OMNISCIA

February 20, 2023

SMART CONTRACT AUDIT REPORT

Qanx Token Round #2

omniscia.io

info@omniscia.io

Online report: qanx-token-round-2

Round 2 Security Audit

Audit Overview

We were tasked with performing a secondary audit round on the Qanx token implementation which represents a gradually-unlockable token that is meant to be distributed via signed "cheques".

Over the course of the audit, we did not identify any security vulnerabilities in the code and our findings consist of purely gas and style related recommendations.

We advise the Qanx Token team to consider all optimizational exhibits identified in the report.

Post-Audit Conclusion

The Qanx Token team iterated through all findings within the report and provided us with a revised commit hash to evaluate all exhibits on.

We evaluated all alleviations performed by Qanx Token and have identified that all exhibits have been adequately dealt with no outstanding issues remaining in the report.

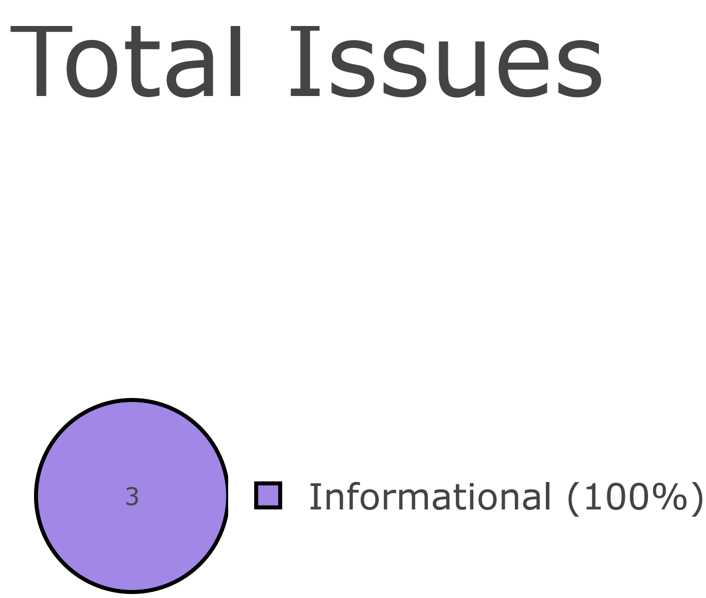
Contracts Assessed

Files in Scope	Repository	Commit(s)
Context.sol (CTX)	qanx-token	54924320c9, f779be3f7c
ERC20.sol (ERC)	qanx-token	54924320c9, f779be3f7c
IERC20.sol (IER)	qanx-token	54924320c9, f779be3f7c
IERC20Metadata.sol (IEC)	qanx-token	54924320c9, f779be3f7c
QANX.sol (QAN)	qanx-token	54924320c9, f779be3f7c

Audit Synopsis

Severity	Identified	Alleviated	Partially Alleviated	Acknowledged
Unknown	0	0	0	0
Informational	3	3	0	0
Minor	0	0	0	0
Medium	0	0	0	0
Major	0	0	0	0

During the audit, we filtered and validated a total of **2 findings utilizing static analysis** tools as well as identified a total of **1 findings during the manual review** of the codebase.



The list below covers each segment of the audit in depth and links to the respective chapter of the report:

- 💻 **Compilation**
- 🔍 **Static Analysis**
- 👁 **Manual Review**
- 🖋 **Code Style**

Compilation

The project utilizes `hardhat` as its development pipeline tool, containing an array of tests and scripts coded in JavaScript.

To compile the project, the `compile` command needs to be issued via the `npx` CLI tool to `hardhat`:

```bash
npx hardhat compile
```

The `hardhat` tool automatically selects Solidity version `0.8.17` based on the version specified within the `hardhat.config.js` file.

The project does not contain any discrepancies with regards to the Solidity version used and all contracts have been set to the same `pragma` version.

All `pragma` versions are locked to `0.8.17` (`=0.8.17`), the same version utilized for our static analysis as well as optimizational review of the codebase.

During compilation with the `hardhat` pipeline, no errors were identified that relate to the syntax or bytecode size of the contracts.

Static Analysis

The execution of our static analysis toolkit identified **17 potential issues** within the codebase of which **15 were ruled out to be false positives** or negligible findings.

The remaining **2 issues** were validated and grouped and formalized into the **2 exhibits** that follow:

ID	Severity	Addressed	Title
CTX-01S	Informational	Nullified	Unused Code
ERC-01S	Informational	Nullified	Unused Code

Manual Review

A **thorough line-by-line review** was conducted on the codebase to identify potential malfunctions and vulnerabilities in Qanx Token's token.

As the project at hand implements a signature-based distribution mechanism of both vested and immediately unlocked tokens, intricate care was put into ensuring that the **flow of funds within the system conforms to the specifications and restrictions** laid forth within the protocol's specification.

We validated that **all state transitions of the system occur within sane criteria** and that all rudimentary formulas within the system execute as expected. We **did not pinpoint any security related vulnerabilities** during this audit round of the codebase.

Additionally, the system was investigated for any other commonly present attack vectors such as re-entrancy attacks, mathematical truncations, logical flaws and **ERC / EIP** standard inconsistencies. The documentation of the project was satisfactory to an exemplary extent, containing in-line documentation for all top-level declarations of the `QANX` contract as well as in-line documentation within its functions.

A total of **1 findings** were identified over the course of the manual review of which **no findings** concerned the behaviour and security of the system. The non-security related findings, such as optimizations, are included in the separate **Code Style** chapter.

Code Style

During the manual portion of the audit, we identified **1 optimizations** that can be applied to the codebase that will decrease the operational cost associated with the execution of a particular function and generally ensure that the project complies with the latest best practices and standards in Solidity.

Additionally, this section of the audit contains any opinionated adjustments we believe the code should make to make it more legible as well as truer to its purpose.

These optimizations are enumerated below:

ID	Severity	Addressed	Title
QAN-01C	Informational	Yes	Inexistent Event Emission

Context Static Analysis Findings

CTX-01S: Unused Code

Type	Severity	Location
Gas Optimization	Informational	Context.sol:L19-L21

Description:

The referenced function remains unutilized in the codebase.

Example:

contracts/Context.sol

```sol
abstract contract Context {
    function _msgSender() internal view virtual returns (address) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes calldata) {
        return msg.data;
    }
}
```

Recommendation:

We advise it to be safely omitted from it, optimizing its deployment cost.

Alleviation:

The Qanx team stated that they wish to retain the current implementation in the codebase to maintain a one-to-one relation with the original code of OpenZeppelin and to avoid touching sensitive dependencies. As the hash of the file will change if the code is removed and Qanx wishes to retain its dependencies intact, we consider this exhibit nullified.

ERC20 Static Analysis Findings

ERC-01S: Unused Code

Type	Severity	Location
Gas Optimization	Informational	ERC20.sol:L282-L298

Description:

The referenced function remains unutilized in the codebase.

Example:

contracts/ERC20.sol

SOL

```solidity
282 function _burn(address account, uint256 amount) internal virtual {
283     require(account != address(0), "ERC20: burn from the zero address");
284 
285     _beforeTokenTransfer(account, address(0), amount);
286 
287     uint256 accountBalance = _balances[account];
288     require(accountBalance >= amount, "ERC20: burn amount exceeds balance");
289     unchecked {
290         _balances[account] = accountBalance - amount;
291         // Overflow not possible: amount <= accountBalance <= totalSupply.
292         _totalSupply -= amount;
293     }
294 
295     emit Transfer(account, address(0), amount);
296 
297     _afterTokenTransfer(account, address(0), amount);
298 }
```

Recommendation:

We advise it to be safely omitted from it, optimizing its deployment cost.

Alleviation:

The Qanx team stated that they wish to retain the current implementation in the codebase to maintain a one-to-one relation with the original code of OpenZeppelin and to avoid touching sensitive dependencies. As the hash of the file will change if the code is removed and Qanx wishes to retain its dependencies intact, we consider this exhibit nullified.

QANX Code Style Findings

QAN-01C: Inexistent Event Emission

Type	Severity	Location
Language Specific	Informational	QANX.sol:L72-L75

Description:

The linked function adjusts a sensitive contract variable yet does not emit an event for it.

Example:

contracts/QANX.sol

```sol
72 function setChequeSigner(address _newChequeSigner) external {
73     require(msg.sender == chequeSigner && _newChequeSigner != address(0), "Invalid che
74     chequeSigner = _newChequeSigner;
75 }
```

Recommendation:

We advise an `event` to be declared and correspondingly emitted to ensure off-chain processes can properly react to this system adjustment.

Alleviation:

A `ChequeSignerUpdated` event was introduced to the codebase and is now properly emitted in the `setChequeSigner` function, alleviating this exhibit in full.

Finding Types

A description of each finding type included in the report can be found below and is linked by each respective finding. A full list of finding types Omniscia has defined will be viewable at the central audit methodology we will publish soon.

Input Sanitization

As there are no inherent guarantees to the inputs a function accepts, a set of guards should always be in place to sanitize the values passed in to a particular function.

Indeterminate Code

These types of issues arise when a linked code segment may not behave as expected, either due to mistyped code, convoluted if blocks, overlapping functions / variable names and other ambiguous statements.

Language Specific

Language specific issues arise from certain peculiarities that the Circom language boasts that discerns it from other conventional programming languages.

Curve Specific

Circom defaults to using the BN128 scalar field (a 254-bit prime field), but it also supports BSL12-381 (which has a 255-bit scalar field) and Goldilocks (with a 64-bit scalar field). However, since there are no constants denoting either the prime or the prime size in bits available in the Circom language, some Circomlib templates like `Sign` (which returns the sign of the input signal), and `AliasCheck` (used by the strict versions of `Num2Bits` and `Bits2Num`), hardcode either the BN128 prime size or some other constant related to BN128. Using these circuits with a custom prime may thus lead to unexpected results and should be avoided.

Code Style

In these types of findings, we identify whether a project conforms to a particular naming convention and whether that convention is consistent within the codebase and legible. In case of inconsistencies, we point them out under this category. Additionally, variable shadowing falls under this category as well which is identified when a local-level variable contains the same name as a toplevel variable in the circuit.

Mathematical Operations

This category is used when a mathematical issue is identified. This implies an issue with the implementation of a calculation compared to the specifications.

Logical Fault

This category is a bit broad and is meant to cover implementations that contain flaws in the way they are implemented, either due to unimplemented functionality, unaccounted-for edge cases or similar extraordinary scenarios.

Privacy Concern

This category is used when information that is meant to be kept private is made public in some way.

Proof Concern

Under-constrained signals are one of the most common issues in zero-knowledge circuits. Issues with proof generation fall under this category.

Disclaimer

The following disclaimer applies to all versions of the audit report produced (preliminary / public / private) and is in effect for all past, current, and future audit reports that are produced and hosted under Omniscia:

IMPORTANT TERMS & CONDITIONS REGARDING OUR SECURITY AUDITS/REVIEWS/REPORTS AND ALL PUBLIC/PRIVATE CONTENT/DELIVERABLES

Omniscia ("Omniscia") has conducted an independent security review to verify the integrity of and highlight any vulnerabilities, bugs or errors, intentional or unintentional, that may be present in the codebase that were provided for the scope of this Engagement.

Blockchain technology and the cryptographic assets it supports are nascent technologies. This makes them extremely volatile assets. Any assessment report obtained on such volatile and nascent assets may include unpredictable results which may lead to positive or negative outcomes.

In some cases, services provided may be reliant on a variety of third parties. This security review does not constitute endorsement, agreement or acceptance for the Project and technology that was reviewed. Users relying on this security review should not consider this as having any merit for financial advice or technological due diligence in any shape, form or nature.

The veracity and accuracy of the findings presented in this report relate solely to the proficiency, competence, aptitude and discretion of our auditors. Omniscia and its employees make no guarantees, nor assurance that the contracts are free of exploits, bugs, vulnerabilities, deprecation of technologies or any system / economical / mathematical malfunction.

This audit report shall not be printed, saved, disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Omniscia.

All the information/opinions/suggestions provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report.

Information in this report is provided 'as is'. Omniscia is under no covenant to the completeness, accuracy or solidity of the contracts reviewed. Omniscia's goal is to help reduce the attack vectors/surface and the high level of variance associated with utilizing new and consistently changing technologies.

Omniscia in no way claims any guarantee, warranty or assurance of security or functionality of the technology that was in scope for this security review.

In no event will Omniscia, its partners, employees, agents or any parties related to the design/creation of this security review be ever liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this security review.

Cryptocurrencies and all other technologies directly or indirectly related to cryptocurrencies are not standardized, highly prone to malfunction and extremely speculative by nature. No due diligence and/or safeguards may be insufficient and users should exercise maximum caution when participating and/or investing in this nascent industry.

The preparation of this security review has made all reasonable attempts to provide clear and actionable recommendations to the Project team (the “client”) with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts in scope for this engagement.

It is the sole responsibility of the Project team to provide adequate levels of test and perform the necessary checks to ensure that the contracts are functioning as intended, and more specifically to ensure that the functions contained within the contracts in scope have the desired intended effects, functionalities and outcomes, as documented by the Project team.

All services, the security reports, discussions, work product, attack vectors description or any other materials, products or results of this security review engagement is provided "as is" and "as available" and with all faults, uncertainty and defects without warranty or guarantee of any kind.

Omniscia will assume no liability or responsibility for delays, errors, mistakes, or any inaccuracies of content, suggestions, materials or for any loss, delay, damage of any kind which arose as a result of this engagement/security review.

Omniscia will assume no liability or responsibility for any personal injury, property damage, of any kind whatsoever that resulted in this engagement and the customer having access to or use of the products, engineers, services, security report, or any other other materials.

For avoidance of doubt, this report, its content, access, and/or usage thereof, including any associated services or materials, shall not be considered or relied upon as any form of financial, investment, tax, legal, regulatory, or any other type of advice.