SOLIDIFIED

Audit Report for OB1. March 6, 2018.

Summary

Audit Report prepared by Solidified for OB1 covering the Rewards smart contract.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the below smart contract. The debrief took place on March 06, 2019 and the final results are presented here.

Audited Files

- OBRewards.sol

The files were audited on commit `82aba3ff17119f62daf3ac54f43355ba46d2ad6d` and Solidity compiler version `0.4.24`

Intended Behavior

OB1 will occasionally hold 'promotions' where users who buy goods from "promoted sellers" become eligible for reward tokens (OBT). Each of these promotions will be managed by an instance of the OBRewards contract.

Issues Found

Critical

No major issues were found.

Major

No major issues were found.

Minor

1. Deviation from specifications

In the current implementation, the user can claim rewards for a given promotion several times (if the claimed value is below `maxRewardToBuyerPerSeller`), the amount is currently capped by `maxRewardPerSeller` and `maxRewardToBuyerPerSeller`.

Per the specifications: Each buyer may be rewarded tokens for purchasing from a given promoted seller only **once** per promotion.

If this ever happens, `sellerVsBuyers` array will contain the multiple instances of the same buyer.

Recommendation

Adjust so that users can claim rewards only once per promotion or update the specification to match the implementation.

Follow up [19.03/2019]

The issue has been fixed and is no longer present in commit `0ba78da7bd4aa13723a4022620ac753d4432e202`.

2. The Owner can exploit the promotion

The owner possesses the ability to change the reward amount or even transfer the tokens to some random address. This will allow the contract to show behaviours that are different than those advertised as per the campaign.

Recommendation

It is recommended to lock the owner from changing the reward parameters or token transfers during the promotion. This can help increase the user's trust in the system.

OB1's response:

"We chose not to make any changes for this issue. We want the owner to have the ability to modify the promotion even while the promotion is underway. We acknowledge that this means

users won't have cryptographic guarantees that the promotion will go as promoted, but we're okay with this."

Notes

3. Adding promoted sellers in constructor can be simplified

Use the already defined function `addPromotedSellers` in the constructor for avoiding code duplication. Also, add the null address check in the function for better validation (The constructor L145 checks that `promotedSeller` is not `address(0)`. The `addPromotedSellers(...)` does not have this check.).

Follow up [19.03/2019]

The issue has been fixed and is no longer present in commit `0ba78da7bd4aa13723a4022620ac753d4432e202`.

4. External contract calls

The contract contains many calls to external functions which can make the system vulnerable. The author has stated them as trusted calls hence it is not marked as an issue.

OB1's response:

"We're okay with this, as the external calls are to a trusted contract and are needed in order to verify eligibility for rewards"

5. Add end date validation in constructor

It is recommended to add a validation which checks whether the end date is greater than the current time to the constructor. Using the existing `setEndDate` function with the same validation will also work.

Follow up [19.03/2019]

The issue has been fixed and is no longer present in commit `0ba78da7bd4aa13723a4022620ac753d4432e202`.

6. Renounce Ownership

The contract inherits from OpenZeppelin's Ownable implementation, and it includes the `renounceOwnership` function.

While there are use cases for renouncing ownership of a contract, the in-scope contracts do not specify such a situation, resulting in unnecessary increase of the attack surface.

Recommendation

Overload the function, allowing the `renounceOwnership` function to be executed only for promotions that are finished, and hold no funds.

OB1's response:

"We did not make any changes to address this note."

7. Consider updating to recent versions

Solidity compiler 0.4.24 and Openzepellin-solidity 2.0.0 were used in the in-scope smart contracts.

Solidity 0.4.25 fixed a bug about the ** operator (not applicable to the in scope contracts, the bug does not manifest when using literals or uint256), as well as some of the findings in the Solidity 0.4.24 audit.

Recommendation

Consider upgrading to the latest stable compiler and OpenZeppelin versions.

OB1's response:

"We decided to stick with the 0.4.24 version of the compiler for this version of the contract and will update it to 0.5+ for future versions."

8. Owner is misspelled onwer in the comment at L264

Consider amending this for increased readability.

Follow up [19.03/2019]

The issue has been fixed and is no longer present in commit `0ba78da7bd4aa13723a4022620ac753d4432e202`.

Issues Found - Escrow.sol

Critical

1. Escrow.sol - Deviation from specifications

L549: Not all ERC-20 tokens return true on successful `transfer(...)`. For example BNB (Binance) token returns true on successful `transferFrom(...)`, but does not return anything on successful `transfer(...)` (For more information see https://medium.com/coinmonks/missing-return-value-bug-at-least-130-tokens-affected-d67bf08521ca)

If such a token is used to fund a trade, it will be stuck in the escrow contract forever.

Recommendation

Adjust the contract to account for non-compliant tokens (or blacklist such tokens from the platform)

OB1's response:

"We don't consider tokens that do not adhere to the ERC20 specification to be ERC20 tokens, and supporting tokens that do not comply with the ERC20 standard are not within the scope of our goals for this contract.

We will never offer non-compliant tokens as a payment option at the UI level (or if we do, we'll wrap them in a contract that makes them compliant). However, we recognize that third-parties may do so via their own UIs. People can, of course, also interact directly with the contract. We adjusted the spec to include an important note about this issue but have not made any code changes.

We like the suggestion of whitelisting tokens that are safe to use with the contract, but this brings up "political" concerns for us that we have to resolve before deciding whether that's

something we're comfortable with. (We're hesitant to put ourselves into a position where a given ERC20-compliant token has to be "approved" by our company before users can use it as a medium of exchange.) We may address this in a future version of the contract."

Minor

2. Unsafe signed message hash calculation in `_verifySignatures(...)` function

The signed message hash includes two arrays of dynamic length - destinations and amounts. When the hash is calculated, those arrays are packed encoded one after another.
As a result, if a user mistakenly signs a message with 3 destinations and 1 amount, such signed message could be validated as 2 destinations and 2 amounts message (one destination would become an amount).

Recommendation

The issue would not be easily exploitable, but it is recommended to separately hash the destinations and amounts arrays and use those hashes to calculate the signed message hash.

OB1's response:

"This is a good catch. Active exploitation of this requires tricking the signer into signing a carefully constructed bad message -- and attackers with that ability are not included in our threat model. The likelihood of passive exploitation (the user accidentally signing a bad message that can be used to exploit this) is negligible, so we've decided this is low enough risk that we won't make any changes for this version. In general, though, we absolutely agree that there should never be any ambiguity with regards to what message was signed by the signer. We intend to address this in a future version of the Escrow contract."

Closing Summary

Although no critical or major issues were found, there are a couple minor issues that can affect the desired behavior and it's recommended that they're addressed before proceeding to deployment.

The Escrow.sol smart contract was not in scope for this audit, but issues found (including a critical one) are reported in a separate section of the report. We have not fully audited Escrow.sol, so the report above should be considered informative and not a comprehensive audit of the smart contract.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of the OB1 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.

Solidified Technologies Inc.