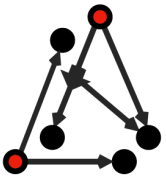
Final Security Audit Report:
Platform Smart Contracts

FundRequest

Report Version: 4 July 2018



Least Authority
Freedom Matters

Table of Contents

- Overview
- Review Scope
  - Target Code and Revision
  - Threat Model
- Findings
  - Excess Authority
    - Security of Funds
    - Legal Liability
    - Dynamic State
  - Documentation
  - Issues
    - Issue A: Excess Authority over Contract State
    - Issue B: Excess Authority over Token Migration
    - Issue C: Security Depends on Contract State
  - Suggestions
    - Suggestion 1: Add Owner Control of Safety Modes
      - Synopsis
      - Remediation
- Recommendations

Overview

Least Authority performed a security review of the *FundRequest Platform Smart Contracts*, at the request of the FundRequest team.

The audit was performed from June 4 - 13, 2018, by Gordon Hall and Dominic Tarr. The initial report was issued on June 15, 2018. An updated report has been issued on July 4, 2018 following a discussion and verification phase.

Review Scope

This scope of this review was all smart contracts in the repository, with the exception of the previously reviewed smart contracts related to the sale and vesting for FundRequest. Third party vendor code is considered out of scope.

Target Code and Revision

For this audit, we reviewed the following repositories:

- https://github.com/FundRequest/contracts/tree/develop/contracts/platform

Specifically, we examined the Git revisions:

```
e4d510171dacc8b817d3201cd0a989b631e01209
```

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

Threat Model

Most of the review effort was focused on the logical interaction of the smart contracts and whether they presented any security vulnerabilities for the platform. The main focus of the security considerations were the theft of funds, but we also reviewed how this could be done through identified vulnerabilities or other ways of gaming the system.

Findings

Excess Authority

Considering that FundRequest aims to be a "decentralized marketplace for open source collaboration", the current versions of the FundRequest smart contracts have significant excess authority problems, similar to the level of control that a centralized service has. This means FundRequest users must totally trust the FundRequest development team to behave honestly, and also to prevent attackers gaining control of their systems.

Nonetheless, we acknowledge that the FundRequest developers are likely to have the best interests of their users at heart. The FundRequest team intends to take a phased approach to decentralizing the platform and acknowledges this excess authority is a strategic choice. However, it should be noted that this audit report findings are based on the version of the codebase noted above, which does not have all of this intended decentralization. Since avoiding excess trust in the platform is a general goal of smart contract systems, we have reported our specific findings of excess control by FundRequest as

vulnerabilities, in the *Issues and Suggestions* sections. In our initial report, we also recommend that the FundRequest team take reasonable measures to secure their devices, especially in the interim before addressing these issues and suggestions. Following the delivery of the initial report, the FundRequest team provided details regarding such measures, which are outlined below.

Operational Security

Our findings in this report reflect vulnerabilities that require an attacker to compromise secrets or gain access to a machine belonging to the FundRequest team. While this type of attack may prove far more difficult than exploiting a vulnerability directly within a contract, we believe it is important to reiterate that avoiding excess trust in the platform is a general goal of smart contract system and we have recommended that steps be taken to eliminate trust and decentralize the platform further - which has been acknowledged and affirmed by the FundRequest team.

The FundRequest team has assured us that the owner of the contracts is only accessible through a hardware wallet and that this hardware wallet is only accessible by a limited number of people. There are 2 copies of this hardware wallet that are stored in separate secure locations.

While we believe this to be a reasonable operational security measure for protecting the secrets that are critical for maintaining control of the FundRequest platform, neither we nor users are able to conduct an audit to verify these measures.

Security of Funds

One important concern that results from this excess authority is the resulting security of the funds managed by the smart contracts, especially considering the irreversibility of stolen cryptocurrency funds. This is perhaps more concerning than the case of centralized repositories of funds because the institutions behind fiat funds have procedures in place for arbitrating stolen funds (like how credit cards can refund unapproved transactions). If someone was to gain unauthorized access and control of the FundRequest platform, they would have no problems stealing the cryptocurrency in use. In the interim phases before this excess authority is addressed, we recommend that the FundRequest team establish processes that can be followed in the event of a breach of control and make these processes that directly impact users transparent to them. According to the FundRequest roadmap, some of this is planned in upcoming releases.

Legal Liability

It is also worth mentioning that there could be legal liability due to this excess authority allowing for the intervention in any transaction. The legally established entity behind the FundRequest platform could be required to take particular actions regarding transaction disputes on their platform. Although this is not legal advice and the legal standing of smart contracts is unclear, we recommend that excess authority is limited or removed if FundRequest intends the performance of transactions to be decentralized. FundRequest has informed us they are seeking legal advice for these concerns and we recommend users be made aware of the terms they need agree to during the interim phases.

Dynamic State

FundRequest uses a modular code style. The contract behaviour as a whole is assembled out of subcontracts which each are responsible for an aspect of that behaviour. From a purely software development perspective, this is an excellent practice, but from a security perspective, this creates some problems. Such as, it's not easy to query the state of a Solidity contract, although it is publicly accessible, it's relatively opaque, and there are currently no good tools for inspecting the state. Consequently, if security depends on the correct state being set, there is no easy way to know that the contract is still in a

secure state. Ideally, designs where the security depends on state should be avoided, or at least, monitoring the state should be implemented (and build such checks into client UIs), and use logs so that state transitions are recorded in a useful way.

Documentation

Although the codebase for the project is new, we think that more documentation of the platform would be beneficial, both for the parts that are completed and what is still under development. Sufficient documentation is a key asset for open source code as it makes technical understanding of the work accessible to non-developers and easier for code reviewers and developers. If FundRequest wants to encourage other parties to review the code or build on the code as part of the strategy, this is especially important. In our initial report, we recommended the FundRequest team look for opportunities to capture key components of the code and expand existing documentation to allow for better maintenance and future audits. Since then, the FundRequest team has added more documentation, code comments, and included an architecture diagram.

Issues

We list the issues we found in the code in the order we reported them.

ISSUE / SUGGESTION	STATUS
Issue A: Excess Authority over Contract State	Reported
Issue B: Excess Authority over Token Migration	Reported
Issue C: Security Depends on Contract State	Reported

Issue A: Excess Authority over Contract State

Synopsis

The owner of the *EternalStorage* contract can set any field to any value, enabling them to completely control any aspect of dependent contracts, both *FundRepositiory* and *ClaimRepositiory*. This means that although the contract infrastructure is technically decentralized, FundRequest users must rely on the *EternalStorage* contract to behave as expected.

Impact

Someone with access to the key that deployed the *EternalStorage* contract has complete control over the funds in and performance of the FundRequest platform.

Preconditions

An attacker compromises any FundRequest developer machine or somehow accesses the key, otherwise.

Feasibility

Unlikely, depending on security measures taken by the FundRequest developers.

Technical Details

FundRepositiory and *ClaimRepositiitory* both store their state by calling out to the *EternalStorage* contract, which they are initialized with in their constructor (stored in their “db” property). These contracts set keys in their *EternalStorage* database by keccak256 hashing a key string, and storing various values at key. The

EternalStorage contract only allows a whitelist of callers or the contract owner to update values, but since the owner can set arbitrary values, and can put any repo or issue being funded into any state. Thus they can steal all money deposited into FundRequest.

Remediation

Instead of directly storing state in *EternalStorage*, *FundRepositiitory* and *ClaimRepositiitory* could maintain their own state, which is only updated via actual transactions from users to the contracts.

Status

Acknowledged by the FundRequest team and stated that decentralizing the platform in a phased approach is on the project roadmap between now and the end of 2019. FundRequest has also stated they have a number of operational security measures in place to prevent attackers from gaining control of the platform.

Verification

Not changed, but mitigated by operational security practices.

Issue B: Excess Authority over Token Migration

Synopsis

Tokens (all tokens held by the contract) can be “migrated” by a contract owner to another contract.

Impact

An attacker gaining control of the deployment keys (thus becoming the contract owner) can simply move all tokens to another contract.

Preconditions

An attacker compromises any FundRequest developer machine or somehow accesses the key, otherwise.

Technical Details

A token held by *FundRequestContract* is simply transferred to another contract. Obviously, this is intended to be an updated version of FundRequest, but an attacker could use this to simply move all tokens held by the contract to a new contract that will let them be spent.

Feasibility

Unlikely, depending on security measures taken by the FundRequest developers.

Mitigation

The simplest improvement which lessens this risk might be to have a safety factor, separate from the contract owner, which can disable token migration until it is needed.

Remediation

Users should retain some control over tokens they have invested into or received through FundRequest. The simple fix would be to migrate tokens on a per-user basis, when that user next interacts with the system. This would still provide a reasonable ability to upgrade the system, but prevent that upgradability from being a hole through which all the user’s money can easily be extracted. It is reasonable for the owner to be able to deactivate a specific version of a contract such that funds deposited can only be withdrawn.

Status

Acknowledged by the FundRequest team and indicated as desired behavior to ease the growth and upgrading of the platform. FundRequest has stated they have a number of operational security measures in place to prevent an attacker from gaining control of the keys required to move funds.

Verification

Not changed, but mitigated by operational security practices.

Issue C: Security Depends on Contract State

Synopsis

Verifying the security of the FundRequest system depends on stored state, not just code. This means verifying whether a running contract is currently secure is not easy.

Impact

Significant. Hidden backdoors can be easily added this way.

Preconditions

An attacker compromises any FundRequest developer machine or somehow gains access to the platform for a few minutes.

Technical Details

Solidity provides a number of high level storage APIs, such as arrays and maps, and these are implemented in terms of the EVM's mstore opcodes. Also, for public fields, Solidity exposes "automatic getters". In some cases, this can completely change the security properties of a system. These properties must be monitored to ensure that they remain in a secure state. For example, *EternalStorage* inherits from *Callable*, and callers (or the owner) can set any value. During contract deployment, *ClaimRepository* and *FundRepository* are set as callers. As long as no other contracts are ever set as callers, then security will be maintained. But there are no safeguards or notifications if these settings change, at any point. If a prepared attacker was able to access a FundRequest developer's machine or accounts for a few minutes, they would only need to issue one command, and could later extract all user's funds. This also affects Preconditions, so a new precondition could be added, possibly affecting overall contract behaviour, likely without anyone realizing.

Also, note, this is similar to the nature of the infamous devops199 parity multisig contract vulnerability - the Parity multisig was vulnerable because it's state wasn't fully initialized, and no one realized until someone accidentally did initialize it, and then destroyed the contract.

Feasibility

Given temporary access to owner permissions, fairly easy. However, ease of access to owner permissions depends on security measures taken by the FundRequest team.

Mitigation

Actively monitor the state of a contract, and log every state change. Have client code check critical state when running the contract.

Remediation

Avoid designs where security depends on stored state.

Status

Acknowledged by the FundRequest team. FundRequest has stated they have a number of operational security measures in place to prevent an attacker from gaining control of the keys required to manipulate the contract state.

Verification

Not changed, but mitigated by operational security practices.

Suggestions

Suggestion 1: Add Owner Control of Safety Modes

Synopsis

The *FundRepository* contracts are not equipped with any methods for remediation. Software development issue tracking can yield more complex scenarios than a simple opened or closed flag. For instance, in the event that a issue is funded but that issue is later superseded by a different issue or the issue is tracking a feature that the maintainer decides is not appropriate (i.e. “wontfix”), there should exist the ability for the contract owner to perform deposit freezes and refunds.

Remediation

Add safety modes that can be controlled by the owners that include particular functions. For example, include the ability to freeze a contract so withdrawals or transfers are allowed but not deposits. Additionally, the ability to “revert” or issue refunds to all funders after freezing the contract could be a desirable method for nullifying the issue in the *FundRepository* in the event that it becomes invalid.

Status

Acknowledged by FundRequest and steps taken to begin resolving.

Verification

Mitigated. A `refundToken` method was added to the `FundRepository` contract that allows caller contracts to issue refunds. FundRequest has stated that further control of safety modes are planned in the future.

Recommendations

We recommend that the *Issues* and *Suggestions* stated above be further addressed as soon as possible in the future development and followed up with security audits. Additionally, we recommend there be further consideration and analysis on all of the overall comments listed in the *Findings* section above.