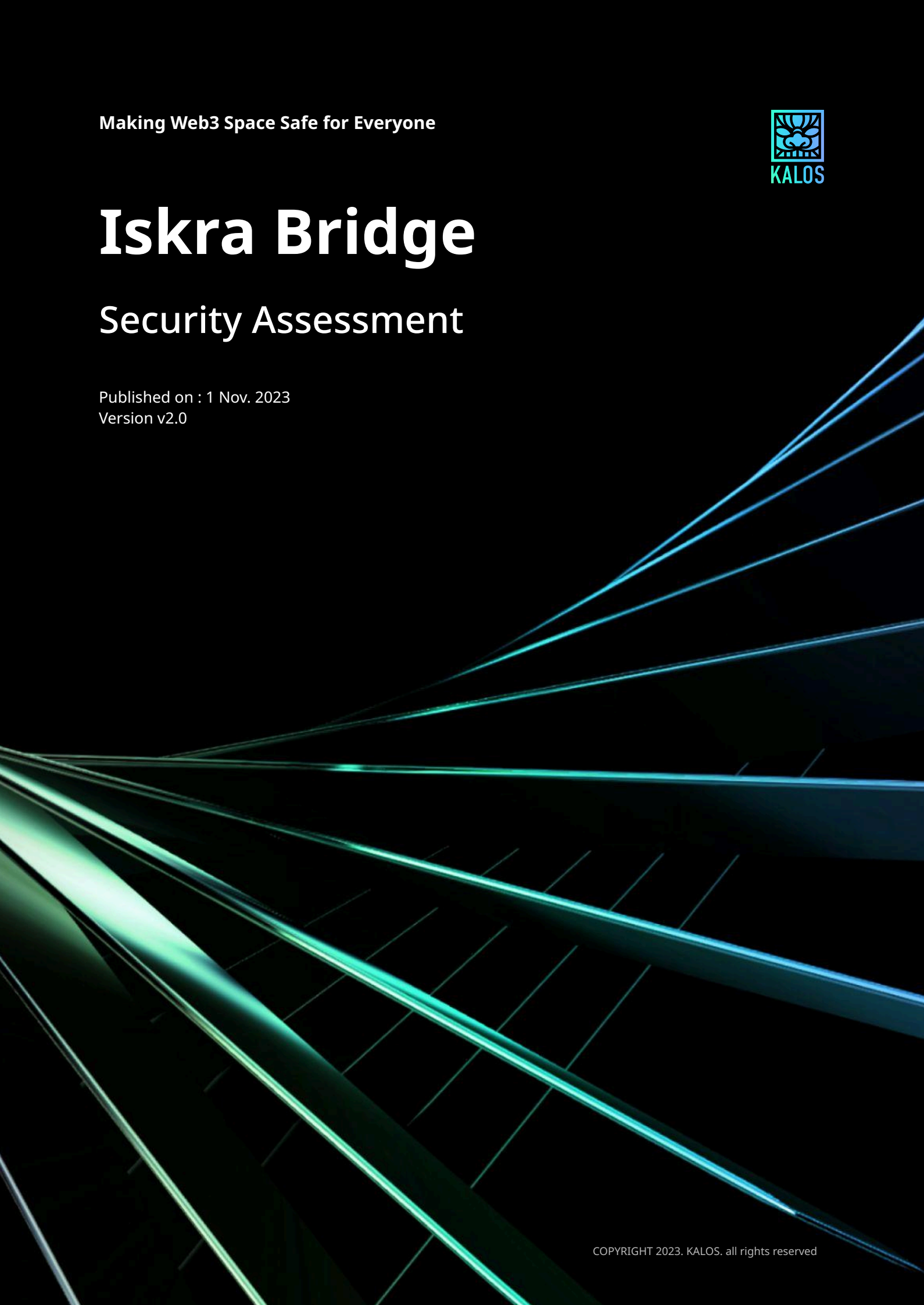
Making Web3 Space Safe for Everyone

KALOS

Iskra Bridge

Security Assessment

Published on : 1 Nov. 2023
Version v2.0



Security Report Published by KALOS

v2.0 1 Nov. 2023

Auditor : Jade

hojung han

Found issues

Severity of Issues	Findings	Resolved	Acknowledged	Comment
Critical	-	-	-	-
High	-	-	-	-
Medium	-	-	-	-
Low	-	-	-	-
Tips	-	-	-	-

TABLE OF CONTENTS

TABLE OF CONTENTS
ABOUT US
Executive Summary
OVERVIEW
- Protocol overview
- Scope
- Access Controls
- Verifications

FINDINGS
DISCLAIMER
Appendix. A
- Severity Level
- Difficulty Level
- Vulnerability Category

ABOUT US

Making Web3 Space Safer for Everyone

KALOS is a flagship service of HAECHI LABS, the leader of the global blockchain industry. We bring together the best Web2 and Web3 experts. Security Researchers with expertise in cryptography, leaders of the global best hacker team, and blockchain/smart contract experts are responsible for securing your Web3 service.

We have secured the most well-known web3 services including 1inch, SushiSwap, Klaytn, Badger DAO, SuperRare, Netmarble, Klaytn and Chainsafe. We have secured $60B crypto assets on over 400 main-nets, Defi protocols, NFT services, P2E, and Bridges.

KALOS is the only blockchain technology company selected for the Samsung Electronics Startup Incubation Program in recognition of our expertise. We have also received technology grants from the Ethereum Foundation and Ethereum Community Fund.

Inquiries: audit@kalos.xyz
Website: https://kalos.xyz

Executive Summary

Purpose of this report

This report was prepared to audit the security of the bridge contracts developed by the Iskra team. KALOS conducted the audit focusing on whether the system created by the Iskra team is soundly implemented and designed as specified in the published materials, in addition to the safety and security of the bridge.
In detail, we have focused on the following

- Signature Replay
- Bridge Logic
- Token Bridging Model
- Side effect related with Base L1 Bridge
- Assets drainage in Bridge Contract
- Assets freezing in Bridge Contract

Codebase Submitted for the Audit

The codes used in this Audit can be found on GitHub (https://github.com/iskraworld/iskra-contracts).

The commit of the code used for this Audit is “ca1f45ab86d91e44f585407ba0016ea418bf208f”

Audit Timeline

Date	Event
2023/09/25	Audit Initiation (Iskra Bridge)
2023/11/01	Delivery of v2.0 report.

Findings

No security issues have been discovered in the project.

OVERVIEW

Protocol overview

This project is based on the Wormhole Bridge, and we have carefully examined the code for other components that connect the Wormhole Bridge Service and the Iskra Bridge Service. One of the primary functions of the project is the transfer of assets between chains, which is handled in a similar manner to the Wormhole Bridge Service.

Iskra Bridge is a bridge that facilitates the exchange of various assets between Ethereum, Klaytn, and Amethyst chains. Only ERC20 assets approved by Iskra Governance can be transferred through Iskra Bridge. ERC20 token owners can create Wrapped Tokens for transferring between different chains via the Iskra Bridge Service. The original Wormhole Bridge Service supports asset custody within the Bridge Contract, allowing anyone to create wrapped tokens and transfer tokens between chains through the Bridge Contract.

In contrast, the Iskra Bridge Service manages Wrapped Tokens using TokenAdapter instead of the Bridge Contract. Wrapped Tokens are created using only tokens approved by Governance. In Iskra Bridge Service, the structure of the Beacon Proxy is modified differently from the original Wormhole Bridge Service due to TokenAdapter management. When a user publishes a message, the message publishing layer varies depending on the source chain type. When users transfer assets from Ethereum or Klaytn to Amethyst, the Original Wormhole Core Contract is used as the Message Publish Layer. Conversely, when transferring assets from Amethyst to Ethereum or Klaytn, the Wormhole Core Contract modified by Iskra is used as the Message Publishing Layer.

Unlike the Original Wormhole Bridge, users who want to transfer assets must pay an additional Service Fee. IskraBridgeGovernance handles various operational tasks related to the Iskra Bridge service, including policy updates and contract updates.

BridgeExtensionHub and BridgeExtensionL1 Contract were added to the second audit of Iskra Bridge. As a code added to send assets from Klaytn to Base, BridgeExtensionL1 ultimately transfers assets to Base chain using Base L1 Bridge Contract.

Unlike Wormhole's NFT Bridge, it provides a separate returnTransfer function that can reverse user asset transfers and provides the ability to batch process NFT asset transfers. Iskra Team provides not only ERC-721 but also ERC-1155 assets and also provides MultiToken Bridge, similar to NFT Bridge.

Notice

This audit scope included only contracts used in Iskra Bridge. Off-chain elements such as Guardian code and Relayer code used in Iskra Bridge are not included in this audit scope.

Scope

```
├── adapter
│   ├── TokenAdapterImplementation.sol
│   └── TokenAdapterState.sol
├── beacon
│   ├── Beacon.sol
│   └── BeaconProxyCreator.sol
├── governance
│   ├── BridgeGovernanceMessage.sol
│   └── IskraBridgeGovernance.sol
├── interface
│   ├── IBeaconProxyCreator.sol
│   ├── IBurnable.sol
│   ├── IFeePolicy.sol
│   ├── IGovernance.sol
│   ├── IMessenger.sol
│   └── IMintable.sol
├── utils
│   └── ERC20Querier.sol
└── wormhole
    ├── Getters.sol
    ├── Governance.sol
    ├── GovernanceStructs.sol
    ├── Implementation.sol
    ├── Messages.sol
    ├── Migrations.sol
    ├── Setters.sol
    ├── Setup.sol
    ├── State.sol
    ├── Structs.sol
    ├── Wormhole.sol
    ├── bridge
    │   ├── Bridge.sol
    │   ├── BridgeExtensionHub.sol
    │   ├── BridgeExtensionHubBase.sol
    │   ├── BridgeExtensionL1.sol
    │   ├── BridgeExtensionL1Base.sol
    │   ├── BridgeGetters.sol
    │   ├── BridgeGovernance.sol
    │   ├── BridgeImplementation.sol
    │   ├── BridgeSetters.sol
    │   ├── BridgeSetup.sol
    │   ├── BridgeState.sol
```

```
│   ├── BridgeStructs.sol
│   ├── BridgeUtil.sol
│   ├── CallerProxy.sol
│   ├── FixedRatioFee.sol
│   ├── TokenBridge.sol
│   ├── migrations
│   │   ├── BridgeImplementationV1.sol
│   │   └── BridgeImplementationV2.sol
│   ├── token
│   │   ├── TokenImplementation.sol
│   │   └── TokenState.sol
│   └── utils
│       └── Migrator.sol
├── interfaces
│   ├── IOptimismL1StandardBridge.sol
│   └── IWormhole.sol
├── libraries
│   └── external
│       └── BytesLib.sol
└── nft
    ├── MultiTokenBridge.sol
    ├── MultiTokenBridgeImplementation.sol
    ├── NFTBridge.sol
    ├── NFTBridgeEntrypoint.sol
    ├── NFTBridgeGetters.sol
    ├── NFTBridgeGovernance.sol
    ├── NFTBridgeImplementation.sol
    ├── NFTBridgeSetters.sol
    ├── NFTBridgeSetup.sol
    ├── NFTBridgeState.sol
    ├── NFTBridgeStructs.sol
    ├── NFTBridgeUtil.sol
    ├── NFTFeePolicy.sol
    ├── NFTMultiTokenCommon.sol
    ├── interfaces
    │   └── INFTFeePolicy.sol
    └── token
        ├── IBridgeNFTCommon.sol
        ├── MultiTokenImplementation.sol
        ├── MultiTokenState.sol
        ├── NFTImplementation.sol
        └── NFTState.sol
```

Access Controls

Iskra Bridge contracts have the following access control mechanisms.

- onlyOwner Modifier
- onlyOperator Modifier
- onlyVoter Modifier
- onlyProposer Modifier
- onlySelfCall Modifier
- call by only Bridge
- call by only recipient
- call by only callerProxy

onlyOwner Modifier : Modifier that controls access related to TokenImplementation mint/burn functions. The following functions are the ones where this modifier is applied.

- TokenImplementation#mint
- TokenImplementation#burn

onlyOperator Modifier : Modifier that controls access related to TokenAdapterImplementation mint/burn functions. The following functions are the ones where this modifier is applied.

- TokenAdapterImplementation#mint
- TokenAdpaterImplementation#burn

onlyVoter Modifier : Modifier that controls access related to proposal voting action in MultiSigContract. The following functions are the ones where this modifier is applied.

- MultiSigContract#confirmTransaction
- MultiSigContract#revokeTransaction
- MultiSigContract#executeTransaction

onlyProposer Modifier : Modifier that controls access related to updating proposal in MultiSigContract and MultiSigContractGovernable. The following functions are the ones where this modifier is applied.

- MultiSigContract#submitTransaction
- MultiSigContractGovernable#proposeAddVoter
- MultiSigContractGovernable#proposeRemoveVoter
- MultiSigContractGovernable#proposeAddProposer
- MultiSigContractGovernable#proposeRemoveProposer
- MultiSigContractGovernable#proposeChangeQuorumSize
- MultiSigContractGovernable#proposeCheckpoint

onlySelfCall Modifier : Modifier that restricts function call exclusively through the executeTransaction function of IskraBridgeGovernance. The following functions are the ones where this modifier is applied.

- IskraBridgeGovernance#executeMessagePublish
- IskraBridgeGovernance#executeChangeConsistencyLevel
- IskraBridgeGovernance#executeSetVerifier

call by only Bridge : The following functions can only be called by Token Bridge.

- CallerProxy#callbyProxy

call by only recipient : The following functions can only be called by Recipient.

- IskraBridgeGovernance#executeMessagePublish
- BridgeExtensionL1#returnTransfer
- NFTMultiTokenCommon#returnTransfer
- NFTMultiTokenCommon#returnBatchTransfer

call by only callerProxy : The following functions can only be called by callerProxy.

- BridgeExtensionL1#relayTransfer

Verifications

Iskra Bridge contracts have the following verification logics.

- Governance#verifyGovernanceVM
- BridgeUtil#verifyTransferVM
- BridgeUtil#verifyCompleteRemoteAssetVM
- BridgeUtil#verifyGovernanceVM
- BridgeUtil#verifyBridgeVM
- NFTBridgeGovernance#_verifyGovernanceVM
- NFTMultiTokenCommon#_verifyTransferVM
- NFTMultiTokenCommon#_verifyBatchTransferVM
- NFTMultiTokenCommon#_verifyTransferFeeTokenVM
- NFTMultiTokenCommon#_verifyBridgeVM

Governance#verifyGovernanceVM : This function verifies whether the user input has been signed by the currently active Guardian Set and created from the specified contract in the Governance Chain. Additionally, it checks whether user input is reused. The following functions validate user input using the mentioned verification function.

- Governance#submitContractUpgrade
- Governance#submitSetMessageFee
- Governance#submitNewGuardianSet
- Governance#submitTransferFees
- Governance#submitRecoverChainId

BridgeUtil#verifyTransferVM : This function verifies whether the user input has been signed by the currently active Guardian Set and created from the specified contract in the Sender Chain. It also checks that the user input was generated through the transferTokens, transferTokensWithPayload, transferTokensWithCall, or returnTransfer functions. Additionally, it checks whether user input is reused. The following functions validate user input using the mentioned verification function.

- Bridge#_completeTransfer
- BridgeExtensionL1#returnTransfer
- Bridge#returnTransfer-1
- Bridge#returnTransfer-2

BridgeUtil#verifyCompleteRemoteAssetVM : This function verifies whether the user input has been signed by the currently active Guardian Set and created from the specified contract in the Sender Chain. It also checks that the user input was generated through the createWrapped or createAdapter. Additionally, it checks whether user input is reused. The following functions validate user input using the mentioned verification function.

- Bridge#completeRemoteAsset

BridgeUtil#verifyGovernanceVM : This function verifies whether the user input has been signed by the currently active Guardian Set and created from the specified contract in the Governance Chain. Additionally, it checks whether user input is reused. The following functions validate user input using the mentioned verification function.

- BridgeGovernance#registerChain
- BridgeGovernance#upgrade
- BridgeGovernance#updateServiceFeePolicy
- Bridge#createWrapped
- Bridge#createAdapter

NFTBridgeGovernance#_verifyGovernanceVM : This function verifies whether the user input has been signed by the currently active Guardian Set and created from the specified contract in the Governance Chain. Additionally, it checks whether user input is reused. The following functions validate user input using the mentioned verification function.

- NFTBridgeGovernance#registerChain
- NFTBridgeGovernance#upgrade

NFTMultiTokenCommon#_verifyTransferVM : This function verifies whether the user input has been signed by the currently active Guardian Set and created from the specified contract in the Sender Chain. Additionally, it checks if the user input was generated through the transferNFT or transferMultiToken functions and ensures that the Chain ID where the user input should be executed matches the current Chain ID. Additionally, it checks whether user input is reused. The following functions validate user input using the mentioned verification function.

- NFTBridgeGovernance#registerChain
- NFTBridgeGovernance#upgrade
- MultiTokenBridge#returnTransfer
- NFTBridge#returnTransfer
- MultiTokenBridge#completeTransfer
- NFTBridge#completeTransfer

NFTMultiTokenCommon#_verifyBatchTransferVM : This function verifies whether the user input has been signed by the currently active Guardian Set and created from the specified contract in the Sender Chain. Additionally, it checks if the user input was generated through the batchTransferNFT or batchTransferMultiToken functions and ensures that the Chain ID where the user input should be executed matches the current Chain ID. Additionally, it checks whether user input is reused. The following functions validate user input using the mentioned verification function.

- NFTBridge#completeBatchTransfer
- MultiTokenBridge#completeBatchTransfer
- NFTBridge#returnBatchTransfer
- MultiTokenBridge#completeBatchTransfer

NFTMultiTokenCommon#_verifyTransferFeeTokenVM : This function verifies whether the user input has been signed by the currently active Guardian Set and created from the specified contract in the Sender Chain. Additionally, it checks if the user input was generated through the transferFeeToken function and ensures that the Chain ID where the user input should be executed matches the current Chain ID. Additionally, it checks whether user input is reused. The following functions validate user input using the mentioned verification function.

- NFTBridge#completeBatchTransfer
- MultiTokenBridge#completeBatchTransfer

FINDINGS

No security issues have been discovered in the project.

DISCLAIMER

This report does not guarantee investment advice, the suitability of the business models, and codes that are secure without bugs. This report shall only be used to discuss known technical issues. Other than the issues described in this report, undiscovered issues may exist such as defects on the main network. In order to write secure codes, correction of discovered problems and sufficient testing thereof are required.

Appendix. A

Severity Level

CRITICAL	Must be addressed as a vulnerability that has the potential to seize or freeze substantial sums of money.
HIGH	Has to be fixed since it has the potential to deny users compensation or momentarily freeze assets.
MEDIUM	Vulnerabilities that could halt services, such as DoS and Out-of-Gas, need to be addressed.
LOW	Issues that do not comply with standards or return incorrect values
TIPS	Tips that makes the code more usable or efficient when modified

Difficulty Level

	Low	Medium	High
Privilege	anyone	Miner/Block Proposer	Admin/Owner
Capital needed	Small or none	Gas fee or volatile as price change	More than exploited amount
Probability	100%	Depend on environment	Hard as mining difficulty

Vulnerability Category

Arithmetic	• Integer under/overflow vulnerability • floating point and rounding accuracy
Access & Privilege Control	• Manager functions for emergency handle • Crucial function and data access • Count of calling important task, contract state change, intentional task delay
Denial of Service	• Unexpected revert handling • Gas limit excess due to unpredictable implementation
Miner Manipulation	• Dependency on the block number or timestamp. • Frontrunning
Reentrancy	•Proper use of Check-Effect-Interact pattern. •Prevention of state change after external call • Error handling and logging.
Low-level Call	• Code injection using delegatecall • Inappropriate use of assembly code
Off-standard	• Deviate from standards that can be an obstacle of interoperability.
Input Validation	• Lack of validation on inputs.
Logic Error/Bug	• Unintended execution leads to error.
Documentation	•Coherency between the documented spec and implementation
Visibility	• Variable and function visibility setting
Incorrect Interface	• Contract interface is properly implemented on code.

End of Document