| Security Classification | Open in the Project Group |
| --- | --- |

SENIOR

# Smart Contract Audit Report

## athenastoken


Test Engineer: QI DAI

KNOWNSEC

# Release notes

| Revised one | Revise the content | The revision | The version | review |
|---|---|---|---|---|
| **Qi Dai** | Written document | **2019/06/21** | V1.0 | **Qi Dai** |

# Document information

| The document name | Document | The document number | Level of |
|---|---|---|---|
| **Smart contract audit report** | V1.0 | 【 SLKJ - DMSJ - 20190621.】 | Project team disclosure |

# Copyright statement

# directory

# 1. review

The effective test period of this report is from June 20, 2019 to June 21, 2019, during which the security and normality of smart contract codes are audited and used as the statistical basis for the report.

In this test,Knownsec engineers conducted a comprehensive analysis of common vulnerabilities of intelligent contracts (see chapter 3) and found that there were logic design problems in the contract code, which was comprehensively assessed as security.

**The result of intelligent contract security audit: pass**

Since this test is conducted in a non-production environment, all codes are the latest backup, and the test process is communicated with relevant interface personnel, and relevant test operations are conducted under controllable operational risks to avoid production and operation risks and code safety risks in the test process.

**Target information of this test:**

| The name of the module | |
|---|---|
| The name of the Token | athenastoken |
| Code type | Issue token codes |
| Contract address | https://eospark.com/account/athenastoken |
| The link address | https://eospark.com/account/athenastoken |
| Code | C++ |

| language | |
|---|---|

## Information of testers for this project:

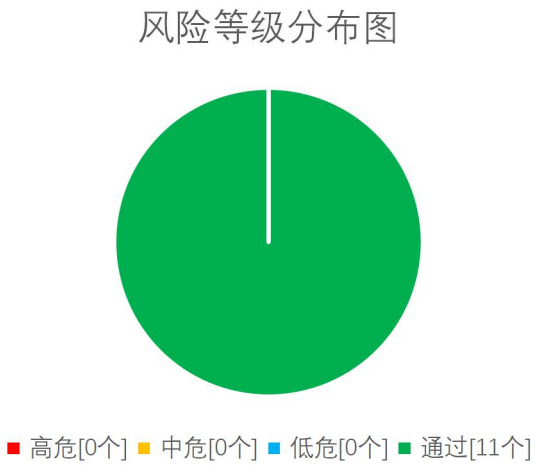
| The name | Email/contact information | position |
|---|---|---|
| **Qi Dai** | daiq@knownsec.com | Safety engineer |

# 2. Code vulnerability analysis

## 2.1. Vulnerability grade distribution

The vulnerability risk is counted by grade:

| Statistical table of vulnerability risk levels | | | |
|---|---|---|---|
| At high risk of | In a crisis | Low risk | Pass |
| 0 | 0 | 0 | 11 |



## 2.2. Summary of audit results

| The audit results | | | |
|---|---|---|---|
| Test project | The test content | state | describe |
| Intelligent contract | Coding normative detection | Pass | Check the security of coding information leakage, type conversion, etc |
| | Numerical overflow detection | Pass | Check balance using security, asset class overflow validation |
| | Limit of authority detection | Pass | Check access control for each operation |
| | API function check | Pass | The types of arguments to EOS API functions. Whether the corresponding parameter type input range is detected in the process of parameter passing |
| | Routine code risk detection | Pass | Check whether the database API usage is lax |
| | Logical design test | Pass | Check the code for security issues related to the business design |
| | Variable initialization risk detection | Pass | Check the code for defects in variable initialization |
| | Apply check | Pass | Check whether each action and code meets the association requirements when handling contract calls. |
| | Transfer false notice detection | Pass | Check the smart contract code for false notification vulnerabilities |
| | Random number detection | Pass | Check that random number generation algorithms |

| | | | |
|---|---|---|---|
| | | | do not introduce controllable or predictable seeds |
| | **Rollback attack detection** | Pass | Check for rollback vulnerabilities |

# 3. Code audit results analysis

## 3.1. Coding normative detection [safety]

Based on the official EOS example of eosio.token, the code normalization of token contracts is reviewed, as follows:

1. In the contract, the asset data structure provided by the government is used to describe the token, and the arithmetic operation of token is also completed by using asset.

```
void transfer(name from, name to, asset quantity, string memo);
```

2. Be sure to check the return value when using the find function of multi_index.

```
302         auto playerLookup = _playergames.find(from.value);
303         if (playerLookup != _playergames.end()){ //game exists before
304             auto currentgame = _playergames.get(from.value);
305             eosio_assert(((currentgame.state == STATE_DRAW) || (currentgame.sta
306             _playergames.erase(playerLookup);
307         }
```

3. All input is checked for validity by assertion, and the type and size of the parameters are checked before the API function is called.

```
341 void v_poker::drawcards(const name account, vector<uint8_t> cardschoice) //0
342 {
343     require_auth(account); //check player account is valid
344
345     eosio_assert(cardschoice.size()==5, "Invalid card choices");
```

* this check is annotated in the contract source code, not as a vulnerability.

Detection process: after detection, the application has detected the return value.

```
    stats statstable( _self, sym.code().raw() );
    auto existing = statstable.find( sym.code().raw() );
    eosio_assert( existing == statstable.end(), "token with symbol alr
eady exists" );

    statstable.emplace( _self, [&]( auto& s ) {
       s.supply.symbol = maximum_supply.symbol;
       s.max_supply    = maximum_supply;
       s.issuer        = issuer;
    });
}
```

Test result: safe

## 3.2. Overflow detection

### 3.2.1. Numerical overflow detection [safety]

Check whether the contract exists and fail to check the security when calculating the token value.In the wrong operation easy to produce integer overflow error, may lead to the token amount to zero or even become a negative number of serious consequences!Risk. Sample problem code:

A sample:

```
void token::add_balance( account_name owner, int64_t value, symbol_name sym_name, account_name ram_payer )
{
   accounts to_acnts( _self, owner );
   auto to = to_acnts.find(sym_name);
   if( to == to_acnts.end() ) {
      to_acnts.emplace( ram_payer, [&]( auto& a ){
        a.balance = value;
      });
   } else {
      to_acnts.modify( to, 0, [&]( auto& a ) {
        a.balance += value;
      });
   }
}
```

The above code has integer overflow problem in add_balance (increase asset) function. When the token is not whitlisted, after using modify method to modify the project, the transaction of token quantity is not detected in the modification of token quantity, resulting in the risk of overflow of numerical operation.

Example 2:

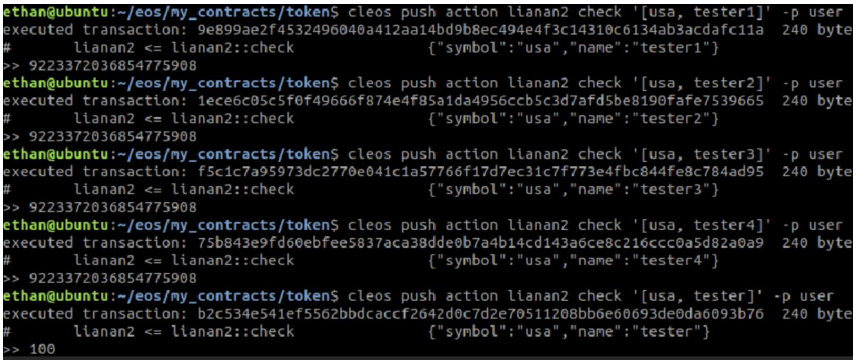
Transfer function source code and transfer money to 4 people at the same time:

```
typedef  struct acnts {
    account_name    name0;
    account_name    name1;
    account_name    name2;
    account_name    name3;
}account_names;

void transfer(symbol_name symbol, account_name from, account_names to, uint64_t balance) {
    print("transfer");
    require_auth(from);
    account fromaccount;

    require_recipient(from);
    require_recipient(to.name0);
    require_recipient(to.name1);
    require_recipient(to.name2);
    require_recipient(to.name3);

    eosio_assert(is_balance_within_range(balance), "invalid balance");
    eosio_assert(balance > 0, "must transfer positive balance");

    uint64_t amount = balance * 4;

    int itr = db_find_i64(_self, symbol, N(table), from);
    eosio_assert(itr >= 0, "Sub--wrong name");
    db_get_i64(itr, &fromaccount, sizeof(account));
    eosio_assert(fromaccount.balance >= amount, "overdrawn balance");

    sub_balance(symbol, from, amount);

    add_balance(symbol, to.name0, balance);
    add_balance(symbol, to.name1, balance);
    add_balance(symbol, to.name2, balance);
    add_balance(symbol, to.name3, balance);
}
```

Suppose the attacker calls the transfer function of the contract to transfer money to four people at the same time and sets the value of the balance parameter to ($2^{63}$), the function call process is shown in the figure below:

```
$ cleos push action lianan2 transfer '[usa,tester,[tester1 tester2 tester3 tester4], 9223372036854775808
b4155c9b36d10a34adb7a4bd40c33fdc3c421892689b1a409ac4c86ae0fea  280 bytes  121856 cycles
          {"symbol":"usa","from":"tester","to":{"name0":"tester1","name1":"tester2","name2":"tester3","n


          {"symbol":"usa","from":"tester","to":{"name0":"tester1","name1":"tester2","name2":"tester3","n

          {"symbol":"usa","from":"tester","to":{"name0":"tester1","name1":"tester2","name2":"tester3","n

          {"symbol":"usa","from":"tester","to":{"name0":"tester1","name1":"tester2","name2":"tester3","n

          {"symbol":"usa","from":"tester","to":{"name0":"tester1","name1":"tester2","name2":"tester3","n

          {"symbol":"usa","from":"tester","to":{"name0":"tester1","name1":"tester2","name2":"tester3","n
```

Next, a query of the above address balance shows that the balance of the transferee (tester) (100) does not decrease, while the account balance of the receiver (tester1, tester2, tester3, tester4) generates a very large amount due to the overflow of the amount variable ($2)^{63}$), as shown in figure 3:




Balance is the uint64 data type when the value is $2^{63}$, because it is smaller than the maximum value of uint64, the overflow boundary check of balance is bypassed.However, when the amount=balance*4 is calculated, the amount overflow occurs, making the value equal to 0. Since the amount bypasses the check that the reduced value is larger than the reduced value, it enables the balance of the transferee to obtain a very large value (2) without consuming the balance of the transferee$^{63}$)

Solutions:

Contract developers use the intelligent contract programming Math API interface provided by the EOS blockchain platform to prevent this type of overflow vulnerability.For example, contract developers can convert uint data into doule data first, and then use double_add, double_mult and other functions in Math API provided by EOS blockchain platform to perform operations, and finally convert the calculated results into uint data output.

Detection process: no overflow point was found after detection

```
23
24  void token::add_balance( name owner, asset value, name ram_payer )
25  {
26     accounts to_acnts( _self, owner.value );
27     auto to = to_acnts.find( value.symbol.code().raw() );
28     if( to == to_acnts.end() ) {
29        to_acnts.emplace( ram_payer, [&]( auto& a ){
30          a.balance = value;
31        });
32     } else {
33        to_acnts.modify( to, same_payer, [&]( auto& a ) {
34          a.balance += value;
35        });
36     }
37  }
```

Test result: safe

Repair suggestion: none

## 3.2.2. Asset class overflow detection [safety]

An asset is a structure provided in an EOS official header file to represent a monetary asset, such as an official currency EOS or other monetary unit published by itself.When using asset for multiplication (operator *=), bugs in the official code invalidated the overflow detection.As a result, there is a risk of overflow if the developer USES asset multiplication in the smart contract.

Question exists in the code: contracts/eosiolib/asset. The HPP detection reference official whether the header file has been updated:

```
      asset& operator*=( int64_t a ) {
        eosio_assert( a == 0 || (amount * a) / a == amount, "multiplication overflow
or underflow" );                                          <== (1)
        eosio_assert( -max_amount <= amount, "multiplication underflow" );  <= (2)
        eosio_assert( amount <= max_amount,  "multiplication overflow" );   <= (3)
        amount *= a;
        return *this;
     }
```

Vulnerability analysis for details please see: https://paper.seebug.org/658/

Detection procedure: no multiplication is used in this contract.

Test result: safe

Repair suggestion: none

## 3.3. Limit of authority detection [safety]

Check whether the contract is correct to check the access rights and method calls, and whether there is a logical loophole caused by the lack of rigorous checking.

Sample code:

```
void token::issue( account_name to, asset quantity, string memo )
{
    /* Other Code */
    statstable.modify( st, 0, [&]( auto& s ) {
       s.supply += quantity;
    });
    /* Other Code */
}

void token::transfer( account_name from,account_name to,asset quantity)
{
    /* Other Code */
    sub_balance( from, quantity, st );
    add_balance( to, quantity, st, from );
}

void token::sub_balance( account_name owner, asset value, const currency_stat& st ) {
    /*Other Code*/
   eosio_assert( !st.can_freeze || !from.frozen, "account is frozen by issuer" );
   eosio_assert( !st.can_freeze || !st.is_frozen, "all transfers are frozen by issuer" );
   eosio_assert( !st.enforce_whitelist || from.whitelist, "account is not white listed" );
    /*Other Code*/
}
```

The sample scrip contract sets up the function of freezing the account and scrip. However, the user only puts the code of checking "freezing" in the transfer function, so that the issue is not affected by the "freezing" state and can issue more scrip at will.

Detection process: the application USES the corresponding permission check

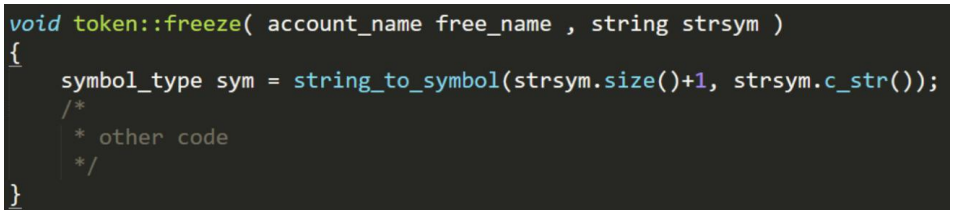
```
void token::close( name owner, const symbol& symbol )
{
   require_auth( owner );
   accounts acnts( _self, owner.value );
   auto it = acnts.find( symbol.code().raw() );
   eosio_assert( it != acnts.end(), "Balance row already deleted or ne -
ver existed. Action won't have any effect." );
   eosio_assert( it->balance.amount == 0, "Cannot close because the ba -
lance is not zero." );
   acnts.erase( it );
}
```

Test result: safe

Repair suggestion: none

## 3.4. Inspection of API function [safety]

Check the parameter types of the EOS API functions.Whether the corresponding parameter type input range is detected in the process of parameter passing.




For example, string_to_symbol(uint8_t, const char *), the integer variable passed in by the first parameter needs to be less than 256. If the input is not checked before using the API, the integer overflow may occur, resulting in the operation of the wrong type of token, with serious consequences.

Test result: safe

Repair suggestion: none

## 3.5. Routine code risk detection [safety]

The use of database apis is lax, such as get and find provided in

multi_index.Where, get will check whether the data is successfully queried, and if the data is not found, the assertion exits, while find will not check the data query, requiring users to determine by themselves. Direct use of the pointer without judgment will lead to problems in the use of Pointers.

```
stats statstable( _self, sym_name );
auto existing = statstable.find( sym_name );
const auto& st = *existing;
```

Detection process:

After testing, eosio_assert was used for verification.

Test result: safe

## 3.6. Logical design detection [safety]

Check the code for security issues related to the business design

Detection process: this problem is not involved in the application after detection.

Test result: safe

## 3.7. * variable initialization risk detection [safety]

Reference:

1, do not initialize the variable in the function first, after the next call, the local variable is still retained the last execution result!!!(codeBlocks version, this has happened before) ***

2. The calculation result is incorrect

3. The program logic is inconsistent with the expected logic

4, for the use of function Pointers in the program, will cause process crash.

5, write to the hard disk data produced an error

6, serious may lead to system or even hardware failure

Detection process: the initial value of each configuration variable is specified in the configuration file, and it is called as specified

Test result: safe

Repair suggestion: none

## 3.8. Apply check [safety]

When handling contract calls, ensure that each action and code meets the association requirements.

```
650  // extend from EOSIO_ABI
651  #define EOSIO_ABI_EX( TYPE, MEMBERS )
652  extern "C" {
653     void apply( uint64_t receiver, uint64_t code, uint64_t action ) {
654        auto self = receiver;
655        if( action == N(onerror)) {
656           /* onerror is only valid if it is for the "eosio" code account and authorized by "eosio"'s "active permission */
657           eosio_assert(code == N(eosio), "onerror action's are only valid from the \"eosio\" system account");
658        }
659        if( code == self || code == N(eosio.token) || action == N(onerror) ) {
660           TYPE thiscontract( self );
661           switch( action ) {
662              EOSIO_API( TYPE, MEMBERS )
663           }
664           /* does not allow destructor of thiscontract to run: eosio_exit(0); */
665        }
666     }
667  }
668
669  EOSIO_ABI_EX(eosio::charity, (hi)(transfer))
670
```

Detection process:

The application does not involve the apply call

Test result: safe

Repair suggestion: none

## 3.9. Detection of transfer false notice [safety]

When handling contract calls, ensure that each action and code meets the association requirements.

When handling notifications that are triggered by require_recipient, ensure that transfer.to is _self.

```
652  void transfer(uint64_t sender, uint64_t receiver) {
653
654      auto transfer_data = unpack_action_data<st_transfer>();
655
656      if (transfer_data.from == _self || transfer_data.from == N(eosbetcasino)){
657          return;
658      }
659
660      eosio_assert( transfer_data.quantity.is_valid(), "Invalid asset");
661  }
```

Detection process:

There is no risk that an action is invoked on a contract that does not involve outside apply.

Test result: safe

Repair suggestion: none

## 3.10. Random number detection [safety]

Random number generation algorithms should not introduce controllable or predictable seeds

```
// source code: https://github.com/loveblockchain/eosdice/blob/3c6f9bac570cac236302e94b62432b73f6e74c3b/eos
uint8_t random(account_name name, uint64_t game_id)
{
    auto eos_token = eosio::token(N(eosio.token));
    asset pool_eos = eos_token.get_balance(_self, symbol_type(S(4, EOS)).name());
    asset ram_eos = eos_token.get_balance(N(eosio.ram), symbol_type(S(4, EOS)).name());
    asset betdiceadmin_eos = eos_token.get_balance(N(betdiceadmin), symbol_type(S(4, EOS)).name());
    asset newdexpocket_eos = eos_token.get_balance(N(newdexpocket), symbol_type(S(4, EOS)).name());
    asset chintailease_eos = eos_token.get_balance(N(chintailease), symbol_type(S(4, EOS)).name());
    asset eosbiggame44_eos = eos_token.get_balance(N(eosbiggame44), symbol_type(S(4, EOS)).name());
    asset total_eos = asset(0, EOS_SYMBOL);
    //攻击者可通过inline_action改变余额total_eos，从而控制结果
    total_eos = pool_eos + ram_eos + betdiceadmin_eos + newdexpocket_eos + chintailease_eos + eosbiggame44_
    auto mixd = tapos_block_prefix() * tapos_block_num() + name + game_id - current_time() + total_eos.amou
    const char *mixedChar = reinterpret_cast<const char *>(&mixd);

    checksum256 result;
    sha256((char *)mixedChar, sizeof(mixedChar), &result);

    uint64_t random_num = *(uint64_t *)(&result.hash[0]) + *(uint64_t *)(&result.hash[8]) + *(uint64_t *)(&
    return (uint8_t)(random_num % 100 + 1);
}
```

Testing process: random Numbers are not used in this contract.

Test result: safe

Repair suggestion: none

## 3.11. Rollback attack detection [safety]

- Method 1: detect execution results in transactions (such as collection amount, account balance, table records, random number calculation results, etc.), and call eosio_assert when the results meet certain conditions, so that the current transaction fails to roll back.
- Trick 2: use the super node blacklist account to initiate a transaction and trick the ordinary node into responding, but the transaction will not be packaged.

Such as:

- Game betting then lottery and transfer, malicious contract can be detected through inline_action balance increase, so as to roll back the failed lottery
- Game type game betting then write lottery results into the table, malicious contract can be recorded through inline_action detection table, so as to roll back the failed lottery
- Game type game lottery results and lottery Numbers in the game are associated, malicious contract can be launched at the same time through a number of small amount of betting transactions and a large amount of betting transactions, when receiving a small amount of money to roll back the transaction, so as to achieve the lottery number can be won "transfer" to the purpose of large amount of betting.
- Game game lottery and betting transactions are not associated, the attacker can use the blacklist account or malicious contract rollback bet

**Detection process: from the source code analysis, the contract does not involve betting lottery, does not involve the detection**

## Test result: safe

# 4. Appendix A: contract code

Source of this test code:

*# code*

```
/ * *
  * @ file
  * @copyright defined in eos/ license.txt
  * /

# include < eosio. Token/eosio. Token. HPP >

The namespace eosio {

Void token: : create (name issuer,
                    Asset maximum_supply)
{
    Require_auth (love);

    Auto sym = maximum_supply. Symbol;
    Eosio_assert (sym.is_valid(), "invalid symbol name");
    Eosio_assert (maximum_supply is_valid (), "the invalid supply");
    Eosio_assert (maximum_supply. Amount >, "max-supply must be positive");

    Stats statstable(_self, sym.code().raw());
    Auto existing = statstable.find(sym.code().raw()); auto existing =
statstable.find(sym.code().raw());
    Eosio_assert (existing == statstable. End (), "token with symbol already exists");

    Statstable. Emplace (_self, [&](auto& s) {
       S.s upply. Symbol = maximum_supply. Symbol;
       S.m ax_supply = maximum_supply;
       S.i ssuer = issuer;
    });
}


Void token::issue(name to, asset quantity, string memo)
{
    Auto sym = quantity. The symbol;
    Eosio_assert (sym.is_valid(), "invalid symbol name");
    Eosio_assert (memo. Size () <= 256, "memo has more than 256 bytes");

    Stats statstable(_self, sym.code().raw());
    Auto existing = statstable.find(sym.code().raw()); auto existing =
statstable.find(sym.code().raw());
    Eosio_assert (existing! = statstable. End (), "token with symbol does not exist, create
token before issue");
    Const auto& st = *existing;

    Require_auth (st. issuer);
    Eosio_assert (quantity. Is_valid (), "invalid quantity");
    Eosio_assert (quantity. Amount > 0, "must issue positive quantity");

    Symbol == st.supply. Symbol, "symbol precision mismatch");
    Eosio_assert (quantity amount < = st. max_supply. Amount - st. supply. The amount of
"quantity exceeds the available supply");

    Statstaby.modify (st, same_payer, [&](auto& s) {
       S.s upply + = quantity;
    });

    Add_balance (st. issuer, quantity, st. issuer);

    If (to! = st. issuer) {
      SEND_INLINE_ACTION (* this, transfer, {{st. issuer, "active" _n}},
                          {st. issuer, to quantity, memo}
      );
    }
}

Void token::retire(asset quantity, string memo)
```

```
{
    Auto sym = quantity. The symbol;
    Eosio_assert (sym.is_valid(), "invalid symbol name");
    Eosio_assert (memo. Size () <= 256, "memo has more than 256 bytes");

    Stats statstable(_self, sym.code().raw());
    Auto existing = statstable.find(sym.code().raw()); auto existing =
statstable.find(sym.code().raw());
    Eosio_assert (existing! = statstable. End (), "token with symbol does not exist");
    Const auto& st = *existing;

    Require_auth (st. issuer);
    Eosio_assert (quantity. Is_valid (), "invalid quantity");
    Eosio_assert (quantity. Amount >, "must retire positive quantity");

    Symbol == st.supply. Symbol, "symbol precision mismatch");

    Statstaby.modify (st, same_payer, [&](auto& s) {
       S.s upply - = quantity;
    });

    Sub_balance (st. issuer, quantity);
}

Void token: : transfer (the name from,
                  Name the to,
                  Asset quantity,
                  String memo)
{
    Eosio_assert (the from! = to, "cannot transfer to self");
    Require_auth (from);
    Eosio_assert (is_account(to), "to account does not exist");
    Auto sym = quantity. The symbol. The code ();
    Stats statstable(_self, sym.raw());
    Const auto& st = statstable. Get (sym.raw());

    Require_recipient (from);
    Require_recipient (to);

    Eosio_assert (quantity. Is_valid (), "invalid quantity");
    Eosio_assert (quantity. Amount >, "must transfer positive quantity");
    Symbol == st.supply. Symbol, "symbol precision mismatch");
    Eosio_assert (memo. Size () <= 256, "memo has more than 256 bytes");

    Auto payer = has_auth(to)?To: the from;

    Sub_balance (the from, quantity);
    Add_balance (to, quantity, payer);
}

Void token::sub_balance(name owner, asset value) {
   Accounts from_acnts(_self, owner.value);

   Const auto& from = from_acnts. Get (value.symbol. Code (). Raw (), "no balance object
found");
   Eosio_assert (from. Balance. Amount >= value.amount, "overdrawn balance");

   From_acnts. Modify (from, owner, [&](auto& a) {
        A. alance - = value;
     });
}

Void token::add_balance(name owner, asset value, name ram_payer)
{
   Accounts to_acnts(_self, owner.value);
   Auto to = to_acnts.find(value.symbol. Code ().raw());
   If (to == to_acnts.end()) {
     To_acnts. Emplace (ram_payer, [&](auto& a){
       A. alance = value;
     });
   } else {
     To_acnts. Modify (to, same_payer, [&](auto& a) {
       A. alance + = value;
     });
   }
}
```

```
Void token::open(name owner, const symbol& symbol, name ram_payer)
{
   Require_auth (ram_payer);

   Auto sym_code_raw = symbol. The code (). The raw ();

   Stats statstable(_self, sym_code_raw);
   Const auto& st = statstable. Get (sym_code_raw, "symbol does not exist");
   Symbol == symbol, "symbol precision mismatch";

   Accounts acnts(_self, owner.value);
   Auto it = acnts. Find (sym_code_raw);
   If (it == acnts. End ()) {
      Acnts. Emplace (ram_payer, [&](auto& a){
        A. alance = asset {0, symbol};
      });
   }
}

Void token::close(name owner, const symbol& symbol)
{
   Require_auth (the owner);
   Accounts acnts(_self, owner.value);
   Auto it = acnts. Find (symbol. Code (). Raw ());
   Eosio_assert (it! End (), "Balance row already deleted or never facilities. Action won't
have any effect.")
   Eosio_assert (it->balance. Amount == 0, "Cannot close because the balance is not zero.");
   Acnts. Erase (it);
}

} / / / namespace eosio

EOSIO_DISPATCH (eosio: : token, (create) (issue) (transfer) (open) (close) (retire))
}
```

# 5. Appendix B: vulnerability risk rating criteria

| *Intelligent contract vulnerability rating criteria* | |
|---|---|
| **Vulnerability rating** | **Vulnerability rating instructions** |
| High risk vulnerabilities | Loopholes that can directly cause the loss of scrip contracts or users' funds, such as: numerical overflow loopholes that can cause the value of scrip to return to zero, false recharge loopholes that can cause exchange loss of scrip, ETH or re-entry loopholes that can cause the loss of contract accounts, etc.;<br>Loopholes that can cause loss of ownership of token contracts, such as: access control defect of key functions, call injection leads to bypass of access control of key functions, etc.;<br>Vulnerabilities that can cause token contracts to not work properly, such as the denial of service vulnerability caused by sending ETH to a malicious address or the denial of service vulnerability caused by gas exhaustion. |
| In the dangerous holes | High-risk vulnerabilities that require a specific address to trigger, such as numerical overflow vulnerabilities that can be triggered by the owner of a token contract; Access control defects of non-critical functions, logic design defects that cannot cause direct capital loss, etc. |
| Low latent loophole | The vulnerabilities that are difficult to be triggered, the ones that have limited harm after being triggered, such as the numerical overflow vulnerabilities that require a large number of ETH or tokens to trigger, the vulnerabilities that cannot directly benefit the attacker after the numerical overflow is triggered, and the transaction sequence dependency risks that are triggered by specifying high gas, etc. |

KNOWNSEC

[ Advisory telephone ]+86(10)400 060 9587
[ Complaints Hotline ] 13811527185
[ E-mail ] sec@knownsec.com
[ Website ] www.knownsec.com
[ Address ] wangjing soho T3-A15,Chaoyang District, Beijing

Beijing KnownSec Information Technology Co., Ltd.