Audit Report

Helix xToken

v1.0

July 4, 2024

Table of Contents

Table of Contents **2**
License **3**
Disclaimer **3**
Introduction **5**
Purpose of This Report 5
Codebase Submitted for the Audit 5
Methodology 6
Functionality Overview 6
How to Read This Report **7**
Code Quality Criteria **8**
Summary of Findings **9**
Detailed Findings **10**
1. Undelivered transfers require fee to be refunded 10
2. Centralization risks 10
3. Daily limits are not communicated across the bridge, leading to inefficiencies 11
4. Input validation could be improved 12
5. The contract locks up ETH received 12
6. Redundant logic in the updateXToken function when approving xToken 13
7. Excessive storage usage in XTokenBacking contract 13
8. Transfer state mappings could be streamlined 13
9. Miscellaneous 14

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by ITERING TECH PTE. LTD. to perform a security audit of Helix xToken.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/helix-bridge/xtoken-monorepo
Commit	`9263b353956994d9d7a7fd323446753509150914`
Scope	Contracts in `packages/xtoken-contract/contracts`, excluding the templates in `packages/xtoken-contract/contracts/templates`
Fixes verified at commit	`e47fcf3f57ca2c247eedb19746d2f0ef4db2d179` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Helix xToken is a Bridge as a Service platform for cross-chain token mapping bridge services.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium	The contracts use cross-chain bridging solutions that transfer ABI-encoded data, increasing the overall complexity.
Code readability and clarity	Medium	-
Level of documentation	Medium-High	Most functions are documented, external documentation was provided.
Test coverage	High	`93.27%` test coverage

Summary of Findings

No	Description	Severity	Status
1	Undelivered transfers require fee to be refunded	Minor	Partially Resolved
2	Centralization risks	Minor	Acknowledged
3	Daily limits are not communicated across the bridge	Minor	Acknowledged
4	Input validation could be improved	Minor	Resolved
5	The contracts locks up ETH received	Informational	Resolved
6	Redundant logic in the `updateXToken` function when approving xToken	Informational	Resolved
7	Excessive storage usage in `XTokenBacking` contract	Informational	Resolved
8	Transfer state mappings could be streamlined	Informational	Acknowledged
9	Miscellaneous	Informational	Resolved

Detailed Findings

1. Undelivered transfers require fee to be refunded

Severity: Minor

In `packages/xtoken-contract/contracts/base/XTokenBacking.sol:121` and `packages/xtoken-contract/contracts/base/XTokenIssuing.sol:82` transfers could fail due to the daily limit set by the DAO. This is not in control of the sender because the limit could be set or consumed after the transfer has been initiated but before it is executed.

The user is free to request a refund of the transfer using the function `xRollbackBurnAndXUnlock` or `xRollbackLockAndXIssue`. However, this requires covering the `protocolFee`.

It also requires having an account on the target chain, namely `_rollbackAccount`, to submit the transaction. This condition should be documented properly and made clear to the users of the bridge.

Recommendation

We recommend checking the status of `expendDailyLimit` calls and call the function `xRollbackBurnAndXUnlock` or `xRollbackLockAndXIssue` automatically by the contract covering the `protocolFee` from the contracts treasury.

Status: Partially Resolved

The protocol fee was removed from the contracts, which limits the impact of the issue.

2. Centralization risks

Severity: Minor

The contracts contain a few centralization concerns that users should be aware of:

- The `xToken` contracts implement the `guard` role. While it might be intended as a safety measure to stop an attack after it is deployed and before it is executed, applying it during a real incident creates a lot of post-incident remediation work. This is caused by ultimately replacing all receivers, both on backing and issuing chains, by the guard. After a potential incident all the transfers would require manual completion or manual refund.

- The `operator` role, which is currently an externally owned account set by the DAO, has the privilege to call the `pause`, `unpause`, and `setProtocolFee` functions at any time. This includes disabling slashing, liquidity withdrawals, and setting protocol fees to any amount at any time.

- The function `updateXToken` could be used to make issued tokens non-redeemable.

Recommendation

We recommend re-using the already defined functions `pause` or `_setDailyLimit`, instead of introducing an additional `guard` role. For the specific scenario of attacks mitigation, `_setDailyLimit` should be invoked with limit `0`, effectively pausing the bridge for one specific token. While `pause` is simpler and useful for freezing a side of the bridge in a whole network, there are cases when finely-grained freeze using `_setDailyLimit` is preferred. Compared to the `guard` role, neither `pause` nor `_setDailyLimit` requires manual refunding of suspended transfers.

Moreover, we recommend explicitly describing the operator's role in the documentation and removing the `updateXToken` function.

Status: Acknowledged

3. Daily limits are not communicated across the bridge, leading to inefficiencies

Severity: Minor

In `packages/xtoken-contract/contracts/base/XTokenBridgeBase.sol:163`, the `setDailyLimit` function is defined and can be used to limit the amount of tokens issued or unlocked during a day. In particular, this limit is checked via calls to function `expendDailyLimit` in `packages/xtoken-contract/contracts/base/XTokenBacking.sol:121` and `packages/xtoken-contract/contracts/base/XTokenIssuing.sol:82`.

When this limit is reached, the transfer can be refunded on the other side of the bridge. However, this introduces several inefficiencies, since the following steps are or need to be unnecessarily executed:

- the tokens have been locked or burned on the remote chain
- the underlying messaging channel has been used to pass the transfer request
- the user has spent fees which will not be refunded
- one more message passing back across the bridge is needed to refund the tokens
- the user needs either to request the refund and cover the protocol fee, or submit a proposal to the DAO which would take time to be processed

Recommendation

We recommend propagating the daily limits across the bridge and using them to reject transfer requests before locking or burning the tokens and charging the fees.

Status: Acknowledged

4. Input validation could be improved

Severity: Minor

In several of functions, the input is not properly validated:

- In `lockAndXIssue` within the file `packages/xtoken-contract/contracts/base/XTokenBacking.sol`, it is not validated that the amount is greater than zero. A value of zero will revert on the target chain, resulting in wasted gas/transfer costs.

- The validation that `_amount` is greater than zero in `packages/xtoken-contract/contracts/base/XTokenIssuing.sol:81` should be moved to the function `lockAndXIssue` in `packages/xtoken-contract/contracts/base/XTokenBacking.sol` in order to avoid the redundant lock.

- In other functions of the `xToken` contracts, it is not checked whether the submitted addresses and/or amounts are not equal to zero. This can lead to unintended behavior in out of scope contracts.

Recommendation

We recommend adding additional input validations.

Status: Resolved

5. The contract locks up ETH received

Severity: Informational

The `XTokenBridgeBase` contract is able to receive ETH. However, there is currently no way to retrieve ETH from the contract. The funds will be locked up.

Recommendation

We recommend removing the `receive` function if the contract is not supposed to handle ETH. Otherwise, we recommend adding functionality to withdraw/use ETH from the contract.

Status: Resolved

6. Redundant logic in the `updateXToken` function when approving xToken

Severity: Informational

The `updateXToken` function of the `XTokenIssuing` contract contains the following check:

```
require(IXToken(_xToken).approve(address(this), type(uint256).max)
== true, "approve xtoken failed");
```

This check contains a redundant condition which reduces readability and increases the gas fees. There is no need to check whether `approve` returns true or false, as the `require` itself already does this.

Recommendation

We recommend removing the `== true` condition from the `require` condition.

Status: Resolved

7. Excessive storage usage in `XTokenBacking` contract

Severity: Informational

In `packages/xtoken-contract/contracts/base/XTokenBacking.sol:15` the mapping `originalToken2xTokens` from type `bytes` to type `address` is defined. This mapping is used to store addresses of the mapped tokens.

However, the only reading of this mapping in `XTokenBacking.sol:63` discards the actual value and only checks that it is not zero. This validation could be done during the address storage.

Recommendation

We recommend using the `bool` type for the values of the mapping to reduce the amount of data to store and query.

Status: Resolved

8. Transfer state mappings could be streamlined

Severity: Informational

In `packages/xtoken-contract/contracts/base/XTokenBridgeBase.sol:39` the mapping `requestInfos` is defined. This mapping is used to track if a transfer was requested or refunded. Further, in the same file, in line `44` the mapping `filledTransfers` is

defined, which is used to track the delivered and refunded states of a transfer. These two mappings are used intricately to check all possible cases in the transfer flow.

However, all possible states of a transfer could be easily enumerated as `REQUESTED`, `DELIVERED`, `REFUND_REQUESTED`, and `REFUNDED`.

Recommendation

We recommend merging the mappings used to track state of a transfer and unifying the possible states to reduce complexity of the codebase.

Status: Acknowledged

9. Miscellaneous

Severity: Informational

- The parameter name `originalSender` is misleadingly used in several occurrences, where it is not related to the "original token":
  - It can mean the party redeemed the `xToken` in `XTokenBacking.sol:114` and `XTokenIssuing.sol:134`
  - It can mean a sender on either chain in `XTokenBridgeBase.sol:150`

- There are various typos and grammatical errors in the contracts. We recommend to:
  - Replace `request exist` with `this request already exists`.
  - Replace `request not exist` with `this request does not exist`.
  - Replace `request has been refund` with `this request has already been refunded`.
  - Replace `issuxToken` with `issueXToken`.
  - Replace `not enough fee` with `not enough fees`.

Recommendation

We recommend fixing these items, and also spell-checking all String constants, comments, variable names and error messages. Ensuring clarity of any human-readable text is a preemptive measure for establishing better readability and hence maintainability of the codebase.

Status: Resolved