Security Audit Report

07/08/2024

Neo X - NeoVM Bridge Contracts

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.

Neo Contracts for **N3-X bridge** is the current implementation of the smart contracts developed in **Neow3j** and deployed in **Neo N3** 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 **Neo X - NeoVM Bridge Contracts** project.

The report includes specifics retrieved from the audit for all the existing vulnerabilities of **Neo X - NeoVM Bridge Contracts**. The performed analysis showed that the smart contract did contain critical and medium risk vulnerabilities.

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 **Neo X - NeoVM Bridge 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**:

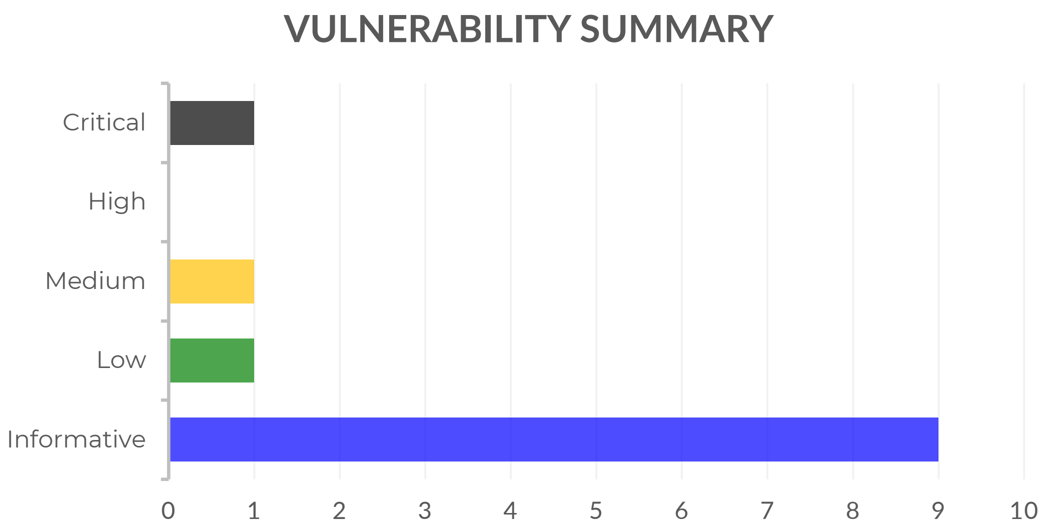
- https://github.com/bane-labs/bridge-neo-contracts
  - commit: `05239826a9b77bdbbc2fa49189d1fef633ba6a6a`

Executive Summary

The security audit against **Neo X - NeoVM Bridge Contracts** has been conducted between the following dates: **01/04/2024** and **05/06/2024**.

Once the analysis of the technical aspects has been completed, the performed analysis showed that the audited source code contained critical and medium risk vulnerabilities that were mitigated by the team.

During the analysis, a total of **12 vulnerabilities** were detected, these vulnerabilities have been classified by the following level of risks, defined in Vulnerabilities Severity annex.



Conclusions

Upon conducting the **Neo X NeoVM Bridge Contracts** audit, several vulnerabilities have been identified, ranging from critical, medium, to low risk, and informative elements. The overall security status of the project is concerning due to the presence of these vulnerabilities, specifically the critical and medium risk vulnerabilities. This scenario exposes the project to potential security breaches that could lead to significant financial and data losses.

The **Neo X** team has worked and resolved most of the issues identified and reported in this document, 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 critical risk identified is the possibility of Funds Draining via Deposit Manipulation as it directly impacts the project's financial security. This issue arises from a business logic vulnerability that, if exploited, could permit unauthorized withdrawal or manipulation of funds. A general solution to this issue would involve a comprehensive review of the smart contract's logic and the implementation of rigorous controls to protect against unauthorized transactions.

Furthermore, the Lack of Input Validation, although a low risk, is a notable issue. It could allow for the insertion of malicious or erroneous data into the contract. Implementing proper input validation methods can help to mitigate this risk.

Several other areas have been classified as “Informative” risks. These include issues such as Front Running Set Fee, Missing Logic on Update, Incomplete Interface Definition, Missing Manifest Information, Unsecured Ownership Transfer, Bad Coding Practices, Outdated Compiler, Lack of Documentation, and GAS Optimizations. While these may not pose immediate threats, they are indicative of a lack of best practices in the development of the smart contract. They could potentially lead to vulnerabilities in the future and should be addressed accordingly.

In conclusion, the smart contract, in its previous form, exposed the project to significant risks. We have advised to rectify the critical and medium risks immediately and also address the informative risks to ensure a robust and secure system. This should be done alongside comprehensive documentation and following best coding practices. Regular updates and audits of the smart contracts should also be carried out to ensure their safety and security.

As noted earlier and confirmed through the review of fixes, we can verify that the **Neo X** team has effectively addressed and resolved the majority of the issues identified and reported in this document.

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
NBN-01	Funds Draining via Deposit Manipulation	**Critical**	**Fixed**
NBN-02	Defective Handling of Deposit Fees	**Medium**	**Fixed**
NBN-03	Lack of Inputs Validation	**Low**	**Fixed**
NBN-04	Front Running Set Fee	**Informative**	**Fixed**
NBN-05	Missing Logic on Update	**Informative**	**Fixed**
NBN-06	Incomplete Interface Definition	**Informative**	**Acknowledged**
NBN-07	Missing Manifest Information	**Informative**	**Acknowledged**
NBN-08	Unsecured Ownership Transfer	**Informative**	**Fixed**
NBN-09	Bad Coding Practices	**Informative**	**Fixed**
NBN-10	Outdated Compiler	**Informative**	**Fixed**
NBN-11	Lack of Documentation	**Informative**	**Acknowledged**
NBN-12	GAS Optimizations	**Informative**	**Fixed**

Vulnerability details

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

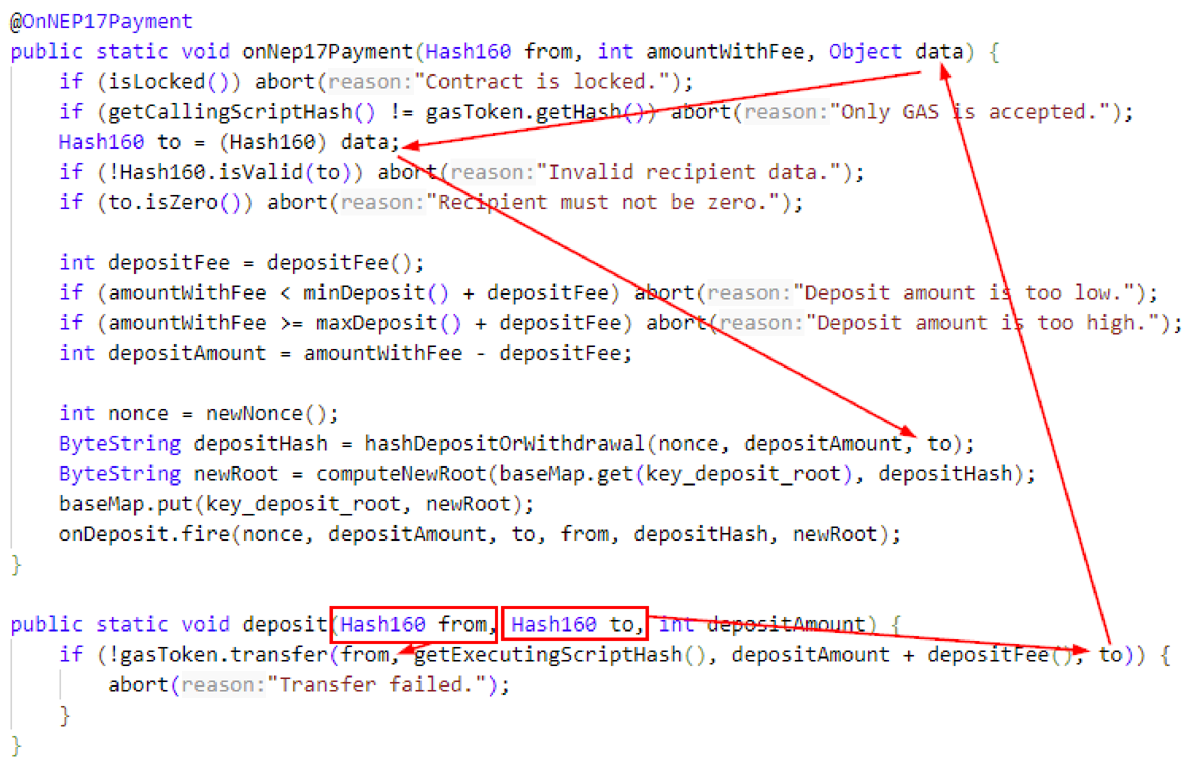
- Category
- Active
- Risk
- Description
- Recommendations

Funds Draining via Deposit Manipulation

Identifier	Category	Risk	State
NBN-01	Business Logic	**Critical**	**Fixed**

The `BridgeContract` contract can be drained by a malicious actor using the gas stored in the contract to simulate a deposit to an arbitrary account under the attacker's control.

The `deposit` method fails to authenticate that the `from` field matches the hash of the `BridgeContract` contract. Consequently, an attacker who can specify from the `from` of the contract's hash, as the contract itself acts as the `gasToken` caller during the transfer. This allows the attacker to be verified and send gas to the contract correctly. Successively, an account controlled by the attacker will be specified in the `to`, creating a deposit record that can be extracted from the other end of the bridge by draining the contract.



```java
@OnNEP17Payment
public static void onNep17Payment(Hash160 from, int amountWithFee, Object data) {
    if (isLocked()) abort(reason:"Contract is locked.");
    if (getCallingScriptHash() != gasToken.getHash()) abort(reason:"Only GAS is accepted.");
    Hash160 to = (Hash160) data;
    if (!Hash160.isValid(to)) abort(reason:"Invalid recipient data.");
    if (to.isZero()) abort(reason:"Recipient must not be zero.");

    int depositFee = depositFee();
    if (amountWithFee < minDeposit() + depositFee) abort(reason:"Deposit amount is too low.");
    if (amountWithFee >= maxDeposit() + depositFee) abort(reason:"Deposit amount is too high.");
    int depositAmount = amountWithFee - depositFee;

    int nonce = newNonce();
    ByteString depositHash = hashDepositOrWithdrawal(nonce, depositAmount, to);
    ByteString newRoot = computeNewRoot(baseMap.get(key_deposit_root), depositHash);
    baseMap.put(key_deposit_root, newRoot);
    onDeposit.fire(nonce, depositAmount, to, from, depositHash, newRoot);
}

public static void deposit(Hash160 from, Hash160 to, int depositAmount) {
    if (!gasToken.transfer(from, getExecutingScriptHash(), depositAmount + depositFee(), to)) {
        abort(reason:"Transfer failed.");
    }
}
```

Flow of the issue

Recommendations

- It is imperative to check that the contract's hash is not used during the `deposit` function.

Source Code References

- src/main/java/network/bane/BridgeContract.java#L173

Fixes Review

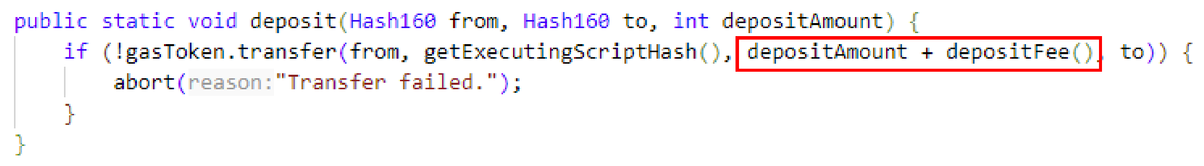
This issue has been addressed in the following pull request:

- https://github.com/bane-labs/bridge-neo-contracts/pull/56

Defective Handling of Deposit Fees

Identifier	Category	Risk	State
NBN-02	Business Logic	**Medium**	**Fixed**

A vulnerability has been identified in the `BridgeContract` smart contract, which affects the `deposit` function. This defect is associated with improper handling of transaction fees, which can result in failed transactions or the unauthorized transfer of funds greater than those intended by the user. The current contract logic adds a flat fee `depositFee` to the `depositAmount` the user wants to transfer. This results in the transfer of a total `totalAmount = depositAmount + depositFee` from the sender's balance.



```
public static void deposit(Hash160 from, Hash160 to, int depositAmount) {
    if (!gasToken.transfer(from, getExecutingScriptHash(), depositAmount + depositFee(), to)) {
        abort(reason:"Transfer failed.");
    }
}
```

Transfer greater than what was established by the user

The function does not reduce the fee of the amount the user wishes to send, leading to a possible overrun of the sender's balance. In scenarios where the user intends to precisely transfer their remaining balance, the sum of the fee will cause the transaction to exceed their balance, resulting in an *"Insufficient Balance"* error or the transfer of more tokens than the user intended, ultimately depleting their balance.

Given the flexibility for an administrator to adjust the transfer fee without clear (`setDepositFee`), there is a significant risk. A malicious administrator could raise the fee to an exorbitant value just before a planned transaction. Consequently, all tokens belonging to the sender might be transferred without their explicit consent, especially if the administrator is aware of pending high-value transactions.

Recommendations

- Ensure that the fee is subtracted from the amount the user wants to send, instead of added to it.
- Implement strict controls to regulate who has the ability to change the fee and the circumstances under which such changes can occur, possibly requiring a vote or a grace period before adjustments take effect.

Source Code References

- src/main/java/network/bane/BridgeContract.java#L173

Fixes Review

The issue has been addressed in the following pull request:

- https://github.com/bane-labs/bridge-neo-contracts/pull/97

Lack of Inputs Validation

Identifier	Category	Risk	State
NBN-03	Data Validation	**Low**	**Fixed**

Certain methods of the different contracts in the `BridgeContract` project do not properly check the arguments, which can lead to major errors.

UInt160 Validations

The general execution of the `BridgeManagementContract` and the `BridgeContract` contracts is not checking the inputs nor the integrity of most of the `UInt160` types.

Source code references

In certain cases it is not checked that the value received is `null`.

- src/main/java/network/bane/structs/Withdrawal.java#L13-L15
- src/main/java/network/bane/structs/BridgeDeploymentData.java#L14

In another scenario it is not verified that it is a contract:

- src/main/java/network/bane/structs/BridgeDeploymentData.java#L14

Additionally, in various methods it is convenient to check that the value is not `IsZero`, leaving the verification as: `hash.IsValid && !hash.IsZero`.

- src/main/java/network/bane/structs/Withdrawal.java#L13-L15
- src/main/java/network/bane/structs/BridgeDeploymentData.java#L14
- src/main/java/network/bane/BridgeManagementContract.java#L90-L91
- src/main/java/network/bane/BridgeManagementContract.java#L100
- src/main/java/network/bane/BridgeManagementContract.java#L106
- src/main/java/network/bane/BridgeManagementContract.java#L138
- src/main/java/network/bane/BridgeManagementContract.java#L144

Set Validators Issues

The verification of `const_max_validators` is not conducted when establishing the validators using the `setValidators` method. However, this check is carried out through initialization with the `deploy` method. Conversely, in the `deploy` method, potential duplicates existing within the validators are not verified, while in `setValidators`, such duplicates are checked.

Source code references

- src/main/java/network/bane/BridgeManagementContract.java#L84-L88
- src/main/java/network/bane/BridgeManagementContract.java#L110

Recommendations

- It is advisable to always check the format of the arguments before using their value, otherwise, a user could send unexpected values through these arguments, being able to make injections or arbitrary reads from the storage, either intentionally or not.

Fixes Review

This issue has been addressed in the following pull request:

- https://github.com/bane-labs/bridge-neo-contracts/pull/70

Front Running Set Fee

Identifier	Category	Risk	State
NBN-04	Timing and State	**Informative**	**Fixed**

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

The `governor` can establish the deposit fee without set maximums through the `setDepositFee` method. This together with the fact that the change is applied from the same block as its call, implies that the governor can conduct a front-running attack in order to extract the entire balance from users who are in the process of depositing funds.

Recommendations

- Use a time lock or apply the administrative changes in the next block.

Source Code References

- src/main/java/network/bane/BridgeContract.java#L439

Fixes Review

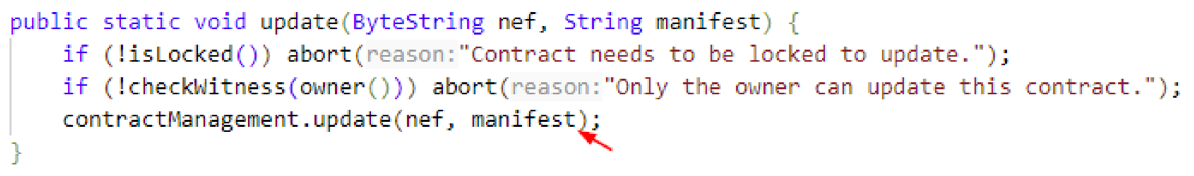
The issue has been addressed in the following pull request:

- https://github.com/bane-labs/bridge-neo-contracts/pull/100

Missing Logic on Update

Identifier	Category	Risk	State
NBN-05	Design Weaknesses	**Informative**	**Fixed**

The `update` method within the native contract `contractManagement` has an argument named "`data`", which is used in the `deploy` method to obtain essential values required for executing the proper update of the contract. However, this argument is not present in any of the audited contracts.



```
public static void update(ByteString nef, String manifest) {
    if (!isLocked()) abort(reason:"Contract needs to be locked to update.");
    if (!checkWitness(owner())) abort(reason:"Only the owner can update this contract.");
    contractManagement.update(nef, manifest);
}
```

Update without data argument

Recommendations

- Include the `data` argument in the `update` methods.

Source Code References

- src/main/java/network/bane/BridgeManagementContract.java#L200
- src/main/java/network/bane/BridgeContract.java#L446

Fixes Review

This issue has been addressed in the following pull request:

- https://github.com/bane-labs/bridge-neo-contracts/pull/94

Incomplete Interface Definition

Identifier	Category	Risk	State
NBN-06	Design Weaknesses	**Informative**	**Acknowledged**

The `BridgeManagement` interface does not currently reflect all available methods within the associated smart contract.

This deficiency can lead to the following problems:

- **Unit Testing:** Without a complete interface definition, it is not possible to ensure that all contract functionalities are adequately tested.
- **Compatibility:** Developers relying on this interface to interact with similar contracts could face runtime errors due to calls to methods not specified in the interface.

Recommendations

- Make sure that the interface contains all the public methods within the smart contract.

Source Code References

- src/main/java/network/bane/interfaces/BridgeManagement.java#L8

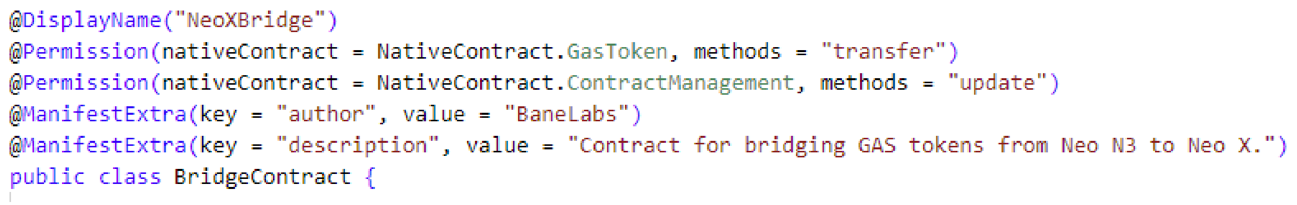
Missing Manifest Information

Identifier	Category	Risk	State
NBN-07	Codebase Quality	**Informative**	**Acknowledged**

The Red4Sec team has detected that the `BridgeContract` contract does not properly specify the information related to the smart contract in its manifest.

Currently, all the contract information found in the manifest (*source*) belongs to the developer of the project. Therefore, it is recommended to customize said content by including your own information, which will also provide the users with relevant information about the project.

For the audited smart contracts, it has been observed that the `source` attribute is not included. The definition of a good manifest increases the trust and improves the SEO of the contract.



```
@DisplayName("NeoXBridge")
@Permission(nativeContract = NativeContract.GasToken, methods = "transfer")
@Permission(nativeContract = NativeContract.ContractManagement, methods = "update")
@ManifestExtra(key = "author", value = "BaneLabs")
@ManifestExtra(key = "description", value = "Contract for bridging GAS tokens from Neo N3 to Neo X.")
public class BridgeContract {
```

Recommendations

- Adding all possible metadata to the contracts favors indexing, traceability and the audit by users, which conveys greater confidence to the user.

References

- https://github.com/neo-project/proposals/blob/master/nep-16.mediawiki#source

Source Code References

- src/main/java/network/bane/BridgeManagementContract.java#L29
- src/main/java/network/bane/BridgeContract.java#L41

Unsecured Ownership Transfer

Identifier	Category	Risk	State
NBN-08	Data Validation	**Informative**	**Fixed**

The process of modifying an owner is very delicate, as it directly affects the governance of our contract and, consequently, the entire project's integrity. For this reason, it is advised to adjust the owner's modification logic, to one that allows to verify that the new owner is in fact valid and does exist.

In N3, the owner modification process can be conducted in a single transaction without unnecessary prolongation. Transactions in N3 can be signed by multiple accounts and `CheckWitness` could be called with the proposed owner and the current owner simultaneously, provided the transaction scope is properly configured.

Source Code References

- src/main/java/network/bane/BridgeManagementContract.java#L100

Fixes Review

This issue has been addressed in the following pull request:

- https://github.com/bane-labs/bridge-neo-contracts/pull/69

Bad Coding Practices

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

During the smart contract audit, certain bad practices have been detected throughout the code that should be improved. It is strongly advised to consider the adoption of improved coding styles and best practices for overall code improvement.

Abi Not Descriptive

The events outlined in the contract do not define the names of the arguments, they are only specified in the comments. However, these names are not ported to the contract's abi, which is the interface that makes their use easier for the developer. It is necessary that the generated abi has the descriptive names of the arguments that each event handles.

```
"events" : [ {
      "name" : "Deposit",
      "parameters" : [ {
        "name" : "arg1",
        "type" : "Integer"
      }, {
        "name" : "arg2",
        "type" : "Integer"
      }, {
        "name" : "arg3",
        "type" : "Hash160"
      }, {
        "name" : "arg4",
        "type" : "Hash160"
      }, {
        "name" : "arg5",
        "type" : "ByteArray"
      }, {
        "name" : "arg6",
        "type" : "ByteArray"
      } ]
    } ...
```

Source code references:

- src/main/java/network/bane/BridgeContract.java#L123

Use Storage Serialization

The logic related to the storage of the claims manually stores the `to` and `amount` values. This approach not only complicates the maintenance of the code but also adds to the serialization cost of an object using the corresponding syscalls.

Source code references:

- src/main/java/network/bane/BridgeContract.java#L308

Fixes Review

This issue has been addressed in the following commit:

- https://github.com/bane-labs/bridge-neo-contracts/commit/b2f5b4a3396231baada69d71d92f0c6c246c4779

Outdated Compiler

Identifier	Category	Risk	State
NBN-10	Outdated Software	**Informative**	**Fixed**

Neo's Java compiler regularly releases new versions, and it is important to stay updated. Operating with an outdated compiler version can pose significant issues, particularly if there are known errors or vulnerabilities associated with that version.

It has been verified that the project is compiled for the version `3.21.2` of the `neow3j` framework, and it uses Neo's framework `3.22.1`. This version can be updated and it contains major improvements.

Source Code References

- build.gradle#L3

Fixes Review

This issue has been addressed and the `neow3j` dependency is up-to-date in the `develop` branch.

Lack of Documentation

Identifier	Category	Risk	State
NBN-11	Testing and Documentation	**Informative**	**Acknowledged**

The project does not contain technical documentation of any sort, nor execution flow diagrams, class diagrams or code properly commented. This is a bad practice that complicates understanding the project and makes it difficult for the team of auditors to analyze the functionalities, since there is no technical documentation to check if the current implementation meets the needs and purpose of the project.

Having updated and accurate documentation of the project is an essential aspect for open source projects, in fact it is closely related with the adoption and contribution of the project by the community.

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

GAS Optimizations

Identifier	Category	Risk	State
NBN-12	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 the N3 blockchain, GAS is an execution fee which is used to compensate the network for the computational resources required to power smart contracts. If the network usage is increasing, so will the value of GAS optimization.

Static Variables

The static variables in N3 are processed at the beginning of the execution of the smart contract, so they must be carefully used. Although these static variables are not used for the call that will be made, their initialization will be processed anyway, and they will occupy elements in the stack with the corresponding cost that this entails.

It is a good practice to reduce the static variables, either through constants or through methods that return the desired value.

Source code references

- src/main/java/network/bane/BridgeManagementContract.java#L35-L49
- src/main/java/network/bane/BridgeContract.java#L44-L73

Cache Array Length Size

Every time an array iteration occurs in a loop, the neo compiler will read the array's length. Thus, this is an additional operation that can be improved in order to save gas.

Source code references

- src/main/java/network/bane/BridgeManagementContract.java#L119
- src/main/java/network/bane/BridgeManagementContract.java#L130
- src/main/java/network/bane/BridgeContract.java#L337
- src/main/java/network/bane/BridgeContract.java#L282
- src/main/java/network/bane/BridgeContract.java#L294

Logic Optimization

The code that verifies that the `nonce` of the `withdraw` matches the current `nonce` is redundant, since the same check is performed again in the call to `subsequentNonces`.

Source code references:

- src/main/java/network/bane/BridgeContract.java#L220-L221

Fixes Review

This issue has been addressed in the following pull request:

- https://github.com/bane-labs/bridge-neo-contracts/pull/68

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.

RED4SEC

Invest in Security, invest in your future