Audit Report

Filecoin FEVM Solidity Library

v1.1

March 9, 2023

Table of Contents

License

Disclaimer

This audit has been performed by

Oak Security

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security has been engaged by Filecoin Foundation to perform a security audit of the Filecoin Solidity Library.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following GitHub repository:

https://github.com/Zondax/filecoin-solidity

Commit hash: `263c3c12794823c63221262068c9c0e4fbab70e6`

The audit focussed on the following files:

```
├── contracts
│   └── v0.8
│       ├── PrecompilesAPI.sol
│       ├── Utils.sol
│       └── utils
│           ├── Actor.sol
│           └── Misc.sol
```

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line by line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

To interact with the Filecoin EVM in Solidity, a library is required. It allows Solidity smart contracts to seamlessly call methods on Filecoin built-in actors, as well as access Filecoin-specific syscalls idiomatically.

The Filecoin Solidity library provides such an API to interact with Filecoin built-in actors.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Medium	Contracts make extensive use of assembly code.
Code readability and clarity	Medium-High	-
Level of documentation	Medium	No technical documentation is provided.
Test coverage	Medium	The codebase does not include a working test suite and precompiles are not tested.

Summary of Findings

No	Description	Severity	Status
1	`addr` with a length that is not a multiple of `0x20` may lead to incorrect results	Major	Resolved
2	Address lookup via `PrecompilesAPI.lookupAddress` fails due to incorrect memory access	Major	Resolved
3	Hardcoded `outputSize` results in data loss during a delegatecall	Major	Resolved
4	Hardcoded `GAS_LIMIT` used to call precompiles is potentially insufficient	Minor	Resolved
5	Missing sanity check for `codec` value	Minor	Resolved
6	Calling the actor precompile conceals the revert reason	Informational	Resolved
7	Inconsistent comment suggesting mandatory `READ_ONLY_FLAG`	Informational	Resolved

Detailed Findings

1. `addr` with a length that is not a multiple of `0x20` may lead to incorrect results

Severity: Major

In `contracts/v0.8/PrecompilesAPI.sol:33`, during the execution of the `resolveAddress` function, the provided `addr` length should be a multiple of `0x20`, but a user can provide any arbitrary length of bytes as input and there is a possibility that the user will receive incorrect output because the current code does not validate the length of the provided `addr` param.

Recommendation

We recommend adding the following Yul code above `contracts/v0.8/PrecompilesAPI.sol:45`:

```
if not(mod(len, 0x20)) {
    revert(0,0)
}
```

Status: Resolved

The client replaced the low-level Yul assembly code with Solidity code.

2. Address lookup via `PrecompilesAPI.lookupAddress` fails due to incorrect memory access

Severity: Major

The `PrecompilesAPI.lookupAddress` library function in `contracts/v0.8/PrecompilesAPI.sol` uses Yul to call the `lookup_address` FEVM precompile. However, the `actor_id` length is incorrectly accessed by using `mload(actor_id)` instead of using a fixed length of 32 bytes. `mload` loads a word from memory at the given address. `actor_id` is a static variable of the type `uint64`, not a memory pointer. This leads to incorrect memory access.

The same issue is present in the `PrecompilesAPI.getActorType` function in line `73`.

Recommendation

We recommend using a fixed length of `0x20` bytes as the `len` parameter.

Status: Resolved

The client replaced the low-level Yul assembly code with Solidity code.

3. Hardcoded `outputSize` results in data loss during a `delegatecall`

Severity: Major

In `contracts/v0.8/utils/Actor.sol:87`, the `MAX_RAW_RESPONSE_SIZE` parameter is used to restrict the `outputSize` of the `delegatecall` Yul instruction, while FVM actors can return data sizes greater than `0x300`. Due to the hardcoded output length, the return data is truncated, resulting in unexpected behavior of the call function.

Recommendation

We recommend using `0` as the `outputSize` and using the `returndatasize` Yul instruction to get the size of the return data.

Status: Resolved

The client replaced the low-level Yul assembly code with Solidity code.

4. Hardcoded `GAS_LIMIT` used to call precompiles may lead to gas exhaustion

Severity: Minor

The library uses a `GAS_LIMIT` constant to specify the maximum amount of gas a precompile can consume.

The actual gas consumption depends on the actual FEVM implementation, which may change over time. This makes it impossible to predict the `Actor.call` precompile execution gas cost, and hence with a hardcoded `GAS_LIMIT`, the precompile call could run out of gas.

Recommendation

We recommend using `msg.gas` (or `gas()`) instead of a hardcoded gas value.

Status: Resolved

The client replaced the low-level Yul assembly code with Solidity code.

5. Missing sanity check for `codec` value

Severity: Minor

According to the comment in `contracts/v0.8/utils/Actor.sol:49`, `codec` should always be `cbor`. However, the user can provide any codec, while the Solidity code assumes it to be `cbor`. Without validation of the `codec` parameter, there is potential for incorrect results if the FEVM returns a response that is encoded with a different codec than `cbor`.

Recommendation

We recommend adding the following code before `contracts/v0.8/utils/Actor.sol:50`:

```
if not(eq(codec, Misc.CBOR_CODEC)) {
    revert(0,0)
}
```

Status: Resolved

6. Calling the actor precompile conceals the revert reason

Severity: Informational

The `Actor` library in `contracts/v0.8/utils/Actor.sol` performs a `delegatecall` to the actor precompile. However, in the event of a revert, the revert reason is suppressed and not returned. This could negatively impact the developer and user experience.

Recommendation

We recommend retrieving the revert reason and propagating the error via `revert` (see OpenZeppelin's `Address` library for reference).

Status: Resolved

7. Inconsistent comment suggesting mandatory `READ_ONLY_FLAG`

Severity: Informational

According to the comment in `contracts/v0.8/utils/Actor.sol:47`, the mandatory `READ_ONLY_FLAG` should get passed to the call function. However, the `READ_ONLY_FLAG` was only previously required by the FEVM while it was still in early development. `DEFAULT_FLAG` is now used correctly instead, while the comment still mentions `READ_ONLY_FLAG`.

Recommendation

We recommend adapting the comment to mention the use of `DEFAULT_FLAG` rather than `READ_ONLY_FLAG`.

Status: Resolved