

Nomic Audit - Response

The entire audit is inlined in gray.

Summary

We conducted a security audit of the Paxos Stablecoin system, and no vulnerabilities were found.

Audit results

Critical severity

No critical severity issues were found.

High severity

No high severity issues were found.

Medium severity

No medium severity issues found.

Low severity

No low severity issues found.

Comments & recommendations

- Consider using `pragma experimental "v0.5.0"`, as it's been recommended by Solidity since version 0.4.21. This pragma opt-ins to upcoming breaking changes.

We have considered adding the experimental pragma in addition to the version one that we already have. Since adding the additional pragma makes us more likely to work

consistently with future versions we have added it in our implementation contract. This way in the event that an upgrade of the implementation contract to a future version of solidity is necessary to get a security fix we are more likely to have the new contract be consistent with the state created by the old contract.

- `StablecoinImplementation#initialize` returns an unused boolean. The intention for doing so is not clear. We recommend documenting it or removing the return value.

The boolean was meant to indicate whether initialization succeeded, but require was used instead. We have now removed the unused boolean.

- `StablecoinImplementation#setSupplyController` and `transferOwnership` emit events before modifying the state. We recommend using the [Checks-Effects-Interactions Pattern](#) in every function that modifies the state.

The Check-Effects-Interactions is standard pattern. However, emitting events are not interactions with other contracts but simply something added to the event log. Since emitting the event after setting the new owner would require keeping a copy of the old owner (and similarly for the supplyController) we have left these functions the same.

- `StablecoinImplementation#setSupplyController` doesn't check that `_newSupplyController` is not `0x0`, which can lead to accidental misconfigurations in the system.

Checking that the address is not the zero address is a standard safety check for the assignment of addresses. We have added this check as recommended.

Notes on ERC-20 compatibility

ERC-20's functions `allowance`, `approve` and `transferFrom` are not present.

Removing this functions will decrease the interoperability of the token, as most contracts dealing with ERC-20 use them. For instance, the 0x protocol uses them to execute trades without getting custody of the users' funds.

Paxos provided us a system design document that links to [an explanation](#) about why they are not needed. We consider it incomplete and misleading. Before EIP-150 this was the only safe

way of getting a contract to transfer an user's funds, and after it new alternatives appeared. But some alternative is needed, just removing them prohibits many use-cases. One example alternative is ERC-223. But none of them has been widely adopted, so ERC-20 is the safest choice.

The design document also mentions the withdraw pattern in this context. We want to clarify that this is not relevant to ERC-20 tokens. When transferring ETH or an ERC-223 token the receiving account can execute arbitrary code, leading to many possible attacks (see The DAO hack). The withdrawal pattern aims to isolate transfers because of this. ERC-20 tokens don't call the receiving account, so it's not affected by this class of attacks.

The 0x protocol is a good example among many use cases for the approve/allowance/transferFrom pattern in ERC-20. We have added standard implementations for these three methods.

Notes on the Proxy Pattern implementation

The Paxos Stablecoin uses an implementation of the Proxy Pattern with Unstructured Storage. This implementation has some unique properties that can lead to confusion, and one of them may increase the attack surface of the system.

Inlined interface in the proxy

The token implementation's interface has been copied into the proxy contract. We understand that the purpose of this is to have a single ABI definition json to simplify the interaction with the system.

The problem with it is that the proxy's interface can't be changed in the future. If a function is added or removed from the implementation, the interfaces won't match anymore. This situation would be more confusing than not inlining the implementation and having to deal with two ABI definitions.

We consider that it's possible that the industry adopts new extensions to the ERC-20 standard in the future, leading to changes in the token interface and making this issue relevant.

The interface declaration on the proxy has been removed. The purpose was to make the ABI for the proxy simple and easy to support. However, since the ABI for the proxy can be specified and possibly changed after deployment the standard route is more flexible.

Mixed responsibilities of the proxy and its implementation

`StablecoinProxy#upgradeTo` doesn't implement the actual upgradeability, but delegates the responsibility of doing so to the implementation. This makes the system harder to understand and test in isolation. Furthermore, if a new version of the implementation breaks the `upgradeTo` functionality it won't be fixable.

The proxy also delegates the authentication of `upgradeTo` to the implementation. We consider that it should be part of the proxy.

Refactoring this can lead to a simpler system, as there wouldn't be a need to access the proxy's `implementation` field from the actual implementation.

Based on all of the feedback on the proxy, and the desire to do something nearly standard, we are switching to zeppelinox/zos-lib `AdminUpgradeabilityProxy`, which stands on its own.

The `owner` role has too many responsibilities

`StablecoinImplementation` has an `owner` role that has some administrative privileges on the system (ie: pausing/unpausing and reassigning the different roles) and is responsible for managing upgrades.

We believe that these responsibilities should be split in two different roles, one to administer the system, and another one to upgrade the system. The reason for doing so is that the former can be used during normal operations, and the latter is the most critical part of the system and rarely used. Assigning these privileges to the same role means exposing the upgrade mechanism every time an admin task is done.

Splitting these into two separate roles would lead to a cleaner implementation, as there wouldn't be a need to access the proxy's role from the implementation.

See below. We now have an additional admin role for upgradeability and the owner is purely responsible for pause.

Upgrades are not atomic

The proxy offers no way to upgrade its implementation and migrate its storage in a single transaction. This functionality is needed to enable future upgrades that can't work before some migration logic is run.

Not changing the implementation and running the migration logic in a single transaction can lead to users interacting with uninitialized parts of the contract.

AdminUpgradeabilityProxy has `upgradeAndCall` for precisely this problem.

Recommendations

Consider using one of the available implementations of upgradability instead of implementing your own. We highly recommend using [ZeppelinOS Lib](#).

We have taken this recommendation. AdminUpgradeabilityProxy from ZeppelinOS stands on its own and is failsafe against bad upgrades. We can no longer update the methods on the proxy, but they are only callable by the admin.