Gnosis Multisig Wallet Audit

MARCH 6, 2018 | IN SECURITY AUDITS | BY OPENZEPPELIN SECURITY

We reviewed and audited the Gnosis multisig wallet contract for our own internal use, and choose to publish our findings for informational purposes. We also shared this report privately with the Gnosis team.

We found the code under scrutiny to be elegant, robust, and secure. The wallet's features are implemented with a minimal amount of code, resulting in a reduced attack surface. Despite such minimalism, its well-thought-out design allows for a surprisingly large feature set.

The audited code is located in the github.com/maraoz/MultiSigWallet repository, which is a fork of the original code found at github.com/gnosis/MultiSigWallet. The version used for this report is commit `585863178330d3c64855d596caba2b7f3271a423`. Only the code in contracts/MultiSigWallet.sol was audited.

Here is our assessment and observations on possible improvements, in order of importance.

Critical Severity

No issues of critical severity.

High Severity

Incomplete test coverage

Of the features implemented in `MultiSigWallet`, only the following are tested:

- Funding a wallet
- Submitting a transaction
- Confirming a transaction
- Changing the required number of confirmations
- Executing a transaction

A considerable amount of features are not tested, such as:

- Adding an owner
- Changing an owner
- Removing an owner
- Trying to add duplicate owners
- Trying to remove all owners
- Trying to set an invalid number of required approvals
- An Owner trying to execute a transaction that is not yet approved by other owners
- Non-owners trying to submit/approve/execute transactions

Consider making use of a coverage library such as solidity-coverage and increasing the test coverage to match all the features, behaviors and edge cases implemented in the `MultiSigWallet` contract.
Update: *The Gnosis team informed us that there are some tests which were not ported to the truffle testing framework.*

Medium Severity

`replaceOwner` can produce an invalid owner set

Since moving funds out of the wallet always requires a transaction, and at least one owner to authorize it, a wallet with no owners or invalid owner addresses (such as the address `0x0`) would be unable to withdraw its funds.
To guard against this problem, the constructor function uses the `validRequirement` modifier to guarantee that at least one valid owner is set.
Similarly, the function `removeOwner` will not allow its last owner to be removed because, in such situation, it would call `changeRequirement` with zero as a parameter, which is not a `validRequirement` and would end up triggering a revert on the `removeOwner` call.
However, the function `replaceOwner` does not check for the validity of the incoming owner address, which means that an address such as `0x0` could be set for a new owner. This will produce the unwanted situation described above, where funds cannot be withdrawn from the wallet.
Consider using the `notNull` modifier in `replaceOwner` to avoid such a situation.

Low Severity

Duplicate code for confirmation count

The function `getConfirmationCount` iterates over the `owners` array and verifies that each owner has confirmed a given transaction id, counting the total verifications. The function `isConfirmed` does the same thing, but finally checks if the total count is larger or equal to the wallet's required approval count for confirming a transaction.
Consider calling `getConfirmationCount` from within `isConfirmed` to avoid code duplication.

Outdated Solidity version

The latest solidity version at the time of this writing is 0.4.20. The contract uses 0.4.18 and truffle is setup for version 3.4.9, which uses the 0.4.15 solidity compiler. As a result, all tests fail because the version specified in solidity is larger than the available compiler version.
Also, despite the solidity version of the contract being 0.4.18, deprecated usage of `constant` in functions still exists.
Consider updating the solidity version in the contracts to 0.4.20, updating to truffle version 4.0.0, and replacing all usage of `constant` in functions to `view` (or `pure` if applicable).

Functions that return variable sized arrays should be external

As addressed in the solidity documentation, section *"Can you return an array or a string from a solidity function call?"*, functions can return variable sized data only in external calls. The functions `getConfirmations` and `getTransactionIds` have public visibility, which implies that they can be called internally. Even though such an internal call does not exist in the contract, a future modification may add one, producing unexpected results.

The bodies of the mentioned functions do construct and return fixed-size arrays, in what might be an attempt from the developer to mitigate this problem.
Consider simplifying the code by not using fixed sized arrays at all, and changing the visibility of the functions to `external`.

Notes & Additional Information

- The contract never specifies `uint` resolution — It is a good practice to specify weather a `uint` is `uint256`, `uint64`, etc, in all cases. Doing so has multiple benefits such as reducing storage space, avoiding overflows, etc.
- Inconsistent variable naming conventions — The contract adopts the convention of using underscore in function parameters in order to distinguish them from contract members. However, it does so partially. For example, the function `changeRequirement` uses underscore, while `addOwner` doesn't. Consider using underscore in all method parameters.
- The code would benefit from minor renaming to make it more explicit and easier to read. For example, the `confirmed` modifier, which checks if a given transaction id has been confirmed by a given owner, could be renamed to `txConfirmedByOwner`. Similarly the modifier `notConfirmed` could be renamed to `txNotConfirmedByOwner`. The central variable `required` determines the minimum number of owner confirmations that a transaction needs for approval, and a name like `requiredConfirmations` would be clearer. Finally, `changeRequirement` could be changed to `changeRequiredConfirmations`.
- `getTransactionIds` does not check if `to` is larger than `from`.
- The fallback function should explicitly declare `public` visibility.

Conclusion

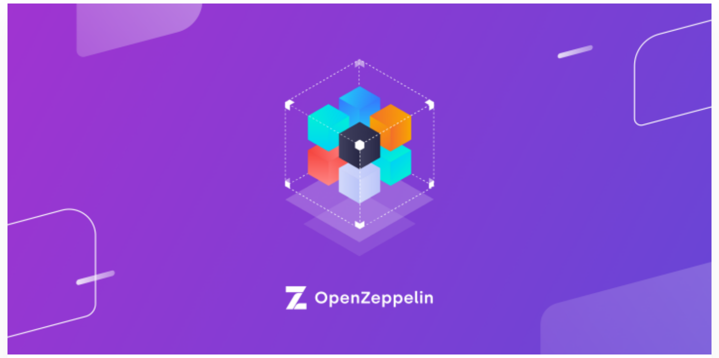
No critical severity and one high severity issue was found and explained, along with recommendations on how to fix it. Some changes were proposed to follow best practices and reduce potential attack surface.

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the Gnosis multisig wallet contracts. 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



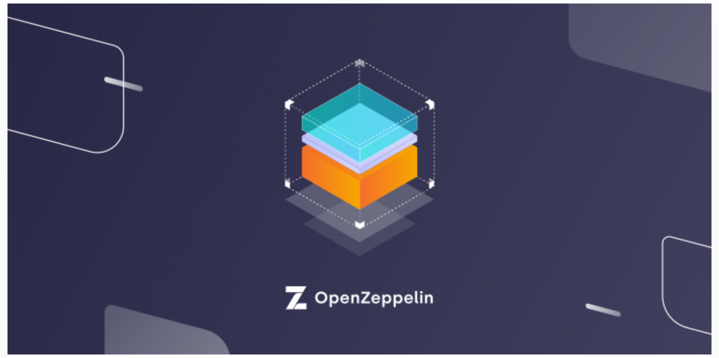
GUIDES

Exploring upgradeability governance in ZeppelinOS with a Gnosis MultiSig

With the first release of ZeppelinOS two months ago, the key issue we wanted to tackle was...

READ MORE

by Santiago Palladino



ANNOUNCEMENTS PRODUCT RELEASES

Announcing the launch of ZeppelinOS

Today we’re excited to announce the first mainnet release of ZeppelinOS, a platform for developing,...

READ MORE

by Demian Brener