SOLIDIFIED

Audit Report for Turms Message Transport March 19, 2019

Summary

Audit Report prepared by Solidified for Turms Message Transport covering the token and transport contracts.

Process and Delivery

Three independent Solidified experts performed an unbiased and isolated audit of the below rewards contract. The debriefing took place on March 19, 2019, and the final results are presented here.

Audited Files

The following files were covered during the audit:

- TurmsToken.sol
- MessageTransport.sol

Notes

- The audit was based on the solidity compiler `0.5.6+commit.b259423e`
- The audit was performed on commit `3e4a2adaecf4b0a57ed1c50993f6e68d72ac8ff5`

Issues

Major

1. A user won't receive spam messages for all fees

MessageTransport.sol, L 186 - 188: The contract allows setting the `messageFee` higher than the `spamFee`. The implementation of `doSendMesasge(...)` however assumes that spamFee is always higher than messageFee. If `messageFee > spamFee`, the receiver won't receive any spam message if the `msg.value < messageFee`. The sender has to spend the extra amount, in this case (`messageFee - spamFee)`, for sending a spam message.

Recommendation

It is recommended to check for either messageFee or spamFee to ensure that the sender is not paying extra for spam messages.

Amended [29.03.2019]

The issue was fixed and is no longer present in commit `7e0721ea88bf7c1dc1409f9dbdaa1b451a32a89d`

2. Trusted accounts can exploit message transfer

MessageTransport.sol, L 171: Trusted accounts added by the owner can modify the from address while transporting messages. These “trusted accounts” can send messages on behalf of someone else using this permission. A malicious actor may abuse this feature in order to send messages utilizing this abuse of trust.

Recommendation

It is recommended to restrict the feature or let other accounts know that it is sent by a different address. Allowing the user to approve the trusted accounts can also be considered.

Amended [29.03.2019]

The issue was fixed and is no longer present in commit `7e0721ea88bf7c1dc1409f9dbdaa1b451a32a89d`

3. Transaction Ordering Dependence - TOD

TurmsToken.sol: Token approval method is susceptible to TOD attack. TOD is a well-known vulnerability where changing an allowance brings the risk that someone may exhaust both the old and the new allowance by unfortunate transaction ordering. Ref

Transaction ordering dependence can also influence the `totalPoint` calculation dependent upon the time that the snapshot is taken. Users should be wary of this.

Recommendation

It is recommended to increase/decrease the approval limit using different functions. Another possible solution is to first reduce the spender's allowance to 0 and set the desired value afterwards, but it requires two transactions. Also, consider using a well-known and audited open-source library like OpenZeppelin as a base.

Amended [29.03.2019]

The issue was fixed and is no longer present in commit
`7e0721ea88bf7c1dc1409f9dbdaa1b451a32a89d`

4. Non-complaint with ERC-20

TurmsToken.sol: The token contract is not completely compliant with the ERC 20 standard.

1. The events are named differently which will cause incompatibility with clients which listens to the standard ERC 20 events.
2. Initial minting should produce a transfer event from address 0x00.

Recommendation

It is recommended to follow the ERC20 standard. It is also recommended to use a well-known and audited open-source library like OpenZeppelin as a base.

Amended [29.03.2019]

The issue was fixed and is no longer present in commit
`7e0721ea88bf7c1dc1409f9dbdaa1b451a32a89d`

Minor

5. Storing encrypted sensitive data on the Blockchain

MessageTransport.sol: In the present state, the Ethereum Blockchain stores data forever - if any time in the future an account is compromised or if its owner is forced to sign a message to obtain the private encryption key, all the previously sent/received messages could be decrypted.

Amended [29.03.2019]

The issue was fixed and is no longer present in commit
`7e0721ea88bf7c1dc1409f9dbdaa1b451a32a89d`.

6. Register method can be used for updating

MessageTransport.sol, L 85 - 92: The current register method for message transport can be used by the same account for updating the value. Allowing a complete update can cause problems like having to pay a different fee or having a different public-private key for different messages.

Recommendation

It is recommended to check for an existing record before registering. If an update feature is necessary, use a different function with a restriction on fields that can be updated.

Amended [29.03.2019]

The issue was fixed and is no longer present in commit `7e0721ea88bf7c1dc1409f9dbdaa1b451a32a89d`

7. DAI directly transferred to the contract will be lost

TurmsToken.sol: If someone transfers DAI directly to the contract, the `holdoverDaiBalance` and `totalDaiReceived` will not be updated.

Recommendation

Maybe it would be good to provide a method to somehow update the `holdoverDaiBalance` and `totalDaiReceived` based on actual DAI balance. For example, `dai.balanceOf(tokenContract)`.

Amended [29.03.2019]

The issue was fixed and is no longer present in commit `7e0721ea88bf7c1dc1409f9dbdaa1b451a32a89d`

Notes

8. Killing the contract might lock DAI funds

TurmsToken.sol, L 285: In case there is any DAI in the contract, they will be lost if the owner calls `killContract()`.

Amended [29.03.2019]

The issue was fixed and is no longer present in commit `7e0721ea88bf7c1dc1409f9dbdaa1b451a32a89d`

9. Compiler Version not Fixed

Having a set and enforced compiler version is suggested. Contracts should be deployed with the same compiler version and flags that they have been tested the most with. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may have higher risks of undiscovered bugs. Contracts may also be deployed by others and the pragma indicates the compiler version intended by the original authors.

Amended [29.03.2019]

The issue was fixed and is no longer present in commit `7e0721ea88bf7c1dc1409f9dbdaa1b451a32a89d`

10. Use ownable for ownership handling

It is recommended to use ownable.sol for handling ownership in the contract. This can further modularize and even handle the ownership transfer if the current owner address becomes unusable for some reason. It gives more flexibility.

Amended [29.03.2019]

The issue was fixed and is no longer present in commit `7e0721ea88bf7c1dc1409f9dbdaa1b451a32a89d`

11. Add more validations and gas optimizations

It is recommended to add more validations in all functions. For example, checking for null address, checking for existing record etc can save some gas and avoid unnecessary transactions. Also, check the whole contract again for duplicate executions which consume unwanted gas.

Amended [29.03.2019]
The issue was fixed and is no longer present in commit
`7e0721ea88bf7c1dc1409f9dbdaa1b451a32a89d`

12. Simplify calculation and use SafeMath

Use SafeMath for all the arithmetic calculations in the contracts where over/underflow is possible. It is also recommended to write a function in TurmsToken.sol which does the points calculation and use them everywhere it is needed. It is safer and easy to maintain.

Amended [29.03.2019]
The issue was fixed and is no longer present in commit
`7e0721ea88bf7c1dc1409f9dbdaa1b451a32a89d`

13. Code refactoring

MessageTransport.sol:

1. *L 13:* The duplicate idx in `MessageEvent` event seems to do nothing additional and it is recommended to remove them.
2. *L 230 - 231:* Using the construction `if (condition) { revert(); }` instead of `require(condition)`; Require is suggested for far greater readability.

TurmsToken.sol:

1. *L 97,103:* The default function visibility in contracts is public. However, function visibility should be explicitly defined to avoid confusion. To avoid confusion, each individual visibility level should be explicitly declared.
2. *L 239 - 245:* The payDai function currently states that one may optionally transfer Ethereum upon calling the function. For the sake of clarity, it is suggested these steps are removed. Reserve an additional function name, if required, for serving an additional action (such as paying ETH).
3. *L 191 - 122:* Remove onlyPayloadSize modifier. This was fixed in Solidity version 0.5.0 (link)
4. *L 265, 274*: Functions `withdrawEthDividends()` and `withdrawDaiDividends()` ignores the `_amount` parameter.
5. *L 146 - 187:* `transfer(...)` and `transferFrom(...)` contain the same code. The duplicated code could be encapsulated in an internal `_transfer(from, to, amount)` function.

Amended [29.03.2019]

The issue was fixed and is no longer present in commit `7e0721ea88bf7c1dc1409f9dbdaa1b451a32a89d`

Closing Summary

Seven issues were found during the audit which could break the intended behaviour, as well as several informational notes with opportunities for improvement. It is recommended for the Turms Message Transport team to address the issues. It is furthermore recommended to post the contracts on public bounty following the implementation of the suggested fixes.

Amended [29.03.2019]
All issues were either fixed or clarified, and are no longer present in commit `7e0721ea88bf7c1dc1409f9dbdaa1b451a32a89d`.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of the Turms platform 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.