PUML TOKEN SMART CONTRACT AUDIT

FOR PUML HEALTH AND FITNESS PTE. LTD.

07.05.2019

Made in Germany by Chainsulting.de

Smart Contract Audit PUML Token

Table of Contents

1. Disclaimer

The audit makes no statements or warrantees about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of Puml Health and Fitness Pte. Ltd. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description	Author
0.1 (04.05.2019)	Layout	Y. Heinze
0.5 (06.05.2019)	Automated Security Testing	Y. Heinze
0.7 (07.05.2019)	Manual Security Testing	Y. Heinze
1.0 (07.05.2019)	Summary and Recommendation	Y. Heinze

2. About the Project and Company

Main Company address:	**Token Issuer Company address:**
Pummel Pty Ltd Unit 1 53 Township Drive West Burleigh, QLD 4219 ABN 79610587089	PUML HEALTH AND FITNESS PTE. LTD. 71 UBI CRESCENT #08-02 EXCALIBUR CENTRE SINGAPORE (408571) REG: 201836880D

Website: https://puml.io and https://pummel.fit

Project Overview:

PUML will power the sweat economy and reward healthier lifestyles. They aim to help individuals, institutions and societies achieve their collective health and fitness goals. The PUML Token is based on EOS Blockchain and is giving out to app members as reward. Pummel recently acquired Zippy and now has over 32,000 users in the PUML ecosystem and they have recorded over 380,000 gym visits.

PUML

3. Vulnerability Level

0-Informational severity – A vulnerability that have informational character but is not effecting any of the code.

1-Low severity - A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.

2-Medium severity – A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.

3-High severity – A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.

4-Critical severity – A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.

4. Overview of the audit

The PUML Token is part of the pumlhealthio Smart Contract and this one was audited. All the functions and state variables are well commented using the natspec documentation for the functions which is good to understand quickly how everything is supposed to work. The Token is based on the default eosio.token contract and is well audited by the block.one team.

4.1 Used Code from other Frameworks/Smart Contracts (3th Party)

1. EOSIO.Token
https://github.com/EOSIO/eosio.contracts/tree/master/contracts/eosio.token/include/eosio.token/eosio.token.hpp
https://github.com/EOSIO/eosio.contracts/blob/master/contracts/eosio.token/src/eosio.token.cpp/eosio.token.cpp

2. EOSIO.CDT (Contract Development Toolkit)
https://github.com/EOSIO/eosio.cdt/blob/master/libraries/eosiolib/eosio.hpp

3. EOSIO.CDT (Contract Development Toolkit)
https://github.com/EOSIO/eosio.cdt/blob/master/libraries/eosiolib/asset.hpp

4.2 Tested Contract Files

File	Checksum (SHA256)
pumlhealthio.cpp	4067DB375239922CED9267ED2247C5BEB8F7ADB0A72E9F9AA2192DAE54DCC984
pumlhealthio.hpp	7E982025876208E4C58BA2593F153A5D5BB3AAC28A59BF55F816F152FA14879E

4.3 Contract Specifications (PUML Token)

Language	C++, webassembly
Token Standard	EOS Token
Most Used Framework	eosio.token
Compiler Version	1.8.0-rc1
Burn Function	No
Mint Function	No
Ticker Symbol	PUML
Total Supply	500 000 000
Timestamps used	No

5. Summary of Distribution

https://www.eosx.io/account/pumlhealthio

pumlhealthio - 238,500,000 (previously Pre/Private Sale) goes into main account
pumlcoreteam - 25,000,000
pumlfounders - 25,000,000
pumlfutureos - 75,000,000
pumlsponsors - 5,000,000
pumlpartners - 5,000,000
betterhealth - 126,500,000

6. Test Suite Results (PUML Token)

Cppcheck : nothing found
A tool for static C/C++ code analysis

Flawfinder : nothing found
A simple program that examines C/C++ source code and reports possible security weaknesses (“flaws”) sorted by risk level

PVS-Studio Analyzer : nothing found
A tool for detecting bugs and security weaknesses in the source code of programs, written in C, C++, C# and Java

Result: The analysis was completed successfully. No issues were detected

7. General Summary

A majority of the code was standard, based on eosio.token and as a result, a lot of the code was reviewed before. The audit identified no major security vulnerabilities, at the moment of audit. We noted that a majority of the functions were self-explanatory, and standard documentation tags were included.

8. Deployed Smart Contract

https://bloks.io/transaction/393FBE4623078DBFE76EC9E0341EC1A8BAE684B9B1A029181C5872424F643D79 (PUML Token)

9. Code

pumlhealthio.cpp

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

#include <eosio.token/eosio.token.hpp>

namespace eosio {

void token::create( name   issuer,
              asset  maximum_supply )
{
   require_auth( _self );

   auto sym = maximum_supply.symbol;
   check( sym.is_valid(), "invalid symbol name" );
   check( maximum_supply.is_valid(), "invalid supply");
   check( maximum_supply.amount > 0, "max-supply must be positive");

   stats statstable( _self, sym.code().raw() );
   auto existing = statstable.find( sym.code().raw() );
   check( existing == statstable.end(), "token with symbol already exists" );

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

pumlhealthio.hpp

```
/**
 * @file
 * @copyright defined in eos/LICENSE.txt
 */
#pragma once

#include <eosiolib/asset.hpp>
#include <eosiolib/eosio.hpp>

#include <string>

namespace eosiosystem {
  class system_contract;
}

namespace eosio {

  using std::string;

  class [[eosio::contract("eosio.token")]] token : public contract {
    public:
      using contract::contract;

      [[eosio::action]]
      void create( name   issuer,
              asset  maximum_supply);

      [[eosio::action]]
      void issue( name to, asset quantity, string memo );

      [[eosio::action]]
```

```
void token::issue( name to, asset quantity, string memo )
{
   auto sym = quantity.symbol;
   check( sym.is_valid(), "invalid symbol name" );
   check( memo.size() <= 256, "memo has more than 256 bytes" );

   stats statstable( _self, sym.code().raw() );
   auto existing = statstable.find( sym.code().raw() );
   check( existing != statstable.end(), "token with symbol does not exist, create token before issue" );
   const auto& st = *existing;

   require_auth( st.issuer );
   check( quantity.is_valid(), "invalid quantity" );
   check( quantity.amount > 0, "must issue positive quantity" );

   check( quantity.symbol == st.supply.symbol, "symbol precision mismatch" );
   check( quantity.amount <= st.max_supply.amount - st.supply.amount, "quantity exceeds available
supply");

   statstable.modify( st, same_payer, [&]( auto& s ) {
     s.supply += 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.symbol;
   check( sym.is_valid(), "invalid symbol name" );
   check( memo.size() <= 256, "memo has more than 256 bytes" );

   stats statstable( _self, sym.code().raw() );
   auto existing = statstable.find( sym.code().raw() );
   check( existing != statstable.end(), "token with symbol does not exist" );
```

```
   void retire( asset quantity, string memo );

   [[eosio::action]]
   void transfer( name    from,
           name    to,
           asset   quantity,
           string  memo );

   [[eosio::action]]
   void open( name owner, const symbol& symbol, name ram_payer );

   [[eosio::action]]
   void close( name owner, const symbol& symbol );

   static asset get_supply( name token_contract_account, symbol_code sym_code )
   {
     stats statstable( token_contract_account, sym_code.raw() );
     const auto& st = statstable.get( sym_code.raw() );
     return st.supply;
   }

   static asset get_balance( name token_contract_account, name owner, symbol_code sym_code )
   {
     accounts accountstable( token_contract_account, owner.value );
     const auto& ac = accountstable.get( sym_code.raw() );
     return ac.balance;
   }

   using create_action = eosio::action_wrapper<"create"_n, &token::create>;
   using issue_action = eosio::action_wrapper<"issue"_n, &token::issue>;
   using retire_action = eosio::action_wrapper<"retire"_n, &token::retire>;
   using transfer_action = eosio::action_wrapper<"transfer"_n, &token::transfer>;
   using open_action = eosio::action_wrapper<"open"_n, &token::open>;
   using close_action = eosio::action_wrapper<"close"_n, &token::close>;
private:
   struct [[eosio::table]] account {
     asset    balance;

     uint64_t primary_key()const { return balance.symbol.code().raw(); }
   };
   const auto& st = *existing;

   require_auth( st.issuer );
   check( quantity.is_valid(), "invalid quantity" );
   check( quantity.amount > 0, "must retire positive quantity" );

   check( quantity.symbol == st.supply.symbol, "symbol precision mismatch" );

   statstable.modify( st, same_payer, [&]( auto& s ) {
     s.supply -= quantity;
   });

   sub_balance( st.issuer, quantity );
}

void token::transfer( name    from,
                      name    to,
                      asset   quantity,
                      string  memo )
{
   check( from != to, "cannot transfer to self" );
   require_auth( from );
   check( is_account( to ), "to account does not exist");
   auto sym = quantity.symbol.code();
   stats statstable( _self, sym.raw() );
   const auto& st = statstable.get( sym.raw() );

   require_recipient( from );
   require_recipient( to );

   check( quantity.is_valid(), "invalid quantity" );
   check( quantity.amount > 0, "must transfer positive quantity" );
   check( quantity.symbol == st.supply.symbol, "symbol precision mismatch" );
   check( memo.size() <= 256, "memo has more than 256 bytes" );

   auto payer = has_auth( to ) ? to : from;

   sub_balance( from, quantity );
   add_balance( to, quantity, payer );
}
```

```
      struct [[eosio::table]] currency_stats {
        asset    supply;
        asset    max_supply;
        name     issuer;

        uint64_t primary_key()const { return supply.symbol.code().raw(); }
      };

      typedef eosio::multi_index< "accounts"_n, account > accounts;
      typedef eosio::multi_index< "stat"_n, currency_stats > stats;

      void sub_balance( name owner, asset value );
      void add_balance( name owner, asset value, name ram_payer );
   };

} /// namespace eosio
```

```
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" );
  check( from.balance.amount >= value.amount, "overdrawn balance" );

  from_acnts.modify( from, owner, [&]( auto& a ) {
      a.balance -= 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.balance = value;
    });
  } else {
    to_acnts.modify( to, same_payer, [&]( auto& a ) {
     a.balance += value;
    });
  }
}

void token::open( name owner, const symbol& symbol, name ram_payer )
{
  require_auth( ram_payer );

  auto sym_code_raw = symbol.code().raw();

  stats statstable( _self, sym_code_raw );
  const auto& st = statstable.get( sym_code_raw, "symbol does not exist" );
  check( st.supply.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.balance = asset{0, symbol};
```

```
      });
   }
}

void token::close( name owner, const symbol& symbol )
{
   require_auth( owner );
   accounts acnts( _self, owner.value );
   auto it = acnts.find( symbol.code().raw() );
   check( it != acnts.end(), "Balance row already deleted or never existed. Action won't have any effect."
);
   check( 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) )
```