MixBytes()

Audit of bridge smart contracts in the POA Network project

Contract

URL of the audited contract: https://github.com/poanetwork/poa-parity-bridge-contracts/tree/d796891477e15823c7bdd5b0b2f9a38e10f17b94/contracts

Classification of identified issues

CRITICAL: possible stealing of Ether/tokens, or their permanent blocking with no opportunity of recovering access, or any other loss of Ether/tokens payable to any party (e.g., as dividends).

SERIOUS: possible violations of contract operations, where the manual amendment or complete replacement of the contract is required to recover proper operations.

WARNINGS: possible violation of the contract's planned logic, or possible DoS attack on the contract.

REMARKS: any other remarks.

Audit methodology

The code of the contract may be viewed manually in order to identify known vulnerabilities, logic errors, and compliance with the White Paper. Unit tests can be written to check any questionable aspects as required.

Identified issues

[CRITICAL]

- not found

[SERIOUS]

1. U_BridgeValidators.sol: 37

No verification is performed to confirm that a `_validator` being removed is an actual validator. If they are not, then the `validatorCount()` no longer complies with the true number of validators. This can only be fixed by updating the contract's code.

Fixed at 40a07ebd12bc2cc0716d4a1eb8004d979619907d.

2. U_ForeignBridge.sol: 124, 139

`hash` should be used instead of `hashSender`, otherwise the signatures number for `message` will never exceed 1, as `msg.sender` is included in `hashSender`. It appears only `hashSender` should be used to exclude dual signatures issued by the same validator (when invoking `messagesSigned` and `setMessagesSigned`).

Fixed at PR 21.

3. U_ForeignBridge.sol: 103

If you increase the number of required validator signatures, tokens can be released multiple times: 1) Let's say that `requiredSignatures` was 2, and you received the signature from the second validator and generated tokens for `recipient`; 2) In the contract `validatorContract` the number of `requiredSignatures` was increased to 3; 3) One of the validators (which is probably malicious) that hasn't yet signed this `transactionHash`, sends a signature; 4) So, the token generation code is invoked once again, as the number `signed` is now 3, which is the same as the number of `requiredSignatures`.

`isAlreadyProcessed` *method was added. A discussion of the solution is available in PR 20.*

4. U_ForeignBridge.sol: 143

When the number of required validator signatures increases, `CollectedSignatures` may be repeatedly generated for this message.

Discussion and fix is available in PR 20.

[WARNINGS]

1. Helpers.sol: 49

Quadratic complexity of the algorithm — normally, there's a risk of hitting block gas limit. We recommend creating either a hash map, or a balanced binary search tree located in the array.

Talking to the customer: Auditor: "You can create a mapping library by executing sload/sstore in the hash address (function id, call id, id maps, key). However, you may not want to waste gas, since you're going to require 20k * array_length. So, it's better to do this in memory. But memory is limited here, and there aren't really any ready-made solutions. The ecosystem is not yet prepared for either an allocator, or for a standard library of structures. Thus, the fastest way may be to limit the number of validators and calculate consumed gas in terms of a worst-case scenario."

Customer: "I've performed a small experiment: Link. Even having 20 array elements, 140 thousand of gas per transaction was still required. For that reason, I don't think that the initial comment from the document is of any importance. It's hardly likely that there is going to be more than 15 validators. When we implement an AVL tree in order to be able to process small data volumes, we're still going to face significant overheads related to the insertions of elements, as each insertion

will require the re-balancing of the tree, as typically a signatures scenario in the array will not have any duplicates, which means that insertions will still be made for each element. In reality, the bypass range for searching an element (for a duplicate) will gradually decline to 5 elements instead of 19, as it is in this case. However, overheads are inevitable when implementing the tree with an array, which is not typical even for a non-balanced tree. As a result, we may probably win 20–30 thousand of gas if there's a large number of validators, then we will most likely lose in cases where 5–7 validators are available. In any event, we're always losing in what concerns code readability. Here is an example of tree implementation: OrderStatisticTree.sol (see methods starting from the insert). In our case, this can be even more complex, as it would be implemented on the top of the arrays, not the hashmaps."

So, we decided to leave it unchanged until the number of validators grows significantly.

2. U_BridgeValidators.sol: 14

Since the contract is initialized by any first participant of the network who invoked `initialize`, for security purposes you should make sure that an initialization transaction that you sent (e.g., here — 3_upgradeable_deployment.js: 33) was successful. The same is true for instantiating any other upgradeable contracts — `HomeBridge`, `ForeignBridge`.

Added to TODO.

3. Helpers.sol: 46

Here an endless loop may occur due to an overflow of a uint8-type loop count when `requiredSignatures` is equal or greater than 256.

Fixed at 69b97796f684ecde0261e32a413bf5dbf3b3f11c.

[REMARKS]

1. Helpers.sol: 50

Is it really a good idea to use `false` here? This most likely needs `require` (not `assert`, as call arguments are created by independent code), or ignoring duplicates without stopping the function, along with counting unique signatures.

Fixed at master Message.sol: 103.

2. Message.sol: 27

We recommend applying a mask to the corresponding bits using the `and(mload(add(message, 20)), 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF)`.

Customer: "mload reads 32 bytes, isn't it? I want to understand what it's for."

Auditor: "Comments to the code say that yes, we're engaging a part of the adjacent field as well, but it always contains zeros, because... Here we're talking exactly of not relying on zeros that may or

may not be there but making sure they are there by means of applying a mask. This way we can read 20 bytes, engage the next field and make a 12 bytes-shift to the right."

Fixed at PR 21.

3. Proxy.sol: 30

The value of the pointer to the free memory is not updated. You should add `mstore(0x40, add(ptr, size))`.

Fixed at PR 21.

4. U_BridgeValidators.sol: 14

We recommend adding a next check to the initialize method: `require(owner != address(0));`.

Fixed at master.

5. U_BridgeValidators.sol: 20

The first condition is redundant because it is duplicated by the `addValidator` method. You'd better include the second check in the `addValidator` method in order to encapsulate the check inside.

Altered at master.

6. U_BridgeValidators.sol: 28

We recommend checking user input using `require`. Using `assert` is only recommended to check the self-consistency of code.

Fixed at PR 21.

7. POA20.sol: 46

May be declared as `view`.

Fixed at PR 21.

8. POA20.sol: 31

You don't need to use the `super` mechanism because the function does not override the function `transfer` of the parent contract.

Fixed at PR 21.

9. POA20.sol: 31

No verification of the `transfer` output is performed. ERC20 specification says: "NOTE: An important point is that callers should handle false from returns (bool success). Callers should not assume that false is never returned!" As a matter of fact, the current code always returns `true`, but by relying on

that, you're thus adding a hidden fault into the code that can show up in the future.

Fixed at PR 21.

10. POA20.sol: 43

The `onTokenTransfer` output is not processed. However, even the authors of the ERC677 specification themselves have no idea of why this function returns a Boolean value, and how this value should be processed.

Customer's question: "So, what should we do? Please let us know if we should migrate to ERC827."

Auditor's reply: "If you have a Boolean value, we recommend performing require. As far as ERC827 is concerned, a week ago it was still being discussed and fixed: https://github.com/ethereum/EIPs/issues/827#issuecomment-381966457. The same is true for any currently existing non-ERC20 token. We've chosen the following solution for the token in our Smartz project: https://github.com/smartzplatform/sale/blob/master/contracts/token/TokenWithApproveAndCallMethod.sol."

Fixed at PR 21.

11. U_ForeignBridge.sol: 45

The `require(address(erc677token()) != address(0x0));` check is redundant because we're already performing it at the time of token assignment. This can be replaced with the `assert` verification.

Changed at PR 21.

12. U_ForeignBridge.sol: 50

Logic does not comply with the description. The `Withdraw` event is generated immediately after tokens are transferred to the foreign bridge, while the README.md page says that this only happens after the validation of transactions (with validators? Or do you mean transaction mining?): "When the transaction is validated, the user should expect to see a Withdrawal Event on the Foreign Network (right-side - Ethereum Foundation)." The same is true for reverse transfers across the bridge.

13. U_ForeignBridge.sol: 49

After reading the README.md file, you might think that the tokens transferred to the foreign bridge are burnt after having collected the required number of signatures. However, tokens are actually burnt immediately after the user transfers them to the foreign bridge contract balance.

Added to TODO.

14. U_HomeBridge.sol: 22

We recommend including the following checks: `require(_homeDailyLimit > 0);` into the corresponding `setHomeDailyLimit` function. The same is recommended for `U_ForeignBridge` contract.

Fixed at master.

[SUGGESTIONS]

1. Basic contract for `HomeBridge` and `ForeignBridge`

Part of `HomeBridge` and `ForeignBridge` code (working with day limit, part of initialization) may be added to the common basic contract.

Approved by the customer. Postponed to a later date.

2. Ownable.sol: 41

We recommend transferring the `newOnwer!= address(0)` verification to the `setOwner` function.

Not implemented. Customer's comment: `transferOwnership` *is already accomplishing this, and the* `BridgeValidators` *contract uses* `setOwner` *because* `transferOwnership` *is not available for use.*

3. U_ForeignBridge.sol: 120 and U_HomeBridge.sol: 62

We recommend adding this message validation procedure in the `library Message`, which is a single point for message de-serialization. We suggest you make de-serialization a single function that returns multiple values and performs validation.

Fixed at PR 20.

4. UpgradeabilityProxy.sol: 24

You may want to create a versioning engine that prevents downgrading to a previous version.

Customer: "In this case it's just a string for convenience of display, it has no actual effect. Regrettably, there are no options to compare strings in Solidity."

Auditor: "It's clear, but you can use a number. Using semver will probably be a bit too much. The same way that migration systems prevent the implementation of a previous migration version, control of the version increase also makes sense."

Fixed at master.

5. U_HomeBridge.sol: 29

We recommend making sure that `msg.data.length == 0` — that the customer/user software does not erroneously invoke another non-existent function, but instead transfers Ether without

parameters.

Fixed at PR 21.

6. U_BridgeValidators.sol: 28

You can use a more specific function: `require(!isValidator(_validator));`.

Fixed at PR 20.

7. U_ForeignBridge.sol: 78

Here and in other similar places. If the function is going to be invoked strictly by means of external calls, it's recommended to replace the `public` visibility modifier with an `external` modifier. First, we provide the compiler with a more accurate description of how we expect the contracting system to perform. For instance, any accidental internal call of the external function in the contract or inheriting contract will cause a compilation error, which requires either the removal of the call or revision of the function role across the contracting system. As a matter of fact, this is similar to using the const specifier in some programming languages. Second, this will save us a little bit of gas (https://medium.com/@gus_tavo_guim/public-vs-external-functions-in-solidity-b46bcf0ba3ac).

Largely fixed at PR 21.

8. Implement other ways to enable the upgradability of contracts

Weak points of the current implementation: - No IDE and compiler-level support. Therefore, for instance, an error in a string literal that actually designates a variable may go unnoticed (e.g., if you make a spelling mistake in `gasLimitDepositRelay` here): U_ForeignBridge.sol: 79) - If you're using complex data types where the storage element hash is built based on multiple components, no proper isolation is possible, and a collision may occur as a result, e.g.: `boolStorage(keccak256("foo", "bar"))` and `boolStorage(keccak256("fooba", "r"))` collide. - We rely on the unchanged current layout fields in the Solidity contract (by this we basically mean the slots for EternalStorage fields). Disadvantages at the stage of contract instantiating: instead of the constructor, we have to use the function without access control.

Suggested option: each version of the contract has its own storage and pulls data from the previous version of storage as required. The suggested option is illustrated by the `UpgradableBridge.sol` file containing the `UpgradableBridge.js` test.

Weak points of the suggested option: - We can't guarantee in advance that you will be able to completely "disable" the contract version when updating it based on its storage required by the next version (e.g., if you have forgotten an `onlyProxy` modifier somewhere). Sometimes the version may still be functioning (though without any effect to the bridge) and corrupting its storage. - This may cause an increase in gas spending.

The customer is experienced in implementing the recommended solution and notes some difficulties in data migration.

[SUMMARY]

We've identified a number of challenges that prevent proper operations and need to be addressed. We've found no risks that funds or tokens may have been stolen by any third parties. We've also identified a number of potentially insecure constructions and provided recommendations with regard to the further development and adjustment of code.

All required alterations have been applied as of 05/28/2018.