Neon Labs

Neon EVM

by Ackee Blockchain

Nov 4, 2022

Contents

1. Document Revisions

0.1	Draft report	Oct 31, 2022
1.0	Final report	Nov 4, 2022
1.1	Fix review	Nov 4, 2022

2. Overview

This document presents our findings in reviewed contracts.

2.1. Ackee Blockchain

Ackee Blockchain is an auditing company based in Prague, Czech Republic, specializing in audits and security assessments. Our mission is to build a stronger blockchain community by sharing knowledge – we run free certification courses School of Solana, Summer School of Solidity and teach at the Czech Technical University in Prague. Ackee Blockchain is backed by the largest VC fund focused on blockchain and DeFi in Europe, RockawayX.

2.2. Audit Methodology

The Ackee Blockchain auditing process follows a routine series of steps:

1. **Code review**
    a. High-level review of the specifications, sources, and instructions provided to us to make sure we understand the project's size, scope, and functionality.
    b. Detailed manual code review, which is the process of reading the source code line-by-line to identify potential vulnerabilities. We focus mainly on common classes of Solana program vulnerabilities, such as:

        missing ownership checks, missing signer authorization, signed CPI of unverified programs, cosplay of Solana accounts, missing rent exemption assertion, bump seed canonicalization, incorrect accounts closing, casting truncation, numerical precision errors, arithmetic overflows or underflows, …

    c. Comparison of the code and given specifications, ensuring that the program logic correctly implements everything intended.

d. Review of best practices to improve efficiency, clarity, and maintainability.

2. **Testing and automated analysis**

    a. Run client's tests to ensure that the system works as expected, potentially write missing unit or fuzzy tests using our testing framework Trdelnik.

3. **Local deployment + hacking**

    a. The programs are deployed locally, and we try to attack the system and break it. There is no specific strategy here, and each project's attack attempts are characteristic of each program audited. However, when trying to attack, we rely on the information gained from previous steps and our rich experience.

2.3. Finding classification

A *Severity* rating of each finding is determined as a synthesis of two sub-ratings: *Impact* and *Likelihood*. It ranges from *Informational* to *Critical*.

If we have found a scenario in which an issue is exploitable, it will be assigned an impact rating of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Info*.

Low to *High* impact issues also have a *Likelihood*, which measures the probability of exploitability during runtime.

The full definitions are as follows:

Severity

		Likelihood			
		High	**Medium**	**Low**	**-**
Impact	**High**	Critical	High	Medium	-
	Medium	High	Medium	Medium	-
	Low	Medium	Medium	Low	-
	Warning	-	-	-	Warning
	Info	-	-	-	Info

Table 1. Severity of findings

Impact

- **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue will result in consequences of serious substance.
- **Low** - Code that activates the issue will have outcomes on the system that are either recoverable or don’t jeopardize its regular functioning.
- **Warning** - The issue cannot be exploited given the current code and/or configuration (such as deployment scripts, compiler configuration, use of multi-signature wallets for owners, etc.), but could be a security vulnerability if these were to change slightly. If we haven’t found a way to exploit the issue given the time constraints, it might be marked as a ”Warning” or higher, based on our best estimate of whether it is currently exploitable.
- **Info** - The issue is on the borderline between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or configuration (see above) was to change.

Likelihood

- **High** - The issue is exploitable by virtually anyone under virtually any circumstance.
- **Medium** - Exploiting the issue currently requires non-trivial preconditions.
- **Low** - Exploiting the issue requires strict preconditions.

2.4. Review team

Member’s Name	Position
Vladimír Marcin	Lead Auditor
Michal Prevratil	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.5. Disclaimer

We’ve put our best effort to find all vulnerabilities in the system, however our findings shouldn’t be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

3. Executive Summary

The Neon EVM is a tool that allows Ethereum-like transactions to be processed on Solana, taking full advantage of the functionality native to Solana, including the ability to execute transactions in parallel. As such, the Neon EVM allows dApps to operate with the low gas fees, high transaction speed, and high throughput of Solana, while also offering access to the growing Solana market.

Revision 1.0

Neon Labs engaged Ackee Blockchain to conduct a security review of their Neon EVM contract with a total time donation of 33 engineering days. The review took place between 26. Sep and 4. Nov 2022.

The scope included the following repositories with a given commits:

- **Repository**: neon-evm
  - commit: `eeed4c4fd55e09d30a6a7ae4253a31bdd0bb7a35`
- **Repository**: rust-evm
  - commit: `49bd848e08502010f6d5f31aa5cea4dac65eaad7`

The beginning of the audit was dedicated to understanding the Neon EVM program. We then took a deep dive into the system.

Since it was not a classic Solana program, two auditors were dedicated to the audit, namely one Solana auditor who checked the evm_loader and one Ethereum auditor who verified the implementation of the EVM itself.

The goal was to support the client in delivering the audit, including the fix review, by the deadline of Nov 4, 2022. The draft report was sent a week before the deadline; at the same time, the work continued. During that, the client sent us fixes that we verified, so we could release the version 1.1 fix review alongside version 1.0.

During the review, we paid particular attention to:

- Is the correctness of the custom EVM ensured?
- Do the program correctly use dependencies or other programs they rely on (e.g., SPL dependencies)?
- Is the code vulnerable to any form of unintended manipulation?

Our review resulted in 2 *Medium* severity issues and another 6 findings ranging from *Informational* to *Low* severity.

AckeeBlockchain recommends Neon Labs:

- address all reported issues.

See [Revision 1.0](#) for the system overview of the codebase.

4. Summary of Findings

The following table summarizes the findings we identified during our review. Unless overridden for purposes of readability, each finding contains:

- a `Description`,
- an `Exploit scenario`,
- a `Recommendation` and if applicable
- a `Solution`.

There might often be multiple ways to solve or alleviate the issue, with varying requirements regarding the necessary changes to the codebase. In that case, we will try to enumerate them all, clarifying which solves the underlying issue better (albeit possibly only with architectural changes) than others.

	Severity	Reported	Status
M1: Selfdestruct early evaluation	Medium	1.0	Fixed
M2: The emulation of the `spl_associated_token_program` will not work	Medium	1.0	Fixed
L1: Precompiled `ecrecover` behaves incorrectly	Low	1.0	Fixed
W1: Differences in the system program emulation	Warning	1.0	Fixed
I1: Redundant account check	Info	1.0	Fixed

	Severity	Reported	Status
I2: Unnecessary owner check	Info	1.0	Fixed
I3: Unnecessary instruction argument	Info	1.0	Fixed
I4: Unnecessary holder owner validation	Info	1.0	Not Fixed

Table 2. Table of Findings

5. Report revision 1.0

5.1. System Overview

This section contains an outline of the audited programs. Note that this is meant for understandability purposes and does not replace project documentation.

Programs

Programs we find important for better understanding are described in the following section.

EVM loader

The main Neon program defines the interface between Solana and the EVM. Includes, for example, signature verification and state and storage handling. For a more extensive description see the official documentation.

Rust EVM

The actual EVM implementation. Implements all EVM opcodes. It is a customized fork of Parity's SputnikVM. For more details see documentation.

5.2. Trust model

There is a trusted set of operators chosen by Neon and they have to be trusted as there is a DoS possibility. See the Denial of Service issue for more details.

M1: Selfdestruct early evaluation

Medium severity issue

Impact:	Medium	Likelihood:	Low
Target:	state.rs	Type:	Application logic

Description

According to the specification, the `selfdestruct(addr)` instruction should:

- transfer all Ether the contract owns to the `addr` account,
- mark the contract as ”to be deleted”,
- exit the execution context.

All changes (except the Ether transfer) to the contract must be applied when finalizing the transaction, not right after the `selfdestruct` instruction.

If the mentioned specification is applied, the functions `code`, `code_size`, `nonce`, `storage`, `code_hash`, and `valids` in the `state.rs` behave incorrectly as all these functions must return the same values as if run before `selfdestruct` (in the same transaction).

Exploit scenario

Imagine the following contract:

```
contract Dummy {
    uint public state_var;

    event E();

    constructor() {
        state_var = 2;
    }
```

```
  function emits() public {
      emit E();
  }

    function kill() public {
        selfdestruct(payable(msg.sender));
    }
}

contract Test {
    function scenario1(Dummy d) public {
        d.kill();
        d.emits(); // does NOT run the `emits` function (but it should)
    }

    function scenario2(Dummy d) public returns(uint size){
        d.kill();
        assembly {
            size := extcodesize(d)
        }
        // returns zero instead of the code size
    }

    function scenario3(Dummy d) public returns(uint) {
        d.kill();
        return d.state_var(); // returns 0 instead of 2
    }

    function scenario4(Dummy d) public returns(bytes32 hash){
        d.kill();
        assembly {
            hash := extcodehash(d)
        }
        // returns keccak256 of an empty array instead of the keccak256
hash of the code
    }
}
```

Recommendation

Remove the `Action::EvmSelfDestruct { address }` match arms from the `code`, `code_size`, `nonce`, `storage`, `code_hash`, and `valids` functions.

Solution

The Neon team removed the `Action::EvmSelfDestruct { address }` match arm from all the listed functions. Ackee Blockchain verified the fix.

Go back to Findings Summary

M2: The emulation of the `spl_associated_token_program` will not work

Medium severity issue

Impact:	Low	Likelihood:	High
Target:	spl_associated_token.rs	Type:	Application logic

Description

The reason behind the wrong functionality is the condition on the first line of the emulate function:

```
if !instruction.is_empty() {
        return Err!(...);
}
```

The mistake arises from the upgrade of the `spl_associated_token_account` crate. The older version of the program had only one instruction, and the value instruction data was hard-coded as an empty vector (`data: vec![]`). However, the new version has three instructions, and the serialization of instruction data look like this: `data: instruction.try_to_vec().unwrap()`. So to serialize the instruction data, the Borsh library is used, i.e., the data vector will not be empty anymore. It means that the `emulate` function will always fail on that condition.

Recommendation

Either fix the condition to check that the instruction equals the `AssociatedTokenAccountInstruction::Create` or add support for all the instructions of the `spl_associated_token_program`.

Solution

The Neon team replaced the broken condition. The emulator now support both, an empty instruction data (older version of the `spl_associated_token_account`), and serialized version of the `AssociatedTokenAccountInstruction::Create` instruction data. Ackee Blockchain verified the fix.

Go back to Findings Summary

L1: Precompiled `ecrecover` behaves incorrectly

Low severity issue

Impact:	Low	Likelihood:	Low
Target:	ecrecover.rs	Type:	Application logic

Description

According to specification, the pre-compiled `ecrecover` contract should:

- return empty bytes (empty `Vec<u8>`) in a case of failure (currently returns `vec![0_u8; 32]`),
- accept less than 128 bytes as input and extend it up to 128 bytes with zero bytes from the right (currently fails → returns `vec![0_u8; 32]`),
- accept more than 128 bytes and ignore bytes at index 128+ (currently fails → returns `vec![0_u8; 32]`).

Recommendation

Fix the `ecrecover` contract to behave according to the specification.

Solution

The Neon team fixed the `ecrecover` precompiled contract according to the specification. Ackee Blockchain verified the fix.

Go back to Findings Summary

W1: Differences in the system program emulation

Impact:	Warning	Likelihood:	N/A
Target:	system.rs	Type:	Application logic

Description

In the `SystemInstruction::Transfer` emulator, there is no check that the source account (i.e., `from` field) contains any data. So the emulator returns a different result than the actual execution. There are no direct ways to misuse this behavior; however, the emulation should not end up with a different result than the actual execution.

Recommendation

Fix the system program emulation to behave exactly as the actual implementation.

Solution

The Neon team fixed the system program emulation by adding the condition checking the emptines of the source transfer account. Ackee Blockchain verified the fix.

Go back to Findings Summary

I1: Redundant account check

Impact:	Info	Likelihood:	N/A
Target:	sysvar.rs	Type:	Compute budget

Description

The check whether the passed-in account is the `Rent` account is redundant (loc: 29) as the function `solana_program::sysvar::rent::Rent::from_account_info` already checks it, so there is no need to do this check explicitly.

Recommendation

Remove the redundant check by which one can save some computational units.

Solution

The code no longer exists.

Go back to Findings Summary

I2: Unnecessary owner check

Impact:	Info	Likelihood:	N/A
Target:	account_create.rs	Type:	Compute budget

Description

The `EvmInstruction::CreateAccountV02` instruction checks if the passed-in code account belongs to the `evm_loader` program (loc: 51). However, this check is unnecessary as the instruction writes to this account. If someone provides a wrong account, the runtime will abort the transaction as only the owner program can modify the account's data.

Recommendation

Remove the unnecessary check by which one can save some computational units.

Solution

The code no longer exists.

Go back to Findings Summary

I3: Unnecessary instruction argument

Impact:	Info	Likelihood:	N/A
Target:	-	Type:	Compute budget

Description

In the `CreateAccountV02` instruction, there is no need to send a `bump_seed` as the `find_program_address` function calculates the bump. So the calculated bump can be used directly without needing to pass in the bump.

Recommendation

Remove the `bump_seed` instruction parameter and use the calculated bump instead. By doing so, some computational units can be saved.

Solution

The code no longer exists.

[Go back to Findings Summary](#)

I4: Unnecessary holder owner validation

Impact:	Info	Likelihood:	N/A
Target:	transaction_step_from_account.rs	Type:	Compute budget

Description

There is no need to validate the owner of a holder account in the `TransactionStepFromAccount` instruction (loc: 56). The owner will be checked in the `State::new` function so there is unnecessary double check of the ownership.

Recommendation

Remove the unnecessary check by which one can save some computational units.

Solution

The Neon team decided to not fix this issue and left the extra layer of security.

Go back to Findings Summary

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain, Neon Labs: Neon EVM, Nov 4, 2022.

Thank You

Ackee Blockchain a.s.

Prague, Czech Republic

hello@ackeeblockchain.com

https://discord.gg/z4KDUbuPxq