Coinfix Token Audit

JANUARY 26, 2018 | IN SECURITY AUDITS | BY OPENZEPPELIN SECURITY

The Coinfix team asked us to review and audit their `MerchantSubscription` contract. We looked at the code and now publish our results.
The audited code is located in the ChainXcom/coinfix.io-smart-contracts repository. The version used for this report is commit `75edd659ce96c88fe2784f3e767d6617787e8b7c`.

Here is our assessment and recommendations, in order of importance.
Update: *The Coinfix team has followed most of our recommendations and updated the contracts. The new version is at commit* `c02c083173fee08fb6e8e20ff854e5fbbbe9ed55`.

Critical Severity

No issues of critical severity.

High Severity

No issues of high severity.

Medium Severity

Merchant address can be null

A `MerchantSubscription` contract can be instantiated with the null address as the `merchant` parameter. Such an instance will never be able to be activated or function at all. Consider adding a sanity check in the constructor to require that the `merchant` parameter is different from `0x0`. This will also be in line with the principle of failing as early and loudly as

possible.
Update: *Fixed in this commit by checking* `merchant != 0x0`.

Low Severity

Solidity version

The contracts can be compiled with versions of the compiler as low as `0.4.11`. We recommend changing the Solidity version pragma to a more recent version (for example, `pragma solidity ^0.4.17`) to enforce the use of an up to date compiler.
Update: *Fixed in this commit.*

Reuse open source contracts

The contract implements similar functionality to code found in OpenZeppelin's `SafeMath`, `Ownable` (or Clai`mable`) and `Pausable`. Reimplementing functionality instead of reusing public and already audited code can bring regression problems and difficult to find bugs. Consider removing the duplicate code from your repo and using the installed versions from OpenZeppelin.
Update: *Fixed in this commit by using OpenZeppelin's contracts.*

Redundant amount state variable

The contract has an `amount` state variable, which intends to track the amount of ether received. However, it is redundant because such a value is already accessible via `this.balance`.
Additionally, bear in mind that the contract can receive ether bypassing the fallback function (see the second warning here. This would leave the `amount` variable out of sync with the actual balance, and the ether received in such a way unrecoverable. Moreover, the `SubscriptionPaymentMade` event will not be emitted for these cases.
Consider removing this variable from the contract, and using `this.balance` instead. After this removal `SafeMath` will be unnecessary.
Update: *Fixed in this commit by using* `this.balance`.

Notes & Additional Information

- Neither the `constructor`, fallback function nor the `claimOwnership` method have explicit visibility. Solidity defaults to `public`. Consider making this explicit. This will also avoid some compiler warnings introduced in the latest versions, motivated by the first Parity wallet hack.
  (***Update:*** *Fixed in this commit.)*
- The contract's `version` state variable could be declared `constant`.
  (***Update:*** *Fixed in this commit.)*
- The `SafeMath` methods could be marked `pure`. Consider using Solidity's `using for` feature for enhanced readability.
  (***Update:*** *Fixed in this commit by reusing OpenZeppelin's* `SafeMath`.)
- Consider indexing event parameters such as `SubscriptionPaymentMade`'s `customer` as it enables faster lookups for consuming apps.
  (***Update:*** *Fixed in this commit.)*
- Only the owner can trigger a withdrawal of funds from the subscription. This requires trust in the owner by the merchant. Unless this limitation is by design, consider also allowing the merchant to trigger it.
- The `withdrawal` function transfers ether to the `merchant` address (by using Solidity's `transfer`). Make sure that merchant addresses can receive ether.

Conclusion

No critical or high severity issues were found. Some changes were proposed to follow best practices and reduce potential attack surface.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to Coinfix's contract. We have not reviewed the related Coinfix project. The above should not be construed as investment advice. For general information about smart contract security, check out our thoughts here.

Security Audits

- If you are interested in smart contract security, you can continue the discussion in our forum, or even better, join the team 🚀
- If you are building a project of your own and would like to request a security audit, please do so here.

RELATED POSTS

SECURITY AUDITS

Eyerus Token Au

Metal team asked u