MixBytes()

Audit of PoPA Smart Contracts

Contract

Address of the audited contract: https://github.com/poanetwork/poa-popa/tree/e259cec1fcfcfdff30a52bffb395d845c774855b/blockchain/contracts, https://github.com/poanetwork/poa-popa/tree/e259cec1fcfcfdff30a52bffb395d845c774855b/web-dapp/server-lib

Classification of Detected Issues

CRITICAL: bugs that enable theft of ether/tokens; or lock access to funds without possibility to restore it; or lead to any other loss of ether/tokens to be transferred to any party (for example, dividends).

MAJOR: bugs that can trigger a contract failure, with further recovery only possible through manual modification of the contract state or contract replacement altogether.

WARNINGS: bugs that can break the intended contract logic or enable a DoS attack on the contract.

COMMENTS: all other issues.

Audit Methodology

The contract code is scanned manually for known vulnerabilities and logic errors as well as for its compliance with the White Paper. To eliminate potential flaws, unit tests may be written.

Detected Issues (Solidity)

[CRITICAL]

- none

[MAJOR]

- none

[WARNINGS]

1. ProofOfPhysicalAddress.sol#L44-L46

totalUsers, totalAddresses, and totalConfirmed counters only increase, but never decrease. Logic suggests that they decrease once an address or user is deleted.
This option was favoured, as the previous version of ERC780 allowed the user to delete a claim about himself/herself in a way that bypassed the smart contract. Going by the number of deleted claims, therefore, rendered the accuracy of these figures unreliable. This issue has been fixed in the new version: PR 136.

[COMMENTS]

1. https://github.com/ethereum/EIPs/issues/780 — EthereumClaimsRegistry.sol contract has been updated in comparison to what is used in the audited code.

When working on the new version, we agreed that removeClaim may call the issuer, but not the subject. Such decision is justified by the following: if the claim contains false information, then the subject may post a counter claim on his/her behalf, whereafter it becomes a matter of trust. On the other hand, if the subject just wants to delete his/her data, then removeClaim will not help him/her as the data can still be retrieved from change history.
Though not used in the contract code, claim removal performed by the subject is checked by the tests.

```
--- EthereumClaimsRegistry.sol  2018-05-10 11:19:04.459670819 +0300
+++ EthereumClaimsRegistry-EIP780-20180510.sol  2018-05-10 11:22:10.510566403 +0300
```

```
@@ -1,6 +1,3 @@
-pragma solidity 0.4.19;
-
-
 contract EthereumClaimsRegistry {

     mapping(address => mapping(address => mapping(bytes32 => bytes32))) public registry;
@@ -18,24 +15,23 @@
         bytes32 indexed key,
         uint removedAt);

-    // create or update claims
-    function setClaim(address subject, bytes32 key, bytes32 value) external {
+    // create or update clams
+    function setClaim(address subject, bytes32 key, bytes32 value) public {
         registry[msg.sender][subject][key] = value;
-        ClaimSet(msg.sender, subject, key, value, now);
+        emit ClaimSet(msg.sender, subject, key, value, now);
     }

-    function setSelfClaim(bytes32 key, bytes32 value) external {
-        registry[msg.sender][msg.sender][key] = value;
-        ClaimSet(msg.sender, msg.sender, key, value, now);
+    function setSelfClaim(bytes32 key, bytes32 value) public {
+        setClaim(msg.sender, key, value);
     }

-    function removeClaim(address issuer, address subject, bytes32 key) external {
-        require(msg.sender == issuer || msg.sender == subject);
-        delete registry[issuer][subject][key];
-        ClaimRemoved(msg.sender, subject, key, now);
+    function getClaim(address issuer, address subject, bytes32 key) public view returns(bytes32)
+{
+        return registry[issuer][subject][key];
     }

-    function getClaim(address issuer, address subject, bytes32 key) external view
returns(bytes32) {
-        return registry[issuer][subject][key];
+    function removeClaim(address issuer, address subject, bytes32 key) public {
+        require(msg.sender == issuer);
+        delete registry[issuer][subject][key];
+        emit ClaimRemoved(msg.sender, subject, key, now);
     }
```

```
}
```

This has been updated in PR 136.

2. ProofOfPhysicalAddress.sol#L98

userAddressConfirmed cannot receive a value for addressIndex that exceeds the length of the physicalAddresses array. The function contains the code to check the user's existence:

```
require(userExists(wallet))
```

However, the array may lack the required index. If the call function has checked the index existence (and this is the case with userAddressByCreationBlock), then require is not needed. Otherwise, the array size should also be checked. Alternatively, we can make helper functions internal, but then the tests will break.

A check has been added in PR 138.

3. ProofOfPhysicalAddress.sol#L200

userAddress function, as well as userAddressConfirmed, does not check the index existence before accessing an array element.
A check has been added in PR 138.

4. ProofOfPhysicalAddress.sol#L214

userAddressInfo does not check the index existence, either.
A check has been added in PR 138.

5. ProofOfPhysicalAddress.sol

It is recommended that you use view or pure, rather than constant, as specifiers in functions.
This change was made in PR 156.

6. ProofOfPhysicalAddress.sol#L97

Here and below, it is recommended that you turn the require(userExists(wallet)); check into a modifier.
This was already done in PR 127.

7. ProofOfPhysicalAddress.sol#L104

Before decoding, a check is recommended to verify the claim's existence: `if (0 == claim) return false;`.
This was added in PR 151.

8. ProofOfPhysicalAddress.sol#L225

It is recommended that you add `require(userAddressConfirmed(wallet, ai));` above. Otherwise, you will be decoding an empty claim here.
In this case, an exception will occur if the address is not confirmed, although the method itself does not imply that the address should be confirmed — it simply returns the information. Probably it would be better to do it this way.

```
confirmationBlock = 0;
if (userAddressConfirmed(wallet, addressIndex)) {
      confirmationBlock = PhysicalAddressClaim.decodeConfirmation(...)
}
return ( ....name, ....creationBlock, confirmationBlock, ....keccakIdentifier)
```

I would rather suggest a point solution:

```
bytes32 claim = registry.getClaim(
    address(this),
    wallet,
    users[wallet].physicalAddresses[addressIndex].keccakIdentifier);
return (
    users[wallet].physicalAddresses[addressIndex].name,
    users[wallet].physicalAddresses[addressIndex].creationBlock,
    uint256(claim) != 0 ? PhysicalAddressClaim.decodeConfirmation(claim) : 0,
    users[wallet].physicalAddresses[addressIndex].keccakIdentifier
);
```

This was added in PR 159.

9. ProofOfPhysicalAddress.sol#L100

Given the logic used in the code, this condition is never satisfied. To check the internal consistency of the code, it is recommended to replace this condition with `assert(keccakIdentifier != 0x0)`.
This was added in PR 152.

10. ProofOfPhysicalAddress.sol#L309

An algorithm with an O(N) complexity for injecting data into the blockchain. The current algorithm for shifting array elements starting with `index + 1` can be replaced by an algorithm for moving the last element with an index equal to `length - 1` by `index`, if `index != length - 1`. So, it will be an algorithm with an O(1) complexity for registering data in the blockchain.
This was done in PR 154.

11. ProofOfPhysicalAddress.sol#L331

According to the diagram, a confirmation code is sent from the DApp web directly to the smart contract bypassing the DApp server. However, the code requires a signature from the DApp server.

This was corrected in https://github.com/poanetwork/wiki/commit/052db2902f3f9f62b4bf22446070270008a83b0f (the image is uploaded from the wiki repository).

[PROPOSALS]

1. The number of possible physical addresses for a single account should be explicitly limited.

Currently, the address list in all locations is processed by iterating through the array elements. The size of the variable used for the counter always equals uint256, and given the fact that the address is added via offline mail, overflow is not likely to happen. However, security issues can be identified in other parts of the system, so we suggest imposing an explicit restriction on the number of addresses on the list and/or the ability to trim the address array to this maximum length in case of an overflow.

The check was added in PR 137.

Detected Issues (JavaScript)

[CRITICAL]

- none

[MAJOR]

It is possible to consume API requests to the service for sending emails for free, as well as to initiate a DoS attack on this service. The attack vector is as follows:

1. The attacker sends N /prepareRegTx requests and gets N session keys sk_1 .. sk_N and N data to send transactions to the PoPA contract.
2. However, the attacker then sends only the first transaction to the PoPA contract and waits for it to be mined, thus receiving txId_1. Other transactions are not sent.
3. After that, the attacker calls /notifyRegTx N times for each call using different sk starting with step 1, but all calls use the same txId = txId_1: data_i = {wallet: wallet, txId: txId_1, sessionKey: sk_i}.
4. As a result, DApp server will successfully find N sessions, block them, find a valid transaction for each of them, retrieve the corresponding address and finally send N cards to the same address.

The easiest way to fix this issue is to save data transaction parameter to the session and check it along with the to and from transaction parameters in the blockchain. This will also help eliminate the problem of insufficient to and fromchecks (because an attacker can send transactions with value = 0 and valid to and from). In general, it is recommended to provide a background process at the DApp server to poll the blockchain and send emails without user intervention, i.e. to avoid transmitting data through an untrusted source.
Client: It seems to me that the method with data will not work (I might have got it wrong, though) because when you call /prepareRegTx, the transaction is still unknown and sessionKey stores only the confirmation code in an open mode to be sent in a card. Maybe it makes sense to store a list of processed transactions on the server in Redis for some time (say 1 hour). It is necessary to add the following checks to the transaction processing code: (1) a check to confirm that the transaction is not on the list yet; (2) if the transaction is not on the list, then its block was mined earlier than 1 hour before.
The data parameter (input for the transaction) is known (at least all of its components are known) and is signed by the server key. We need to confirm that the same parameters are passed in session X (with ABI encoding applied). At the /prepareRegTx stage, we do not need to know the transaction ID. Alternatively, we can store the prepare_reg_tx.js#L60-L62 signature in the session. As a result, the attacker will not be able to send the same transaction to different sessions because the signature in the transaction will not match the signature in the session (the signature stored in the session is different every time, as the signed parameters contain a random confirmation code). However, there will be some difficulties with ABI-decoding of the transaction input.

Fixed in PR 178

[WARNINGS]

1. recalc_price.js#L24

Leaving this code as it is now appears to be risky because if priceWei is commented out in the config, this code will start working in production. It is suggested to add

```
if (process.env.NODE_ENV !== 'test')
    throw new Error('not implemented');
```

Fixed in PR 171

2. notifyRegTx.js#L119

Race condition during reading/writing of the limit of cards for the current day can lead to exceeding the set limit for sending cards. Let's say postcard_limiter.get() equals 9, and postcard_limiter.MAX_POSTCARDS_PER_DAY equals 10. If several requests retrieve 9 from the database, postcard_limiter.canSend() will return true for all of them, and they will send requests to post_api.create_postcard. They probably will execute postcard_limiter.inc() only after that, as mutual order of callbacks for different requests is not predetermined.
It is recommended that you use a strict synchronisation mechanism, such as semaphores.

Fixed in PR 179.

3. notifyRegTx.js#L51

The session will be blocked here by means of renaming the key in session store. However, there is no session unlock operation. Therefore, any errors, even non-fatal ones, will prevent the subsequent user access to this session, and the money that was transferred to the PoPA contract will be lost. In this case, the user will suffer losses and will not get the desired result.

The session blocking is necessary; otherwise, the user will be able to send several cards. It is recommended that you implement automatic unlocking of the session after processing the request. When the data is stored outside the web server process, it is also recommended to consider the possibility of the web server restarting and crashing, as this will require automatic unlocking (blocking by renaming the key will no longer work).

Fixed in PR 180

[COMMENTS]

1. req_id.js#L19

Before you retrieve the host from a host:port pair, you should first parse the header value into hosts, i.e. simply put, you should exchange the code blocks starting from lines 19 and 22. Otherwise, hosts may be mistakenly dropped, for example in the following case: 192.168.0.10:50000, 37.10.5.2.
This was added in PR 172.

2. redis.js#L74

We do not wait for the DEL command to complete before successfully completing the Promise.

Corrections were introduced in PR 173.

3. prepareRegTx.js#L15

Here and in subsequent cases, it is recommended to limit the maximum length for lines entered by the user (e.g. 500 characters).

Client: The limit on the total size of POST body before parsing was added to PR 174.

This is one possible way. The goal is to protect the system against big data, such as megabyte addresses. In this case, the algorithms that are considered to be O(1) up to a point will be O(N), and they can cause at least minor inconveniences.