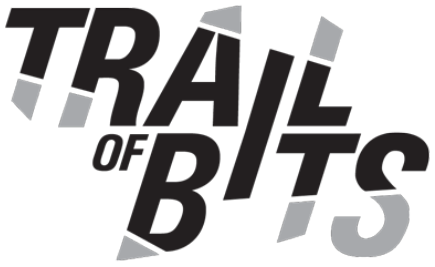

Nervos

Security Assessment

November 2, 2020

Prepared For:
Ben Morris | *Decrypted Sapiens*
ben.morris@nervos.org

Kevin Wang | *Decrypted Sapiens*
kevin@nervos.org

Prepared By:
Artur Cygan | *Trail of Bits*
artur.cygan@trailofbits.com

Brad Larsen | *Trail of Bits*
brad.larsen@trailofbits.com

Will Song | *Trail of Bits*
will.song@trailofbits.com

Executive Summary

Project Dashboard

Engagement Goals

Coverage and Methods

Recommendations Summary
- Short term
- Long term

Findings Summary
- 1. Docker-based contract build process depends on moleculec in PATH
- 2. Use of an outdated ckb-c-stdlib dependency
- 3. GCC versions 9.2 through 10.2 miscompile certain memcmp calls
- 4. Implementation of sbrk does not set errno upon failure
- 5. Uninitialized variables are read
- 6. CKB-only cells invoke undefined behavior
- 7. Duplicated logic in the anyone-can-pay lock contract
- 8. The mbedtls library is built in non-production mode
- 9. nervosnetwork/riscv-newlib is severely outdated

Appendix A. Vulnerability Classifications

Appendix B. Code Quality Recommendations

Executive Summary

From October 13 to October 30, 2020, Decrypted Sapiens engaged Trail of Bits to review the security of the Nervos Simple User-Defined Token (SUDT) and the anyone-can-pay lock contracts. Trail of Bits conducted this assessment over six person-weeks, with three engineers working from commit `deac6801` of the `ckb-anyone-can-pay` repository and commit `175b8b09` of the `ckb-miscellaneous-scripts` repository.

Trail of Bits identified nine findings; these include two medium-severity issues stemming from the use of uninitialized memory, which could invoke undefined behavior. The other findings are of low or informational severity and mostly pertain to weaknesses in the dependencies of the contracts.

The contracts and related Nervos components under review appear to have been carefully engineered and constructed but lack thorough test suites.

We recommend that Decrypted Sapiens take the following steps to improve the security posture of the Nervos project:

- Expand the automated testing of the contracts.
- Create a simulator for contract development.
- Keep third-party dependencies up to date.

Project Dashboard

Application Summary

Name	Nervos
Versions	ckb-anyone-can-pay: deac6801
	ckb-miscellaneous-scripts: 175b8b09
Type	Blockchain contracts
Platform	RISC-V

Engagement Summary

Dates	October 13 – 30, 2020
Method	Full knowledge
Consultants Engaged	3
Level of Effort	6 person-weeks

Vulnerability Summary

Total Medium-Severity Issues	2	■ ■
Total Low-Severity Issues	4	■ ■ ■ ■
Total Informational-Severity Issues	2	■ ■
Total Undetermined-Severity Issues	1	■
Total	9	

Category Breakdown

Patching	4	■ ■ ■ ■
Undefined Behavior	2	■ ■
Error Reporting	1	■
Configuration	1	■
Cryptography	1	■
Total	9	

Engagement Goals

The engagement was scoped to provide a security assessment of two low-level contracts developed for the Common Knowledge Base (CKB) blockchain, written in C, and compiled for `riscv64`.

Specifically, we sought to answer the following questions:

- Is it possible for a user to execute fraudulent transactions?
- Is it possible for an unauthorized participant to unlock a cell?
- Are there known bugs in any of the dependencies?
- Does the code avoid the common pitfalls of the C language?

Coverage and Methods

We reviewed the contracts manually, assisted by static analysis tools including `clang-tidy`, `cppcheck`, and CodeQL.

Recommendations Summary

This section aggregates all the recommendations made during the engagement. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals.

Short Term

❑ **Update the build documentation for the `ckb-miscellaneous-scripts` and `ckb-anyone-can-pay` repositories to note the requirement on the `moleculec` compiler and to explain how to obtain it.** TOB-NERVOS-001

❑ **Update both smart contracts to use the most recent version of the `ckb-c-stdlib` dependency.** TOB-NERVOS-002

❑ **Refrain from updating the `ckb-riscv-gnu-toolchain` repository to use a newer upstream version of GCC until the code generation bug has been resolved there.** Monitor the GCC bug report to ensure you are aware of which release resolves the issue. TOB-NERVOS-003

❑ **Ensure that the `_sbrk` function sets `errno` upon a failure (probably to `ENOMEM`).** TOB-NERVOS-004

❑ **Modify the anyone-can-pay lock contract so that all data is initialized before it is read.** TOB-NERVOS-005

❑ **Modify the UDT logic to depend on `is_ckb_only` and to execute only when `is_ckb_only` is false.** This will guarantee that UDT amounts are initialized and will prevent the invocation of undefined behavior. TOB-NERVOS-006

❑ **Extract the duplicated code into a `load_type_hash_and_amounts` function and turn the differing variables into the function call arguments.** The `InputWallet` structure can be generalized to also hold the output cell's data, as the input and output cells share all fields except the `output_cnt` field. This will further simplify the code. TOB-NERVOS-007

❑ **Turn off the `MBEDTLS_TEST_NULL_ENTROPY` option.** If its use is intentional and necessary, document that fact extensively. TOB-NERVOS-008

❑ **Incorporate the upstream changes into `nervosnetwork/riscv-newlib` and update the users of this dependency.** This will address the known bugs. TOB-NERVOS-009

Long Term

❑ **Incorporate the generation of serialization-related dependencies into the Docker-based build process via the `all-via-docker` `make` target.** TOB-NERVOS-001

❑ **Consider adopting development and release processes in which downstream users of `ckb-c-stdlib` are automatically updated and tested when a new release of `ckb-c-stdlib` is created.** TOB-NERVOS-002

❑ **Consider using GCC's `-fno-builtin-memcmp` option, which appears to prevent the code generation bug from manifesting itself even in affected versions of GCC.** TOB-NERVOS-003

❑ **Consider using instrumentation tools such as `AddressSanitizer`, `UndefinedBehaviorSanitizer`, or `Valgrind` in the routine testing of this smart contract.** These tools are effective in identifying memory errors and reliance on undefined behavior. TOB-NERVOS-005, TOB-NERVOS-006

❑ **Focus on identifying duplicated code during code reviews.** This will help prevent the use of repeated logic, which is a liability for development and code maintenance. TOB-NERVOS-007

❑ **Always use the best possible security options for all components.** TOB-NERVOS-008

❑ **Adopt a development and release process that involves periodically updating forked dependencies from their upstream sources.** TOB-NERVOS-009

Findings Summary

#	Title	Type	Severity
1	Docker-based contract build process depends on moleculec in PATH	Configuration	Low
2	Use of an outdated ckb-c-stdlib dependency	Patching	Low
3	GCC versions 9.2 through 10.2 miscompile certain memcmp calls	Patching	Informational
4	Implementation of sbrk does not set errno upon failure	Error Reporting	Low
5	Uninitialized variables are read	Undefined Behavior	Medium
6	CKB-only cells invoke undefined behavior	Undefined Behavior	Medium
7	Duplicated logic in the anyone-can-pay lock contract	Patching	Informational
8	The mbedtls library is built in non-production mode	Cryptography	Undetermined
9	nervosnetwork/riscv-newlib is severely outdated	Patching	Low

1. Docker-based contract build process depends on moleculec in PATH

Severity: Low | Difficulty: High
Type: Configuration | Finding ID: TOB-NERVOS-001
Target: ckb-miscellaneous-scripts/Makefile, ckb-anyone-can-pay/Makefile

Description
The contracts in the `ckb-miscellaneous-scripts` repository and the `ckb-anyone-can-pay` repository try to provide reproducible builds, but each depends on the binary `moleculec`, which is not managed in the build process. This could prevent builds from being reproduced.

The `ckb-miscellaneous-scripts` and `ckb-anyone-can-pay` repositories use a top-level `Makefile` to automate their build processes via the `all-through-docker` target. This target has a handful of serialization-related dependencies produced by the `moleculec` compiler, such as the `build/or.h` header file. Because of the dependencies' structure, the commands to produce these files are executed outside of the Docker build. If the expected `moleculec` binary is not found in the user's `PATH`, the dependencies will be produced incorrectly, and the entire build will fail.

Note that the latest version of each repository uses an updated version of the `ckb-c-stdlib` dependency, which resolves this issue by eliminating the need to use `moleculec` in the build process.

Exploit Scenario
An attacker injects a malicious `moleculec` binary into a user's build environment and inserts a backdoor into a compiled contract.

Recommendations
Short term, update the build documentation for the `ckb-miscellaneous-scripts` and `ckb-anyone-can-pay` repositories to note the requirement on the `moleculec` compiler and to explain how to obtain it.

Long term, incorporate the generation of serialization-related dependencies into the Docker-based build process via the `all-via-docker` `make` target.

2. Use of an outdated ckb-c-stdlib dependency

Severity: Low | Difficulty: High
Type: Patching | Finding ID: TOB-NERVOS-002
Target: `ckb-miscellaneous-scripts`, `ckb-anyone-can-pay`

Description
The contracts in the `ckb-miscellaneous-scripts` repository and the `ckb-anyone-can-pay` repository use out-of-date versions of the `ckb-c-stdlib` dependency. The `ckb-miscellaneous-scripts` repository uses the `ckb-c-stdlib` commit eae8c4c9, from May 26, 2020, and `ckb-anyone-can-pay` uses commit `c056b00f`, from April 13, 2020.

During their initial development, the `ckb-miscellaneous-scripts` and `ckb-anyone-can-pay` repositories both used the latest version of the `ckb-c-stdlib` dependency. Since that time, there have been no changes made to `ckb-c-stdlib` that affect the binary code generated for the two contracts: the reproducible build process produces identical binaries when using the updated `ckb-c-stdlib` version.

This issue has been resolved in the current versions of the `ckb-miscellaneous-scripts` and `ckb-anyone-can-pay` repositories, as both contracts use the latest version of `ckb-c-stdlib`.

Exploit Scenario
An attacker observes that a contract uses the out-of-date `ckb-c-stdlib` dependency and exploits a bug in its implementation to cause a denial of service or to steal funds.

Recommendations
Short term, update both smart contracts to use the most recent version of the `ckb-c-stdlib` dependency.

Long term, consider adopting development and release processes in which downstream users of `ckb-c-stdlib` are automatically updated and tested when a new release of `ckb-c-stdlib` is created.

3. GCC versions 9.2 through 10.2 miscompile certain memcmp calls

Severity: Informational | Difficulty: High
Type: Patching | Finding ID: TOB-NERVOS-003
Target: [ckb-miscellaneous-scripts](), [ckb-anyone-can-pay](), [ckb-riscv-gnu-toolchain]()

Description
Decrypted Sapiens asked us to evaluate the impact of an unresolved code generation bug in GCC (#95189) on the Nervos smart contracts. This bug does not currently appear to negatively affect the project but should be addressed when an updated GCC with a fix for it is released.

The GCC code generation bug causes certain calls to `memcmp` to be miscompiled at `-O2` and higher optimization levels when using GCC versions 9.2 through 10.2 (inclusive). This bug is evident only when one of the buffers provided to `memcmp` is a compile-time constant with embedded `0` or `NUL` values. This code pattern is unusual, which is likely the main reason that the bug was not reported for many months.

The Nervos smart contracts reviewed during this audit are built with a forked GCC 8.3 toolchain that does not appear to be affected by the code generation bug.

The bug appears to have been fixed on the GCC trunk, but the fix has not yet been included in an official numbered release.

Exploit Scenario
A Nervos smart contract is built with an affected version of GCC and includes the unusual `memcmp` pattern required for the manifestation of the code generation bug. An attacker exploits the resulting miscompilation to bypass script validation and to ultimately steal users' funds.

Recommendations
Short term, refrain from updating the `ckb-riscv-gnu-toolchain` repository to use a newer upstream version of GCC until the code generation bug has been resolved there. Monitor the GCC bug report to ensure you are aware of which release resolves the issue.

Long term, consider using GCC's `-fno-builtin-memcmp` option, which appears to prevent the code generation bug from manifesting itself even in affected versions of GCC.

References
- The GCC bug report: https://gcc.gnu.org/bugzilla/show_bug.cgi?id=95189
- A related `secp256k1` issue: https://github.com/bitcoin-core/secp256k1/issues/823

4. Implementation of sbrk does not set errno upon failure

Severity: Low
Type: Error Reporting
Target: riskv-newlib/libgloss/riskv/sys_sbrk.c

Difficulty: High
Finding ID: TOB-NERVOS-004

Description
The forked version of `riskv-newlib` used by Nervos includes support for the `sbrk` function on systems that lack an MMU. However, if an `sbrk` call fails, the implementation does not set `errno` as expected. The details of the implementation appear in the `_sbrk` function, shown in figure 4.1.

```
#include <sys/types.h>
/*
* For simplicity, we are allocating 1MB - 3MB memory range as the
* memory space for brk. This preserves 1MB for code size, and 1MB for
* stack size. Notice the parameters here are tunable, so a script with
* special requirements can still adjust the values as they want.
* Since ckb-vm doesn't have MMU, we don't need to make real syscalls.
*/
#ifndef CKB_BRK_MIN
#define CKB_BRK_MIN 0x00100000
#endif  /* CKB_BRK_MIN */
#ifndef CKB_BRK_MAX
#define CKB_BRK_MAX 0x00300000
#endif  /* CKB_BRK_MAX */
void*
_sbrk(ptrdiff_t incr)
{
 static uintptr_t p = CKB_BRK_MIN;
 uintptr_t start = p;
 p += incr;
 if (p > CKB_BRK_MAX) {
   return (void *) (-1);  /* NOTE: errno is not set in this failure case */
 }
 return start;
}
```

Figure 4.1: The implementation of the `_sbrk` function when `CKB_NO_MMU` is defined

The implementation's failure to set `errno` does not appear to have a runtime effect on either the anyone-can-pay lock contract or the SUDT contract. However, it could be an issue for other third-party contracts.

Exploit Scenario
An attacker provides input to a CKB smart contract that causes the contract to behave unexpectedly after a failed call to `sbrk`.

Recommendations
Short term, ensure that the `_sbrk` function sets `errno` upon a failure (probably to `ENOMEM`).

5. Uninitialized variables are read

Severity: Medium
Type: Undefined Behavior
Target: ckb-anyone-can-pay/c/anyone_can_pay.c

Difficulty: High
Finding ID: TOB-NERVOS-005

Description
In the anyone-can-pay lock contract, the data used to represent input wallets is not fully initialized before it is used. In C, this invokes undefined behavior that cannot reliably be reasoned about. The program could behave unexpectedly.

```
typedef struct {
 int is_ckb_only;
 unsigned char type_hash[BLAKE2B_BLOCK_SIZE];
 uint64_t ckb_amount;
 uint128_t udt_amount;
 uint32_t output_cnt;   /* NOTE: field is not initialized before use */
} InputWallet;

int check_payment_unlock(uint64_t min_ckb_amount, uint128_t min_udt_amount) {
 unsigned char lock_hash[BLAKE2B_BLOCK_SIZE];
 InputWallet input_wallets[MAX_TYPE_HASH];    /* NOTE: stack-allocated array */

 /* iterate outputs wallet cell */
 i = 0;
 while (1) {
   /* ... */
   for (int j = 0; j < input_wallets_cnt; j++) {
     /* ... */
     /* increase counter */
     found_inputs++;
     input_wallets[j].output_cnt += 1;    /* NOTE: used uninitialized here on line 223 */
     /* ... */
   }
   /* ... */
 }
/* ... */
}
```

Figure 5.1: The `output_cnt` struct member is used uninitialized in `anyone_can_pay.c`.

Exploit Scenario
An attacker crafts a transaction input that uses uninitialized memory, causing the anyone-can-pay lock contract to behave erratically. The contract's validation logic fails on lines 242 – 249, ultimately causing a denial of service.

Recommendations
Short term, modify the anyone-can-pay lock contract so that all data is initialized before it is read.

Long term, consider using instrumentation tools such as `AddressSanitizer`, `UndefinedBehaviorSanitizer`, or `Valgrind` in the routine testing of this smart contract. These tools are effective in identifying memory errors and reliance on undefined behavior.

Additionally, develop a simulator for CKB contracts that can run on mainstream platforms such as x64 Linux, and use it extensively in testing. This will enable the use of a wide variety of development tools that may not yet be well supported on `riscv64`, including debuggers, fuzzers, static analyzers, and error-checking instrumentation.

References

- `UndefinedBehaviorSanitizer`: https://clang.llvm.org/docs/UndefinedBehaviorSanitizer.html
- `AddressSanitizer`: https://clang.llvm.org/docs/AddressSanitizer.html
- `Valgrind`: https://www.valgrind.org/

6. CKB-only cells invoke undefined behavior

Severity: Medium
Type: Undefined Behavior
Target: [ckb-anyone-can-pay/c/anyone_can_pay.c]

Difficulty: High
Finding ID: TOB-NERVOS-006

Description
The anyone-can-pay lock contract uses uninitialized variables when cells hold only CKB. The `input_wallets[j].udt_amount` and `udt_amount` variables are passed to a `ckb_load_cell_data` syscall, but memory is initialized solely for CKB-only cells. The logic that operates on UDT amounts is subsequently executed regardless of whether a cell holds only CKB, invoking undefined behavior when uninitialized memory is read (figure 6.1).

```
int check_payment_unlock(uint64_t min_ckb_amount, uint128_t min_udt_amount) {
  /* ... */
  InputWallet input_wallets[MAX_TYPE_HASH];
  /* ... */
  while (1) {
    /* ... */
    len = UDT_LEN;
    ret = ckb_load_cell_data((uint8_t *)&input_wallets[i].udt_amount, &len, 0,
                             i, CKB_SOURCE_GROUP_INPUT);
    if (ret != CKB_ITEM_MISSING && ret != CKB_SUCCESS) {
      return ERROR_SYSCALL;
    }

    if (input_wallets[i].is_ckb_only) {
      /* ckb only wallet should has no data */
      if (len != 0) {
        return ERROR_ENCODING;
      }
    } else {
      if (len < UDT_LEN) {
        return ERROR_ENCODING;
      }
    }

    i++;
  }

  int input_wallets_cnt = i;

  /* iterate outputs wallet cell */
  i = 0;
  while (1) {
    /* ... */
    uint128_t udt_amount;
    /* ... */
    len = UDT_LEN;
    ret = ckb_load_cell_data((uint8_t *)&udt_amount, &len, 0, i,
                             CKB_SOURCE_OUTPUT);
    if (ret != CKB_ITEM_MISSING && ret != CKB_SUCCESS) {
      return ERROR_SYSCALL;
    }
```

```
    if (is_ckb_only) {
      /* ckb only wallet should has no data */
      if (len != 0) {
        return ERROR_ENCODING;
      }
    } else {
      if (len < UDT_LEN) {
        return ERROR_ENCODING;
      }
    }

    /* find input wallet which has same type hash */
    int found_inputs = 0;
    for (int j = 0; j < input_wallets_cnt; j++) {
      /* ... */
      overflow = uint128_overflow_add(
          &min_output_udt_amount, input_wallets[j].udt_amount, min_udt_amount);
      int meet_udt_cond = !overflow && udt_amount >= min_output_udt_amount;

      /* fail if can't meet both conditions */
      if (!(meet_ckb_cond || meet_udt_cond)) {
        return ERROR_OUTPUT_AMOUNT_NOT_ENOUGH;
      }
      /* output coins must meet condition, or remain the old amount */
      if ((!meet_ckb_cond && ckb_amount != input_wallets[j].ckb_amount) ||
          (!meet_udt_cond && udt_amount != input_wallets[j].udt_amount)) {

        return ERROR_OUTPUT_AMOUNT_NOT_ENOUGH;
      }
      /* ... */
    }
    /* ... */
  }
  /* ... */
}
```

Figure 6.1: [c/anyone_can_pay.c, lines 40 – 252]

Exploit Scenario
Because of undefined behavior on a node, the anyone-can-pay lock contract's execution fails on that node but succeeds on another, causing consensus issues and a network fork.

Recommendations
Short term, modify the UDT logic to depend on `is_ckb_only` and to execute only when `is_ckb_only` is false. This will guarantee that UDT amounts are initialized and will prevent the invocation of undefined behavior.

Long term, consider using instrumentation tools such as `AddressSanitizer`, `UndefinedBehaviorSanitizer`, or `Valgrind` in the routine testing of this smart contract. These tools are effective in identifying memory errors and reliance on undefined behavior.

7. Duplicated logic in the anyone-can-pay lock contract

Severity: Informational
Type: Patching
Target: ckb-anyone-can-pay/c/anyone_can_pay.c

Difficulty: High
Finding ID: TOB-NERVOS-007

Description
The `check_payment_unlock` function in the anyone-can-pay lock contract has a significant amount of duplicated logic. The same logic is used in two places (an input cell and an output cell) and loads the type hash, CKB amount, and UDT amount. Figure 7.1 shows the input cell's code, with the portions of it that differ from the output cell's code highlighted in yellow. See c/anyone_can_pay.c, lines 139 –183, for the corresponding portion of the output cell's code.

```
ret = ckb_checked_load_cell_by_field(input_wallets[i].type_hash, &len, 0, i,
                                     CKB_SOURCE_GROUP_INPUT,
                                     CKB_CELL_FIELD_TYPE_HASH);
if (ret == CKB_INDEX_OUT_OF_BOUND) {
  break;
}

if (ret == CKB_SUCCESS) {
  if (len != BLAKE2B_BLOCK_SIZE) {
    return ERROR_ENCODING;
  }
} else if (ret != CKB_ITEM_MISSING) {
  return ERROR_SYSCALL;
}

input_wallets[i].is_ckb_only = ret == CKB_ITEM_MISSING;

/* load amount */
len = CKB_LEN;
ret = ckb_checked_load_cell_by_field(
    (uint8_t *)&input_wallets[i].ckb_amount, &len, 0, i,
    CKB_SOURCE_GROUP_INPUT, CKB_CELL_FIELD_CAPACITY);
if (ret != CKB_SUCCESS) {
  return ERROR_SYSCALL;
}
if (len != CKB_LEN) {
  return ERROR_ENCODING;
}
len = UDT_LEN;
ret = ckb_load_cell_data((uint8_t *)&input_wallets[i].udt_amount, &len, 0,
                         i, CKB_SOURCE_GROUP_INPUT);
if (ret != CKB_ITEM_MISSING && ret != CKB_SUCCESS) {
  return ERROR_SYSCALL;
}

if (input_wallets[i].is_ckb_only) {
  /* ckb only wallet should has no data */
  if (len != 0) {
    return ERROR_ENCODING;
  }
} else {
```

```
  if (len < UDT_LEN) {
    return ERROR_ENCODING;
  }
}
```

Figure 7.1: c/anyone_can_pay.c, lines 61 – 105

Other code is duplicated in the `has_signature` and `verify_secp256k1_blake160_sighash_all` functions. The `has_signature` logic, including two syscalls, is almost entirely repeated in `verify_secp256k1_blake160_sighash_all`.

Exploit Scenario
A vulnerability is discovered in the duplicated code. A developer patches the code in one place but not the others, leaving the program vulnerable.

Recommendations
Short term, extract the duplicated code into a `load_type_hash_and_amounts` function and turn the differing variables into the function call arguments. The `InputWallet` structure can be generalized to also hold the output cell's data, as the input and output cells share all fields except the `output_cnt` field. This will further simplify the code.

Long term, focus on identifying duplicated code during code reviews. This will help prevent the use of repeated logic, which is a liability for development and code maintenance.

8. The mbedtls library is built in non-production mode

Severity: Undetermined
Type: Cryptography
Target: Contracts depending on `mbedtls`

Difficulty: High
Finding ID: TOB-NERVOS-008

Description
The `mbedtls` library is explicitly configured to be built without entropy sources. As indicated in the documentation (figure 8.1), `MBEDTLS_TEST_NULL_ENTROPY` is suitable only for development, and enabling the switch negates any security provided by the library. The build log indicates that the option is enabled during the build process and warns that the build is not suitable for production use (figure 8.2).

The `mbedtls` library is not used by the contracts reviewed during this assessment.

```
/**
 * \def MBEDTLS_TEST_NULL_ENTROPY
 *
 * Enables testing and use of mbed TLS without any configured entropy sources.
 * This permits use of the library on platforms before an entropy source has
 * been integrated (see for example the MBEDTLS_ENTROPY_HARDWARE_ALT or the
 * MBEDTLS_ENTROPY_NV_SEED switches).
 *
 * WARNING! This switch MUST be disabled in production builds, and is suitable
 * only for development.
 * Enabling the switch negates any security provided by the library.
 *
 * Requires MBEDTLS_ENTROPY_C, MBEDTLS_NO_DEFAULT_ENTROPY_SOURCES
 *
 */
#define MBEDTLS_TEST_NULL_ENTROPY
```

Figure 8.1: deps/mbedtls-config-template.h, lines 530 – 545

```
$ make all-via-docker
...
make -C deps/mbedtls/library CC=riscv64-unknown-linux-gnu-gcc
LD=riscv64-unknown-linux-gnu-gcc CFLAGS="-Os -fPIC -nostdinc -nostdlib
-DCKB_DECLARATION_ONLY -I ../../ckb-c-stdlib/libc -fdata-sections -ffunction-sections"
libmbedcrypto.a
make[1]: Entering directory '/code/deps/mbedtls/library'
...
  CC    entropy.c
entropy.c:31:2: warning: #warning "**** WARNING!  MBEDTLS_TEST_NULL_ENTROPY defined! "
[-Wcpp]
   31 | #warning "**** WARNING!  MBEDTLS_TEST_NULL_ENTROPY defined! "
      |  ^~~~~~~
entropy.c:32:2: warning: #warning "**** THIS BUILD HAS NO DEFINED ENTROPY SOURCES " [-Wcpp]
   32 | #warning "**** THIS BUILD HAS NO DEFINED ENTROPY SOURCES "
      |  ^~~~~~~
entropy.c:33:2: warning: #warning "**** THIS BUILD IS *NOT* SUITABLE FOR PRODUCTION USE "
[-Wcpp]
   33 | #warning "**** THIS BUILD IS *NOT* SUITABLE FOR PRODUCTION USE "
```

```
      |   ^~~~~~
...
```

Figure 8.2: Warnings that appear when building `mbedtls`

Exploit Scenario
The `mbedtls` library is used in a security-sensitive context, but the security guarantees of the library are negated by the use of the enabled `MBEDTLS_TEST_NULL_ENTROPY` option.

Recommendations
Short term, turn off the `MBEDTLS_TEST_NULL_ENTROPY` option. If its use is intentional and necessary, document that fact extensively.

Long term, always use the best possible security options for all components.

9. nervosnetwork/riscv-newlib is severely outdated

Severity: Low | Difficulty: High
Type: Patching | Finding ID: TOB-NERVOS-009
Target: ckb-riscv-newlib, ckb-miscellaneous-scripts, ckb-anyone-can-pay

Description
The nervosnetwork/riscv-newlib repository contains a forked version of the riscv-newlib library. The forked repository lacks more than 700 recent commits and is missing several bug fixes, including for `riscv`-specific bugs.

Based on our review, it does not appear that the known bugs fixed in the more than 700 upstream commits have had any specific security effects on the contracts in the `ckb-miscellaneous-scripts` and `ckb-anyone-can-pay` repositories. However, a full evaluation of each of the commits was outside the scope of our assessment. When we quickly looked through the commit messages, we identified numerous bug fixes in C math functions and in core functions such as `memcpy` on `riscv`.

The `nervosnetwork/riscv-newlib` library is used in both the CKB virtual machine and the `ckb-riscv-gnu-toolchain` Docker image used to compile contracts.

Exploit Scenario
An attacker uses one of the many bugs fixed in the upstream `riscv-newlib` library to cause the CKB virtual machine or contracts to malfunction, which could lead to undefined behavior, chain splits, or theft of funds.

Recommendations
Short term, incorporate the upstream changes into `nervosnetwork/riscv-newlib` and update the users of this dependency. This will address the known bugs.

Long term, adopt a development and release process that involves periodically updating forked dependencies from their upstream sources.

Appendix A. Vulnerability Classifications

Vulnerability Classes	
Class	**Description**
Access Controls	Related to authorization of users and assessment of rights
Auditing and Logging	Related to auditing of actions or logging of problems
Authentication	Related to the identification of users
Configuration	Related to security configurations of servers, devices, or software
Cryptography	Related to protecting the privacy or integrity of data
Data Exposure	Related to unintended exposure of sensitive information
Data Validation	Related to improper reliance on the structure or values of data
Denial of Service	Related to causing a system failure
Error Reporting	Related to the reporting of error conditions in a secure fashion
Patching	Related to keeping software up to date
Session Management	Related to the identification of authenticated users
Timing	Related to race conditions, locking, or the order of operations
Undefined Behavior	Related to undefined behavior triggered by the program

Severity Categories	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices or Defense in Depth.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is relatively small or is not a risk the customer has indicated is important.
Medium	Individual users' information is at risk; exploitation could pose reputational, legal, or moderate financial risks to the client.
High	The issue could affect numerous users and have serious reputational, legal, or financial implications for the client.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is commonly exploited; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of a complex system.
High	An attacker must have privileged insider access to the system, may need to know extremely complex technical details, or must discover other weaknesses to exploit this issue.

Appendix B. Code Quality Recommendations

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

ckb-anyone-can-pay/c/anyone_can_pay.c:235: The `else if (found_inputs > 1)` clause of the conditional is dead code and will never execute. This is because `found_inputs` will be either 0 or 1 after exiting the previous loop.

c/anyone_can_pay.c, lines 49 – 51: The check is not necessary, as the `ckb_load_script_hash` syscall fills in 32 bytes.

c/anyone_can_pay.c, lines 142 – 144: The condition is already checked a couple of lines earlier, after the syscall to `ckb_checked_load_cell_by_field`.