SOLIDIFIED

Audit Report for SKALE - November 1, 2022

Summary

Audit Report prepared by Solidified covering security features that were added to the IMA bridge.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on October 18, 2022, and the results are presented here.

Audited Files

The source code has been supplied in the following source code repository:

Repo: https://github.com/skalenetwork/IMA

NOTE: The scope of this audit was reduced to the following features added to the existing bridge contract:

1. Pull Request 1209 at commit hash 40d2f0e
2. Pull Request 1234 at commit hash b884d65

The reviewed mitigations were provided in the following pull requests:

- 1304 (issue 1)
- 1310 (issue 2)
- 1307 (issue 4)
- 1305 (issue 8)
- 1306 (issue 9)

Findings

Smart contract audits are an important step to improve 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 a smart contract 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**.

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

Criteria	Status	Comment
Code complexity	Medium	-
Code readability and clarity	High	-
Level of Documentation	High	-
Test Coverage	High	-

Issues Found

Solidified found that the SKALE contracts contain 1 critical issue, 2 major issues, 1 minor issue, and 5 informational notes.

We recommend issues are amended, while informational notes are up to the team's discretion, as they refer to best practices.

Issue #	Description	Severity	Status
1	DepositBoxERC20.sol: retrieveFor always reverts in some scenarios, making withdrawals impossible	Critical	Fixed
2	DepositBoxERC20.sol: Failing transfer blocks all withdrawals for user	Major	Fixed
3	DepositBoxERC20.sol: _transfer reverts for some successful transfers	Major	Fixed
4	DepositBoxERC20.sol: retrieveFor and getNextUnlockTimestamp implicitly assume that delayedTransfersByReceiver is sorted	Minor	Acknowledged
5	Delayed transfers can be circumvented with multiple messages	Note	Acknowledged
6	DepositBoxERC20.sol: All transfers are delayed when bigTransferThreshold is not set for a token	Note	Acknowledged
7	MessageProxyForMainnet.sol: Inconsistent role usage for pausing / resuming	Note	Acknowledged
8	MessageProxyForMainnet.sol: pause() and resume() should emit event	Note	Fixed
9	DepositBoxERC20.sol: setBigTransferValue(), setBigTransferDelay() and setArbitrageDuration() should emit event	Note	Fixed

Critical Issues

1. DepositBoxERC20.sol: retrieveFor always reverts in some scenarios, making withdrawals impossible

The function `retrieveFor` iterates over `_QUEUE_PROCESSING_LIMIT` items at most. All completed items at the beginning of the queue are removed while iterating over it. However, the function requires that at least one transfer was retrieved, otherwise it will revert. This leads to the following problem: Let's say that `_QUEUE_PROCESSING_LIMIT` is 10 and that for the given user, there are 10 items with status `COMPLETED` at the beginning of the queue and 5 with status `DELAYED` after them. On a call to `retrieveFor`, the function will try to remove the 10 completed items from the beginning. However, `retrieved` is never set to `true` (because no transfers are retrieved with this call), meaning that the function will revert in the end. This will happen on every call to the function, resulting in locked funds (the items after the completed ones).

Recommendation
Do not revert when the function removed completed transfers.
Status: Resolved

Major Issues

2. DepositBoxERC20.sol: Failing transfer blocks all withdrawals for user

In `retrieveFor`, when there is one delayed transfer in the queue for a user where the transfer fails, the retrieval of all other delayed transfers is not possible. A transfer can fail for valid tokens (e.g., because of blocklists) or an attacker may also be able to send a message with a malicious

ERC20 token (that always reverts for some users). While the latter would not be problematic for instant transfers (only the processing of this one message would fail), it affects all delayed transfers here.

Recommendation
Skip the queue item when the transfer was unsuccessful.
Status: Resolved

3. DepositBoxERC20.sol: _transfer reverts for some successful transfers

---

Because USDT does not return a boolean value when calling `transfer`, `DepositBoxERC20._transfer` contains custom logic for the transfer of it. However, there are other tokens (such as BNB or OMG, see also here for a slightly outdated list of them) that do not return a boolean value. When they are used within the system, `_transfer` will revert, even if the transfer was successful. In combination with the delayed transfer mechanism, this can lead to fatal situations, because the tokens will end up in the queue for a user, but any attempt to transfer them out will fail (meaning that no progress is possible for the user and all other delayed transfers are also stuck).

Recommendation
Consider using OpenZeppelin's SafeERC20 (or implementing the same transfer logic) to handle all of these tokens and to avoid having special cases for specific addresses.
Status: Resolved

Minor Issues

4. DepositBoxERC20.sol: retrieveFor and getNextUnlockTimestamp implicitly assume that delayedTransfersByReceiver is sorted

`DepositBoxERC20.retrieveFor` (and `getNextUnlockTimestamp`) break as soon as there is a transfer with status `DELAYED` that is not unlocked yet. While this behavior would be correct when `delayedTransfersByReceiver` is sorted, this is not always the case. Because of the `depthLimit` in `_addToDelayedQueueWithPriority`, it can happen that there are transfers with a lower unlock timestamp at a position with a higher index (for instance, when the transfer delay was shortened afterwards). This can lead to situations where transfers cannot be executed (although their timestamp has passed) because they are blocked by some earlier ones.

Recommendation

Do not break when there is an unlocked transfer with status `DELAYED`.

Team response

Acknowledged

Informational Notes

5. Delayed transfers can be circumvented with multiple messages

Because the delay decision works on a per-message basis, it is easy to circumvent it by splitting up a withdrawal into multiple smaller ones. For instance, in the hypothetical scenario where an attacker can perform arbitrary withdrawals because of a security vulnerability, it would still be possible to withdraw all tokens by sending messages just below the withdrawal limit.

Recommendation
Consider restricting the withdrawal amount for a given time (e.g., per hour or per 100 blocks) such that the mechanism cannot be easily circumvented.

Team response
Acknowledged

6. `DepositBoxERC20.sol`: All transfers are delayed when `bigTransferThreshold` is not set for a token

In `DepositBoxERC20.postMessage`, when `bigTransferThreshold` is not (yet) set for a specific token, but a delay is configured, all transfers for this token will be delayed, no matter how small the amount is.

Recommendation
Unless this behavior is intended, consider checking if the threshold is greater than 0 and only delaying the transfer if that is the case.

Team response
Acknowledged

7. MessageProxyForMainnet.sol: Inconsistent role usage for pausing / resuming

---

The `DEFAULT_ADMIN_ROLE` and schain owner do not have permission to call `pause`, and the `PAUSABLE_ROLE` role does not have permission to call `resume`.

Recommendation
Consider allowing the `DEFAULT_ADMIN_ROLE` and owner roles to pause the contract, and the `PAUSABLE_ROLE` to resume the contract.

Team response
Acknowledged

8. MessageProxyForMainnet.sol: pause() and resume() should emit event

---

Functions `pause` and `resume` are responsible for pausing and resuming a skale chain indefinitely. Such important methods should be emitting events to show the current status of a particular skale chain.

Recommendation
Consider adding events for skale chain pausing and resuming and emit them inside `pause` and `resume` methods respectively.
Status: Resolved

9. DepositBoxERC20.sol: setBigTransferValue(), setBigTransferDelay() and setArbitrageDuration() should emit event

---

Functions `setBigTransferValue`, `setBigTransferDelay` and `setArbitrageDuration` are responsible for changing the sensitive transfer configurations on a particular Skale Chain. Such methods should be emitting events to show the current status of transfer configurations of a particular skale chain.

Recommendation
Consider adding events for the above methods and emit them inside their respective methods.
Status: Resolved

Audit Report for SKALE - November 1, 2022

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of SKALE Protocol or its products. This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Oak Security GmbH