Audit for POA Bridge

v1.0.0

April 13th, 2018

LEVELK

Introduction

The POA Network team asked us to review their version of the Parity Bridge contracts that incorporate upgradable smart contract patterns sourced from the ZepplinOS Labs repo. The code is located in POA Network's poa-parity-bridge-contracts repo and the commit that was used for this auditted is c63b44f4cd4f32b3b4c68d6577a96f306993e5db.

We reviewed the code and laid out the results below. Overall, the code is very well written, clean, easy to understand, and generally follows best practices regarding structure and security. There were no critical issues found in the code. We recommended a few changes which have been outlined below.

Issues

1. Fix input validation

Medium

U_BridgeValidators.sol#L29 should use `require` instead of `assert`. Conditional checks should generally only use `assert` for conditions that are never expected to happen.

Also, the `isValidator` function should be used here to avoid logic duplication.

Update: *Fixed in* `7ab6c35`

2. Disallow 0 required signatures

Medium

The `setRequiredSignatures` function U_BridgeValidators.sol#L43 should include a check for `_requiredSignatures != 0`

Update: *Fixed in* `0c2e5e2`

3. Refactor repeated logic

Low

In HomeBridge, we recommend refactoring `totalSpentPerDay(getCurrentDay()).add(...)` U_HomeBridge.sol#L35 to an internal function, since the logic is repeated in `withinLimit` U_HomeBridge.sol#L100. The same refactoring can also be done in ForeignBridge.

Update: *The POA team is planning on implementing this optimization in v2 of the POA bridge.*

4. Consider refactoring common code

Low

Considering refactoring common Bridge functionality into a base class. `deployedAtBlock`, `getCurrentDay`, `withinLimit`, `isInitialized`, `setTotalSpentPerDay` could be shared between ForeignBridge and HomeBridge.

Update: *The POA team is planning on implementing this optimization in v2 of the POA bridge.*

5. Add 0x0 address check

Low

Add a check to require that `_owner` is a non-zero address when setting owner U_BridgeValidators.sol#L16. If the intention is to make the contract un-owned, the `0x000000000000000000000000000000000000dEaD` address can be used.

Update: *Fixed in* `a22b871`

6. Missing event in setRequiredSignatures function

Low

Consider adding a `RequiredSignaturesSet` event in `setRequiredSignatures` U_BridgeValidators.sol#L43.

Update: *Fixed in* `b928a0e`

7. Duplicate checks for _foreignDailyLimit > 0

Low

`require(_foreignDailyLimit > 0)` is checked in initialize U_ForeignBridge.sol#L35 and in `setForeignDailyLimit`. Consider removing the duplicate check.

Update: *Fixed in* `abccf2b`

7. Move MessageSigning library to its own file

Low

Consider separating the `MessageSigning` library out into its own file. Helpers.sol#L59.

Update: *Fixed in* `b572717`

8. Use SafeMath for arithmetic operations

Low

The following lines are using arithmetic operations. None of these instances are at risk of an underflow or overflow and do not present a security risk. However, it is best practice to use the OpenZeppelin SafeMath library when possible. Consider updating the following lines to use SafeMath functions in place of the arithmetic operations.

```
contracts/libraries/Helpers.sol
  26:20    warning    Avoid use of arithmetic operation '-' directly. Use SafeMath
instead.    zeppelin/no-arithmetic-operations
  29:31    warning    Avoid use of arithmetic operation '+' directly. Use SafeMath
instead.    zeppelin/no-arithmetic-operations

contracts/upgradeable_contracts/U_ForeignBridge.sol
  134:22    warning    Avoid use of arithmetic operation '+' directly. Use SafeMath
instead.    zeppelin/no-arithmetic-operations
  146:42    warning    Avoid use of arithmetic operation '-' directly. Use SafeMath
instead.    zeppelin/no-arithmetic-operations
  168:15    warning    Avoid use of arithmetic operation '/' directly. Use SafeMath
instead.    zeppelin/no-arithmetic-operations

contracts/upgradeable_contracts/U_HomeBridge.sol
  92:15    warning    Avoid use of arithmetic operation '/' directly. Use SafeMath
instead.    zeppelin/no-arithmetic-operations
```

Update: *The POA team has elected to use arithmetic operations. None of the arithmetic operations are at risk of an underflow or overflow and do not present a security risk.*

9. Use uint256

Low

The variable `i` could overflow when there are 256 or more required signatures in Helpers.sol#L46. Additionally, in Zeppelin's Augur audit, they state "the EVM word size is 256 bits, so there is no additional benefit to using a smaller integer variable. There is, in fact, an additional cost due to the operations required to simulate overflow semantics."

Update: *Fixed in* `69b9779`

10. Check that the owner address is not 0x0

Low

Consider adding a sanity check to ensure owner is a non-zero address. Ownable.sol#L49

Update: *Fixed in* `bcd6084`

11. Message bytes data length check is incorrect

Low

The comments in Message.sol indicate that message bytes data should be 116 bytes, and that the last 32 bytes is reserved for “home gas price”. However, there is no code that accesses the last 32 bytes of data for messages. Additionally, there are two checks for message length U_HomeBridge.sol#L65 and U_ForeignBridge.sol#L130 that assume the message data is 116 bytes.

When the message bytes array is passed as a parameter to a public function, as is the case with the submitSignature function in ForeignBridge, the value of the memory that was intended to be used for “home gas price” can be manipulated by the sender. This allows the sender to submit the recipient address, token value, and transaction hash many times, but always hash to a different result. The hash value on this line can be manipulated in this way.

This does not result in an exploit, since all validators would still need to agree to sign the new message. Additionally, this require in HomeBridge ensures that the transaction hash within the message is unique in order to execute a withdrawal.

This could be prevented by changing these require checks (U_HomeBridge.sol#L65 and U_ForeignBridge.sol#L130) to `message.length == 84`.

Update: *Message lengths of 116 bytes are required for compatibility with the current parity-bridge implementation. The POA Network team plans to build a new bridge implementation from scratch upon which the contracts will be updated.*

Spam Attack

The current bridge design is susceptible to a spam attacks where an attacker purposefully makes many small deposits to a bridge with the intent of wasting validators' ETH and other tokens in transaction fees. This attack is possible on both `ForeignBridge` and `HomeBridge`.

With a validator set size of NUM_VALIDATORS, each deposit to HomeBridge requires NUM_VALIDATORS transactions by the validators on the foreign chain. Each deposit to ForeignBridge requires NUM_VALIDATORS transactions on the foreign chain and one transaction on the home chain. The cost effectiveness of this attack depends on the cost to transact on each respective chain, and which chain is the "home chain" and which chain is the "foreign chain". If there is a large difference in the cost to transact on each chain this attack becomes more cost effective as the attacker can make many deposits on the cheaper chain causing validators to make transactions on the more expensive chain wasting much more of the validators' resources than were spent for the attack.

Update: *This attack has been mitigated in* `987dc06`*. A minimum deposit amount (*`MIN_PER_TX`*) has been added, limiting the amount of transactions that are possible per day. For each bridge, there is now a maximum of* `DAILY_LIMIT/MIN_PER_TX` *user transactions per day where* `DAILY_LIMIT` *is the maximum amount of tokens that can transferred across the bridge on a given day. This limits the spam attack to producing a maximum of* `DAILY_LIMIT/MIN_PER_TX` *validator transactions on the home chain and* `DAILY_LIMIT/MIN_PER_TX * NUM_VALIDATORS * 2` *transactions on foreign.*

The POA team plans on implementing a bridge fee that would cover the validators gas fees shortly in the future further mitigating this attack.

EVM Assembly Code Observations

We reviewed the assembly code in the MessageSigning library, Message.sol, and Proxy.sol. No issues related to this code were found.

Proxy.sol

The Proxy.sol code, which has been developed by the Zeppelin Solutions team, differs slightly from their latest implementation of Proxy.sol in zeppelinos/upgradeability-lib. There is a gas cost optimization (7d97b2c) that the Zeppelin team has since introduced. Note that they have also marked this entire repo as “do not use in production” at this time.

We wrote inline comments for the Proxy.sol assembly code. We would suggest adding these to a contract level description, similar to the description in Message.sol. We will make the same suggestion to Zeppelin Solutions, so that future projects can easily understand how it works before adopting it.

Inline comments can be found in this gist. This file is only used for commenting, and not intended to be used in production.

Update: *The POA team has added the the new proxy code along with the inline comments explaining the solidity code in this commit* 277cdab.

Code Analysis Tools

We ran the solidity code through two analysis tools: Solium and Oyente. The Solium report found that arithmetic operators were being used, which should be replaced with SafeMath. The Oyente report found a few potential overflows/underflows - however, none of these presented a potential exploit. The analysis reporting did not indicate any problems other than these low priority issues.

Reporting output for Solium and Oyente will be submitted along with this report

General Observations

uint256 vs. uint

Although not necessary because the compiled result is the same in either case, it has been recommended to use uint256 in place of uint, in order to be more explicit about type. Relevant discussion can be found in this issue:
https://github.com/OpenZeppelin/zeppelin-solidity/issues/226

Conclusion

No critical or high severity issues were found. We made a few small suggestions to better follow Solidity and general coding best practices. Overall the code was very clean and well thought out.