Security Audit

Report

22/07/2024

Neo X - EVM Bridge Contracts

All information collected here is strictly confidential and may only be distributed with Red4Sec express authorization.

Content

Introduction

The **N3-X bridge** is an infrastructure designed to facilitate the transfer and management of assets between the **Neo N3** and **Neo X** blockchains. This system is built on a set of smart contracts that ensure effective and efficient interoperability between both platforms.

The **N3-X bridge** smart contracts enable asset transitions between the **Neo N3** and **Neo X** chains, ensuring transfers are secure, transparent, reliable, and verifiable.

The **bridge-evm-contracts** is the current implementation of the smart contracts developed in Solidity and deployed in **Neo X** to fulfill the functionality of the bridge.

As solicited by **Neo** and as part of the vulnerability review and management process, Red4Sec has been requested to perform a security code audit in order to evaluate the security of the **EVM Bridge Contracts** project.

The report includes specifics retrieved from the audit for all the existing vulnerabilities **Neo X EVM Bridge Contracts**. The performed analysis showed that the smart contract did contain a high-risk issue.

Disclaimer

This document only represents the results of the code audit conducted by Red4Sec Cybersecurity and should not be used in any way to make investment decisions or as investment advice on a project.

Likewise, the report should not be considered either "endorsement" nor "disapproval" of the guarantee of the correct business model of the analyzed project, nor as guarantee on the operation or viability of the implemented financial product.

Red4Sec makes full effort and applies every resource available for each audit, however it does not warrant the function, nor the safety of the project and it cannot be deemed a sufficient assessment of the code's utility and safety, bug-free status, or any other declarations of the project. Additionally, Red4Sec makes no security assessments or judgments about the underlying business strategy, or the individuals involved in the project.

Blockchain technology and cryptographic assets come with their own new risks and challenges, where the ecosystem, platform, its programming language, and other software related to said technology can have vulnerabilities that could lead to exploits. As a result, the audit cannot guarantee the explicit security of the audited projects.

The audit reports can be used to improve the code quality of smart contracts, to help limit the vectors of attack and to lower the high level of risks associated with utilizing new and continually changing technologies such as cryptographic tokens and blockchain, but they are unable to detect any future security concerns with the related technologies.

Scope

Red4Sec Cybersecurity has made a thorough audit of the **bridge-evm-contracts** security level against attacks, identifying possible errors in the design, configuration, or programming; therefore, guaranteeing the availability, integrity and confidentiality of the project and the possible assets treated and stored.

The scope of this evaluation includes the following items provided by **Neo**:

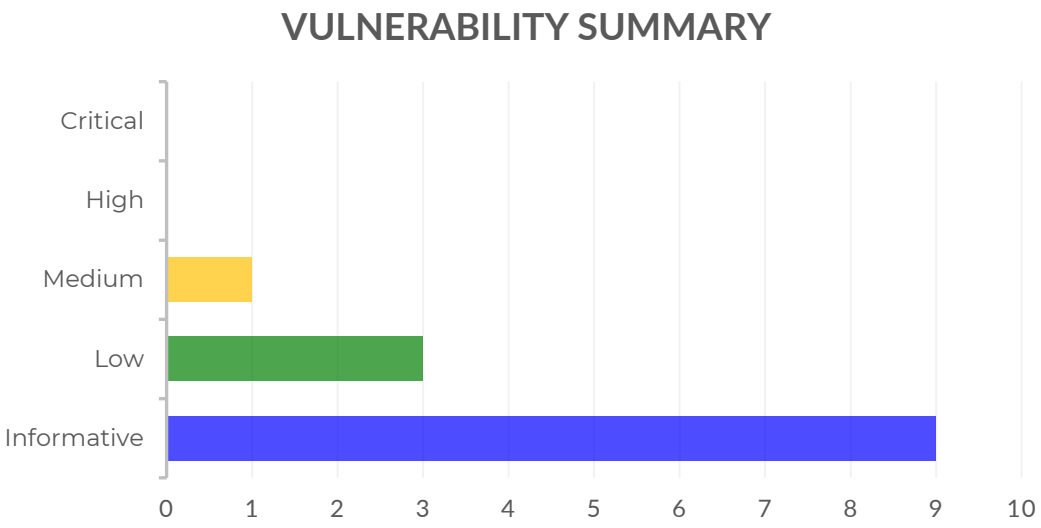
- https://github.com/bane-labs/bridge-evm-contracts
  - Commit: `14cd3efd3f618fd18a4cda358c791fdca0e66cff`
  - July review commit: `0e3ad1c40c10877e5446ba350a09207cc08bea5b`

Executive Summary

The security audit against **bridge-evm-contracts** has been conducted between the following dates: **01/04/2024** and **22/07/2024**.

Once the analysis of the technical aspects has been completed, the performed analysis showed that the audited source code contained issues that were mitigated by the team.

During the analysis, a total of **13 vulnerabilities** were detected, these vulnerabilities have been classified by the following level of risks, defined in Vulnerabilities Severity annex.



Conclusions

Based on the **Neo X EVM Bridge Contracts** audit, it is evident that the project had a number of vulnerabilities that expose it to certain security risks. While the risks are mostly low and medium, they cannot be ignored as they can potentially compromise the integrity, reliability, and efficiency of the smart contract system.

The **Neo X** team has worked on and resolved most of the issues identified and reported in this report, as reflected in their status.

Note that the conclusions of this security audit are provisional, as the development of the smart contracts is ongoing. Future changes in the architecture or functionality of the project could significantly alter both its operability and security. Therefore, continuous monitoring and periodic audits are recommended to assess the impacts of any adjustments and ensure the system's integrity.

The most significant vulnerability detected is the Hardcoded Signature Requirement which carries a medium risk. This posed a potential threat to the system as it can be exploited by malicious entities. Hardcoding can lead to a lack of flexibility and potential security risks if the hardcoded values contain sensitive information and are exposed. It is recommended to have a more dynamic and secure method for signature requirements.

The Contracts Management Risks and the Contract with a Maximum Duration also pose low-level threats. Contracts management risks can lead to mismanagement of contracts, which can result in unauthorized access or misuse. On the other hand, contracts with a maximum duration could potentially limit the flexibility and utility of the contract for the users.

The Outdated Compiler and Outdated Third-party Libraries, although only informative risks, could potentially lead to inefficiencies and vulnerabilities in the system. It is generally recommended to keep compilers and libraries up-to-date to ensure optimal performance and security.

The Lack of Documentation, Improvable Code Quality, Lack of Event Index, and Wrong Event Emit are also informative risks that indicate areas for improvement. These issues do not pose immediate threats but addressing them can improve the overall quality and reliability of the system.

In terms of GAS Optimization, the project could benefit from more efficient gas usage. This can be achieved by optimizing the contract code to use less gas, which can lead to cost savings.

In conclusion, while the project has a number of vulnerabilities, the risks associated with these are mostly low to medium and **most of them** are **currently fixed**. It is always recommended to address these vulnerabilities to ensure the security, efficiency, and reliability of the smart contract system.

UPDATE: A re-audit of the updated scope of work detailed in this report has been conducted, revealing two new vulnerabilities.

Vulnerabilities

In this section, you can find a detailed analysis of the vulnerabilities encountered upon the security audit.

List of vulnerabilities

Below, we have gathered a complete list of the vulnerabilities detected by Red4Sec, presented, and summarized in a way that can be used for risk management and mitigation.

Table of vulnerabilities			
ID	**Vulnerability**	**Risk**	**State**
NBE-01	Hardcoded Signature Requirement	**Medium**	**Fixed**
NBE-02	Contracts Management Risks	**Low**	**Acknowledged**
NBE-03	Contract with Maximum Duration	**Low**	**Fixed**
NBE-04	Outdated Compiler	**Informative**	**Fixed**
NBE-05	Outdated Third-Party Libraries	**Informative**	**Fixed**
NBE-06	GAS Optimization	**Informative**	**Fixed**
NBE-07	Solidity literals	**Informative**	**Fixed**
NBE-08	Lack of Documentation	**Informative**	**Acknowledged**
NBE-09	Improvable Code Quality	**Informative**	**Fixed**
NBE-10	Lack of Event Index	**Informative**	**Acknowledged**
NBE-11	Wrong Event Emit	**Low**	**Fixed**
NBE-12	Lack of neoN3Token Validation on TokenConfig	**Informative**	**Fixed**
NBE-13	Lack of neoXToken Verification	**Informative**	**Fixed**

Vulnerability details

In this section, we provide the details of each of the detected vulnerabilities indicating the following aspects:

- Category
- Active
- Risk
- Description
- Recommendations

Hardcoded Signature Requirement

Identifier	Category	Risk	State
NBE-01	Business Logic	**Medium**	**Fixed**

The `verifyValidatorSignatures` method defines in a hardcoded manner the number of existing validators and the number of signatures required for their validation, ignoring those established during the construction of the contract.

```
// check if all recovered addresses are in the validator set
uint covered = 0;
uint n = 0;
uint j;
for (uint i = 0; i < 5; i++) {
    for (j = n; j < 7; j++) {
        if (recovered[i] == validators[j]) {
            covered++;
            break;
        }
    }
    n = j + 1;
}
return covered == 5;
```

Hardcoded values

This implies that if only 3 validators are defined during deployment, the contract will automatically be denied because it requires 5 signatures for proper verification.

Recommendations

- It is necessary to obtain the mentioned values from those existing in the state variables: `validators` and `requiredValidatorSignaturesForDeposit`.

Source Code References

- contracts/BridgeContract.sol#L195-L196
- contracts/BridgeContract.sol#L204-L213

Fixes Review

This issue has been addressed in the following commit:

- https://github.com/bane-labs/bridge-evm-contracts/commit/fdae4a26955f39ed795cfe27135c7407576e6c79

Contracts Management Risks

Identifier	Category	Risk	State
NBE-02	Design Weaknesses	**Low**	**Acknowledged**

The logical design of the `BridgeContract` contract introduces certain minor risks that should be reviewed and considered for their improvement.

Emergency ERC20 Token Sending

Even though this logic is intentional, it is important to mention that the ERC20 tokens received by the contract cannot be returned or transferred by either the validators or the `governor`. Adding an emergency sending functionality for ERC20 tokens could be a good practice, allowing the contract to return tokens mistakenly sent by users, even though the contract is primarily designed to manage the native token.

Hardcoded Values

It should be noted that the data of the `validators`, `relayer`, `owner`, `governor`, `securityGuard`, as well as those related to the limits of deposits or required signatures cannot be modified. It is necessary to be able to modify these values in order to increase the resilience of the contract, deal with possible leaks of private keys or managing dishonest validators.

Centralized initial values

Although it is mentioned that the values of validators must be changed during the deployment, it should be noted that the first 3 validators are the same as the `owner`, `governos` and `securityGuard`. It is advisable to ensure decentralization and separation of roles, so that validators exclusively focus on validation tasks without additional responsibilities. Appropriately separating the roles of the contract will prevent possible future problems in the contract.

```
address[] public validators = [
    0x70997970C51812dc3A010C7d01b50e0d17dc79C8,
    0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC,
    0x90F79bf6EB2c4f870365E785982E1f101E93b906,
    0x15d34AAf54267DB7D7c367839AAf71A00a2C6A65,
    0x9965507D1a55bcC2695C58ba16FB37d819B0A4dc,
    0x976EA74026E726554dB657fA54763abd0C3a0aa9,
    0x14dC79964da2C08b23698B3D3cc7Ca32193d9955
];

uint256 public constant minWithdrawalAmount =     1_00000000_0000000000;
uint256 public constant maxWithdrawalAmount = 10000_00000000_0000000000;

bytes32 public depositRoot;
bytes32 public withdrawalRoot;

uint64 public depositNonce = 0;
uint64 public withdrawalNonce = 0;

uint8 public requiredValidatorSignaturesForDeposit = 5;
uint8 public maxDepositsPerDistribution = 100;

bool public locked = false;
address public owner = 0x70997970C51812dc3A010C7d01b50e0d17dc79C8;
address public governor = 0x3C44CdDdB6a900fa2b585dd299e03d12FA4293BC;
address public securityGuard = 0x90F79bf6EB2c4f870365E785982E1f101E93b906;
```

Source Code References

- contracts/BridgeContract.sol#L13-L40

Contract with Maximum Duration

Identifier	Category	Risk	State
NBE-03	Design Weaknesses	**Low**	**Fixed**

The logic related to `deposits` and `withdrawals` requires having certain values correctly signed according to the `depositNonce` and `withdrawalNonce` variables, which are incremental, of type `uint64`. However, no logic has been established to account for the scenario in which these values reach their maximum value, which would currently trigger a Denial of Service.

Although it is true that these values are large enough to extend over time, it is advisable to review this logic in order to make the contract resilient to the passage of time, or possible attacks with the aim of increasing the nonce in the future.

Recommendations

- Establish a logic that considers the end of the type used for `nonce` fields.

Source Code References

- contracts/BridgeContract.sol#L31-L32

Fixes Review

This issue was addressed during the upgradable refactor of `BridgeStorage`.

Outdated Compiler

Identifier	Category	Risk	State
NBE-04	Outdated Software	**Informative**	**Fixed**

Solc frequently launches new versions of the compiler. Using an outdated version of the compiler can be problematic, especially if there are errors that have been made public or known vulnerabilities that affect this version. The project has set the `0.8.18` compiler version in the config.

```
/** @type import('hardhat/config').HardhatUserConfig */
const config: HardhatUserConfig = {
  solidity: {
    version: "0.8.18",
    settings: {
      optimizer: {
        enabled: true,
        runs: 200,
```

Solidity has introduced important bug fixes in versions up to `0.8.25`, including version `0.8.22`. Therefore, it is recommended to use the most up-to-date version of pragma.

The source file requires a different compiler version than the Hardhat configuration. It is advisable to unify both definitions with the latest stable version of the compiler.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.9;
```

Recommendations

- It is always a good policy to use the most up to date version of the pragma.

References

- https://github.com/ethereum/solidity/blob/develop/Changelog.md

Source Code References

- hardhat.config.ts#L10
- contracts/BridgeContract.sol#L2

Fixes Review

This issue has been addressed in the following commit:

- https://github.com/bane-labs/bridge-evm-contracts/commit/53b0f3e67ad400edf4eb5b9413950ab943ea86d3

Outdated Third-Party Libraries

Identifier	Category	Risk	State
NBE-05	Outdated Software	**Informative**	**Fixed**

The Hardhat project contains dependencies flagged as vulnerable according to the `npm audit` analysis. While this does not indicate a direct vulnerability in the project itself, it does imply that an update of third-party packages or libraries is not carried out.

```
# npm audit report

follow-redirects  <=1.15.5
Severity: moderate
Follow Redirects improperly handles URLs in the url.parse() function - https://github.com/advisories/GHSA-jchw-25xp-jwwc
follow-redirects' Proxy-Authorization header kept across hosts - https://github.com/advisories/GHSA-cxjh-pqwp-8mfp

undici  <=5.28.3
Undici proxy-authorization header not cleared on cross-origin redirect in fetch - https://github.com/advisories/GHSA-3787-6prv-h9w3
Undici's fetch with integrity option is too lax when algorithm is specified but hash value is in incorrect - https://github.com/advisories/GHSA-9qxr-qj54-h672
Undici's Proxy-Authorization header not cleared on cross-origin redirect for dispatch, request, stream, pipeline - https://github.com/advisories/GHSA-m4v8-wqvr-p9f7

2 vulnerabilities (1 low, 1 moderate)
```

Recommendations

- Delete unused dependencies in the project.
- Update the libraries to the most recent version.

Fixes Review

This issue has been addressed in the following commit:

- https://github.com/bane-labs/bridge-evm-contracts/commit/6a5d43a63943e1a12661ced851e41a3b17ee392f

GAS Optimization

Identifier	Category	Risk	State
NBE-06	Codebase Quality	**Informative**	**Fixed**

Software optimization is the process of modifying a software system to make an aspect of it work more efficiently or use less resources. This premise must be applied to smart contracts as well, so that they execute faster or in order to save GAS.

On EVM blockchain, GAS is an execution fee which is used to compensate miners for the computational resources required to power smart contracts. If the network usage is increasing, so will the value of GAS optimization.

These are some of the requirements that must be met to reduce GAS consumption:

- Short-circuiting.
- Remove redundant or dead code.
- Delete unnecessary libraries.
- Explicit function visibility.
- Use of proper data types.
- Use hard-coded constant instead of state variables.
- Avoid expensive operations in a loop.
- Pay special attention to arithmetical operations and comparisons.

Logic Optimization

The condition of the `deposit` method relative to the beginning of the first stored nonce is double-checked in the next call to `subsequentNonces`. Therefore, it can be safely removed.

```
require(
    _deposits[0].nonce == depositNonce + 1,
    "Only the next nonce is allowed in the first proof."
);
require(
    subsequentNonces(_deposits, depositNonce),
    "The nonces of the proofs must be subsequent."
);
```

Duplicate condition

Source Code References

- contracts/BridgeContract.sol#L97-L100

Dead Code

In programming, a part of the source code that is never used is known as dead code. The execution of this type of code consumes more GAS during deployment, which is not necessary.

Values related to the `owner`, or `onlyowner` modifier, are not used and can be removed from the contract.

Source Code References

- BridgeContract.sol#L237-L240
- BridgeContract.sol#L38

Use Custom Errors Instead of Require

Solidity **0.8.4** introduced custom errors, a more efficient way to notify users of an operation failure.

It has been verified that the `BridgeContract` contract is managing the error messages through the `require` command. This instruction is more expensive than the use of custom errors because it requires placing the error message on the stack, whether or not the transaction is reverted in addition to being more susceptible to using long error messages that will produce a higher cost of gas, regardless of the result of the condition.

```
// Modifier

modifier onlyRelayer() {
    require(msg.sender == relayer, "Not relayer");
    _;
}

modifier onlyGovernor() {
    require(msg.sender == governor, "Not governor");
    _;
}

modifier onlySecurityGuard() {
    require(msg.sender == securityGuard, "Not securityGuard");
    _;
}

modifier onlyOwner() {
    require(msg.sender == owner, "Not owner");
    _;
}

modifier unlocked() {
    require(!locked, "Contract is locked");
    _;
}
```

Custom errors are more gas efficient than revert strings in terms of deployment and runtime cost when the revert condition is met. In order to save gas it is recommended to use custom errors instead of revert strings.

The new custom errors are defined through the `error` statement, and they can also receive arguments. As indicated in the following example:

```
error Unauthorized();
error InsufficientPrice(uint256 available, uint256 required);
```

Afterwards, it is enough to simply call them when needed using `if` conditionals:

```
if (msg.sender != owner) revert Unauthorized();
if (amount > balance[msg.sender]) {
            revert InsufficientPrice({
                available: balance[msg.sender],
                required: amount
            });
}
```

References

- https://blog.soliditylang.org/2021/04/21/custom-errors

Source Code References

- contracts/BridgeContract.sol#L92-L109
- contracts/BridgeContract.sol#L223-L243
- contracts/BridgeContract.sol#L256
- contracts/BridgeContract.sol#L273-L286
- contracts/BridgeContract.sol#L345

Redundant Code

The `computeNewRoot` method and the `computeNewWithdrawalRoot` method share the same logic, arguments, and return, it is convenient to unify both calls to reduce the bytecode size.

Source Code References

- contracts/BridgeContract.sol#L318
- contracts/BridgeContract.sol#L169

Caching Arrays Length in Loops

Every time an array iteration occurs in a loop, the solidity compiler will read the array's length. Thus, this is an additional `sload` operation for storage arrays (100 more extra gas for each iteration except for the first) and an additional `mload` action for memory arrays (3 additional gas for each iteration except for the first).

```
for (uint i; i < array.length; ++i) {
    // ...
}
```

The array length (in stack) might be cached to reduce these additional costs:

```
uint length = array.length;
for (uint i; i < length; ++i) {
    // ...
}
```

The `sload` or `mload` action is only performed once in the example above before being replaced by the inexpensive `dupX` instruction.

Source Code References

- contracts/BridgeContract.sol#L121

Optimize Require Messages

Ethereum Virtual Machine operates under a 32-byte word memory model where an additional gas cost is paid by any operation that expands the memory that is in use.

Therefore, exceeding error messages of this length means increasing the number of slots necessary to process the require, reducing the error messages to 32 bytes or less would lead to saving gas.

Source Code References

- contracts/BridgeContract.sol#L99
- contracts/BridgeContract.sol#L103
- contracts/BridgeContract.sol#L108

Storage Optimization

It has been detected that the mappings `claimableAmount` and `claimableTo` are always modified and queried simultaneously. Therefore, using a common structure is convenient in terms of gas savings.

```
struct ClaimData {
    address payable to;
    uint64 amount;
}

mapping(uint64 => ClaimData) public claimable;
```

Source Code References

- contracts/BridgeContract.sol#L254-L259

The use of the `immutable` keyword is recommended to obtain less expensive executions, by having the same behavior as a constant. However, by defining its value in the constructor we have a significant save of GAS.

Source Code References

- contracts/BridgeContract.sol#L13-L26
- contracts/BridgeContract.sol#L34-L35
- contracts/BridgeContract.sol#L38-L40

Fixes Review

This issue has been partially addressed in the following pull request:

- https://github.com/bane-labs/bridge-evm-contracts/pull/80

`_computeNewTopRoot` Method Optimization

The `_computRoot` method is not optimal in terms of gas consumption due to the declaration and use of the `parent` variable.

The `parent` variable is declared with the value of `_previousRoot` and then used within a `for` loop and reassigned at each iteration.

```
function _computeNewTopRoot(
    bytes32 _previousRoot,
    BridgeLib.DepositData[] calldata _deposits
) internal pure returns (bytes32) {
    bytes32 parent = _previousRoot;
    uint256 depositsLength = _deposits.length;
    for (uint256 i = 0; i < depositsLength; i++) {
        BridgeLib.DepositData calldata depositData = _deposits[i];
        bytes32 depositHash = _hashGasBrideOp(
            depositData.nonce,
            depositData.to,
            depositData.amount
        );
        parent = BridgeLib._computeNewRoot(parent, depositHash);
    }
    return parent;
}
```

In order to optimize gas usage in this method, the direct use of `previousRoot` is suggested. Instead of assigning `parent = previousRoot` and using `parent` within the loop, you can directly operate on `_previousRoot`. This change can avoid the need for an additional variable, optimizing the gas cost of the method accordingly.

Source Code References

- contracts/library/TokenBridgeLib.sol#L48
- contracts/library/GasBridgeLib.sol#L11

Fixes Review

This issue has been addressed in the following pull request:

- https://github.com/bane-labs/bridge-evm-contracts/pull/80

Solidity literals

Identifier	Category	Risk	State
NBE-07	Codebase Quality	**Informative**	**Fixed**

To improve code readability and reduce the risk of human error, Solidity recommends using literals. This approach makes the code more user-friendly and easier to understand.

In the case of `BridgeContract`, the `10**10` operation is frequently used to add or eliminate the decimal difference between gas and ether. This multiplication can be avoided by using the literal `1e10`.

```
// Adds 10 decimals to the amount. GasToken originally has 8 decimals and on this chain it has 18 decimals.
function addTenDecimals(uint64 _value) private pure returns (uint256) {
    return uint256(_value) * (10 ** 10);
}

// Removes 10 decimal points from the amount. GasToken originally has 8 decimals and on this chain it has 18 decimals.
function removeTenDecimals(uint256 _value) private pure returns (uint64) {
    return uint64(_value / (10 ** 10));
}
```

Unnecessary operations

References

- https://docs.soliditylang.org/en/latest/units-and-global-variables.html#units-and-globally-available-variables

Source Code References

- contracts/BridgeContract.sol#L335
- contracts/BridgeContract.sol#L330
- contracts/BridgeContract.sol#L276

Fixes Review

This issue has been addressed in the following commit:

- https://github.com/bane-labs/bridge-evm-contracts/commit/746f15126ef3a85b4b16703eb49b41192591af64

Lack of Documentation

Identifier	Category	Risk	State
NBE-08	Testing and Documentation	**Informative**	**Acknowledged**

Although the project has excellent test coverage, it is notable that it lacks documentation. Updated and accurate documentation is crucial for open-source projects, as it directly impacts community adoption and contributions. Providing comprehensive documentation improves the project's accessibility and encourages broader involvement.

Documentation is an integral part of the Secure Software Development Life Cycle (SSDLC), and it helps to improve the quality of the project. Therefore, it is recommended to add the code documentation with the according descriptions of the functionalities, classes, and public methods.

References

- https://snyk.io/learn/secure-sdlc/
- https://www.freecodecamp.org/news/why-documentation-matters-and-why-you-should-include-it-in-your-code-41ef62dd5c2f

Improvable Code Quality

Identifier	Category	Risk	State
NBE-09	Codebase Quality	**Informative**	**Fixed**

The code exhibits a lack of order and structure, which complicates reading and analysis. Issues such as inconsistent variable ordering and disorganized mappings contribute to this problem. While this is not a vulnerability in itself, improving code organization can enhance readability and reduce the likelihood of introducing new vulnerabilities.

High-quality code is important for various reasons, including maintainability, reliability, efficiency, scalability, and security. Well-written code that adheres to best practices offers several advantages, such as easier to maintain and extend, less prone to errors, more efficient, improved collaboration, scalability, and heightened security. Ultimately, good code quality contributes to a more dependable, maintainable, and secure software product, improving the overall user experience while reducing the risk of introducing new vulnerabilities.

Use Constructor to Define Values

The `BrigeContract` contract requires its correct configuration for its correct use, however, it is mentioned that the values must be modified manually prior to deployment, instead of defining them in the contract's constructor. This approach will not only prevent human errors during the deployment process but also improve the comparison of source code against similar ones in browsers, improving both maintenance and testing.

```
// Todo: Before compiling byte code for genesis script, make sure to set the correct
values for the following variables:
// - relayer
// - validators
// - minWithdrawalAmount
// - maxWithdrawalAmount
// - remove the receive function as it is only used for testing purpose.
```

Source code reference

- contracts/BridgeContract.sol#L4-L9

Lack of Comments

The `deposit` method requires that the signatures be ordered in the same way that the validators have been stored in the array. This special case should be reflected in a comment of the `deposit` method.

Source code reference

- contracts/BridgeContract.sol#L88

Good Naming

The variable `hashAmount` of the `withdraw` method makes it appear that it is a hash, when in reality it is the value converted to 8 decimal places. It is advisable to modify the name of the variable in order to improve the readability and understanding of the method.

Source code reference

- contracts/BridgeContract.sol#L289

Named Mappings

Solidity *0.8.18* allows to use named parameters in mapping types. This new feature enables the assignment of descriptive names to mappings, improving the code's readability, interpretation, and audit process.

Reference:

- https://soliditylang.org/blog/2023/02/01/solidity-0.8.18-release-announcement

Source code reference

- contracts/BridgeContract.sol#L42-L44

Open TODO

Certain to-do comments have been detected that reflect that the code is unfinished. Thus, many changes that have not been audited could introduce new vulnerabilities in the future.

Source code reference

- contracts/BridgeContract.sol#L4
- contracts/BridgeContract.sol#L154
- contracts/BridgeContract.sol#L299

Recommendations

As a reference, it is always recommendable to apply coding style/good practices that can be found in multiple standards such as:

- Solidity Style Guide

These references are very useful to improve smart contract quality. Many of these practices are widely accepted and recognized as a popular and effective approach to software development.

References

- https://solidity.readthedocs.io/en/latest

Fixes Review

This issue has been addressed in the following commits:

- https://github.com/bane-labs/bridge-evm-contracts/commit/dd999d58417acfc2954b436a219dd8ce89696cfe
- https://github.com/bane-labs/bridge-evm-contracts/commit/f292c0bf1fd8b8274e1417d4596af52dd2f1a7b8
- https://github.com/bane-labs/bridge-evm-contracts/pull/91

Lack of Event Index

Identifier	Category	Risk	State
NBE-10	Auditing and Logging	**Informative**	**Acknowledged**

Event indexing of smart contracts can be used to filter during the querying of events. This can be very useful when making dApps or in the off-chain processing of the events in our contract, as it allows filtering by specific addresses, making it much easier for developers to query the results of invocations.

Recommendations

- It could be convenient to review the `BridgeContract` contract to ensure that all the events have the necessary indexes for the correct functioning of the possible DApps. Addresses are usually the best argument to filter an event.
- It is advisable to add index in the `to` field of the events: `Deposit`, `Claimable` and `Claimed`.

Source Code References

- contracts/BridgeContract.sol#L48-L51

Wrong Event Emit

Identifier	Category	Risk	State
NBE-11	Auditing and Logging	**Low**	**Fixed**

The events emitted during the execution of the `lock` and `unlock` methods of the `BridgeImpl` contract are incorrect. Due to human error, these events are triggered by the opposite methods they are intended to represent. This misalignment can cause dApps or users relying on these events to respond incorrectly to the contract's locking and unlocking functions, leading to inappropriate behavior.

```
// Contract Locking

function lock() external onlySecurityGuard unlocked {
    _lock();
    emit Unlocked();
}

function unlock() external onlyGovernor {
    _unlock();
    emit Locked();
}
```

Events emitted in reverse

Recommendations

- Swap both events so that they are emitted in the correct methods.

Source Code References

- contracts/BridgeImpl.sol#L185-L193

Fixes Review

This issue has been addressed in the following pull request:

- https://github.com/bane-labs/bridge-evm-contracts/pull/113

Lack of neoN3Token Validation on TokenConfig

Identifier	Category	Risk	State
NBE-12	Data Validation	**Informative**	**Fixed**

The `_isValidConfig` method validates the configuration of a token but does not verify whether the `neoN3Token` field is correctly configured (i.e., not *address(0)*).

Using `_isValidConfig` without verifying the `neoN3Token` field can result in situations where an incomplete or incorrect configuration is deemed valid. This could potentially compromise system integrity in other parts of the code where this validation is crucial.

```
function _isValidConfig(
    StorageTypes.TokenConfig memory _config
) internal pure returns (bool) {
    // The fee must always be greater than 0.
    return
        _config.fee > 0 &&
        _config.minAmount > 0 &&
        _config.maxAmount > _config.minAmount &&
        _config.maxDeposits > 0;
}
```

_isValidConfig method

```
struct TokenConfig {
    address neoN3Token;
    uint256 fee;
    uint256 minAmount;
    uint256 maxAmount;
    uint256 maxDeposits;
    // Execution details
    ExecutionType executionType;
}
```

TokenConfig struct

It is important to mention that the exploitability is null due to the verification in the `registerToken` method, the consistency and robustness of the code are compromised, which could make the contract difficult to maintain and understand in the long term.

```
function registerToken(
    address _neoXToken,
    StorageTypes.TokenConfig calldata _tokenConfig
) external override onlyGovernor {
    if (_neoXToken == address(0)) revert InvalidTokenAddress();
    if (_tokenConfig.neoN3Token == address(0)) revert InvalidAddress();

    _registerToken(_neoXToken, _tokenConfig);
    emit TokenRegister(_neoXToken, _tokenConfig);
}
```

registerToken method validation

Source Code References

- contracts/library/StorageTypes.sol#L57-L58
- contracts/library/TokenBridgeLib.sol#L90-L99
- contracts/bridge/BridgeImpl.sol#L269

Recommendations

To ensure complete and consistent validation under all circumstances, it is recommended that you modify the `_isValidConfig` method to include verification of the `neoN3Token` field. An updated version of the method with this verification would be the following:

```
function _isValidConfig(
    StorageTypes.TokenConfig memory _config
) internal pure returns (bool) {
    // The fee must always be greater than 0.
    return
        _config.neoN3Token != address(0) &&
        _config.fee > 0 &&
        _config.minAmount > 0 &&
        _config.maxAmount > _config.minAmount &&
        _config.maxDeposits > 0;
}
```

Fixes Review

This issue has been addressed in the following commit:

- https://github.com/bane-labs/bridge-evm-contracts/pull/187/commits/a5b63b916ad65ad0da4259f76b8c33881365c29c

Lack of neoXToken Verification

Identifier	Category	Risk	State
NBE-13	Data Validation	**Informative**	**Fixed**

The `BridgeStorage` contract does not perform any check to verify that the `neoXToken` token exists in any of the internal methods `setTokenWithdrawalFee`, `setTokenMinWithdrawalAmount`, `setTokenMaxWithdrawalAmount`, `_setMaxTokenDeposits`.

These methods are called by the corresponding public methods of the `BridgeImpl` contract, where the previously mentioned lack of verifications is also observed. This allows the governor to set different values for non-existent tokens.

Recommendations

- Check that the token referenced by the `_neoXToken` argument exists before setting the corresponding values.

Source Code References

- contracts/bridge/BridgeStorage.sol#L248-L276
- contracts/bridge/BridgeImpl.sol#L545-L595

Fixes Review

The issue has been addressed in the following pull request:

- https://github.com/bane-labs/bridge-evm-contracts/pull/172

Annexes

Methodology

A code audit is a thorough examination of the source code of a project with the objective of identifying errors, discovering security breaches, or contraventions of programming standards. It is an essential component to the defense in programming, which seeks to minimize errors prior to the deployment of the product.

Red4Sec adopts a set of cybersecurity tools and best security practices to audit the source code of the smart contract by conducting a search for vulnerabilities and flaws.

The audit team performs an analysis on the functionality of the code, a manual audit, and automated verifications, considering the following crucial features of the code:

- The implementation conforms to protocol standards and adheres to best coding practices.
- The code is secure against common and uncommon vectors of attack.
- The logic of the contract complies with the specifications and intentions of the client.
- The business logic and the interactions with similar industry protocols do not contain errors or lead to dangerous situations to the integrity of the system.

In order to standardize the evaluation, the audit is executed by industry experts, in accordance with the following procedures:

Manual Analysis

- Manual review of the code, line-by-line, to discover errors or unexpected conditions.
- Assess the overall structure, complexity, and quality of the project.
- Search for issues based on the SWC Registry and known attacks.
- Review known vulnerabilities in the third-party libraries used.
- Analysis of the business logic and algorithms of the protocol to identify potential risk exposures.
- Manual testing to verify the operation, optimization, and stability of the code.

Automatic Analysis

- Scan the source code with static and dynamic security tools to search for known vulnerabilities.
- Manual verification of all the issues found by the tools and analyzes their impact.
- Perform unit tests and verify the coverage.

Vulnerabilities Severity

Red4Sec determines the severity of vulnerabilities found in risk levels according to the impact level defined by CVSSv3 (Common Vulnerability Scoring System) by the National Institute of Standards and Technology (NIST), classifying the risk of vulnerabilities on the following scale:

Severity	Description
Critical	Vulnerabilities that possess the highest impact over the systems, services, and/or sensitive information. The existence of these vulnerabilities is dangerous and should be fixed as soon as possible.
High	Vulnerabilities that could compromise severely compromise the service or the information it manages even if the vulnerability requires expertise to be exploited.
Medium	Vulnerabilities that on their own can have a limited impact and/or that combined with other vulnerabilities could have a greater impact.
Low	These vulnerabilities do not suppose a real risk for the systems. Also includes vulnerabilities which are extremely hard to exploit or whose impact on the service is low.
Informative	It covers various characteristics, information or behaviors that can be considered as inappropriate, without being considered as vulnerabilities by themselves.

Invest in Security, invest in your future