SOLIDIFIED

Audit Report for SingularityNET. February 12, 2019.

Summary

Audit Report prepared by Solidified for SingularityNET covering their smart contracts which implement the SingularityNET platform.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the below contracts. The debrief took place on February 12, 2019 and the final results are presented here.

Audited Files

The following files were covered during the audit:

- IRegistry.sol
- Registry.sol
- MultiPartyEscrow.sol

Notes:
The audit was performed on commit `2242c2871ff31d01e8c07d2ad492bd5f0faa9eb7`
The audit was based on the solidity compiler `0.4.24+commit.e67f0147`

Intended Behavior

The specification of the contracts is found in the interface file IRegistry.sol. Additional information can be found in

Issues Found

Critical

No critical vulnerabilities were identified.

Major

No major vulnerabilities were identified.

Minor

1. Iterating unbound arrays can cause Denial of Service (DoS)

In Registry.sol, arrays `OrganizationRegistration.memberKeys`, `OrganizationRegistration.serviceKeys`, `OrganizationRegistration.typeRepoKeys`, `ServiceRegistration.tags`, `TypeRepositoryRegistration.tags` have no upper bounds. Since all of these arrays are being iterated in functions such as `deleteOrganization()`, `deleteServiceRegistrationInternal()`, `deleteTypeRepositoryRegistrationInternal()`, an organization can potentially face Denial of Service (DoS) if the array count exceeds the block gas limit required to execute its respective function.

Recommendation

Determine the maximum number of elements that will allow each function not to exceed the block gas limit and enforce that on each array.

2. Incorrect nonces in Events

In `MultiPartyEscrow.sol`, `channelClaim()` and `channelClaimTimeout()` emit events for the post-update state, communicating a wrong nonce about the channels to the clients.

Recommendation:
Emit the event before incrementing the nonce.

Notes

3. Gas Optimization during channel closing

The variable `channel.value` is written twice when `channelClaim()` is called with `isSendback`, here and here. The code can be refactored to only write once and save gas.

Recommendation

The following proposed diff creates 9.3% gas savings when calling `channelClaim()`. Patch here.

4. Re-implementation of `supportsInterface()` is unnecessary

The re-implementation of `supportsInterface()` is unnecessary as the OpenZeppelin ERC165 contract already provides a `_registerInterface()` function.

Recommendation
Remove the implementation of `supportsInterface()` and add `_registerInterface(0x91372c6a)` to the contract's constructor instead.

5. Channel expiration time can be set in the past

During initialization of a channel in `MultiPartyEscrow.sol`, its expiration can be set to a past value. This may result in an unexpected scenario where a sender initiates a payment channel with a receiver, and they instantly settle the payment channel after trading the first signature.

Recommendation

Consider setting the channel expiration to `block.number + expiration` so that a user only needs to specify relative time, and not absolute time. Alternatively, add a check that the timeout is in the future, or add preset timeout values. Also consider adding a check that enforces `newExpiration > block.number` to `channelExtend`.

6. Uniqueness of `groupId` is not enforced

In `MultiPartyEscrow.sol` The code comments indicate that the `groupId` value should not be reused between channels with the same sender/recipient pair. The code does not enforce this at any point.

Recommendation

Consider removing the `groupId` parameter altogether, since it is not used anywhere in the smart contract logic. The triple (sender, receiver, channelId) is enough to identify the channel from the client-side. Alternatively, consider generating the groupId by hashing together the (sender, receiver, channelId).

7. Consider using latest version of Solidity and lock contract compiler versions

The contracts use solidity version 0.4.24. It is suggested to use the latest version (`0.5.3`) and fix all compiler errors or warnings that arise. Also consider locking the version of the compiler in the pragma statement on the top of each file.

8. Add error strings to require statements

Since version `0.4.22` of solidity, `require` statements can include an error string. Consider adding appropriate error messages to all require statements.

9. Consider using `external`

Consider using `external` for function visibility if the method will only be accessed from outside. This can help save some gas especially in the case of `multiChannelClaim()` where multiple arrays are passed as arguments.

10. Consider following the Solidity style guide

Formatting of the code should be adjusted for maximum readability by making sure you follow the solidity style guide rules. Consider using a linter such as ethlint.

In Registry.sol some internal function names are suffixed with Internal, while others are not. Consider settling on a naming scheme, either suffix all internal/private functions, or prefix them with an underscore, "_". Finally, update fields that use `this` to typecast it to `address(this)`.

11. Consider using Solidity's `modifier` pattern instead of regular guard functions

It is best practice to use Solidity's `modifier` pattern instead of using regular guard functions.

Recommendation
Consider replacing the `requireAuthorization()`, `requireOrgExistenceConstraint()`, `requireServiceExistenceConstraint()`, `requireTypeRepositoryExistenceConstraint()` guard functions with their respective modifiers.

12. Consider writing a function to delete array elements

Instead of repeating code that swaps elements each time an array element is deleted, consider implementing a single function that does that. This should significantly improve code readability and help eliminate many potential bugs.

13. Block timestamp is more convenient than block number

Although timestamp can be, to a certain degree, manipulated by miners, it is considered a more convenient way to track time in smart contracts.

Recommendation
Change the expiration parameter in MultiPartyEscrow.sol to use block timestamp instead of block.number.

14. Sanitize inputs of organization management functions

In `Registry.sol`, most of management functions do not properly sanitize inputs and therefore zeroed address can be passed as organizations members and owner, as well empty strings can be given as organization names.

Recommendation
Assert that given addresses and strings are existent and valid .

15. Unnecessary wrapping of functions that return true or revert, with `require`

Public functions such as `channelExtendAndAddFunds` adopt the pattern of returning true on success and reverting on failure. If that's the case, there's no need to wrap function calls in `require` since the result will never be false.

Recommendation
Remove the wrapping `require` statements when calling such functions.

Closing Summary

Beyond the minor issues mentioned, the contracts were also checked for overflow/underflow issues, DoS, and re-entrancy vulnerabilities. None were discovered. The code could have been more thoroughly tested for edge cases and can be refactored to improve gas efficiency.

The automated scanning tools Securify, Myth and Slither did not produce any true-positive results with respect to known vulnerabilities.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of SingularityNET or its products. This audit does not provide a security or correctness guarantee of the audited smart contracts. 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.