



SOFTWARE AUDIT REPORT

for

NERVOS FOUNDATION



Prepared By: Shuxiao Wang

Hangzhou, China

Aug. 4, 2020

Document Properties

Client	Nervos Foundation
Title	Software Audit Report
Target	Nervos System Contracts
Version	1.0-rc2
Author	Edward Lo
Auditors	Edward Lo, Ruiyi Zhang
Reviewed by	Chiachih Wu
Approved by	Xuxian Jiang
Classification	Confidential

Version Info

Version	Date	Author	Description
1.0-rc1	May. 20, 2020	Edward Lo	Release Candidate #1
1.0-rc2	Aug. 4, 2020	Edward Lo	Release Candidate #2

Contact

For more information about this document and its contents, please contact PeckShield Inc.

Name	Shuxiao Wang
Phone	+86 173 6454 5338
Email	contact@peckshield.com

Contents

1	Introduction	4
1.1	About Nervos System Contracts	4
1.2	About PeckShield	5
1.3	Methodology	5
1.4	Disclaimer	6
2	Findings	8
2.1	Finding Summary	8
2.2	Key Findings	8
3	Detailed Results	9
3.1	Informational issue while parsing ELF - #1	9
3.2	Informational issue while parsing ELF - #2	10
4	Conclusion	11
	References	12

1 | Introduction

Given the opportunity to review the **Nervos System Contracts** design document and related source code, we in this report outline our systematic method to evaluate potential security issues in the Nervos System Contracts implementation, expose possible semantic inconsistencies between the source code and the design specification, and provide additional suggestions and recommendations for improvement. Our results show that the given branch of Nervos System Contracts can be further improved due to the presence of several issues related to either security or performance. This document describes our audit results in detail.

1.1 About Nervos System Contracts

Nervos network [1] is a public blockchain system designed by Nervos Foundation [2]. Nervos is designed as a layered blockchain network, and it separates the network infrastructure into two layers: a verification layer (layer 1) as the consensus or common trust/knowledge storage layer, and a generation or computation layer (layer 2) for high throughput transaction generations. The goal of this layered design is to scale up the blockchain performance/throughput without sacrificing its security and decentralization.

The basic information of Nervos System Contracts is shown in Table 1.1, and its Git repository and the commit hash value (of the audited branch) are in Table 1.2.

Table 1.1: Basic Information of Nervos System Contracts

Item	Description
Issuer	Nervos Foundation
Website	https://nervos.org
Type	Nervos CKB Blockchain's Contracts
Platform	C
Audit Method	White-box
Latest Audit Report	Aug. 4, 2020

Table 1.2: The Commit Hash List Of Audited Branches

Git Repository	File	Commit Hash
https://github.com/nervosnetwork/ckb-anyone-can-pay.git	anyone_can_pay.c	deac680
https://github.com/nervosnetwork/ckb-miscellaneous-scripts.git	simple_udt.c	3b2bf65
https://github.com/nervosnetwork/ckb-miscellaneous-scripts.git	secp256k1_blake2b_sighash_all_dual.c	0759a65
https://github.com/nervosnetwork/ckb-miscellaneous-scripts.git	and.c	0759a65
https://github.com/nervosnetwork/ckb-miscellaneous-scripts.git	or.c	0759a65
https://github.com/nervosnetwork/ckb-c-stdlib.git	ckb_dlfcn.h	eae8c4c

1.2 About PeckShield

PeckShield Inc. [3] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystems by offering top-notch, industry-leading services and products including security audits. We are reachable at Telegram (<https://t.me/peckshield>), Twitter (<http://twitter.com/peckshield>), or Email (contact@peckshield.com).

1.3 Methodology

Table 1.3: Vulnerability Severity Classification

Impact	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low
		Likelihood		

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [4]:

- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact measures the technical loss and business damage of a successful attack;

- Severity demonstrates the overall criticality of the risk.

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact and can be classified into four categories accordingly, i.e., *Critical*, *High*, *Medium*, and *Low* shown in Table 1.3.

To better describe each issue we identified, we also categorize the findings based on Common Weakness Enumeration (CWE-699) [5], which is a community-developed list of software weakness types to better classify and organize weaknesses around concepts frequently encountered in software development. We use the CWE categories in Table 1.4 to classify our findings.

1.4 Disclaimer

Note that this audit does not give any warranties on finding all possible security issues of the given blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of blockchain software. Last but not least, this security audit should not be used as an investment advice.




Table 1.4: Common Weakness Enumeration (CWE) Classifications Used in This Audit

Category	Summary
Configuration	Weaknesses in this category are typically introduced during the configuration of the software.
Data Processing Issues	Weaknesses in this category are typically found in functionality that processes data.
Numeric Errors	Weaknesses in this category are related to improper calculation or conversion of numbers.
Security Features	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software)
Time and State	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.
Error Conditions, Return Values, Status Codes	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function.
Resource Management	Weaknesses in this category are related to improper management of system resources.
Behavioral Issues	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
Business Logic	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.
Initialization and Cleanup	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
Arguments and Parameters	Weaknesses in this category are related to improper use of arguments or parameters within function calls.
Expression Issues	Weaknesses in this category are related to incorrectly written expressions within code.
Coding Practices	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an exploitable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained.

2 | Findings

2.1 Finding Summary

Here is a summary of our findings after analyzing the Nervos System Contracts. During the first phase of our audit, we studied the contract source code and ran our in-house static code analyzer through the codebase. We further manually review the business logic, and place aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	# of Findings	
Critical	0	
High	0	
Medium	0	
Low	0	
Informational	2	
Total	2	

We have so far identified a list of potential issues: some of them involve subtle corner cases that might not be previously thought of, while others refer to unusual interactions among multiple modules.

2.2 Key Findings

Overall, the Nervos System Contracts are well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 2 informational recommendations.

Table 2.1: Key Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Informational	Informational issue while parsing ELF - 1	Coding Practices	Fixed
PVE-002	Informational	Informational issue while parsing ELF - 2	Coding Practices	Fixed

3 | Detailed Results

3.1 Informational issue while parsing ELF - #1

- ID: PVE-001
- Severity: Informational
- Likelihood: High
- Impact: Undetermined
- Target: ckb-c-stdlib/ckb_dlfcn.h
- Category: Coding Practices [6]
- CWE subcategory: CWE-190 [7]

Description

There is an informational issue in the ELF loading process of OR/AND lock scripts, which might lead to undefined behavior or cause some damages while used in a combination with other scripts.

OR/AND are simple composable lock scripts. They iterate multiple lock scripts in sequence. While finishing the iteration, OR returns success as long as any lock script passes, otherwise it returns fail. On the other hand, AND returns success only if all lock scripts pass.

```

140     for (int i = 0; i < header.e_phnum; i++) {
141         const Elf64_Phdr *ph = &program_headers[i];
142         if (ph->p_type == PT_LOAD && ph->p_memsz > 0) {
143             if ((ph->p_flags & PF_X) != 0) {
144                 uint64_t prepad = ph->p_vaddr % RISCVP_GSIZE;
145                 uint64_t vaddr = ph->p_vaddr - prepad;
146                 uint64_t memsz = ROUNDUP(prepad + ph->p_memsz, RISCVP_GSIZE);
147                 if (vaddr + memsz > aligned_size) {
148                     return ERROR_MEMORY_NOT_ENOUGH;
149                 }
150                 ret = ckb_load_cell_code(aligned_addr + vaddr, memsz, ph->p_offset,
151                                         ph->p_filesz, index, CKB_SOURCE_CELL_DEP);
152                 if (ret != CKB_SUCCESS) {
153                     return ret;
154                 }

```

Listing 3.1: ckb-c-stdlib/ckb_dlfcn.h

In the current implementation, we identified that the sanity check `vaddr + memsz > aligned_size` (line 147) can be bypassed by overflowing the summation. By providing a very large `vaddr`, the content will then be loaded into an unanticipated (overflowed) address `aligned_addr + vaddr` (line 150). It may not cause immediate damages in a simple straightforward usage such as loading single instance, however, could be risky when combined with other scripts.

Recommendation Add necessary sanity checks in the `ckb_dlopen` handler.

3.2 Informational issue while parsing ELF - #2

- ID: PVE-002
- Severity: Informational
- Likelihood: High
- Impact: Undetermined
- Target: `ckb-c-stdlib/ckb_dlfcn.h`
- Category: Coding Practices [6]
- CWE subcategory: CWE-190 [7]

Description

Here's another informational issue in the ELF loading process of OR/AND lock scripts, which might lead to undefined behavior or cause some damages while used in a combination with other scripts.

```

157     uint64_t filesz = ph->p_filesz;
158     uint64_t consumed_end = ROUNDUP(ph->p_vaddr + filesz, RISCVPGSIZE);
159     if (consumed_end > aligned_size) {
160         return ERROR_MEMORY_NOT_ENOUGH;
161     }
162     ret = ckb_load_cell_data(aligned_addr + ph->p_vaddr, &filesz,
163                             ph->p_offset, index, CKB_SOURCE_CELL_DEP);
164     if (ret != CKB_SUCCESS) {
165         return ret;
166     }

```

Listing 3.2: `ckb-c-stdlib/ckb_dlfcn.h`

In the current implementation, we identified that the sanity check `consumed_end > aligned_size` (line 159) can be bypassed by overflowing the summation of `ROUNDUP(ph->p_vaddr + filesz, RISCVPGSIZE)`. By providing a very large `ph->p_vaddr`, the content will then be loaded into an unanticipated (overflowed) address `aligned_addr + ph->p_vaddr` (line 162). It may not cause immediate damages in a simple straightforward usage such as loading single instance, however, could be risky when combined with other scripts.

Recommendation Add necessary sanity checks in the `ckb_dlopen` handler.

4 | Conclusion

Our impression through this audit is that the Nervos System Contracts are neatly organized and elegantly implemented and those identified issues are promptly confirmed and fixed. We'd like to commend Nervos Foundation for a well-done software project, and for quickly fixing issues found during the audit process. Also, as expressed in Section [1.4](#), we appreciate any constructive feedback or suggestions about this report.



References

- [1] Nervos Foundation. Nervos Network. <https://github.com/nervosnetwork/>.
- [2] Nervos Foundation. Nervos Foundation. <https://nervos.org>.
- [3] PeckShield. PeckShield Inc. <https://www.peckshield.com>.
- [4] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.
- [5] MITRE. CWE VIEW: Development Concepts. <https://cwe.mitre.org/data/definitions/699.html>.
- [6] MITRE. CWE CATEGORY: Bad Coding Practices. <https://cwe.mitre.org/data/definitions/1006.html>.
- [7] MITRE. CWE-190: Integer Overflow or Wraparound. <https://cwe.mitre.org/data/definitions/190.html>.