Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR

Bridge

Veridise Inc.

May 20, 2024

► **Prepared For:**

Inception
https://www.inceptionlrt.com/

► **Prepared By:**

Daniel Dominguez
Alberto Gonzalez

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

► **Version History:**

April 5, 2024. Initial Draft
April 10, 2024. Official Report V1
May 20, 2024. Official Report V1.1

Contents

1 Executive Summary

From April. 2, 2024 to April. 4, 2024, Inception engaged Veridise to review the security of their Bridge contracts. Specifically, the review covered the bridge contract which handles the minting and burning of tokens during cross-chain transfers. Compared to the previous version, which Veridise has audited before*, the new version prevents the possibility of users making more than one deposit per transaction, includes short-term and long-term rate limits on the amount of tokens burned and minted, as well as performing minor refactoring to the previous codebase such as changing function names.

Veridise conducted the assessment over 6 person-days, with 2 engineers reviewing code over 3 days on commit `68abc01`†. 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 Bridge developers provided the source code of the Bridge contracts for review. To assist the Veridise auditors in comprehending the code, the developers met with the Veridise team to provide a walk through of the code and point out areas of potential concern.

The source code included a test suite, which the Veridise auditors noted tested both positive and negative paths, verifying access-control related paths as well as covering key protocol user-flow scenarios.

The Veridise auditors found the code to be well-structured and adhering to Solidity best practices. It is worth emphasizing the code's clarity and organized structure, which allowed the auditors to focus on its security aspects.

Summary of issues detected. The audit uncovered 6 issues, 0 of which are assessed to be of high or critical severity by the Veridise auditors. Among these, 1 medium-severity issue was identified. Specifically, V-ICB-VUL-001 pointed out a potential double-spending issue if a new destination bridge is deployed, as the code failed to include validation of the intended destination. In addition, the Veridise auditors uncovered 1 warning finding and 4 informational findings.

Recommendations. After auditing the protocol, the auditors had a few suggestions to improve the Bridge. Most notably, the off-chain component of the bridge was not within the scope of the audit. As the on-chain bridge contracts require trust in the off-chain logic, we highly recommend that the team seek a security review of this component.

In addition, Veridise auditors recommend implementing proper key management standards. This recommendation stems from the critical role played by the proxy's admin in upgrading the contract's implementation, and by the bridge owner, who holds the authority to modify the

* The previous audit report can be found at the following URL: `https://github.com/davos-money/davos-contracts/blob/main/audits/Veridise_280623.pdf`

† The source code can be found at the following repo `https://github.com/inceptionlrt/bridge/tree/testnet`

operator account. As the signature of the operator account is central to the bridge's operation, ensuring robust key management practices is essential for maintaining the integrity of the system.

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.

Project Dashboard 2

Table 2.1: Application Summary.

Name	Version	Type	Platform
Bridge	`68abc01`	Solidity	EVM

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
April. 2 - April. 4, 2024	Manual & Tools	2	6 person-days

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	0	0	0
Warning-Severity Issues	1	0	1
Informational-Severity Issues	4	2	4
TOTAL	6	3	6

Table 2.4: Category Breakdown.

Name	Number
Maintainability	3
Data Validation	1
Usability Issue	1
Access Control	1

3 Audit Goals and Scope

3.1 Audit Goals

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

- Can a user cause a denial of service to the bridge?
- Are users able to withdraw from the bridge if and only if they have valid bridging proofs?
- Does a valid bridging proof cover all the information needed to avoid any misuse of them?
- Are replay attacks possible?
- Can users bypass the short and long caps established by the system?
- Can users benefit at the expense of the protocol due to changes in the configurations of the system?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions above, our audit involved a combination of human experts and automated program analysis & testing tools. In particular, we conducted our audit with the aid of our in-house static analyzer, Vanguard.

- *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 was limited to the contracts of the inception-bridge repository provided by the Bridge developers. Specifically, the Veridise auditors reviewed the following contracts:

- contracts/bridge/InceptionBridge.sol
- contracts/bridge/InceptionBridgeStorage.sol

It is important to note that the following files, while they affect the in-scope code, were not reviewed* and are assumed to not contain any issues:

- contracts/lib/CallDataRLPReader.sol
- contracts/lib/EthereumVerifier.sol
- contracts/lib/ProofParser.sol
- contracts/lib/Utils.sol

* These contracts were previously audited. The previous audit report can be found at the following URL: `https://github.com/davos-money/davos-contracts/blob/main/audits/Veridise_280623.pdf`

Methodology. Veridise auditors reviewed the reports of previous audits for Bridge, inspected the provided tests. They then began a manual audit of the code assisted by both static analyzers and automated testing. Previous to the audit, the Veridise auditors met with the Bridge 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.

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

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

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

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

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

Vulnerability Report 4

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-ICB-VUL-001	Double spending if new bridge deployment	Medium	Fixed
V-ICB-VUL-002	Owner change should be a 2 step process	Warning	Acknowledged
V-ICB-VUL-003	Increase upgrade documentation	Info	Fixed
V-ICB-VUL-004	Centralization Risk	Info	Acknowledged
V-ICB-VUL-005	Proof's length is hardcoded instead of constant	Info	Fixed
V-ICB-VUL-006	Bridge contract assume tokens have the same amo.	Info	Acknowledged

4.1 Detailed Description of Issues

4.1.1 V-ICB-VUL-001: Double spending if new bridge deployment

Severity	Medium	**Commit**	68abc01
Type	Data Validation	**Status**	Fixed
File(s)	`InceptionBridge.sol`		
Location(s)	withdraw()		
Confirmed Fix At	cd92399		

During the `withdraw` function the bridge contract makes some validations before minting the correspondent tokens:

- It validates that the intended destination chain id matches with the current chain id:
  - `if(state.chainId != block.chainid) revert();`
- It validates that the receipt was emitted by the expected source bridge:
  - `if(bridgeAddressByChainId[proof.chainId] != state.contractAddress) revert();`
- It validates that the proof has not been used:
  - `if(usedProofs[payload]) revert();`
- It validates that the burned tokens in the source bridge are correct:
  - `if(getDestination(state.toToken, proof.chainId) != state.fromToken) revert();`

However, there is no validation that the current bridge contract matches the expected destination bridge contract when the tokens were burned in the source bridge. This oversight allows for replay attacks since the `usedProofs` mapping will be entirely if a new bridge contract is deployed.

Impact If a new bridge contract is deployed with the same settings as the previous one, such as token addresses and operator account, users can replay proofs already processed by the deprecated bridge contract.

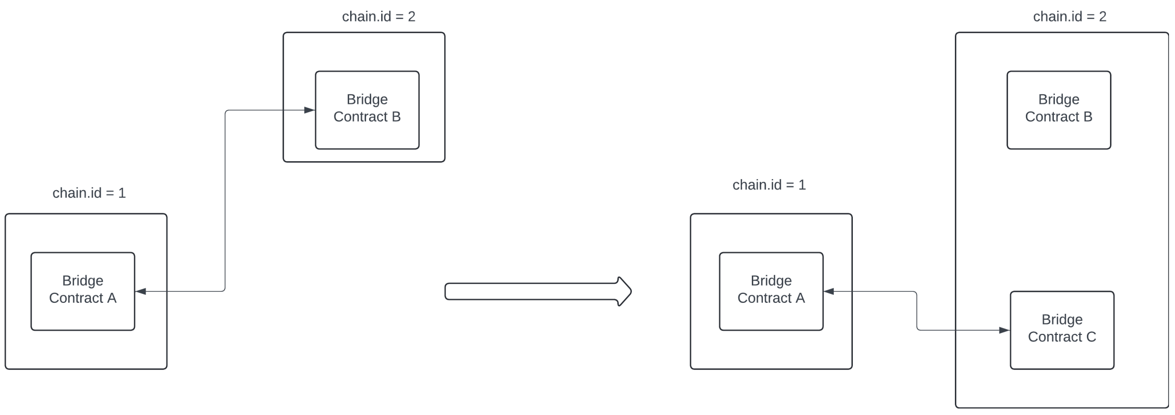

Figure 4.1: Bridge contract C is deployed as the new destination for chainID = 2.

In the above image configuration the validations explained previously should pass:

- Destination's chain id is the same.
- Source bridge contract is the same.
- `usedProofs` is empty.
- tokens configuration remained the same.

Recommendation

1. Add the bridge destination contract as part of the `Deposited` event emitted when tokens are burned in the source bridge. This will provide a record of the intended destination contract for each transaction.
2. Modify the `withdraw` function to include a validation that the address of the destination contract on the receipt matches the current bridge contract that is executing the `withdraw` function. For example:

```
require(state.destinationContract == address(this));
```

Snippet 4.2: Validation that the current contract is equal to the destination contract in the transaction receipt.

Developer Response [If applicable]

4.1.2 V-ICB-VUL-002: Owner change should be a 2 step process

Severity	Warning	**Commit**	68abc01
Type	Usability Issue	**Status**	Acknowledged
File(s)	`InceptionBridge.sol`		
Location(s)			
Confirmed Fix At	N/A		

The Inception Bridge contract currently implements a one-step ownership transfer logic, where ownership is transferred to another account in a single transaction. However, best practices recommend using a two-step transfer logic for increased security and error recovery.

Impact With the current one-step ownership transfer logic, a mistake when transferring the owner role to another would need a contract upgrade.

Recommendation Implement a two-step ownership transfer procedure. This should involve an initial step to initiate the transfer request and a second step to confirm and finalize the transfer. This approach adds an extra layer of security and allows for easier error recovery in case of mistakes during the transfer process.

OZ provides an implementation for this via the `Ownable2StepUpgradeable` contract.

Developer Response

4.1.3 V-ICB-VUL-003: Increase upgrade documentation

Severity	Info	**Commit**	68abc01
Type	Maintainability	**Status**	Fixed
File(s)	`InceptionBridge.sol, InceptionBridgeStorage.sol`		
Location(s)	See description		
Confirmed Fix At	cd92399		

The `InceptionBridge` and `InceptionBridgeStorage` contracts are missing comments about how to properly upgrade the contract in case of a change on the implementation logic. This is critical as there are issues that arises if the storage layout is not consistent between the old and new implementations.

Impact Without proper documentation on how to upgrade the implementation contract, there is a risk of introducing bugs related to inconsistent storage layouts. This could lead to unexpected behavior or vulnerabilities in the upgraded contract.

Recommendation It is recommended to add comments to the `InceptionBridgeStorage` contract explaining how to update storage variables during an upgrade. This should include instructions on how to ensure the storage layout remains consistent between the old and new implementations, such as only appending state variables at the end of `InceptionBridgeStorage` contract updating the storage gaps accordingly and to not modify the inheritance tree of the `InceptionBridge` contract.

Developer Response Comment about storage updates was added.

4.1.4 V-ICB-VUL-004: Centralization Risk

Severity	Info	**Commit**	68abc01
Type	Access Control	**Status**	Acknowledged
File(s)	`InceptionBridge.sol`		
Location(s)	See description		
Confirmed Fix At	N/A		

The Inception Bridge contract designates an owner account with special permissions, including the ability to pause deposits and withdrawals, change the short and long caps and duration, and most crucially, change the operator account, allowing the owner to mint tokens at its discretion. Additionally, the Inception Bridge is an upgradeable contract using the Transparent Proxy Pattern, meaning the admin of the proxy can change the implementation logic.

Impact If the private key of the owner account of the bridge or the admin of the proxy is stolen, the attacker could mint any amount of tokens by upgrading the vault or changing the operator account.

Recommendation

1. Utilize a decentralized governance or multi-sig contract for sensitive operations, such as changing the operator account or upgrading the contract logic. This reduces the risk of a single point of failure and ensures that such operations require consensus among multiple parties.
2. Implement key management standards to protect the private keys associated with sensitive operations.

Developer Response

4.1.5 V-ICB-VUL-005: Proof's length is hardcoded instead of constant

Severity	Info	**Commit**	68abc01
Type	Maintainability	**Status**	Fixed
File(s)	`InceptionBridge.sol`		
Location(s)	withdraw()		
Confirmed Fix At	cd92399		

The `withdraw` function computes the `proofHash` using the hardcoded value of `0x100` for the proof's length, which is not a best practice for maintainability purposes.

Impact Using hardcoded values for the length of the `proof` struct can make the code harder to maintain and update in the future. For example, it can lead to bugs if the length of the `proof` struct needs to be changed in another version of the code.

Recommendation Define a `constant` variable to represent the length of the expected proof instead of using hardcoded values. For example, you can define a constant `PROOF_LENGTH = 0 x100`.

Developer Response Applied recomendation.

4.1.6 V-ICB-VUL-006: Bridge contract assume tokens have the same amount of decimals

Severity	Info	**Commit**	68abc01
Type	Maintainability	**Status**	Acknowledged
File(s)	`InceptionBridge.sol`		
Location(s)	See description		
Confirmed Fix At	N/A		

The bridge contract assumes that all bridgeable tokens across supported chains have the same number of decimal places. During a deposit, one of the event fields records the amount of tokens burned. However, during withdrawal, the same amount retrieved from the transaction receipt is used for minting tokens to users. This can lead to underpaying or overpaying if the tokens have different decimal scaling.

For example, USDC on Ethereum has 6 decimals, while USDC on Binance Smart Chain (BSC) has 18 decimals. Reference:

- USDC-ETH
- USDC-BSC

Impact Given that the bridgeable tokens are not sourced from external issuers, it is expected that they would all be deployed with the same number of decimals. Consequently, this inconsistency is not anticipated to have an impact on the current codebase.

Recommendation We recommend documenting this assumption to prevent any potential errors in the future.

Developer Response