verichains

SECURITY AUDIT OF

CHINTAI SMART CONTRACT

CHINTAI

Public Report

July 15, 2022

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
EOS	EOS (EOSIO) is a highly performant open-source blockchain platform, built to support and operate safe, compliant, and predictable digital infrastructures.
EOSIO.CDT	EOSIO.CDT (Contract Development Toolkit) is a suite of tools used to build EOSIO contracts
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
IPO	Initial public offering (IPO) refers to the moment a private company starts offering its shares to the public for the first time. The term "going public" may also be used to refer to IPOs in some casual instances.
AMM	An automated market maker (AMM) is a type of decentralized exchange (DEX) protocol that relies on a mathematical formula to price assets. Instead of using an order book like a traditional exchange, assets are priced according to a pricing algorithm.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on July 15, 2022. We would like to thank the Chintai for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the Chintai Smart Contract. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some vulnerable issues in the application, along with some recommendations.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About Chintai

Chintai provides a comprehensive blockchain solution that modernizes capital markets for asset managers, banks and enterprise.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of the Chintai Smart Contract.

It was conducted on the source code provided by Chintai team. The latest version of the following repositories were made available in the course of the review:

Repository	Commit
`https://github.com/chintai-platform/sc-balances`	`f458ea310990241b4fa685c6e1ceabf899467e8b`
`https://github.com/chintai-platform/sc-compliance`	`ac755cb4fd20b2066ef6d70d8b7c963a77afc8b7`
`https://github.com/chintai-platform/sc-sale-ipo`	`7e3b47fc8c8e55c93c03482a3242287917940672`
`https://github.com/chintai-platform/sc-token`	`15d33b28e06afa5a2438efb3024178dfb459c582`
`https://github.com/chintai-platform/sc-token-debt`	`593bc1cbf515b91d58a5709367fb74bff7eaaa4b`
`https://github.com/chintai-platform/sc-token-equity`	`2a12d25a5498222d83f7553db158356be458a8e1`
`https://github.com/chintai-platform/sc-token-fiat`	`f0074af646d60101ad306d1b4420cc5ef50b94df`
`https://github.com/chintai-platform/sc-token-realestate`	`add4751ff11f6333f4c8280868919dbfd08640e7`

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and our in-house smart contract security analysis tool.

- 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:

- Integer Overflow and Underflow
- Improper Token Precision Handling
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- Unsafe Memory Handling
- Reentrancy
- Action Authorizations
- Unsafe Random Number Generator
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The Chintai Smart Contract was written in `C++` language which requires the `EOSIO.CDT` (Contract Development Toolkit) version to be `1.8.1`. These contracts are used to deploy to the EOS blockchain which operates on Delegated Proof of Stake, built on the open-source software framework of EOSIO.

Below is the summary of the repositories in the audit scope:

2.1.1. Balances contract

The balance contract is used to keep track user token balances from multiple token contracts.

2.1.2. Token contracts

There are multiple types of token contracts, which include debt token, equity token, fiat token, and real-estate token which will be used with the trade AMM contract. A standard token contract provides multiple features like freezing, issuing, transferring, securing (locking), vesting,...

2.1.3. Sale IPO contract

This contract is used for sale IPO.

2.1.4. Compliance contract

This contract is used to blacklist some countries from receiving tokens and also using to calculate the risk score when transferring.

2.2. Findings

During the audit process, the audit team found some vulnerabilities in the given version of Chintai Smart Contract. Chintai fixed the code, according to Verichains's private reports.

2.2.1. Missing token contract account check CRITICAL

Affected files:

- sale-ipo/src/ipo.cpp

The `ipo` contract has a `deposit` action which is a payable action. The `deposit` action is declared in the header file as below:

```
[[eosio::on_notify("*::transfer")]]
void deposit(eosio::name const &from,
              eosio::name const &to,
              eosio::asset const &quantity,
              std::string const &memo);
```

By specifying the attribute `eosio::on_notify("*::transfer")`, the ipo contract will listen for all token transfer action events which call the `require_recipient(ipo)` method.

The implementation logic for the `deposit` function is shown below:

```
void ipo::deposit(eosio::name const &from,
                  eosio::name const &to,
                  eosio::asset const &quantity,
                  std::string const &memo)
{
  require_auth( from );
  eosio::check( quantity.amount > 0, "Must transfer positive quantity" );
  offerings offerings_table( global::get_self(), global::get_self().value );
  uint8_t id = std::stoi(memo); //TODO find a better way to parse the memo
  auto offerings_itr = offerings_table.require_find(id, ("There is no ipo on this contract
with id " + std::to_string(id)).c_str() );
  eosio::check( offerings_itr->get_start_date().sec_since_epoch() <=
eosio::current_time_point().sec_since_epoch(), "Trying to deposit to an IPO that has not
begun yet. Please check the start date and try again later" );
  eosio::check( offerings_itr->get_max_tokens_accepted().quantity.symbol ==
quantity.symbol, "Incorrect token transferred - transferred '" +
quantity.symbol.code().to_string() + "," + std::to_string(quantity.symbol.precision()) +
"," + get_first_receiver().to_string() + ", expected '" + offerings_itr-
>get_max_tokens_accepted().quantity.symbol.code().to_string() + "," +
std::to_string(offerings_itr->get_max_tokens_accepted().quantity.symbol.precision()) + ","
+ offerings_itr->get_max_tokens_accepted().contract.to_string() + "'.");
  eosio::asset tokens_remaining = eosio::asset{ offerings_itr-
>get_max_tokens_accepted().quantity.amount - offerings_itr-
>get_current_tokens_accepted().quantity.amount, offerings_itr-
>get_current_tokens_accepted().quantity.symbol };
  eosio::check( offerings_itr->get_current_tokens_accepted().quantity + quantity <=
offerings_itr->get_max_tokens_accepted().quantity, "Attempting to transfer more tokens than
this ipo can accept. Trying to transfer: " + quantity.to_string() + ". Tokens remaining: "
+ tokens_remaining.to_string());

  eosio::extended_asset tokens_offered = offerings_itr->get_max_tokens_offered();
  eosio::extended_asset tokens_accepted = offerings_itr->get_max_tokens_accepted();
  double ratio = static_cast<double>( tokens_offered.quantity.amount ) /
static_cast<double>( tokens_accepted.quantity.amount );
  eosio::asset issue_quantity = eosio::asset{ static_cast<int64_t>( quantity.amount * ratio
) , tokens_offered.quantity.symbol };

  ::token_contract::token::issue_action( tokens_offered.contract, {
tokens_offered.contract, "issue"_n }).send( from, issue_quantity, memo );
```

```
  offerings_table.modify(offerings_itr, eosio::same_payer, [&](auto &entry){
    entry.set_current_tokens_offered(entry.get_current_tokens_offered() +
eosio::extended_asset{issue_quantity, entry.get_current_tokens_offered().contract});
    entry.set_current_tokens_accepted(entry.get_current_tokens_accepted() +
eosio::extended_asset{quantity, entry.get_current_tokens_accepted().contract});
  });
}
```

We can see that, the code below is missing some important checks:

- Skip processing if the `from` account is `get_self()`. The `require_auth(from)` check is unnecessary here.
- The `to` account must be `get_self()`, since transfer notifications can be triggered without sending to the `to` account.
- Validate the contract token account using `get_first_receiver()`. If not, the attacker can deploy a new token contract and create a new token with the same symbol as the accepted token.

UPDATES

- *Jul 15, 2022*: This issue has been acknowledged and fixed by Chintai team.

2.2.2. Token issuance fee must be subtracted from issuer balance MEDIUM

Affected files:

- token/src/fee_handling_issuer_pays_native_token.cpp

When tokens are issued, the issuance fee should be subtracted from the issuer balance if the current fee handler is `fee_handling_issuer_pays_native_token_t`. However, in the `fee_handling_issuer_pays_native_token_t::handle_fees` method, the token balance of the issuer (which can be got from the `stat` table via `get_issuer()`) is not reduced. Moreover, after issuance, the current supply will be increased by `issuance_quantity + issuance_fee`.

```
eosio::asset fee_handling_issuer_pays_native_token_t::handle_fees(
    eosio::asset const &quantity,
    eosio::name const &receiver_name,
    double const fee_percentage,
    double const price,
    eosio::extended_asset const &minimum_fee_size
) {
    int64_t fee_int = static_cast<int64_t>(fee_percentage *
static_cast<double>(quantity.amount));
    eosio::asset fee{fee_int, quantity.symbol};

    security_checks::fee_handling_issuer_pays_native_token(receiver_name, fee,
minimum_fee_size.quantity);

```

```
    token_contract::stats statstable( global::get_self(), quantity.symbol.code().raw() );
    statstable.modify(statstable.begin(), eosio::same_payer, [&]( auto& s ) {
                        s.set_supply( s.get_supply() + fee );
                        });
    token::add_balance( receiver_name, fee );

    eosio::asset empty_asset{0, quantity.symbol};
    return empty_asset;
}
```

UPDATES

- *Jul 15, 2022*: This is not a problem, as the issuer and the token contract are always the same account. Moreover, the fee is not deducted from the issuance amount, but is conducted as an additional issuance.

2.2.3. Checking wrong entry of vesting fund table MEDIUM

Affected files:

- token/src/security_checks.cpp

In the `buyback_vesting` function below, the vesting `id` is used to get the corresponding vesting item. However, `vesting.begin()` is used instead of `itr` which is not correct.

```
void buyback_vesting(eosio::name const &user,
                      eosio::asset const &buyback_quantity,
                      uint64_t const id)
{
  // ...
  vesting_funds vesting( global::get_self(), user.value );
  const auto& itr = vesting.require_find( id, ("No entry on the vesting table for id '" +
std::to_string(id) + "'.").c_str());
  eosio::asset unclaimed_quantity = vesting.begin()->get_total_quantity() -
vesting.begin()->get_claimed_quantity();
  eosio::check( unclaimed_quantity == buyback_quantity, "You must buyback the remaining
quantity when performing a buyback for vesting funds. For table entry with id '" +
std::to_string(id) + "'. Attempted to buyback '" + buyback_quantity.to_string() + "',
unclaimed quantity: '" + unclaimed_quantity.to_string() + "'.");
}
```

UPDATES

- *Jul 15, 2022*: This issue has been acknowledged and fixed by Chintai team.

2.2.4. Modify wrong entry of releasing fund table MEDIUM

Affected files:

- token/src/buyback_releasing.cpp

In the `buyback_releasing_t::buyback` method below, the releasing `id` is used to get the corresponding releasing item. However, `releasing.begin()` is used instead of `releasing_entry` which is not correct.

```
void buyback_releasing_t::buyback(
    eosio::name const &user,
    eosio::name const &receiver,
    eosio::name const &payer,
    eosio::asset const &buyback_quantity,
    eosio::extended_asset const &compensation,
    uint64_t const id,
    std::string const &memo) const
{
  // ...
  releasing_funds releasing( global::_self, user.value );
  auto releasing_entry = releasing.find(id);

  if ( releasing.begin()->get_quantity() > buyback_quantity )
  {
    releasing.modify( releasing.begin(), eosio::same_payer, [&](auto &entry){
                      entry.set_quantity( entry.get_quantity() - buyback_quantity );
                      });
  }
  else
  {
    releasing.erase(releasing.begin());
  }
  // ...
}
```

UPDATES

- *Jul 15, 2022*: This issue has been acknowledged and fixed by Chintai team.

2.2.5. Modify wrong entry of secured fund table MEDIUM

Affected files:

- token/src/buyback_secured.cpp

In the `buyback_secured_t::buyback` method below, the secured `id` is used to get the corresponding secured item. However, `secured.begin()` is used instead of `secured_entry` which is not correct.

```
void buyback_secured_t::buyback(
    eosio::name const &user,
    eosio::name const &receiver,
    eosio::name const &payer,
    eosio::asset const &buyback_quantity,
    eosio::extended_asset const &compensation,
    uint64_t const id,
    std::string const &memo) const
{
  // ...
  secured_funds secured( global::get_self(), user.value);
  auto secured_entry = secured.find(id);
  if ( secured_entry->get_quantity() > buyback_quantity )
  {
    secured.modify( secured.begin(), eosio::same_payer, [&](auto &entry){
                    entry.set_quantity( entry.get_quantity() - buyback_quantity );
                    });
  }
  else
  {
    secured.erase(secured.begin());
  }
  // ...
}
```

UPDATES

- *Jul 15, 2022*: This issue has been acknowledged and fixed by Chintai team.

2.2.6. Modify wrong record of releasing fund table in paybackloan MEDIUM

Affected files:

- token-debt/src/paybackloan_releasing.cpp

In the `paybackloan_releasing_t::paybackloan` method below, the releasing `id` is used to get the corresponding releasing item. However, `releasing.begin()` is used instead of `releasing.find(id)` which is not correct.

```
void paybackloan_releasing_t::paybackloan(
    eosio::name const &user,
    eosio::name const &payer,
    eosio::asset const &quantity,
    eosio::extended_asset const &compensation,
    uint64_t const id,
    std::string const &memo) const
{
  // ...
  token_contract::releasing_funds releasing( global::get_self(), user.value );
  if ( releasing.begin()->get_quantity() > quantity )
  {
    releasing.modify( releasing.begin(), eosio::same_payer, [&](auto &entry){
      entry.set_quantity( entry.get_quantity() - quantity );
    });
  }
  else
  {
    releasing.erase(releasing.begin());
  }
  // ...
}
```

UPDATES

- *Jul 15, 2022*: This issue has been acknowledged and fixed by Chintai team.

2.2.7. Smallest possible transfer must be rounded up LOW

Affected files:

- trade-amm/src/security_checks.cpp

In the `trade` method below, the variable `smallest_possible_transfer` is the minimum amount that could be transfer which is used to ensured that the trading fee can be transferred. However, the division `1 / smallest_fee` must be rounded up.

```
void trade(eosio::name const &account,
           uint64_t const buy_token_id,
           eosio::asset const &quantity,
           eosio::name const &first_receiver)
{
  // ...

  double smallest_fee = get_smallest_fee(sell_token_id);
  int64_t smallest_possible_transfer = 1 / smallest_fee; // MUST BE ROUNDED UP
  eosio::asset smallest_asset{smallest_possible_transfer, quantity.symbol};
  eosio::check(quantity >= smallest_asset, "The transfer size is too low for this token.
Smallest allowed transfer: '" + smallest_asset.to_string() + "'.");

}
```

UPDATES

- *Jul 15, 2022*: This issue has been acknowledged and fixed by Chintai team.

2.2.8. Fee is deducted from contract account instead of issuer account LOW

Affected files:

- token/src/fee_handling_issuer_pays_other_token_amm_price.cpp
- token/src/fee_handling_issuer_pays_other_token_fixed_price.cpp

Based on the names of two fee handlers `fee_handling_issuer_pays_other_token_amm_price` and `fee_handling_issuer_pays_other_token_fixed_price`, the issuance fee should be deducted from the issuer balance. However, the transfer inline action from the `fee_handling_issuer_pays_other_token_fixed_price_t::handle_fees` method below shows that the fee payer is actually the token contract account.

```
eosio::asset fee_handling_issuer_pays_other_token_fixed_price_t::handle_fees(
    eosio::asset const &quantity,
    eosio::name const &receiver_name,
    double const fee_percentage,
    double const price,
    eosio::extended_asset const &minimum_fee_size
) {
    eosio::check(is_account(receiver_name), "The fee receiver does not exist: '" +
receiver_name.to_string() + "'." );

    //calculate the fee and perform a transfer
    double token_fee = fee_percentage * static_cast<double>(quantity.amount) * pow(10, -
quantity.symbol.precision());
    double fee_token_quantity = token_fee * price * pow(10,
minimum_fee_size.quantity.symbol.precision());
    eosio::asset fee{static_cast<int64_t>(fee_token_quantity),
minimum_fee_size.quantity.symbol};
    //perform a transfer from the self to the receiver
    eosio::action(eosio::permission_level( global::get_self(), "active"_n ),
minimum_fee_size.contract, "transfer"_n, std::tuple( global::get_self(), receiver_name,
fee, "Fee from token: " + quantity.symbol.code().to_string())).send();

    //check that the fee is not less than the minimum fee size
    eosio::check(fee.amount >= minimum_fee_size.quantity.amount, "The fee is lower than the
minimum fee size for receiver '" + receiver_name.to_string() + "'. Min fee: '" +
minimum_fee_size.quantity.to_string() + "', calculated fee: '" + fee.to_string() + "'.");

    eosio::asset empty_asset{0, quantity.symbol};
    return empty_asset;
}
```

UPDATES

- *Jul 15, 2022*: This is not a problem, as the issuer account and the contract account are always the same.

2.2.9. `occupied_space` should be less than or equal to `total_space` LOW

Affected files:

- token-realestate/src/security_checks.cpp

In the `addoccinfo` function, the `occupied_space` should be less than or equal to `total_space`.

```
void addoccinfo(double const total_space,
                double const occupied_space)
{
  auth_and_freeze_check( global::get_self(), 0, "addoccinfo"_n );
  guarantee_positive_quantity("Total space", total_space);
  guarantee_non_negative_quantity("Occupied space", occupied_space);
  eosio::check(occupied_space / total_space < 1, "The occupied space must be lower than the
total space.");
}
```

UPDATES

- *Jul 15, 2022*: This issue has been acknowledged and fixed by Chintai team.

2.2.10. Incomplete code in `token.cpp` INFORMATIVE

Affected files:

- token/src/token.cpp

In the `token.cpp` file, we found a warning directive as below shows that some code would need to be done here. Consider completing it.

```
void token::freezealltok(
    eosio::name const &user,
    eosio::symbol_code const &symbol
) {
    // ...
    vesting_funds vesting_table( global::get_self(), user.value );
    while (vesting_table.begin() != vesting_table.end())
    {
      vesting_table.erase( vesting_table.begin() );
    }
    #warning "The order entries will also need to be deleted when this occurs, do not
forget to code this in"
}
```

UPDATES

- *Jul 15, 2022*: This issue has been acknowledged by Chintai team. This warning is for the advanced exchange, which has not been coded out yet.

2.2.11. Misleading name of `stats_table_empty` variable INFORMATIVE

Affected files:

- trade-amm/src/security_checks.cpp

The name of the `stats_table_empty` here is misleading, it should be `stats_table_not_empty`.

```
void on_transfer(eosio::name const &from,
                 eosio::name const &to,
                 eosio::asset const &quantity,
                 std::string const &memo,
                  eosio::name const &first_receiver)
{
  // ...
  stats stats_table( global::get_self(), quantity.symbol.code().raw() );
  bool stats_table_empty = stats_table.begin() != stats_table.end(); // MISLEADING NAME
  bool memo_is_liquidity = strcmp(memo.c_str(), "liquidity") == 0;
  bool symbol_is_relay_symbol = eosio::extended_symbol{quantity.symbol, first_receiver} ==
relay_itr->get_liquidity().get_extended_symbol();

  if (memo_is_liquidity)
  {
    if(!stats_table_empty)
    {
      if (symbol_is_relay_symbol)
      {
        guarantee_non_frozen(0, "addrelayliq"_n);
      }
    }
    else
    {
      eosio::extended_symbol symbol =
amm::get_nonLP_symbol(eosio::extended_symbol{quantity.symbol, first_receiver});
      bool symbol_is_relay_LP_symbol = symbol == relay_itr-
>get_liquidity().get_extended_symbol();
      if (stats_table_empty && symbol_is_relay_LP_symbol)
      {
        guarantee_non_frozen(0, "remrelayliq"_n);
      }
    }
  }
  // ...
}
```

UPDATES

- *Jul 15, 2022*: This issue has been acknowledged and fixed by Chintai team.

2.2.12. Unused variable in `token::add_user_balance` method INFORMATIVE

Affected files:

- token/src/token.cpp

The `token::add_user_balance` method in the `token` contract declare an unused variable `empty_asset` at the last statement.

```
void token::add_user_balance(eosio::name const &owner,
                             eosio::asset const &value)
{
    accounts accounts_table( global::get_self(), owner.value );
    auto to = accounts_table.find( value.symbol.code().raw() );
    if( to == accounts_table.end() ) {
        eosio::asset empty_asset{0, value.symbol};
        accounts_table.emplace( global::get_self(), [&]( auto& entry ){
                                entry.set_liquid( value );
                                entry.set_secured( empty_asset );
                                entry.set_releasing( empty_asset );
                                entry.set_in_order( empty_asset );
                                entry.set_vesting( empty_asset );
                                entry.set_frozen( empty_asset );
                                });
    } else {
        accounts_table.modify( to, eosio::same_payer, [&]( auto& entry ) {
                                entry.set_liquid( entry.get_liquid() + value );
                                });
    }
    eosio::asset empty_asset = eosio::asset{0, value.symbol}; // UNSUED VARIABLE
}
```

UPDATES

- *Jul 15, 2022*: This issue has been acknowledged and fixed by Chintai team.

2.2.13. Never-free allocations INFORMATIVE

Affected files:

- token/src/fee_handling_factory.cpp
- token/src/buyback_factory.cpp
- token-debt/src/paybackloan_factory.cpp

In the `token::get_issuance_fees` method, we found some heap memory allocations which are not frozen after being used. This may not affect the program flow too much in this case, however, it's still being considered bad practice.

```
eosio::asset token::get_issuance_fees(eosio::asset const &quantity)
{
    // ...
    if (chintai_itr != feetable.end())
    {
        fee_handling_t *factory = fee_handling_factory::get(
            quantity.symbol,
            chintai_itr->get_minimum_fee_size(),
            chintai_itr->get_price(),
            chintai_itr->get_issuer_pays()
        );
        eosio::asset chintai_fee = factory->handle_fees(
            quantity,
            chintai_itr->get_receiver_name(),
            chintai_itr->get_fee_percentage(),
            chintai_itr->get_price(),
            chintai_itr->get_minimum_fee_size()
        );
        fee_quantity += chintai_fee;
    }
    // ...
}

fee_handling_t *fee_handling_factory::get(
    eosio::symbol const &native_token,
    eosio::extended_asset const &minimum_fee_size,
    double const price,
    bool issuer_pays
) {
    bool price_zero = false;
    if(price <= 0 + std::numeric_limits<double>::epsilon()
        && price >= 0 - std::numeric_limits<double>::epsilon()
    ) {
      price_zero = true;
    }

    fee_handling_t *fee_handler;
    if( !issuer_pays && minimum_fee_size.quantity.symbol == native_token
        && minimum_fee_size.contract == global::get_self() && price_zero
    ) {
      fee_handler = new fee_handling_recipient_pays_t();
    }
    // ...
    return fee_handler;
}
```

UPDATES

- *Jul 15, 2022*: This issue has been acknowledged by Chintai team.

2.2.14. Non-constant `max_supply` INFORMATIVE

Affected files:

- token/src/token.cpp

The `max_supply` is created once on token creation:

```
void token::create(eosio::name  const &issuer,
                     eosio::asset  const &maximum_supply)
{
    security_checks::create(maximum_supply);
    stats statstable( global::get_self(), maximum_supply.symbol.code().raw() );

    statstable.emplace( global::get_self(), [&]( auto& entry ) {
        entry.set_supply( eosio::asset{0, maximum_supply.symbol} );
        entry.set_max_supply( maximum_supply );
        entry.set_issuer ( issuer );
        entry.set_liquid( eosio::asset{0, maximum_supply.symbol} );
        entry.set_secured( eosio::asset{0, maximum_supply.symbol} );
        entry.set_releasing( eosio::asset{0, maximum_supply.symbol} );
        entry.set_in_order( eosio::asset{0, maximum_supply.symbol} );
        entry.set_vesting( eosio::asset{0, maximum_supply.symbol} );
        entry.set_frozen( eosio::asset{0, maximum_supply.symbol} );
    });
}
```

And is always reduced on burn:

```
void token::burn(eosio::name const &owner,
               eosio::asset const &quantity,
               std::string const &memo)
{
    security_checks::burn(owner, quantity);
    stats statstable( global::get_self(), quantity.symbol.code().raw() );
    auto itr = statstable.find(quantity.symbol.code().raw());

    statstable.modify( itr, eosio::same_payer, [&]( auto & entry ) {
       entry.set_supply( entry.get_supply() - quantity );
       entry.set_max_supply( entry.get_max_supply() - quantity );
    });
    sub_balance( owner, quantity );
}
```

This behavior is not common comparing to current existing tokens, but there's no standard on it. So this is a reference-only issue.

UPDATES

- *Jul 15, 2022*: This issue has been acknowledged and fixed by Chintai team.

2.2.15. Redundant code in `fee_checks` function INFORMATIVE

Affected files:

- token/src/security_checks.cpp

```
void fee_checks(eosio::symbol const &native_token,
                eosio::name const &receiver_type,
                eosio::name const &receiver_name,
                double const fee_percentage,
                double const price,
                eosio::extended_asset const &minimum_fee_size,
                bool const issuer_pays)
{
  // ...
  const bool fee_paid_in_another_token_with_amm_price = issuer_pays &&
(minimum_fee_size.quantity.symbol != native_token || minimum_fee_size.contract !=
global::get_self()) && price == 0;

  eosio::check(!receiver_pays_with_non_native_token, "When the receiver is paying, the fee
must be taken in the native token.");
  eosio::check(!native_token_fee_price_non_zero, "When the fee is taken in the native
token, price must be set to 0.");

  if ( fee_paid_in_another_token_with_amm_price )
  {
    amm_property_check();
    if (receiver_type == "issuer"_n)
    {
      eosio::check( !issuer_pays, "You cannot set a fee for the issuer when the issuer pays
for the fee.");
    }
  }
}
```

In the `fee_checks` function above, we can see that if the `fee_paid_in_another_token_with_amm_price` variable is `true`, then the `issuer_pays` variable will also be `true`. So we don't need to check the `issuer_pays` variable again. Instead of that, this function should require that the `receiver_type` must not be the `issuer`.

```
if ( fee_paid_in_another_token_with_amm_price )
{
  amm_property_check();
  eosio::check(receiver_type != "issuer"_n, "You cannot set a fee for the issuer when the
issuer pays for the fee.");
}
```

UPDATES

- *Jul 15, 2022*: This issue has been acknowledged and fixed by Chintai team.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*June 17, 2022*	Private Report	Verichains Lab
1.1	*June 22, 2022*	Private Report	Verichains Lab
1.2	*July 15, 2022*	Public Report	Verichains Lab

Table 2. Report versions history