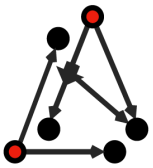
FundRequest: ICO Smart Contracts

Security Audit Report



Least Authority
Freedom Matters

Overview

FundRequest has requested Least Authority perform a security audit of the token sale smart contracts for the upcoming ICO on February 12, 2018.

Target Code and Revision

For this audit, we reviewed the following repositories:

- https://github.com/FundRequest/contracts/tree/master/contracts/token/*.sol (standard MiniMe token)
- https://github.com/FundRequest/contracts/tree/master/contracts/crowdsale/FundRequestTokenGeneration.sol
- Third party vendor code is considered out of scope.

Specifically, we examined the Git revisions:

```
74b28af65b98df348041563a95934339a0d243fd
```

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

Code Review

In manually reviewing all of the contract code, we looked for any potential issues with code logic, error handling, and interaction with contracts that are dependencies. We also considered areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus was on the contract code, we examined some dependency code and behavior when it was relevant to a particular line of investigation.

Findings

The code is well organized and not too long, follows Ethereum best practices, and avoids known bugs such as re-entrancy.

Issues

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

Issue / Suggestion	Status
Issue A: `FundRequestTokenGeneration#withdraw` can be used to move/steal funds if the contract is ever intended in the future to store value	Reported: 07.02.2018
Issue B: `FundRequestTokenGeneration#onTransfer` and `FundRequestTokenGeneration#onApprove` are unimplemented and rely on implicit behavior from the EVM	Reported: 07.02.2018
Issue C: `MiniMeToken#isContract` uses extcodesize check, which can be fooled if method is called from a constructor	Reported: 07.02.2018

Issue A: `FundRequestTokenGeneration#withdraw` can be used to move/steal funds if the contract is ever intended in the future to store value

Reported: 07.02.2018

Synopsis:
https://github.com/FundRequest/contracts/blob/74b28af65b98df348041563a95934339a0d243fd/contracts/crowdsale/FundRequestTokenGeneration.sol#L183-L186

This method is implemented as a safety measure to provide recourse in the event the funds are inadvertently sent to this contract.

Impact: While this is a pretty standard procedure, it's important to take care in the future that if the contract (or another that implements this method) is ever intended to store funds, this can be used by the owner to move or steal funds.

Preconditions: In current implementation, an attacker would have to trick users into sending funds to this contract and possess ownership of the contract in order to move or steal funds. Without ownership, an attacker might attempt to only trick users into sending funds here for the sole purpose of creating a large burden on the contract owner to properly resolve the erroneous transactions.

Feasibility: Difficult to perform and with little incentive to do so.

Mitigation: None suggested for current implementation.

Remediation: None suggested for current implementation.

Verification: *Unverified.*

Issue B: `FundRequestTokenGeneration#onTransfer` and `FundRequestTokenGeneration#onApprove` are unimplemented and rely on implicit behavior from the EVM

Reported: 07.02.2018

Synopsis:
https://github.com/FundRequest/contracts/blob/74b28af65b98df348041563a95934339a0d243fd/contracts/control/TokenController.sol#L11-L27

The `TokenController` interface expects 3 methods to be implemented: `proxyPayment`, `onApprove`, and `onTransfer`. In `FundRequestTokenGeneration`, only `proxyPayment` is implemented, leaving any calls made to the controller by the `MiniMeToken` for these unimplemented methods resulting in implicit behavior.

Impact: The unimplemented function will return immediately. If any value is attached to the call, it will be transferred.

Preconditions: The methods are left unimplemented.

Feasibility: This will occur every time `onApprove` or `onTransfer` is called:

- https://github.com/FundRequest/contracts/blob/74b28af65b98df348041563a95934339a0d243fd/contracts/token/MiniMeToken.sol#L233
- https://github.com/FundRequest/contracts/blob/74b28af65b98df348041563a95934339a0d243fd/contracts/token/MiniMeToken.sol#L188

Mitigation: Best not to rely on implicit behavior of the EVM.

Remediation: Implement the unimplemented methods to be explicit about what these methods should do, even if only returning true.

Verification: *Verified via Telegram discussion on date reported.*

Issue C: `MiniMeToken#isContract` uses extcodesize check, which can be fooled if method is called from a constructor

Reported: 07.02.2018

Synopsis:
https://github.com/FundRequest/contracts/blob/74b28af65b98df348041563a95934339a0d243fd/contracts/token/MiniMeToken.sol#L480-L492

This method is implemented in order to check if the controller for the `MiniMeToken` is a contract to determine whether or not to call certain methods on the contract in question, but this check can be fooled in an edge case.

Impact: In its current implementation there is no meaningful impact. However, since the check against `extcodesize` will return `0` if the contract in question is still being constructed, the check can be tricked if it's called from a constructor.

Preconditions: `MiniMeToken#isContract` must be called from a constructor.

Feasibility: None, this method's usage is okay currently.

Mitigation: Avoid using this check from constructors.

Remediation: None suggested for current implementation.

Verification: *Unverified.*