CERTIK

Certik Report For Nervos

Contents

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Verification Services Agreement between CertiK and Nervos Network (the "Company"), or the scope of services/verification, and terms and conditions provided to the Company in connection with the verification (collectively, the "Agreement"). This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

About CertiK

CertiK is a technology-led blockchain security company founded by Computer Science professors from Yale University and Columbia University built to prove the security and correctness of smart contracts and blockchain protocols.

CertiK's mission of every audit is to apply different approaches and detection methods, ranging from manual, static, and dynamic analysis, to ensure that the project is checked against known attacks and potential vulnerabilities. CertiK leverages a team of seasoned engineers and security auditors to apply testing methodologies and verifications on the project, in turn creating a more secure and robust software system.

CertiK has served more than 100 clients with high quality auditing and consulting services, ranging from stablecoins such as Binance's BGBP and Paxos Gold to decentralized oracles such as Band Protocol and Tellor. CertiK customizes its engineering tool kits, while applying cutting-edge research on smart contracts, for each client on its project to offer a high quality delivery. As it utilizes technologies from blockchain and smart contracts, CertiK team will continue to support the project as a service provider and collaborator.

Executive Summary

This report has been prepared for **Nervos Network** to review the implementation, security and soundness of their **Nervos Network system**. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Review the implementation and security of the **consensus** mechanism.
- Review the implementation and security of the **transaction** mechanism.
- Review the implementation and security of the **utxo** model.
- Review the **incentive** model.
- Review the **economic** model.
- Review the implementation of the **Eaglesong Pow** algorithm.

Review Summary

SECURITY LEVEL

This report has been prepared as a product of the Code Audit request by Nervos Network.
This audit was conducted to discover issues and vulnerabilities in the source code of Nervos Network.

TYPE	Network
SOURCE CODE	https://github.com/nervosnetwork
PLATFORM	Custom
LANGUAGE	Rust
REQUEST DATE	March 12.2020
DELIVERY DATE	April 7, 2020
METHODS	Dynamic Analysis, Static Analysis, and Manual Review, has been performed.

Manual Review Notes

Introduction

CertiK team has been engaged by the Nervos team to audit the design and implementations of its system. The audited source code links:

- https://github.com/nervosnetwork/ckb
- https://github.com/nervosnetwork/ckb-vm
- https://github.com/nervosnetwork/neuron/tree/develop/packages/neuron-wallet/src/models/keys
- https://github.com/nervosnetwork/ckb-system-scripts/blob/master/c/secp256k1_blake160_sighash_all.c
- https://github.com/nervosnetwork/ckb/tree/develop/pow/src

The goal of this audit is to review Nervos implementation of its core mechanisms general design and architecture, study potential security vulnerabilities, and uncover bugs that could compromise the software in production.

Documentation

We used the following sources in respect to our work:

1. Website: https://www.nervos.org/
2. Whitepaper: https://github.com/nervosnetwork/rfcs/tree/master/rfcs
3. Specs : https://github.com/nervosnetwork/rfcs/tree/master/rfcs

Summary

The results of the review and automated tools along the manual examination of the code bases provided with a number of relevant findings regarding the application reviewed. The codebase in scope was mainly in Rust language, as the project's chain, proof of work algorithm and vm, and a small part in TypeScript(JS) regarding the account layer and functionality.

Starting off with the Rust codebase and the **nervosnetwork/ckb** repository, the audit has found the code base **to be in a very high level of code design and implementation** and **findings are language related with no severity**.

Moving forward to the eaglesong proof of work algorithm, the audit has examined the codebase on the **ckb/pow** folder **and the eaglesong own crate** under **nervosnetwork/eaglesong**. Alongside with the documentation provided by the team under **rfcs/0010-eaglesong/0010-eaglesong.md** the audit has checked the implementation manually and **found it to be accurate to the specifications given**.

Finally the audit examined **the vm folder and the RISC-V implementation** of the vm engine for design and implementation correctness and found it **to be very well treated**.Due to the complexity of the mechanism the audit was not able to summarize in depth about the security of the implementation as it was not possible to be addressed within the timeline of the scope. To conclude on the vm implementation, the automated tools and manual review **did not raise any issues**.

To summarize, the audit has come to the conclusion that **the coding team has done stellar work regarding the Rust implementation** of the chain (ckb) , the proof of work(Eaglesong) and vm, using the **language best practices and implementing the designs at a very high level**.

Findings

TITLE	TYPE	SEVERITY	LOCATION
1.Integer type suffix should be separated by an underscore.	Semantic	None	File : genesis_verification.rs Line: 77
2.Unused self argument	Info	None	File: block_verifiers.rs Line: 74
3.Called map(f).unwrap_or(a)on an Option value. This can be done more directly by calling `map_or(a, f)` instead	Info	None	File: block_verifiers.rs Line: 78
4.This expression borrows a reference that is immediately dereferenced by the compiler	Info	None	File: contextual_block_verifier.rs Line: 141
5.It is more concise to loop over references to containers instead of using explicit iteration methods	Info	None	File: contextual_block_verifier.rs Line: 223
6.Casting usize to u32 may truncate the value on targets with 64-bit wide pointers	Info	None	File: contextual_block_verifier.rs Line: 428

Recommendations

Since the findings **are all language related** and are repeated in many places on the codebase, we have included a small sample to avoid repetition and keep the document more easy to read. Given the exceptional work on the low level primitives and design of the codebase **these small semantic and info issues pose no threat to the complete model** and since most of them are not included in the report(ex not use of Self as return value and so on).

Issue: 1

[Semantic]
File: genesis_verification.rs
Line: 77
Integer type suffix should be separated by an underscore.

```
R: if block.parent_hash().raw_data()[..] != [0_u8; 32][..]
                                              ^^
```

Issue: 2

[Info]
File: block_verifiers.rs
Line: 74
Unused `self` argument

```
R: pub fn verify(block: &BlockView) -> Result<(), Error>
                    ^^  removed &self
```

Issue: 3

[Info]
File: block_verifiers.rs
Line: 78
Called `map(f).unwrap_or(a)` on an Option value. This can be done more directly by calling `map_or(a, f)` instead

```
R: if !cellbase_transaction
        .outputs_data()
        .get(0)
        .map_or(true, |data| data.is_empty())
        ^ use map or here
```

Issue: 4

[Info]
File: contextual_block_verifier.rs
Line: 141
This expression borrows a reference that is immediately dereferenced by the compiler
S: self.epoch
    ^ remove & (borrow)

Issue: 5

[Info]
File: contextual_block_verifier.rs
Line: 219
It is more concise to loop over references to containers instead of using explicit iteration methods
R: for committed_id in &committed_ids.iter
                        ^ use & here to iter over references

Issue: 6

[Info]
File: contextual_block_verifier.rs
Line: 428
Casting usize to u32 may truncate the value on targets with 64-bit wide pointers

Nervos Specs Implementation Analysis

Wallet Account Model

We have reviewed the account model, the core of which is present in **`neuron/packages/neuron-wallet/src/models/keys/**/*`**, and the use of which is throughout **`neuron/packages/neuron-wallet/**/*`**.

We have examined **the related codebase in Typescript, the modules used and all related functionality** and **we haven't found any discrepancies from the specifications of BIP--0032, 0039, 0043 and 0044.**

Consensus

The CKB consensus game is a variant of the Nakamoto Consensus. It aims to increase transaction processing throughput, decrease transaction confirmation latency, and enhance security by addressing the selfish-mining strategy.

All parts of the specification were found to be present and well translated to the code. **We have found it achieves these goals without introducing any new security issues.**

Incentive Model

The Nervos incentive model is derived from its consensus. It shares some properties with the Nakamoto Consensus - namely miners are incentivized to mine blocks by issuing block rewards. It also eliminates some undesired incentives though form the Nakamoto Consensus - selfish mining and purposefully including only recent transactions in a block to increase block propagation latency - this is described as *de facto selfish mining attack* in the consensus specifications. **We have not found any game-theoretical issues in the Nervos incentive model.**

Economic Model

The economic model is a subset of the incentive model of the system. The combination of the consensus model plus the eaglesong proof of work algorithm creates a strong security environment around the economic model.

Since the consensus model addresses some key problems found in many blockchains, the addition of a custom proof of work algorithm like Eaglesong **hardens the model to an elevated security level.**

We have not found any issues arising out of the economics of the system.

Test Cases and Coverage

Considering the fact that the team **has provided evidence of stellar performance on the code design and implementation**, **the small percentage of code coverage does not reflect the overall performance of the tests.**

ckb/chain	->	21.73% coverage, 8406/38681 lines covered
ckb/ckb-bin	->	0.00% coverage, 0/37817 lines covered
ckb/db	->	0.69% coverage, 253/36476 lines covered
ckb/error	->	0.00% coverage, 0/36354 lines covered
ckb/indexer	->	14.57% coverage, 5484/37640 lines covered
ckb/miner	->	0.00% coverage, 0/36715 lines covered
ckb/network	->	8.94% coverage, 3328/37206 lines covered
ckb/notify	->	0.00% coverage, 0/36604 lines covered
ckb/pow	->	0.09% coverage, 32/36589 lines covered
ckb/protocols	->	0.00% coverage on all 3 discovery-identify-ping

ckb/resource	->	0.23% coverage, 83/36386 lines covered
ckb/rpc	->	23.05% coverage, 8691/37706 lines covered
ckb/script	->	11.90% coverage, 4594/38619 lines covered
ckb/shared	->	0.00% coverage, 0/37045 lines covered
ckb/spec	->	6.13% coverage, 2261/36865 lines covered
ckb/store	->	7.66% coverage, 2817/36773 lines covered
ckb/sync	->	?
ckb/test	->	?
ckb/traits	->	0.00% coverage, 0/36354 lines covered
ckb/tx-pool	->	5.28% coverage, 1984/37576 lines covered
ckb/utils	->	0.10% coverage, 37/36370 lines covered
ckb/verification	->	16.90% coverage, 6472/38292 lines covered

Dependencies

Spin 0.5.2 is not actively maintained.

https://github.com/mvdnes/spin-rs