Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.05.22, the SlowMist security team received the Nervos team's security audit application for ccToken, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.

Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Reentrancy Vulnerability
- Replay Vulnerability
- Reordering Vulnerability
- Denial of Service Vulnerability
- Transaction Ordering Dependence Vulnerability
- Race Conditions Vulnerability
- Authority Control Vulnerability
- Integer Overflow and Underflow Vulnerability
- TimeStamp Dependence Vulnerability
- Unsafe External Call Audit
- Scoping and Declarations Audit

3 Project Overview

3.1 Project Introduction

ccToken is an xUDT Token with added governance logic.

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Debug functions buffer offset stack overflow	Others	Suggestion	Fixed
N2	`calculate_inputs_len` integer overflow risk	Integer Overflow and Underflow Vulnerability	Low	Fixed
N3	char2hex Logic error	Design Logic Audit	High	Fixed
N4	Risk of excessive authority	Authority Control Vulnerability Audit	Low	Ignored
N5	The `goto exit;` statement is not recommended.	Others	Suggestion	Ignored
N6	`find_cells_by_code_hash` no error returned	Design Logic Audit	Medium	Fixed
N7	simple_udt should check the size of the input and output amounts	Design Logic Audit	Low	Ignored
N8	Missing array bounds checking	Others	Low	Fixed
N9	Avoid unnecessary memory copies	Others	Suggestion	Fixed
N10	Ambiguous error handling	Others	Low	Ignored
N11	Missing system shutdown status check	Authority Control Vulnerability Audit	Low	Ignored
N12	`DeployConfig` can be called repeatedly.	Design Logic Audit	Medium	Fixed
N13	It is recommended to use the encapsulated method	Others	Suggestion	Fixed

4 Code Overview

4.1 Contracts Description

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

Audit version:

https://github.com/dotbitHQ/ccToken-contracts/tree/testnet

Initial audit commit: 893823aea57313fc56afaca672dc49b24aae0b44

Final audit commit: 0ea8ff2d80c0372486ac5f5653d1a4d62d1e0c42

Audit scope:

```
├── contracts
│ ├── always-success
│ │ ├── Cargo.toml
│ │ └── src
│ │     ├── entry.rs
│ │     ├── error.rs
│ │     └── main.rs
│ ├── c
│ │ ├── Makefile
│ │ ├── always_success.c
│ │ ├── common.h
│ │ ├── structures.h
│ │ ├── tx_parser.h
│ │ ├── utils.h
│ │ └── xudt_owner.c
│ ├── config-cell-type
│ │ ├── Cargo.toml
│ │ └── src
│ │     ├── entry.rs
│ │     ├── error.rs
│ │     └── main.rs
│ ├── governance-member-cell-type
│ │ ├── Cargo.toml
│ │ └── src
│ │     ├── entry.rs
│ │     ├── error.rs
│ │     └── main.rs
│ └── tick-cell-type
│     ├── Cargo.toml
│     └── src
│         ├── entry.rs
│         ├── error.rs
│         ├── main.rs
```

```
|           └── parser.rs
└── libs
    ├── core
    │  ├── Cargo.toml
    │  └── src
    │      ├── config.rs
    │      ├── constants.rs
    │      ├── data_parser
    │      │  ├── config_cell.rs
    │      │  ├── governance_member_cell.rs
    │      │  ├── mod.rs
    │      │  └── tick_cell.rs
    │      ├── error.rs
    │      ├── lib.rs
    │      ├── macros.rs
    │      ├── util.rs
    │      └── verifiers
    │          ├── basic.rs
    │          ├── mod.rs
    │          └── permission.rs
    └── types
        ├── Cargo.toml
        ├── rust
        │  ├── src
        │  │  ├── constants.rs
        │  │  ├── convert.rs
        │  │  ├── data_parser
        │  │  │  └── mod.rs
        │  │  ├── lib.rs
        │  │  └── util.rs
        │  └── tests
        │      ├── basic.rs
        │      └── util.rs
        └── schemas
            ├── basic.mol
            ├── cell.mol
            └── xudt_rce.mol
```

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

The contract implements the basic governance logic of the token in these TYPES:

```
config-cell-type
governance-member-cell-type
```

```
tick-cell-type
```

4.3 Vulnerability Summary

[N1] [Suggestion] Debug functions buffer offset stack overflow

Category: Others

Content

There is no check to see if offset exceeds the size of debug_buffer. If data_len is too large, it may cause a stack overflow.

- contracts/c/common.h

```
static void debug_print_data_impl(const char *prefix,
                                  const uint8_t *data,
                                  uint32_t data_len) {
   int offset = 0;
   offset += sprintf_(debug_buffer, "%s", prefix);
   for (size_t i = 0; i < data_len; i++) {
       offset += sprintf_(debug_buffer + offset, "%02x", data[i]);
   }
   debug_buffer[offset] = '\0';
   ckb_debug(debug_buffer);
}
static void debug_print_int_impl(const char *prefix, int ret) {
   int offset = 0;
   offset += sprintf_(debug_buffer, "%s(%d)", prefix, ret);
   debug_buffer[offset] = '\0';
   ckb_debug(debug_buffer);
}
static void debug_print_string_impl(const char *prefix,
                                    const uint8_t *data,
                                    uint32_t data_len) {
   int offset = 0;
   offset += sprintf_(debug_buffer, "%s", prefix);
   for (size_t i = 0; i < data_len; i++) {
       offset += sprintf_(debug_buffer + offset, "%c", data[i]);
   }
   debug_buffer[offset] = '\0';
   ckb_debug(debug_buffer);
}
```

Solution

Check the remaining space before writing.

Status

Fixed

[N2] [Low] `calculate_inputs_len` integer overflow risk

Category: Integer Overflow and Underflow Vulnerability

Content

`hi *= 2` does not use a safe calculation method and may cause an overflow.

- contracts/c/common.h

```
int calculate_inputs_len() {
   uint64_t len = 0;
   /* lower bound, at least tx has one input */
   int lo = 0;
   /* higher bound */
   int hi = 4;
   int ret;
   /* try to load input until failing to increase lo and hi */
   while (1) {
       ret = ckb_load_input_by_field(NULL, &len, 0, hi, CKB_SOURCE_INPUT,
                                     CKB_INPUT_FIELD_SINCE);
       if (ret == CKB_SUCCESS) {
           lo = hi;
           hi *= 2;


       } else {
           break;
       }
   }
//...
```

Solution

Check if `h2` will overflow.

Status

Fixed

[N3] [High] char2hex Logic error

Category: Design Logic Audit

Content

This function assumes that the input hexChar is a valid hexadecimal character, but does not check if the character is in the valid range. If the input hexChar is an illegal character, the result of the calculation will be wrong. For example, entering 'G' or 'H' will result in undesired output.

Also, the function only handles hexadecimal characters with uppercase letters ('A' through 'F'), but hexadecimal characters also include lowercase letters ('a' through 'f').

- contracts/c/utils.h

```
char char2hex(char hexChar) {
   char tmp;


   if(hexChar<='9') {
       tmp = hexChar-'0';
   }
   else if(hexChar<='F') {
       tmp = hexChar-'7';
   }
   else {
       tmp = hexChar-'W';
   }
   return tmp;
}
```

Solution

Add handling of lowercase letters.

Add handling of illegal characters.

Status

Fixed

[N4] [Low] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

`Custodian` has the following unlock permissions:

```
update_merchants
confirm_mint
reject_mint
confirm_burn
reject_burn
```

`Merchant` has the following unlock permissions:

```
update_merchants
Request mint
Request burn
```

Solution

In the short term, transferring owner ownership to multisig contracts is an effective solution to avoid single-point risk. But in the long run, it is a more reasonable solution to implement a privilege separation strategy and set up multiple privileged roles to manage each privileged function separately. And the authority involving user funds should be managed by the community, and the authority involving emergency contract suspension can be managed by the EOA address. This ensures both a quick response to threats and the safety of user funds.

Status

Ignored; The business model based on escrow-issued tokens requires some special role permissions.

[N5] [Suggestion] The `goto exit;` statement is not recommended.

Category: Others

Content

The use of the goto statement, while simplifying the error handling process, is not recommended in modern C/C++ programming because it tends to make the code difficult to maintain and understand.

- contracts/c/common.h

```
#define CHECK(_code)     \
 do {                    \
   int code = (_code); \
   if (code != 0) {     \
     err = code;        \
     goto exit;         \
   }                    \
 } while (0)


#define CHECK2(cond, code) \
 do {                     \
   if (!(cond)) {         \
     err = code;          \
     goto exit;           \
   }                      \
 } while (0)
```

Solution

Using Loop Control Statements

Status

Ignored; Using goto is used to facilitate error handling.

[N6] [Medium] `find_cells_by_code_hash` no error returned

Category: Design Logic Audit

Content

The return value of a function is always 0, which means that even if an error occurs, the caller will not be able to correctly determine it.

- contracts/c/tx_parser.h

```
int find_cells_by_code_hash(size_t *cell_indices, size_t *cell_indices_len, int
field, int source, uint8_t *expected_hash)
{
   int err = 0;
   size_t i = 0, j = 0;


   debug_print_data("try to find cells by code hash: ", expected_hash,
BLAKE2B_BLOCK_SIZE);
   debug_print_int("search source: ", source);
   debug_print_int("search field: ", field);
   uint8_t buff[TMP_SIZE];
   uint64_t len = TMP_SIZE;
   for (i = 0; i < MAX_CELL_NUM; i++)
   {
       len = TMP_SIZE;
       debug_print_int("try to load cell by field: idx", (int)i);
       err = ckb_checked_load_cell_by_field(buff, &len, 0, i, source, field);
       debug_print_int("load cell by field: ret", err);
       // debug_print_data("loaded cell[0..64]: ", buff, 64);
       if (err == CKB_INDEX_OUT_OF_BOUND)
       {
           break;
       }
       if (err == CKB_ITEM_MISSING)
       { // just no type script
           continue;
       }
       if (err != 0)
       {
           break;
       }


       if (memcmp(expected_hash, buff + 16, BLAKE2B_BLOCK_SIZE) == 0)
       {
           debug_print_int("cell found, index", i);
           // check if it's a type script
           if (buff[16 + BLAKE2B_BLOCK_SIZE] != 1)
           {
               break;
           }
           cell_indices[j] = i;
           j++;
           // avoid overflow
           if (j == *cell_indices_len)
           {
               break;
           }
       }
   }
   *cell_indices_len = j;
   return 0;
}
```

Solution

Returns a value greater than 0 when `err ! = 0` returns a value greater than 0.

Status

Fixed

[N7] [Low] simple_udt should check the size of the input and output amounts

Category: Design Logic Audit

Content

simple_udt should check the size of the input and output amounts.

- contracts/c/tx_parser.h

```
int simple_udt(uint128_t *ia, uint128_t *oa)
{
   //...
   *ia = input_amount;
   *oa = output_amount;
   return CKB_SUCCESS;
}
```

Solution

Check `input_amount>output_amount`

Status

Ignored; The xudt_extension code is deprecated.

[N8] [Low] Missing array bounds checking

Category: Others

Content

Accessing `data[0]` and `data[1...]` without boundary checking may result in an array out-of-bounds error.

without boundary checking may result in an array out-of-bounds error.

- libs/core/src/data_parser/config_cell.rs

```
pub fn parse(data: Vec<u8>) -> Result<(u8, Vec<(ConfigKey, Vec<u8>)>), CoreError> {
   let version = data[0];
   let mut configs = vec![];
   match version {
       0 => {
           let config_mol =
               BytesVec::from_compatible_slice(&data[1..]).map_err(|_|
CoreError::ParseCellDataFailed {
        //...
}
```

The code does not check if the input args and data are of sufficient length. Direct access to `args[0]` and `data[0]` may result in out-of-bounds access, which can trigger a crash.

- libs/core/src/data_parser/governance_member_cell.rs

```
pub fn parse_type_args(args: Vec<u8>) -> Result<(GovernanceMemberRole, Vec<u8>), CoreError> {
    let role = GovernanceMemberRole::try_from(args[0]).map_err(|_|
CoreError::ParseCellDataFailed {
        cell_name: String::from("GovernanceMemberCell"),
        msg: format!("Unknown role value {}", args[0]),
    })?;
    let cell_id = (&args[1..]).to_vec();
    //...
    Ok((role, cell_id))
}
pub fn parse_data(data: Vec<u8>) -> Result<(u8, GovernanceMembers), CoreError> {
    let version = data[0];
    //...
               GovernanceMembers::from_compatible_slice(&data[1..]).map_err(|_|
CoreError::ParseCellDataFailed {
    //...
}
```

- libs/core/src/data_parser/tick_cell.rs

```
pub fn parse_data(data: Vec<u8>) -> Result<(u8, Tick), CoreError> {
    let version = data[0];
    //...
           tick = Tick::from_compatible_slice(&data[1..]).map_err(|_|
CoreError::ParseCellDataFailed {
    //...


    Ok((version, tick))
}
```

- libs/core/src/verifiers/permission.rs

```
pub fn verify_input_has_deploy_lock(index: usize) -> Result<(), CoreError> {
    //...
    cc_assert!(cells[0] == index, CoreError::DeployLockIsRequired { index });


    Ok(())
}


pub fn verify_input_has_owner_lock(index: usize) -> Result<(), CoreError> {
    //...
    cc_assert!(cells[0] == index, CoreError::OwnerLockIsRequired { index });


    Ok(())
}
```

Solution

Check if the data length is sufficient.

Status

Fixed

[N9] [Suggestion] Avoid unnecessary memory copies

Category: Others

Content

Function arguments can accept slice references (&[u8]) instead of ownership (Vec), which avoids unnecessary memory copies.

- libs/core/src/data_parser/config_cell.rs

```
pub fn parse(data: Vec<u8>) -> Result<(u8, Vec<(ConfigKey, Vec<u8>)>), CoreError> {
        //...
}
```

- libs/core/src/data_parser/governance_member_cell.rs

```
pub fn parse_type_args(args: Vec<u8>) -> Result<(GovernanceMemberRole, Vec<u8>),
CoreError> {
    //...
}
pub fn parse_data(data: Vec<u8>) -> Result<(u8, GovernanceMembers), CoreError> {
    //...
}
```

- libs/core/src/data_parser/tick_cell.rs

```
pub fn parse_data(data: Vec<u8>) -> Result<(u8, Tick), CoreError> {
    //...
}
```

- libs/core/src/util.rs

```
pub fn get_tx_action() -> Result<Action, CoreError> {
    //...
    let version_byte = witness[0];
    //...
    let action_bytes = &witness[1..];
    //...
}
```

Solution

Use slice references instead of ownership.

Status

Fixed

[N10] [Low] Ambiguous error handling

Category: Others

Content

Use the `warn!` log to record potential parsing errors, which is not uncommon in smart contracts, without explicitly indicating whether the error should be interrupted or accepted.

- libs/core/src/data_parser/config_cell.rs

```
pub fn parse(data: Vec<u8>) -> Result<(u8, Vec<(ConfigKey, Vec<u8>)>), CoreError> {
   //...
               let key = match ConfigKey::try_from(u32::from_le_bytes(key_bytes)) {
                   Ok(key) => key,
                   Err(_) => {
                       warn!(
                           "[{}] Parse [0..4]({}) to config key failed, the key is
removed or not defined.",
                           i,
                           hex_string(key_bytes.as_ref())
                       );
                       continue;
                   }
               };
   //...
   }


   Ok((version, configs))
}
```

Solution

Returns an error or interrupt logic, which can be handled with `debug!` if you don't need to.

Status

Ignored; Forcing logs to be typed with warn! is a specification in our contracts.

[N11] [Low] Missing system shutdown status check

Category: Authority Control Vulnerability Audit

Content

When the system is processing the shutdown state, all other features except configuration should be disabled, including disabling the following features:

```
init_governance
update_owner
update_custodians
update_merchants
confirm_mint
reject_mint
confirm_burn
reject_burn
```

Currently only the request function `request` for token minting and destruction checks if the system is disabled.

Solution

Call `check_system_status()` at the entry of the main function to check if the system has been shutdown.

Status

Ignored; These are intentionally designed business logic.

[N12] [Medium] `DeployConfig` can be called repeatedly.

Category: Design Logic Audit

Content

`DeployConfig` as a configuration initialization function should only be allowed to be called once, subsequent calls should use `UpdateConfig`, otherwise it will cause the system to have multiple configurations in existence, and inconsistent configuration parameters may be exploited to attack the system.

- contracts/config-cell-type/src/entry.rs

```
pub fn main() -> Result<(), Box<dyn AsI8>> {
   debug!("====== Running config-cell-type ======");


   let action = util::get_tx_action()?;
   let self_script =
Script::from(high_level::load_script().map_err(ConfigError::from)?);


   debug!("==== Action {} ====", action.to_string());


   verify_script_args_is_empty(&self_script)?;


   let (input_config_cells, output_config_cells) =
       util::find_cells_by_script_in_inputs_and_outputs(ScriptType::Type,
self_script.as_reader())?;
   match action {
       DeployConfig => {
           verifiers::basic::verify_cell_number_and_position(
               "ConfigCell",
               &input_config_cells,
               &[],
               &output_config_cells,
               &[0],
           )?;
       }
       UpdateConfig => {
           verifiers::basic::verify_cell_number_and_position(
               "ConfigCell",
               &input_config_cells,
               &[0],
               &output_config_cells,
               &[0],
           )?;
       }
       _ => {
           return Err(CoreError::ActionNotSupported {
               action: action.to_string(),
           }
           .into());
       }
   }


   verify_output_lock(output_config_cells[0])?;
   verify_output_data_format(output_config_cells[0])?;


   Ok(())
}
```

Solution

Limit configuration to the owner.

Status

Fixed

[N13] [Suggestion] It is recommended to use the encapsulated method

Category: Others

Content

A wrapped lock validation function already exists, and reusing it improves the readability of the code.

- contracts/config-cell-type/src/entry.rs

```
fn verify_output_lock(index: usize) -> Result<(), Box<dyn AsI8>> {
    //...
```

```
            lock.as_slice() == owner_lock.as_slice(),
        //...
        Ok(())
    }
```

Solution

Use `util::is_entity_eq(&lock, &owner_lock)` instead.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002406030001	SlowMist Security Team	2024.05.22 - 2024.06.03	Low Risk

Summary conclusion: The SlowMist security team use a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 high risk, 2 medium risk, 6 low risk, 4 suggestion vulnerabilities. And 4 low risk, 1 suggestion vulnerabilities were ignored; All other findings were fixed. The code was not deployed to the mainnet.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.