TokenSoft Token Smart Contract Security Assessment

TokenSoft Inc

September 3, 2021

Prepared for
Will Binns

Prepared by
Aleks Kircanski

Executive Summary

Synopsis

During the summer of 2021, TokenSoft Inc. engaged NCC Group to conduct a security assessment of the TokenSoft Token smart contract. The contract is implemented in Clarity, an interpreted LISP-like language used to specify contracts on the Stacks blockchain. The reviewed smart contract implements the SIP 10 standard, loosely based on the ERC-20 token standard. It exposes state-changing operations for transferring, minting and revoking fungible tokens, supports administrative roles and blacklisting and, importantly, leverages native Clarity functions for token handling.

The review lasted for 3 person days. One consultant worked on the project. A kickoff and readout call were held at the beginning and at the end of the project.

Scope

The scope of this review were the following three files:

- `tokensoft-token.clar`: A specific implementation of the SIP 10 standard proposed for use for new token creation
- `ft-trait.clar`: The token trait, as specified by SIP 10
- `restricted-token-trait.clar`: The trait that allows specifying transfer restrictions, not covered by SIP 10, but implemented by `tokensoft-token.clar`.

NCC Group reviewed the following commit: `d3288069bbb0c4a5c45946eb363e0a9ace359222` of the `tokensoft/tokensoft_token` repository on Github. Out of scope was the underlying Clarity fungible token implementation, including functions such as `ft-transfer`, `ft-mint`, `ft-burn`, etc.

Testing Methodology and Key Findings

It should be noted that the Clarity bug landscape is unexplored and the most common bug patterns are not well known at this stage. This is due to the fact that (1) Clarity is a relatively novel language and (2) many of the bug patterns common to Solidity do not apply.[1]

The strategy used during the review was to search for general unexpected behavior in contract modules. Since the contract is simple, for each function, an intuition on what the expected behavior of the function was built. Such an intuition was built based on both the function code and available documentation (source code comments and SIP 10). Next, edge-case analysis of the function was performed using `Clarinet`, in attempt to violate expected behavior of the function.

Using this strategy, two findings were uncovered:

- **Lack of Token Supply Limiting**: The TokenSoft Token contract does not use the Clarity's native functionality for limiting the supply of tokens. All TokenSoft Token contracts will have the same maximal supply of $2^{128} - 1$ tokens. On an attempt to mint more than the maximal number of tokens, the Clarity REPL results with a panic, however, the actual Clarity VM running inside the blockchain client is expected to just abort the transaction.
- **Error Code Ambiguity in Token Transfer:** After receiving an error on a token transfer transaction, the caller will not be able to tell what went wrong. This may introduce unintended behavior in higher-level applications of the contract. In addition, the transfer function implementation diverges from what is described by SIP 10, which may introduce incompatibilities with competing implementations.

Several informational severity observations were made as well, see the findings section.

[1] https://blog.blockstack.org/bringing-clarity-to-8-dangerous-smart-contract-vulnerabilities/

Table of Findings

For each finding, NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. For an explanation of NCC Group's risk rating and finding categorization, see Appendix A on page 10.

Title	ID	Risk
Lack of Token Supply Limiting	001	Low
Error Code Ambiguity in Token Transfer	002	Low
Unused Blacklist Amount Parameter	003	Informational
Lack of Restriction on Role Codes	004	Informational

Finding **Lack of Token Supply Limiting**

Risk **Low** Impact: Low, Exploitability: Low

Identifier NCC-2021-001

Category Data Validation

Location https://github.com/wrappedfi/tokensoft_token_stacks/blob/d3288069/contracts/tokensoft-token.clar#L25

Impact The TokenSoft Token contract does not use the Clarity's native functionality for limiting the supply of tokens. All TokenSoft Token contracts will have the same maximal supply of $2^{128}-1$ tokens. On an attempt to mint more than the maximal number of tokens, the Clarity REPL results with a panic, however, the actual Clarity VM running inside the blockchain client is expected to just abort the transaction.

Description As per Clarity documentation, the `define-fungible-token` function allows specifying the maximal token supply. Also mentioned in the Clarity smart contract bug blog post:

> Numeric overflows and underflows cause the transaction to abort in Clarity. Also, Clarity has a first-class type for a fungible token with a fixed supply, which prevents the supply from accidentally being inflated (a transaction that tries to mint more tokens than can exist will be forced to abort).

The TokenSoft Token contract omits the maximal supply parameter, which prevents the users of the contract to control the maximal supply:

```
;; A second optional parameter can be added here to set an upper limit on max tot
➜ al-supply
(define-fungible-token tokensoft-token)
```

When it comes to arithmetic overflow handling in the absence of maximal supply, the `mint-tokens` function allows participants with `MINTER_ROLE` to mint new tokens. If more than $2^{128}-1$ tokens are minted, the Clarity REPL used inside Clarinet panics. Verifying whether the actual Clarity VM running inside the blockchain would panic as well indicates that this should not be the case and as such this finding is rated as low.

The reason Clarity REPL panics is the `unwrap` call on `execute_contract`, see clarity-repl/src/repl/interpreter.rs:200. The `execute_contract` calls are not unwrapped inside the `stacks-blockchain` repository. The native minting function inside the Clarity VM does contain code that can potentially panic on an arithmetic overflow due to minting, however, this is prevent by returning errors before panicking:

```
pub fn special_mint_token(
    args: &[SymbolicExpression],
    env: &mut Environment,
    context: &LocalContext,
) -> Result<Value> {

// .. SNIP ..

      env.global_context.database.checked_increase_token_supply(
            &env.contract_context.contract_identifier,
            token_name,
            amount,
            ft_info,
        )?;

        let to_bal = env.global_context.database.get_ft_balance(
            &env.contract_context.contract_identifier,
            token_name,
            to_principal,
            Some(ft_info),
        )?;
        let final_to_bal = to_bal.checked_add(amount).expect("STX overflow");
```

The `expect` call in the last line is prevent from panicking by the `checked_increase_token_supply`, which returns an error in case of overflow. As such, the assumption that the Clarity VM can be trusted with graceful $2^{128} - 1$ overflow handling is not breached.

Reproduction Steps Use `Clarinet` to mint more than $2^{128} - 1$ tokens and observe the panic:

```
(contract-call? .tokensoft-token mint-tokens u340282366920938463463374607431768211
→  1455 'ST1PQHQKV0RJXZFY1DGX8MNSNYVE3VGZJSRTPGZGM)
(contract-call? .tokensoft-token mint-
→  tokens u1 'ST1PQHQKV0RJXZFY1DGX8MNSNYVE3VGZJSRTPGZGM)
→
```

Recommendation As per the comment in the code, allow users to set the limit on the token supply. In the case users opt to not set such a limit, the Clarity VM inside the blockchain client is expected to guard from panic and Denial of Service.

Finding	**Error Code Ambiguity in Token Transfer**
Risk	**Low** Impact: Low, Exploitability: Low
Identifier	NCC-2021-002
Category	Other
Location	https://github.com/tokensoft/tokensoft_token_stacks/blob/d3288069/contracts/tokensoft-token.clar#L56
Impact	After receiving an error on a token transfer transaction, the caller will not be able to tell what went wrong. This may introduce unintended behavior in higher-level applications of the contract. In addition, the transfer function implementation diverges from what is described by SIP 10, which may introduce incompatibilities with competing implementations.

Description As for the token transfer function, SIP 10 outlines the following error code specification:

error code	reason
u1	`sender` does not have enough balance
u2	`sender` and `recipient` are the same
u3	`amount` is non-positive
u4	`sender` is not the same as `tx-sender`

The TokenSoft transfer function is as follows:

```
(define-public (transfer (amount uint) (sender principal) (recipient principal )
➜  (memo (optional (buff 34) )))
  (begin
    (try! (detect-transfer-restriction amount sender recipient))
     ➜  ;; Ensure there is no restriction
    (asserts! (is-eq tx-sender sender) (err u4))
     ➜  ;; Ensure the originator is the sender principal
    (ft-transfer? tokensoft-token amount sender recipient) ) ) ;; Transfer
```

The `ft-transfer` function will return `(err u1)` if the sender does not have sufficient funds, conforming to the SIP 10 standard. At the same time, the `detect-transfer-restriction` will also return `(err u1)` if the participant is blacklisted:

```
(define-constant RESTRICTION_NONE u0) ;; No restriction detected
(define-constant RESTRICTION_BLACKLIST u1)
➜  ;; Sender or receiver is on the blacklist

;; Checks to see if a transfer should be restricted.  If so returns an error code
➜  that specifies restriction type.
(define-read-only (detect-transfer-restriction (amount uint) (sender principal)
➜  (recipient principal))
  (if (or (is-blacklisted sender) (is-blacklisted recipient))
    (err RESTRICTION_BLACKLIST)
    (ok RESTRICTION_NONE)))
```

The caller will not be able to discern which of the two events happened. The error codes produced by the custom TokenSoft token transfer conflict with native Clarity `ft-transfer` error codes and diverge from the SIP 10 spec.

Reproduction Steps

Set up the two accounts as follows:

```
(contract-call? .tokensoft-
 ➜  token initialize "test" "HC" u5  'ST1PQHQKV0RJXZFY1DGX8MNSNYVE3VGZJSRTPGZGM)
 ➜

(contract-call? .tokensoft-token add-principal-to-
 ➜  role u4  'ST1PQHQKV0RJXZFY1DGX8MNSNYVE3VGZJSRTPGZGM)
 ➜

(contract-call? .tokensoft-token mint-
 ➜  tokens u10 'ST1PQHQKV0RJXZFY1DGX8MNSNYVE3VGZJSRTPGZGM)
 ➜

(contract-call? .tokensoft-token mint-
 ➜  tokens u10 'STNHKEPYEPJ8ET55ZZ0M5A34J0R3N5FM2CMMMAZ6)
 ➜
```

Observe the error that indicates lack of funds:

```
>> (contract-call? .tokensoft-token transfer u20 'ST1PQHQKV0RJXZFY1DGX8MNSNYVE3VG
 ➜  ZJSRTPGZGM  'STNHKEPYEPJ8ET55ZZ0M5A34J0R3N5FM2CMMMAZ6 none)
(err u1)
```

Blacklist one of the participants:

```
(contract-call? .tokensoft-token update-
 ➜  blacklisted 'ST1PQHQKV0RJXZFY1DGX8MNSNYVE3VGZJSRTPGZGM true)
 ➜
```

Attempt the transfer and observe that the error is the same:

```
>> (contract-call? .tokensoft-token transfer u2 'ST1PQHQKV0RJXZFY1DGX8MNSNYVE3VGZ
 ➜  JSRTPGZGM  'STNHKEPYEPJ8ET55ZZ0M5A34J0R3N5FM2CMMMAZ6 none)
(err u1)
```

Recommendation

In SIP-10, a notion of post-condition is introduced. Eventual existence of post-conditions impacts the list of possible error codes returned by the transfer function. Consider modifying SIP 10 to allocate error codes for post-condition handling, including black listing related error codes. Consider modifying the implementation to conform to the new spec.

Finding	**Unused Blacklist Amount Parameter**
Risk	**Informational** Impact: None, Exploitability: None
Identifier	NCC-2021-003
Category	Other
Location	https://github.com/tokensoft/tokensoft_token_stacks/blob/d3288069/contracts/tokensoft-token.clar#L182
Impact	This is an informational level finding and there is no associated security risk. A publically exposed function requires a parameter that is not used in the contract function.

Description

The current blacklisting functionality inside the TokenSoft Token contract allows preventing chosen addresses to perform any transfers. Each time tokens are transferred, the contract verifies if any of the two addresses are blacklisted using the following function:

```
(define-read-only (detect-transfer-restriction (amount uint) (sender principal)
➜  (recipient principal))
  (if (or (is-blacklisted sender) (is-blacklisted recipient))
    (err RESTRICTION_BLACKLIST)
    (ok RESTRICTION_NONE)))
```

The `amount` parameter to this function is not used inside the function.

Recommendation

Before users deploy the contract for their tokens, implement the amount-wise blacklists, or remove the `amount` parameter from the `detect-transfer-restriction` function.

Finding	**Lack of Restriction on Role Codes**
Risk	**Informational** Impact: None, Exploitability: None
Identifier	NCC-2021-004
Category	Data Validation
Location	https://github.com/tokensoft/tokensoft_token_stacks/blob/d3288069bbb0c4a5c45946eb363e0a9ace359222/contracts/tokensoft-token.clar#L84
Impact	This is an informational level finding and there is no associated security risk. The contract does not verify if added role codes relate to meaningful roles or not. The contract owner can tag users with meaningless role codes.

Description The list of possible roles in the contract is as follows:

```
(define-constant OWNER_ROLE u0) ;; Can manage RBAC
(define-constant MINTER_ROLE u1) ;; Can mint new tokens to any account
(define-constant BURNER_ROLE u2) ;; Can burn tokens from any account
(define-constant REVOKER_ROLE u3)
 ➜  ;; Can revoke tokens and move them to any account
(define-constant BLACKLISTER_ROLE u4)
 ➜  ;; Can add principals to a blacklist that can prevent transfers
```

The map that keeps track of the user-role association:

```
(define-map roles { role: uint, account: principal } { allowed: bool })
```

The function that adds new user-role entries does not verify if the role code is meaningful or not:

```
(define-public (add-principal-to-role (role-to-add uint) (principal-to-add
 ➜  principal))
   (begin
    ;; Check the contract-caller to verify they have the owner role
    (asserts! (has-role OWNER_ROLE contract-caller) (err
     ➜  PERMISSION_DENIED_ERROR))
    ;; Print the action for any off chain watchers
    (print { action: "add-principal-to-role", role-to-add: role-to-add,
     ➜  principal-to-add: principal-to-add })
    (ok (map-set roles { role: role-to-add, account: principal-to-add } {
     ➜  allowed: true }))))
```

It is also worth noting that instead of having the `allowed` field for each map entry, it may be simpler to simply delete the map entry on role revocation. The (`map-delete`)[2] function could be used for that purpose.

Recommendation Consider restricting the role codes according to the role list. Additionally, if there is no particular reason for the boolean flag inside the `roles` map, consider refactoring the code. Existence of an entry inside the map could imply that the user has the role and, on role revocation, the map entry can be deleted. This would prevent old role entries from lingering inside the map forever.

[2] https://docs.stacks.co/references/language-functions#map-delete

Appendix A: Finding Field Definitions

The following sections describe the risk rating and category assigned to issues NCC Group identified.

Risk Scale

NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. The risk rating is NCC Group's recommended prioritization for addressing findings. Every organization has a different risk sensitivity, so to some extent these recommendations are more relative than absolute guidelines.

Overall Risk

Overall risk reflects NCC Group's estimation of the risk that a finding poses to the target system or systems. It takes into account the impact of the finding, the difficulty of exploitation, and any other relevant factors.

Critical	Implies an immediate, easily accessible threat of total compromise.
High	Implies an immediate threat of system compromise, or an easily accessible threat of large-scale breach.
Medium	A difficult to exploit threat of large-scale breach, or easy compromise of a small portion of the application.
Low	Implies a relatively minor threat to the application.
Informational	No immediate threat to the application. May provide suggestions for application improvement, functional issues with the application, or conditions that could later lead to an exploitable finding.

Impact

Impact reflects the effects that successful exploitation has upon the target system or systems. It takes into account potential losses of confidentiality, integrity and availability, as well as potential reputational losses.

High	Attackers can read or modify all data in a system, execute arbitrary code on the system, or escalate their privileges to superuser level.
Medium	Attackers can read or modify some unauthorized data on a system, deny access to that system, or gain significant internal technical information.
Low	Attackers can gain small amounts of unauthorized information or slightly degrade system performance. May have a negative public perception of security.

Exploitability

Exploitability reflects the ease with which attackers may exploit a finding. It takes into account the level of access required, availability of exploitation information, requirements relating to social engineering, race conditions, brute forcing, etc, and other impediments to exploitation.

High	Attackers can unilaterally exploit the finding without special permissions or significant roadblocks.
Medium	Attackers would need to leverage a third party, gain non-public information, exploit a race condition, already have privileged access, or otherwise overcome moderate hurdles in order to exploit the finding.
Low	Exploitation requires implausible social engineering, a difficult race condition, guessing difficult-to-guess data, or is otherwise unlikely.

Category

NCC Group categorizes findings based on the security area to which those findings belong. This can help organizations identify gaps in secure development, deployment, patching, etc.

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 mathematical protections for 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 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 order of operations.

Appendix B: Project Contacts

The team from NCC Group has the following primary members:

- Aleksandar Kircanski — Consultant
  aleksandar.kircanski@nccgroup.com
- Javed Samuel — Cryptography Services Director
  javed.samuel@nccgroup.com

The team from TokenSoft Inc has the following primary member:

- Will Binns — TokenSoft
  will@tokensoft.io