought to answer questions such as:

- Can relayers intentionally cause application calls to fail?
- Can relayers mutate any of the passed values?
- Are the incentives aligned with the interests of the application protocol users?
- Are relayers required to provide enough gas for calls to succeed?
- Does the implementation conform to the specification in the README?
- Are arguments properly validated?
- Are any common Solidity vulnerabilities present (reentrancy, front-running risks, etc.)?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a combination of human experts and automated program analysis. In particular, we conducted our audit with the aid of the following techniques:

- *Static analysis.* To identify potential common vulnerabilities, we leveraged our custom smart contract analysis tool Vanguard, as well as the open-source tool Slither. These tools are designed to find instances of common smart contract vulnerabilities, such as reentrancy and uninitialized variables.

Scope. The scope of this audit is limited to the `src/` folder of the source code provided by the Generalised Incentives developers, which contains the smart contract implementation of the Generalised Incentives.

Methodology. Veridise auditors reviewed the provided documentation for Generalised Incentives and inspected the tests. They then began a manual audit of the code assisted by static analyzers. During the audit, the Veridise auditors regularly met with the Generalised Incentives developers to ask questions about the code.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-GNI-VUL-001	Unchecked low level call in IncentivizedMessage. .	Medium	Fixed
V-GNI-VUL-002	Block number may not be unique across transactio	Low	Intended Behavior
V-GNI-VUL-003	submitMessage() does not validate refundGasTo	Low	Fixed
V-GNI-VUL-004	Missing zero-checks on address parameters	Warning	Fixed
V-GNI-VUL-005	GeneralisedIncentives Typos and Comment clarifi.	Info	Fixed
V-GNI-VUL-006	Unused errors	Info	Fixed
V-GNI-VUL-007	Code Duplication	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-GNI-VUL-001: Unchecked low level call in IncentivizedMessageEscrow

Severity	Medium	**Commit**	bb61af1
Type	Data Validation	**Status**	Fixed
File(s)	src/IncentivizedMessageEscrow.sol		
Location(s)	_handleAck()		
Confirmed Fix At	c2bf4ef		

`IncentivizedMessageEscrow._handleAck` makes a low level call to the `receiveAck` function of the application:

```
1    bytes memory payload = abi.encodeWithSignature("receiveAck(bytes32,bytes32,bytes)
     ", destinationIdentifier, messageIdentifier, message[CTX1_MESSAGE_START: ]);
2   assembly ("memory-safe") {
3       // Because Solidity always create RETURNDATACOPY for external calls, even low
     -level calls where no variables are assigned,
4       // the contract can be attacked by a so called return bomb. This incur
     additional cost to the relayer they aren't paid for.
5       // To protect the relayer, the call is made in inline assembly.
6       let success := call(maxGasAck, fromApplication, 0, add(payload, 0x20), mload(
     payload), 0, 0)
7        // This is what the call would look like non-assembly.
8        // fromApplication.call{gas: maxGasAck}(
9        //     abi.encodeWithSignature("receiveAck(bytes32,bytes32,bytes)",
     destinationIdentifier, messageIdentifier, message[CTX1_MESSAGE_START: ])
10       // );
11   }
```

Snippet 4.1: The status of this call is never checked for reverts

Impact An ack provider may be able to provide less gas than is needed for the call to succeed in order to save on funds. Since only 63/64 of the gas is passed along to the call (see EIP150 for details), it may be possible for `_handleAck` to succeed without providing enough gas for the call to `recieveAck` to succeed.

Recommendation Require the relayer to provide at least `maxGasAck` to the `call(...)`.

Developer Response We know that, if the relayer intentionally denied gas to the application to provoke a failure, then they supplied less than `maxGasAck * 64/63` gas to the application immediately before the `call` (so that the application receives less than `maxGasAck` gas). If this call then reverts, then all the `maxGasAck` gas will be spent, so less than `maxGasAck / 63` gas will remain.

We added a check that `gasleft() >= maxGasAck / 63` in cases where the call fails in `_handleAck()`, `_handleTimeout()`, and `_handleMessage()`.

Note that the primary downside here is that applications which explicitly check gas usage (e.g. an application which reverts if it does not receive `maxGasAck`) may still revert, but not use enough gas for this post-check to fail. However, we feel the benefit of only checking on failures and allowing relayers to optimize for the common case is worth this extra risk.

We have also documented the additional risk.

4.1.2 V-GNI-VUL-002: Block number may not be unique across transactions

Severity	Low	**Commit**	bb61af1
Type	Denial of Service	**Status**	Intended Behavior
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	N/A		

The function comment over the abstract function `_getMessageIdentifier()` recommends the block number or block hash as a unique identifier over time.

```
/// @notice Generates a unique message identifier for a message
/// @dev Should be overwritten. The identifier should:
///  - Be unique over time: Use blocknumber or blockhash
///  - Be unique on destination chain: Use a unique source identifier
///  - Be unique on the source chain: Use a unique destinationIdentifier
///  - Depend on the message
function _getMessageIdentifier(
```

Snippet 4.2: Function comment from `IncentivizedMessageEscrow._getMessageIdentifier`.

This approach is taken in both example implementations (`IncentivizedMockEscrow` and `IncentivizedWormholeEscro
), as shown in the below excerpt from `IncentivizedWormholeEscrow`.

```
function _getMessageIdentifier(
    bytes32 destinationIdentifier,
    bytes calldata message
) internal override view returns(bytes32) {
    return keccak256(
        abi.encodePacked(
            bytes32(block.number),
            chainId(),
            destinationIdentifier,
            message
        )
    );
}
```

Snippet 4.3: Definition of `_getMessageIdentifier()` in `IncentivizedWormholeEscrow`. The implementation is almost identical to that of `IncentivizedMockEscrow`.

However, multiple transactions with the same message may occur within the same block, meaning that these message identifiers are not necessarily unique across time.

Impact Since `_setBounty()` reverts when setting a bounty for a message which already has an existing bounty, batch sending of identical messages is not possible. This may affect protocols relying on this incentive framework which have high volume, and in which certain messages are frequently identical.

Further, since `submitMessage()` does not validate that `refundGasTo` is non-zero (see issue V-GNI-VUL-003), this reversion may be bypassed intentionally, opening the protocol up to unexpected control flow and possibly unintended behavior.

Recommendation Recommend use of a nonce, rather than the `block.number`.

Developer Response This is intended. If a protocol wants to do batch sending, they should add a nonce to their payload. This saves gas for other users who do not need batch operations.

4.1.3 V-GNI-VUL-003: submitMessage() does not validate refundGasTo

Severity	Low	**Commit**	bb61af1
Type	Data Validation	**Status**	Fixed
File(s)	src/IncentivizedMessageEscrow.sol		
Location(s)	submitMessage()		
Confirmed Fix At	f9bb505		

The `refundGasTo` field of an `IncentiveDescription` is frequently used to check whether a bounty is in place. For example, the below snippet shows a check in `increaseBounty()`.

```
function increaseBounty(
    bytes32 messageIdentifier,
    uint96 deliveryGasPriceIncrease,
    uint96 ackGasPriceIncrease
) external payable {
    if (_bounty[messageIdentifier].refundGasTo == address(0)) revert
    MessageDoesNotExist();
```

Snippet 4.4: Check based on `refundGasTo` in `increaseBounty()`.

Other, similar checks occur in `_handleAck()`, `recoverAck()`, and `_setBounty()`. All of these checks appear to be operating based on the intended invariant that `refundGasTo` is non-zero exactly when a bounty is present.

However, `submitMessage()` never checks that the supplied `incentive` has a non-zero `refundGasTo` .

Impact Due to error or malicious action, `refundGasTo` may be set to zero. This may prevent benign users from increasing bounties, or allow the unintended execution of `recoverAck()` before bounty collection (though only once the packet has been verified).

Recommendation Require `incentive.refundGasTo` to be non-zero in `submitMessage()`.

Developer Response We applied the recommendation.

4.1.4 V-GNI-VUL-004: Missing zero-checks on address parameters

Severity	Warning	**Commit**	bb61af1
Type	Data Validation	**Status**	Fixed
File(s)	See issue description		
Location(s)	See issue description		
Confirmed Fix At	b72295f		

The following methods take `addresses` as arguments, but do not validate them as non-zero:

1. src/apps/mock/IncentivizedMockEscrow.sol:
   - Parameters `sendLostGasTo` and `signer` in the `constructor()` are unchecked. A check in `IncentivizedMessageEscrow` would fix this.
2. src/apps/mock/OnRecvIncentivizedMockEscrow.sol:
   - Parameter `sendLostGasTo` in the `constructor()` is unchecked. A check in `IMETimeoutExtension` would fix this.
3. src/apps/wormhole/external/callworm/GettersGetter.sol:
   - Parameter `wormholeState` in the `constructor()` is unchecked.
4. src/apps/wormhole/external/callworm/WormholeVerifier.sol:
   - Parameter `wormholeState` in the `constructor()` is unchecked. A check in `GettersGetters` would fix this.
5. src/apps/wormhole/IncentivizedWormholeEscrow.sol:
   - Parameter `sendLostGasTo` in the `constructor()` is unchecked. A check in `IMETimeoutExtension` would fix this.
6. src/IncentivizedMessageEscrow.sol:
   - Parameter `sendLostGasTo` in the `constructor()` is unchecked.
7. src/TimeoutExtension.sol:
   - Parameter `sendLostGasTo` in the `constructor()` is unchecked. An added check in `IncentivizedMessageEscrow` would fix this.

Impact Contracts may be deployed with unexpected values, wasting gas.

Recommendation Check that the addresses are non-zero.

Developer Response

1. This is intended. While it sounds dumb, it might in some cases be better to send it to `address(0)` than an actual address. Which in both cases is better than it staying the escrow.
2. Same as 1.
3. We applied the recommendation.
4. Same as 3.
5. 6., and 7. are all intended, for the same reason as 1.

4.1.5 V-GNI-VUL-005: GeneralisedIncentives Typos and Comment clarifications

Severity	Info	**Commit**	bb61af1
Type	Maintainability	**Status**	Fixed
File(s)	`src/IncentivizedMessageEscrow.sol`		
Location(s)	see issue description		
Confirmed Fix At	a47f437		

The following locations in the code have minor typos or potentially confusing comments.

- src/IncentivizedMessageEscrow.sol:
  - Line 35-39: The function comment for `MESSAGE_REVERTED` exactly matches the comment for `NO_AUTHENTICATION`. This may be a small copy-and-paste error.
  - Line 175: The phrase "identifier to recover" is repeated twice by mistake.

```
An unique identifier to recover identifier to recover
```

  - Lines 460-461: We believe the second line of the below comment should read `If more time than double the target passed, the ack relayer should get everything.`

```
// More time than target passed and the ack relayer should get a larger chunk.
// If more time than the target passed, the ack relayer should get everything.
```

  - Lines 468-469: Two minor typos occur in the below comment. They can be fixed by changing `destinaion` to `destination` and `encorages` to `encourages`.

```
// This doesn't discourage relaying, since executionTime first begins counting once the destinaion call has been executed.
// As a result, this only encorages delivery of the ack.
```

  - Lines 60-62: Callers of the `_sendPacket()` function assume that it performs cost management, as shown in the below snippet from `processPacket()`. However, this assumption is not documented on the function comment of `_sendPacket()`. We recommend documenting this assumption on `_sendPacket()` directly.

```
// The cost management is made by _sendPacket so we don't have to check if enough gas has been provided.
_sendPacket(chainIdentifier, implementationIdentifier, receiveAckWithContext);
```

- Common typos across the project:
  - The word `recipitent` is used instead of the word recipient.

Impact Clearer comments/variable names will improve maintainability for future developers.

Developer Response We applied the recommended fixes.

4.1.6 V-GNI-VUL-006: Unused errors

Severity	Info	**Commit**	bb61af1
Type	Maintainability	**Status**	Fixed
File(s)	`src/interfaces/IMessageEscrowErrors.sol`		
Location(s)	IMessageEscrowErrors		
Confirmed Fix At	0844578		

The following errors are defined, but never used.

- `InvalidTotalIncentive`
- `feeRecipitentIncorrectFormatted`
- `TargetExecutionTimeInvalid`
- `DeliveryGasPriceMustBeIncreased`
- `AckGasPriceMustBeIncreased`

Impact Developers may be confused about the correct error-type to use.

Recommendation Remove the unused errors.

Developer Response We removed the unused errors.

4.1.7 V-GNI-VUL-007: Code Duplication

Severity	Info	**Commit**	bb61af1
Type	Maintainability	**Status**	Fixed
File(s)	`src/utils/Bytes65.sol`		
Location(s)	See issue description		
Confirmed Fix At	4eae977		

In the `Bytes65` contract, `thisBytes65()`'s function definition exactly matches the `convertEVMTo65()` function, with the `evmAddress` argument inlined to be `address(this)`.

```
function convertEVMTo65(address evmAddress) public pure returns(bytes memory) {
    return abi.encodePacked(
        uint8(20),                                // Size of address. Is always 20 for EVM
        bytes32(0),                               // First 32 bytes on EVM are 0
        bytes32(uint256(uint160((evmAddress)))) // Encode the address in bytes32.
    );
}

function thisBytes65() public view returns(bytes memory) {
    return abi.encodePacked(
        uint8(20),                                // Size of address. Is always 20 for EVM
        bytes32(0),                               // First 32 bytes on EVM are 0
        bytes32(uint256(uint160(address(this)))) // Encode the address in bytes32.
    );
}
```

Snippet 4.5: Definition of `convertEVMTo65()` and `thisBytes65()`.

Impact Changes in one implementation may not be made in the other, leading to inconsistencies down the road.

Recommendation Create a shared internal function, or define `thisBytes65()` via a call to `convertEVMTo65()`.

Developer Response Considering that the 2 functions are nearly identical and placed next to each other we didn't thing it would be an issue.

Based on our testing, thisBytes65() is slightly cheaper than convertEVMTo65. We will redo testing and fix correct the code based on our findings.

Upon testing, the cost decreased even though the function is never used anywhere.

Glossary

AMB arbitrary message bridge. 1

arbitrary message bridge A protocol designed to relay any data between blockchains. This can include tokens, votes, chain state etc. For more information, see https://li.fi/knowledge-hub/navigating-arbitrary-messaging-bridges-a-comparison-framework/ . 1, 17

smart contract A self-executing contract with the terms directly written into code. Hosted on a blockchain, it automatically enforces and executes the terms of an agreement between buyer and seller. Smart contracts are transparent, tamper-proof, and eliminate the need for intermediaries, making transactions more efficient and secure.. 1, 17

Solidity The standard high-level language used to develop smart contracts on the Ethereum blockchain. See https://docs.soliditylang.org/en/v0.8.19/ to learn more. 5