Endaoment Audit

OPENZEPPELIN SECURITY | OCTOBER 4, 2020

Security Audits

Introduction

The `Endaoment` team asked us to review and audit their smart contracts. We looked at the code and now publish our results.

We audited commit `f60aa253d3d869ad6460877f23e6092acb313add` of the endaoment/endaoment-contracts repository. The scope includes all the smart contracts in the repository, except the mocks used for testing.

About Endaoment

The `Endaoment` project is a Donor-Advised Funds platform, where everyone is free to transfer their `ERC20` tokens to a `Fund` contract to fund approved organizations with the use of grants. More grants can be created for the same fund and when a specific grant is finalized an amount of donated tokens is transferred to the recipient organization. The funds received by an organization from the grant can be then cashed out.

Anyone can propose claims containing basic information about an organization but an administrative governance choses which claims to approve, which organizations to allow and which grants can be finalized. The entire system is governed by owners of the `EndaomentAdmin` contract which stores and provides specific role accesses to users interacting with the system. They also manage two factories to deploy `Org` contracts, corresponding to allowed organizations, and `Fund` contracts.

functions, included the ones to move tokens from `Fund` contracts to `Org` contracts and to cash them out.
The creator of this contract can transfer their ownership with a two-step transfer and is in charge of setting and revoking the following roles to the corresponding addresses:

- `PAUSER`: to pause other roles and disable them temporary. Unpause them is an exclusive right of the contract `owner`.
- `ACCOUNTANT`: to finalize grants, cash out organizations and to create `Fund` or `Org` contracts.
- `REVIEWER`: to change the wallet associated to an organization and overwrite the fund's manager address.
- `FUND_FACTORY` and `ORG_FACTORY`: to deploy new `Fund` and `Org` contracts.
- `ADMIN`: to have full control over the system and the exclusive role of deploying new factories.

The `Fund` contract has a specific fund manager that can create new grants.

A two-step transfer of ownership is, in general, a safe pattern to follow, but users must be aware of the fact that a transfer is not completed when initialized. Instead, this is completed when the new `owner` accepts the transfer. If an `owner` is trying to cancel the ownership transfer, it can be front-runned by the recipient owner trying to mine a transaction that accepts the transfer before the cancellation happens.

Summary

Overall the code is extensively commented and easy to understand with very well encapsulated contracts and functions. However, our main concern are the vulnerabilities we found in the roles implementation that can lead to loss of funds. The privileged roles built in the system are in general a safe way to protect access to funds management, but they should be finely implemented to avoid any possible attack from malicious actors. Also the lack of technical documentation makes it hard to understand the exact purpose and functionality of each one of the contracts, and thus makes them hard to audit.

once updated.

Update: *The Endaoment team did a great job reviewing the code, refactoring when necessary and fixing all the major issues. The fixes changed many lines of code, but the governance model has still the same overall architecture. We are happy to see that a very detailed and complete documentation has been created and made available for users and developers willing to learn more about the system. Even if all major issues have been fixed, the changes done produced a completely different code base which may benefit from another audit process, perhaps with a restricted scope made out of the contracts that had more changes. We recommend the Endaoment team to do extensive beta testing on their system before releasing it to production.*

Critical severity

[C01] Governance restrictions can be bypassed

Update: *Fixed in pull requests #76 and #77. The Endaoment team created a new `EndaomentAdminStorage` contract that keeps track of the current `EndaomentAdmin` implementation saved in the `endaomentAdmin` parameter. This contract extends from the `Administratable` one and also provides a restricted method to eventually change the address of the implementation. The factories now extend from such contract and their addresses are stored in the `Fund` and `Org` contract whenever a new one is created. In this way the `endaomentAdmin` of both factories is passed as input parameter in the `Administratable` modifiers whenever a restricted function of the `Fund` or `Org` contract is called. It has to be noticed that, before interacting with the system, the correct roles must be set in the `EndaomentAdmin` contract configured in the `EndaomentAdminStorage` one. Even if this fixes the issue, an address is still passed as input parameter in the `Administratable` modifiers and a new contract has been generated, slightly increasing the gas cost.*

The entire system relies on the `EndaomentAdmin` contract to restrict access to functions and to separate user roles.

The creator of the `EndaomentAdmin` contract is the one in charge of setting specific roles to addresses.

modifiers, used to restrict function calls in the derived contracts.
These modifiers receive the `adminContractAddress` as input parameter and use it to instantiate the governance contract `EndaomentAdmin`.
They then ask to the `EndaomentAdmin` contract at the specified address if the `msg.sender` has the proper rights and it eventually reverts if it's not the case.

The issue is that all the functions making use of these modifiers are `public` and receive an arbitrary `adminContractAddress` as input parameter.
Functions restricted by the `onlyAdminOrRole` modifier are:

- `changeManager`, `finalizeGrant` of the `Fund` contract.
- `createFund` of the `FundFactory` contract.
- `approveClaim`, `cashOutOrg` and `setOrgWallet` of the `Org` contract.
- `createOrg` of the `OrgFactory` contract.

While the `onlyAdmin` modifier is exclusively used to protect `FundFactory` and `OrgFactory` constructors.

This opens the doors for the following attack:

- Alice deploys her own `EndaomentAdmin` contract and sets her address as having full access (i.e. by setting every role to her address).
- She can now call any of the restricted functions listed above or deploy new factories, passing the address of her malicious `EndaomentAdmin` contract as `adminContractAddress` in the function call parameters.

There are several ways now Alice can benefit from this scenario by just using the malicious `adminContractAddress`:

- Every `Org` contract instance, if it has been deployed by the `OrgFactory`, would result in an `allowed` organization. The `createGrant` function of the `Fund` contract checks allowed organizations calling the `checkRecipient` function. Alice can:
- Create a malicious `Org` contract by calling the `createOrg` function of the `OrgFactory` contract making it an allowed organization.

`msg.sender == manager`. She will bypass the `require` statement in line `101` and successfully create a grant to her malicious organization.

- She can also finalize any grant moving out of the contract the funds needed to cover it by calling the `finalizeGrant` function. Together with the previous scenario, she can definitely steal all the funds from the contract.
- Once every fund is transferred out to the malicious organization she can cash out everything by calling the `cashOutOrg` function of the `Org` factory.
- She can finally change the `orgWallet` of any of the deployed `Org` contracts by calling the `setOrgWallet` function or approve any arbitrary malicious claim by calling the `approveClaim` function.

Moreover, all the constructors of the cited contracts are protected by both modifiers, where the `adminContractAddress` is passed as constructor parameter. Since these modifiers can be skipped, the constructors of the factories and of `Fund` and `Org` contracts can be easily called.

To solve the issue consider doing the following:

- Set the current implementation of the `EndaomentAdmin` contract in the constructor of the `Administratable` contract whenever it is called by the derived contract constructors. Save it as state variable.
- Remove the `adminContractAddress` input parameter from any function making use of it and from the modifier definitions.
- Change the modifier implementations to ask directly to the previously set `EndaomentAdmin` contract for `msg.sender` roles.
- Implement an auxiliary function, with restricted access, to change, eventually, the address of the correct `EndaomentAdmin` implementation.

[C02] Fees are wrongly calculated

are bounded by the web application.

In the `finalizeGrant` function of the `Fund` contract, the `fee` parameter is calculated as `grant.value/100`. The entire system assumes that fees are paid to the admin of the fund whenever a grant is finalized.

Due to the Solidity truncating rule for division, if `grant.value < 100`, then `fee` will be zero and nothing will be paid to the `admin`.

The rounding error introduced when dividing will always round down the fees paid to the `admin`.

Consider handling the rounding errors using a library for decimals or introducing specific logic to handle them. Moreover, consider calculating the `finalGrant` variable as the `grant.value - fee`. This would make consistent the fact that both, if summed up, equal the `grant.value`.

High severity

[H01] Unsupported `ERC20` tokens can be stuck in the contract

Update: *Fixed in pull request #68.*

Contract interfaces are contracts that declare function signatures without implementing them. They are used as an specification that decouples the signature of the functions from their implementation, allowing to interoperate with different implementations as long as they adhere to the specification.
In the codebase, the `ERC20` token implementation by OpenZeppelin is used in the `Fund` and `Org` contracts, instead of using the interface.
The address of the `ERC20` token to be used is passed as input parameter.

Given the fact that one can pass arbitrary addresses whether or not they represent valid `ERC20` tokens and that not every `ERC20` contract is implemented exactly as the OpenZeppelin one, it can happen that transactions calling those functions fail due to a mismatch between the `ERC20` OpenZeppelin contract and the target implementation represented by the `tokenAddress`.

Another consequence is that, since anyone can transfer arbitrary `ERC20` tokens to the `Fund` or `Org` contract, if someone accidentally deposits an `ERC20` token which has a different

As suggested in [N03], consider using the `IERC20` interface importing it from the `openzeppelin-contracts` package to allow interoperation with any possible implementation of the `ERC20` token standard and avoid the loss of the funds deposited in the contracts.

[H02] Not following the Checks-Effects-Interactions pattern

Update: *Fixed in pull request #63.*

The `finalizeGrant` function of the `Fund` contract is setting the `grant.complete` storage variable after a token `transfer`.

Solidity recommends the usage of the Check-Effects-Interaction Pattern to avoid potential security issues, such as reentrancy.

The `finalizeGrant` function can be used to conduct a reentrancy attack, where the `token transfer in line 129` can call back again the same function, sending to the `admin` multiple times an amount of `fee`, before setting the `grant` as completed.

In this way the `grant.recipient` can receive less than expected and the contract funds can be drained unexpectedly leading to an unwanted loss of funds.

Consider always following the “Check-Effects-Interactions” pattern, thus modifying the contract’s state before making any external call to other contracts.

[H03] Token transfers can silently fail

Update: *Fixed in pull request #69.*

The `finalizeGrant` function of the `Fund` contract and the `cashOutOrg` function of the `Org` contract are both doing token `transfers` from the contract to some recipients.

In both cases, the `balanceOf(sender)` is not checked to be greater than `0` or greater or equal than the amount that is going to be transferred to the recipient.
Moreover, the returned values of the `transfer` methods, if present, are not checked.

To avoid finalizing grants without actually transferring tokens to the recipient organization, consider always checking the result of a token transfer and to validate first the amount to be transferred. This could also save some gas by not emitting unnecessary events.

Medium severity

[M01] Lack of input validation

Update: *Fixed in pull request #74.*

There is a general lack of input validation in the entire code base. Some examples are:

- The `onlyAdmin` and `onlyAdminOrRole` modifiers, the `changeManager`, `finalizeGrant`, `checkRecipient` and `getSummary` of the `Fund` contract and the `cashOutOrg` and `setOrgWallet` of the `Org` contract, are not checking whether the addresses passed as input parameter are the zero address.
- The `createGrant` function of the `Fund` contract doesn't check any of the grant input parameters before creating a `grant` and push it into the `grants` array. Only the recipient is checked to be an allowed organization.
- The constructor of the `Org` contract is not checking if the passed `ein` value is zero.
- The `claimRequest` function of the `Org` contract is not validating if the `fName`, `lName` and `eMail` parameters are empty.
- The `approveClaim` function of the `Org` contract and the `finalizeGrant` of the `Fund` contract are not checking if the `index` parameter is accessing to a non existing element in the corresponding array.

The lack of validation on user-controlled parameters may result in erroneous or failing transactions. Note also that some user interfaces may default to sending null parameters if none are specified.

Consider reviewing the entire code base and add input validations, especially when an user-controlled input parameter is used to store a value in the contracts.

[M02] Non flexible data objects

Update: *Fixed in pull request #75.*

lacking of auxiliary functions to modify their elements after creation.

In particular, allowed organizations cant be disallowed, grants can’t be modified nor deleted after being finalized, and organization claims can’t be modified.

Moreover, `claims[]` and `grants[]` have no protection from duplicates when new elements are added.

In order to handle unexpected errors and increase the flexibility of the system, consider adding auxiliary functions, with the proper access restriction, to modify and delete, when necessary, such objects from the arrays or mappings.
Also consider adding a duplicate protection to the arrays using a key-value mapping to rapidly check if an element has been already added or not.

Low severity

[L01] Missing error messages

Update: *Fixed in pull request #53.*

There are several require statements in the `Fund` and `EndaomentAdmin` contracts without error messages. In particular:

- In line 188 of the `EndaomentAdmin` contract.
- In line 43, 52,101 and 123 of the `Fund` contract.

Consider including specific and informative error messages in all require statements.

[L02] README file is missing important information

Update: *Fixed in pull request #60.*

The README.md of the Endaoment project has little information about what is the purpose of the project nor how to use it. README files on the root of git repositories are the first documents that most developers often read, so they should be complete, clear, concise, and accurate.

workflows, the public APIs and how the code relates to other key components of the project.

Furthermore, it is highly advisable to include instructions for the responsible disclosure of any security vulnerabilities found in the project.

[L03] Missing docstrings

Update: *Fixed in pull request #47. Moreover some of the mentioned functions are not existing anymore or they have been refactored.*

The `setOrgWallet`, `getTokenBalance` and the `getClaimsCount` functions of the `Org` contract are lacking any sort of docstrings.

This hinders understanding of the code's intention, which is fundamental to correctly assess not only security, but also correctness. Additionally, docstrings improve readability and ease maintenance.

Consider thoroughly documenting the missing functions (and their parameters) that are part of the contracts' public API.

[L04] `SafeMath` library is not used

Update: *Fixed in pull request #54.*

In the code base, the `SafeMath` contract library is declared to be used against `uint256` variables in the `Fund` contract, but then it's not used in lines 127-128.

To avoid any possible underflow/overflow or division by zero, consider always relying on the proven safety of the `SafeMath` library by OpenZeppelin and using it when performing mathematical operations. Also consider removing line 19 of the `Org` contract if the contract doesn't need to perform calculations.

[L05] Misleading docstrings

Update: *Partially fixed in pull request #78. Most of the docstrings have been fixed, some of them have been added or refactored.*

- Line 16 of the `Fund` contract is incorrectly assuming that the token transfer is done using the `SafeMath` library, but since the token address used to do the transfer is arbitrary, then the `transfer` implementation can be different and then the assumption doesn't hold.
- Lines 14,15 and 16 of the `FundFactory` contract state that it provides a way for fetching individual `Org` contract addresses and a list of `allowedOrgs` but there is no way of retrieving such information from the contract.
- Lines 231 and 232 of the `EndaomentAdmin` contract are stating that the permitted roles to pass the modifier's restrictions have to be a "bot commander (0)" or a "pauser (1)". In reality, the modifier can be called with any of the six roles permitted and it only checks if that role exists or if it's paused. Moreover it's not clear what "bot commander" means.
- Line 182, 172 and 158 of the `EndaomentAdmin` contract say "External" when the function visibility is `public`.
- Lines 27 an 28 of the `Administratable` contract, the comment says that the only admitted roles are `ADMIN`, `ACCOUNTANT` and `REVIEWER` while the modifier actually checks also for the `FUND_FACTORY` and `ORG_FACTORY` roles.
- Line 184 of the `EndaomentAdmin` contract says `admin` where it should be the `holder` or `roleAddress` as the returned named parameter. The same happens in the `IEndaomentAdmin.getRoleAddress` function definition.
- Line 79 and 116 of the `Fund` contract and line 88 of the `Org` contract are mentioning stablecoins. There is actually no restriction on the type of `ERC20` tokens that can be passed as input parameter in the functions making use of them and any mention should therefore be removed.

[L06] Multiple getters for the same state variable

Update: *Fixed in pull request #73.*

Some contracts in the code base contain multiple public getter functions for the same state variable.

For example:

- In the `FundFactory` contract, the automatically generated getter for the `public` `createdFunds` array is duplicated by the explicit `getFund` getter.

To avoid duplication, ensure that there is at most one publicly exposed getter for each contract state variable. Consider making all the state variables private, following the style of the openzeppelin-contracts package.

[L07] Useless input parameters

Update: *Fixed in pull request #66.*

In the `claimRequest` function of the `Org` contract, the `orgAdminAddress` is passed as input parameter. In line 60 then, the `msg.sender` is required to be the same address as the `orgAdminAddress`.
The same happens with the `fSub` parameter which is required, in line 59 to be `true`.

The function is `public` and anyone can call it. The `msg.sender` can freely choose their address as `orgAdminAddress` and `fSub` as `true`.

Given the fact that the `orgAdminAddress` and `fSub` are not used for any other purpose we can conclude that the require statements in line 59 and 60 are useless and so the `orgAdminAddress` and `fSub` arguments. Consider removing them.

[L08] Lack of event emission

Update: *Fixed in pull request #52.*

The `changeManager`, `createGrant`, `finalizeGrant` functions of the `Fund` contract, the `claimRequest` and the `setOrgWallet` functions of the `Org` contract are not emitting any event after changing the value of state variables.

To easily inform clients about state changes in the contract and to let the applications subscribe to such changes, consider always emitting an event when changing important parameters of each contract.

[L09] Missing getter

Update: *Fixed in pull request #55.*

To have access to this state variable without parsing the contract storage in the blockchain, consider adding a agetter function to retrieve its value.

[L10] Incorrect require statement

Update: *Fixed in pull request #56.*

In the `Administratable` contract the `onlyAdminOrRole` modifier in line 48 is requiring the `msg.sender` to be the `ADMIN` address of the `EndaomentAdmin` contract, but this statement is inside an conditional structure with the condition that the `msg.sender` is not the `ADMIN`.
So the `require` statement will be never satisfied and will always revert if evaluated.

This `require` statement should reflect that the caller is not the `ADMIN` or that the role requested is paused and should therefore be a `revert` with an explicative error message if the `else` block is evaluated.

Consider refactoring the conditional statements by providing a cleaner structure, and reverting when the caller is accessing to a paused role and it's not the `ADMIN`.

Notes & Additional Information

[N01] `uint` as `uint256`

Update: *Fixed in pull request #49.*

There are several cases in the code base where the `uint` keyword is used implicitly assuming the `uint256` type.
Some examples can be line 81 of the `Fund` contract, and line 44 of the `Org` contract.

To favor explicitness, all instances of `uint` should be declared as `uint256`.

[N02] Unused state variables

Update: *Fixed in pull request #46.*

[N03] Unused contracts

Update: *Fixed in pull request #50.*

There are files which are not used at all in the code base:

- The `library/SafeMath.sol` file is not imported anywhere. Instead, the `Fund` contract, which is the only one performing mathematical operations, uses the `SafeMath` library that automatically comes from the OpenZeppelin `ERC20` import in the `Org` contract.

- The same happens with the `IERC20` interface. As before, the `ERC20` import in the `Org` contract is used to call `ERC20` token transfers.

To improve readability, security and clarity of the code, consider replacing the `ERC20` import with the `IERC20` interface and removing the `SafeMath` file, importing it, where needed, directly from the OpenZeppelin contracts.

[N04] Style recommendations not applied

Update: *Fixed in pull request #51. Solhint is now used as linter tool.*

The style used in the entire code base is following the `Solidity style recommendations`, however there are still some parts in the code that have some deviations. In particular:

- Events are not named using the CapWord style. Examples are in `cashOutComplete` event of the `Org` contract or in the `fundCreated` event of the `FundFactory` contract.
- There are extraspaces in function definitions between the name and the list of input parameters, but also inside functions implementation or trailing whitespaces at the end of the lines.
- Sometimes, indentation is not correct nor made out of four spaces, like the entire `EndaomentAdmin` contract, that uses two spaces for the indentation while the majority of

some are not.

- The `Org` input parameter of the `getAllowedOrg` function of the `OrgFactory` contract should be in mixedCase format.
- Error messages are inconsistent in style since sometimes they are introduced by the contract name and sometimes they are not.
- An extra space should be added in several functions where the list of input parameters is not separated by the body's brackets of the function.
- Some variables have not explicit and unclear names. Like the `x` and the `t` of the `Fund` contract, the `ein` of the `OrgFactory` contract or the `bal` of the `Org` contract.
- Some variables have names that can be improved to give more context and consistency:
- `newOwner` input parameter of the `transferOwnership` function of the `TwoStepOwnable` contract could be changed to `newPotentialOwner`.
- `_newPotentialOwner` in line 85 of the `TwoStepOwnable` contract could better reflect that that's the `oldPotentialOwner` or the `cancelledPotentialOwner`.
- The `creator` input parameter of the `Fund` constructor can be renamed `_admin` to be consistent with the variable that is assigned to.

Taking into consideration how much value a consistent coding style adds to the project's readability, enforcing a standard coding style with help of linter tools such as Solhint is recommended.

[N05] Incorrect functions visibility

Update: *Fixed in pull request #62.*

Whenever a function is not being called internally in the code, it can be easily declared as `external`, saving also gas. While the entire code base have explicit visibilities for every function, some of them can be changed to be `external`.
Examples are the getters in the `OrgFactory` contract or each one of the public functions of the `EndaomentAdmin` contract.

To improve clarity following Solidity recommendations and to better reflect the scope of each function, consider reviewing the visibilities applied.

[N06] Unnecessary import

this contract is never used in the contract's code.

[N07] `Fund` contract can't receive ethers

Update: *Fixed in pull request #61.*

The `Fund` contract is in charge of recollecting the funds to later support organizations with the recollected tokens. The `getSummary` function is returning the `ETH` balance of the contract but the contract has no payable functions to receive them.

Consider whether removing any reference to `ETH` if the contract is not intended to use it as a valid currency, or implementing the missing functionality to correctly receive and move them out from the contract.

[N08] Typos in the code

Update: *Fixed in pull request #42.*

There are some typos to fix in the code base. In particular:

- Line 76 of the `Org` contract is repeated.
- In line 59 of the `OrgFactory` contract, "is provided" should be "if provided".
- In line 16 of the `Org` contract, "direct received" should be "directly receive".
- In the docstring and in the input parameter of the `cashOutOrg` function of the `Org` contract the `desiredWithdrawlAddress` and `withdrawl` words should be changed to `desiredWithdrawalAddress` and `withdrawal`.
- In the `Fund` contract the word `reccomendation` should be changed for `recommendation` in several lines.
- In the `Administratable` contract in line 8 `Provides a of modifiers` should be `Provides two modifiers`.

[N09] Unclear function purpose

Update: *Fixed in pull request #48.*

input parameter.

On the other side, the `setOrgWallet` function is public but called internally only by the `approveClaim` function and it is a one line function setting the `orgWallet` state variable to the `providedWallet` input parameter.
Moreover the `orgWallet` variable is not used anywhere in the code base and it’s not clear what’s its purpose.

If is an explicit design decision, consider commenting on the docstrings why `approveClaim` is approving a claim or why `setOrgWallet` sets that variable and what’s its purpose.

[N10] Default value as sensible information

Update: *Fixed in pull request #44.*

In the list of all possible user’s roles in the `IEndaomentAdmin` interface, the `ADMIN` role is the `0` value of the `enum` structure which is the default value of the integer type. Relying the `ADMIN` role on such value can be risky and should be avoided since many clients can default to that value if none is provided.

Consider moving the `ADMIN` role to the end of the list and to assign to the `0` value an `EMPTY` role.

Conclusions

2 critical and 3 high severity issues were found. Some changes were proposed to follow best practices and reduce potential attack surface.

Update: *All the reported issues have been addressed and fixed by the Endaoment team.*

Related Posts

Beefy Zap Audit

BeefyZapRouter serves as a versatile intermediary designed to execute users' orders through routes...

Security Audits

OpenBrush Contracts Library Security Review

OpenBrush is an open-source smart contract library written in the Rust programming language and the...

Security Audits

Linea Bridge Audit

Linea is a ZK-rollup deployed on top of Ethereum. It is designed to be EVM-compatible and aims to...

Security Audits