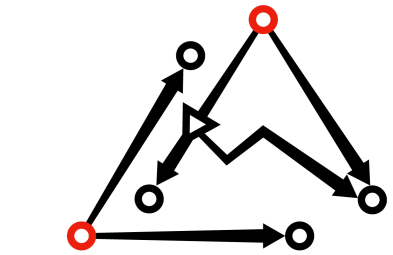

Least Authority

PRIVACY MATTERS

Endaoment V2 Smart Contracts

Security Audit Report

Endaoment

Final Audit Report: 20 July 2022

Table of Contents

Overview

Background

Endaoment has requested that Least Authority perform a security audit of the Endaoment V2 smart contracts.

Project Dates

- **May 11 - June 8:** Initial Code Review *(Completed)*
- **June 10:** Delivery of Initial Audit Report *(Completed)*
- **July 7:** Delivery of Updated Initial Audit Report *(Completed)*
- **July 18 - 19:** Verification Review *(Completed)*
- **July 20:** Delivery of Final Audit Report *(Completed)*

Review Team

- Nicole Ernst, Security Researcher and Engineer
- Ahmad Jawid Jamiulahmadi, Security Researcher and Engineer
- Rosemary Witchaven, Security Researcher and Engineer

Coverage

Target Code and Revision

For this audit, we performed research, investigation, and review of the Endaoment V2 Smart Contracts followed by issue reporting, along with mitigation and remediation instructions as outlined in this report.

The following code repository is considered in-scope for the review:

- https://github.com/endaoment/endaoment-contracts-v2

Specifically, we examined the Git revision for our initial review:

`1d98c02bd1272edb950926ed8e396cd95f378a99`

For the verification, we examined the Git revision:

8bed0b971ad26a6b4ad2c529eedcd78b720cdd2b

For the review, this repository was cloned for use during the audit and for reference in this report:

- https://github.com/LeastAuthority/Endaoment-V2-Smart-Contracts

All file references in this document use Unix-style paths relative to the project's root directory.

In addition, any dependency and third-party code, unless specifically mentioned as in-scope, were considered out of scope for this review.

Supporting Documentation

The following documentation was available to the review team:

- Endaoment Organization documentation: https://docs.endaoment.org/
- Implementation documentation:

https://endaoment.gitbook.io/contracts-v2-documentation/
- OpenZeppelin Security Audit Report: https://blog.openzeppelin.com/endaoment-audit/
- Endaoment V2 Contracts Diagram.pdf *(shared with Least Authority via email on 31 January 2022)*

Areas of Concern

Our investigation focused on the following areas:

- Correctness of the implementation;
- Common and case-specific implementation errors;
- Adherence to the specification and best practices;
- Adversarial actions and other attacks on the smart contracts;
- Potential misuse and gaming of the smart contracts;
- Attacks that impact funds, such as the draining or manipulation of funds;
- Mismanagement of funds via transactions;
- Denial of Service (DoS) and other security exploits that would impact the intended use or disrupt the execution of the smart contracts;
- Vulnerabilities in the smart contract code;
- Protection against malicious attacks and other ways to exploit the smart contracts;
- Inappropriate permissions and excess authority;
- Data privacy, data leaking, and information integrity; and
- Anything else as identified during the initial analysis phase.

Findings

General Comments

The Endaoment system is a Decentralized Autonomous Organization (DAO) that facilitates the donation of cryptocurrency to tax-exempt entities under section 501(c)(3) of the US tax code. The Endaoment V2 smart contracts compose the on-chain components of the system, which are designed to address the regulatory and compliance requirements for an entity to qualify for this tax-exempt status.

Our team reviewed the smart contracts design and implementation for security vulnerabilities that could affect the system or its users. We explored opportunities to drain the funds of an entity by either passing oneself off as a party authorized to do so or through the use of the `delegatecall` method and could not identify any vulnerabilities. We investigated the possibility of overwriting the manager of an existing fund by registering a new fund that clashes with an existing fund and found that this is not feasible. We examined the possibility of the smart contracts being halted as a result of an error or a DoS attack, thereby preventing users from interacting with contracts and found that appropriate use of pull-over-push patterns prevents the contracts from becoming blocked. Furthermore, we reviewed the implementation of the fee collection mechanism and attempted to identify ways to circumvent the fee paid to the Endaoment Organization but could not identify any.

We reviewed the mathematical expressions in the codebase, particularly those in unchecked blocks, and could not identify any possibility for an under or overflow. We examined arithmetic operations and did not identify any implementation errors.

Our team reviewed the implementation of the Curve and Uniswap V3 wrappers. The Curve wrapper calls into a Curve interface function and any errors in the Curve contracts were considered out of scope for this review. In the Uniswap V3 wrapper, we verified that `amountOutMin` is used correctly in the contracts, which makes the possibility of funds leaking to an external address highly unlikely.

Our team investigated potential vulnerabilities resulting from the interaction of non-standard tokens with the Endaoment system of smart contracts and did not identify any. However, this does not guarantee that vulnerabilities do not exist.

Our team was unable to identify any issues in the design or implementation of the Endaoment V2 smart contracts. We found that comprehensive tests and project documentation facilitated this review. However, during the review, the Endaoment team identified a vulnerability in the smart contracts resulting from a missing check to verify that a transfer from a fund is sent to an approved recipient. This would break the business requirements of the Endaoment organization and put the tax-exempt status and operation of the organization at risk. The Endaoment team has implemented a check to verify that the recipient of a transfer is an approved entity (Issue A).

System Design

We found that security has been taken into consideration in the design of the Endaoment V2 smart contracts as demonstrated by use of patterns such as role-based access control, a pause function to improve security, in addition to general adherence to best practices in performing input validation and appropriate implementation of reentrancy guards. Our team checked the locations mentioned in the code comments that could be vulnerable to reentrancy attacks and did not identify any issues. We identified a missed zero check, which we recommend be added (Suggestion 2.)

Code Quality

The Endaoment V2 smart contracts codebase is well organized and generally adheres to best practices. We identified some suggestions to improve the overall quality of the implementation. We recommend that the Solidity compiler versions be declared consistently and that they be pinned to a recent version to avoid unintended behavior resulting from different versions of the compiler (Suggestion 1). We also recommend that functions be defined according to their intended functionality in order to improve readability and reduce confusion about the intended behavior of the implemented functions (Suggestion 3).

Tests

There is extensive unit test coverage implemented, which helps identify implementation errors and check that the system functions as intended.

Documentation

Our team was provided project documentation that sufficiently describes the system. Each file has a brief description in the `README`.

Code Comments

Functions are described in accompanying NatSpec code comments, which proved to be helpful and accurate.

Scope

The scope of this review included all security-critical components.

Dependencies

The implementation uses standard, well-audited libraries as well as more experimental Solmate libraries. Solmate is a more modern, gas-saving alternative to standard that requires the use of assembly in its implementation and can be difficult to reason about. We recommend that well-audited libraries be used.

Specific Issues & Suggestions

We list the issues and suggestions found during the review, in the order we reported them. In most cases, remediation of an issue is preferable, but mitigation is suggested as another option for cases where a trade-off could be required.

ISSUE / SUGGESTION	STATUS
Issue A: No Restriction Preventing Transfer to Unapproved Recipient (Known Issue)	Resolved
Suggestion 1: Use a Non-floating Pragma Version Consistently across the...	Resolved
Suggestion 2: Add a Zero Address Check	Unresolved
Suggestion 3: Define Functions Appropriately	Resolved

Issue A: No Restriction Preventing Transfer to Unapproved Recipient (Known Issue)

Location

`/Endaoment-V2-Smart-Contracts/src/Entity.sol#L251`

Synopsis

The Endaoment team identified a vulnerability, which violates the business requirement that tokens can only move between entities and that any transfer outside of the Endaoment V2 smart contract system, unless performed by an Admin in adherence to strict conditions, is a violation of the legal regulations governing 501(c)(3) entities. This could result in the loss of the tax-exempt status of the Endaoment organization, preventing business operations from continuing. The system of smart contracts is intended to enforce this requirement.

The `transfer` function is intended to be used by a fund manager to transfer donated funds to approved recipients within the Endaoment system. To execute the transfer, the function calls the internal function `_transferWithFeeMultiplier`, which performs the transfer of balances from the fund address to the recipient address. However, no check is performed to verify that the recipient is an approved account. As a result, a fund manager could transfer funds outside of the Endaoment system, breaking the requirement that is intended to be enforced by the smart contracts.

Impact

A transfer of funds by a fund manager to an address outside of the Endaoment system could jeopardize the tax-exempt status of the organization.

Preconditions

The transfer function is used with a recipient address that is outside the Endaoment system.

Feasibility

Straightforward.

Remediation

A check in the internal function `_transferWithFeeMultiplier` must be performed to verify that the recipient address of the transfer operation is an approved entity.

Status

The Endaoment team implemented an additional check to prevent the funds from being sent to an entity that is not approved by Endaoment.

Verification

Resolved.

Suggestions

Suggestion 1: Use a Non-floating Pragma Version Consistently across the Project

Location

[Endaoment-V2-Smart-Contracts/tree/main/src]

Synopsis

Most smart contracts have their pragma set to ^0.8.12. We found several instances of smart contracts with the compiler version set to >= 0.8.0. In other instances, the compiler is set to >= 0.8.12. Additionally, there are instances with no pragma statement set at all. Compiling with different compiler versions may cause conflicts and unexpected results and possibly lead to the smart contracts being deployed with an unintended compiler version, which could result in unexpected behavior.

Mitigation

In order to maintain consistency and to prevent unexpected behavior, we recommend that the Solidity compiler version be pinned by removing "^" and “>=”, updated to the latest version, and used consistently across the system. Additionally, in files where no pragma is declared, we recommend the setting of a pragma statement that is consistent with the rest of the project.

Status

The compiler version has been pinned to 0.8.13 for all project contracts.

Verification

Resolved.

Suggestion 2: Add a Zero Address Check

Location

[src/lib/auth/Auth.sol#L48]

Synopsis

In the `setOwner` function, the zero address check is missing. If the owner is set to zero address accidentally, the ownership of the contract will be lost.

Mitigation

In order to prevent the accidental loss of contract ownership, we recommend adding a zero address check for the parameter “newOwner” in the referenced function.

Status

The Endaoment team has not implemented the recommended mitigation.

Verification

Unresolved.

Suggestion 3: Define Functions Appropriately

Location

Defined Internal:

src/Registry.sol#L120

src/Entity.sol#L130

src/Entity.sol#L180

src/Entity.sol#L251

src/Entity.sol#L440

src/Entity.sol#L455

src/swapWrappers/UniV3Wrapper.sol#L98

src/swapWrappers/UniV3Wrapper.sol#L103

src/swapWrappers/MultiSwapWrapper.sol#L115

src/OrgFundFactory.sol#L161

src/OrgFundFactory.sol#L169

src/portfolios/SingleTokenPortfolio.sol#L68

Defined Public:

src/Entity.sol#L272

src/NVT.sol#L286

src/NVT.sol#L308

src/NVT.sol#L329

src/NDAO.sol#L23

src/AtomicClaim.sol#L19

src/RollingMerkleDistributor.sol#L103

Synopsis

The above-referenced functions in the first set are defined as `internal`. However, they are not being used in any of the derived smart contracts. The functions referenced in the second set are defined as `public` when they should actually be defined as `external`. It is considered best practice to define function access modifiers based on where the function is going to be used in order to improve the readability of the code and make it easier to identify incorrect assumptions about who can call the function.

Mitigation

We recommend defining the referenced internal and public functions, in the function definition, as private and external respectively by replacing the `internal` keyword with `private` and the `public` keyword with `external`.

Status

The Endaoment team has modified function definitions in accordance with the suggested mitigation.

Verification

Resolved.

About Least Authority

We believe that people have a fundamental right to privacy and that the use of secure solutions enables people to more freely use the Internet and other connected technologies. We provide security consulting services to help others make their solutions more resistant to unauthorized access to data and unintended manipulation of the system. We support teams from the design phase through the production launch and after.

The Least Authority team has skills for reviewing code in multiple Languages, such as C, C++, Python, Haskell, Rust, Node.js, Solidity, Go, JavaScript, ZoKrates, and circom, for common security vulnerabilities and specific attack vectors. The team has reviewed implementations of cryptographic protocols and distributed system architecture in cryptocurrency, blockchains, payments, smart contracts, and zero-knowledge protocols. Additionally, the team can utilize various tools to scan code and networks and build custom tools as necessary.

Least Authority was formed in 2011 to create and further empower freedom-compatible technologies. We moved the company to Berlin in 2016 and continue to expand our efforts. We are an international team that believes we can have a significant impact on the world by being transparent and open about the work we do.

For more information about our security consulting, please visit https://leastauthority.com/security-consulting/.

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis

Our audit techniques include manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's website to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. As we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation. We hypothesize what vulnerabilities may be present and possibly resulting in Issue entries, then for each, we follow the following Issue Investigation and Remediation process.

Documenting Results

We follow a conservative and transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even before having verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions

We search for immediate and comprehensive mitigations that live deployments can take, and finally, we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our Initial Audit Report, and before we perform a verification review.

Before our report, including any details about our findings and the solutions are shared, we like to work with your team to find reasonable outcomes that can be addressed as soon as possible without an overly negative impact on pre-existing plans. Although the handling of issues must be done on a case-by-case basis, we always like to agree on a timeline for a resolution that balances the impact on the users and the needs of your project team.

Resolutions & Publishing

Once the findings are comprehensively addressed, we complete a verification review to assess that the issues and suggestions are sufficiently addressed. When this analysis is completed, we update the report and provide a Final Audit Report that can be published in whole. If there are critical unaddressed issues, we suggest the report not be published and the users and other stakeholders be alerted of the impact. We encourage that all findings be dealt with and the Final Audit Report be shared publicly for the transparency of efforts and the advancement of security learnings within the industry.