SOLIDIFIED

Audit Report for Snowbridge - September 10, 2021

Summary

Audit Report prepared by Solidified covering the Snowbridge smart contracts.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code below. The final debrief took place on September 10, 2021, and the results are presented here.

Audited Files

The source code has been supplied in a public source code repository:

https://github.com/Snowfork/snowbridge

Commit number: `b706a56cd9687f0e66115b32fdc89b50f40ef764`

Update: A new PR that replaces the `MMRVerification` contract with `SimplifiedMMRVerification` was audited on September 20, 2021 and no issues were found.
https://github.com/Snowfork/snowbridge/pull/495

Intended Behavior

Snowbridge is a trustless bridge between Polkadot and Ethereum.

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	-

SOLIDIFIED

Audit Report for Snowbridge - September 10, 2021

Issues Found

Solidified found that the Snowbridge contracts contain no critical issues, no major issues, 2 minor issues, and 4 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	ETHApp.sol: Functions reward() and unlock() can potentially fail when transferring ETH to a smart contract	Minor	Acknowledged
2	ERC20App.sol: It is possible that some ERC-20 tokens might get locked forever within the contract	Minor	Acknowledged
3	ChannelAccess.sol: Function _authorizeDefaultOperator() does not emit its respective event	Note	Acknowledged
4	BasicInboundChannel.sol: Inability to have calls that send ETH	Note	Acknowledged
5	BasicInboundChannel.sol: Use memory variable instead of incrementing storage slot	Note	Acknowledged
6	Miscellaneous Notes	Note	Acknowledged

Critical Issues

No critical issues have been found.

Major Issues

No major issues have been found.

Minor Issues

1. ETHApp.sol: Functions reward() and unlock() can potentially fail when transferring ETH to a smart contract

Functions `reward()` and `unlock()` call `transfer()` when sending ETH to their respective recipients, which only forwards 2300 gas. In cases where the recipient address is a smart contract whose fallback function consumes more than 2300 gas, the call will always fail. This will have the side effect of potentially preventing smart contracts (e.g. DAOs) from receiving any ETH.

For a more in-depth discussion of issues with `transfer()` and smart contracts, please refer to `https://diligence.consensys.net/blog/2019/09/stop-using-soliditys-transfer-now/`

Recommendation
Replace instances of `transfer()` with `call()`.

Status

Acknowledged. Team's response: "*We'll keep in mind how/if we want to support smart contracts for relaying in future. For now, due to the design of needing to charge users the full gas cost of the relay on the parachain-side as described above, allowing a possibility of more than 2300 gas being used for the reward could introduce unnecessary charges to end users*".

2. ERC20App.sol: It is possible that some ERC-20 tokens might get locked forever within the contract

---

Some `ERC-20` tokens implementations of `transfer()`/`transferFrom()` do not return a `bool` result. This might either prevent the tokens from being deposited or even result in tokens being forever stuck within the `ERC20App` contract.

Recommendation

Consider using the `SafeERC20` library.

Status

Acknowledged.

Informational Notes

---

3. ChannelAccess.sol: Function _authorizeDefaultOperator() does not emit its respective event

---

Consider having function `_authorizeDefaultOperator()` emit the `OperatorAuthorized()` event.

Status

Acknowledged.

4. BasicInboundChannel.sol: Inability to have calls that send ETH

Certain contracts only work with the ability to send ETH as part of the transaction, and BasicInboundChannel is not able to interface with these contracts.

Recommendation

Consider extending the message format to allow value calls.

Status

Acknowledged. Team's response: "*This could be a potentially useful future feature/addition. For now we're expecting developers to deploy a proxy contract between the Channel and any applications they want to interact with that could do this*".

5. BasicInboundChannel.sol: Use memory variable instead of incrementing storage slot

`processMessages()` in `BasicInboundChannel` increments a storage slot over the entire messages array, which is far more expensive than using a temporary variable and storing the nonce at the end of the loop.

Recommendation

Consider using a temporary stack or memory variable for nonce.

Status

Acknowledged.

6. Miscellaneous Notes

- MMRVerification.sol: Function `verifyInclusionProof()` - when the MMR has only one leaf, it is unnecessary to check `leafPos` when verifying inclusion. Checking that `leafNodeHash == root` is enough. **Recommendation**: consider removing the redundant check.
- `channelId` checks in `ERC721App`, `ERC20App`, `DOTApp`, would be better as an inherited modifier. **Recommendation**: Rather than copy pasting the check in a require it would be better to move those into a modifier and a contract that inherits the enum.
- Consider having `MAX_GAS_PER_MESSAGE` and `GAS_BUFFER` be parametrized rather than constants.
- Consider adding additional documentation stating operator restrictions on the Polkadot side for `BasicInboundChannel` and `IncentivizedInboundChannel`.
- The repository contains contracts that are compiled with different major compiler versions. This may lead to unexpected behavior down the line. **Recommendation**: Consider updating all contracts to use the same major compiler version.

Status

Acknowledged.

Audit Report for Snowbridge - September 10, 2021

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of Snowbridge 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.

Solidified Technologies Inc.