智能合约安全审计报告

慢雾安全团队于 2019-05-08 日，收到超脑链团队对 ultrain 系统智能合约安全审计申请。如下为本次智能合约安全审计细节及结果：

合约地址：

https://github.com/ultrain-os/ultrain-contracts

合约版本：

commit: 28642754bd4e4880f0816d1eec4d02a6d18a0a92

本次审计项及结果：

（其他未知安全漏洞不包含在本次审计责任范围）

序号	审计大类	审计子类	审计结果
1	溢出审计	-	通过
2	权限控制审计	权限漏洞审计	通过
		权限过大审计	通过
3	安全设计审计	硬编码地址安全	通过
		显现编码安全	通过
		异常校验审计	通过
		类型安全审计	通过
4	性能优化审计	-	通过
5	设计逻辑审计	-	通过
6	拒绝服务审计	-	通过
7	性能优化审计	-	通过
8	设计逻辑审计	-	通过
9	假通知审计	-	通过
10	假错误通知审计	-	通过
11	假币审计	-	通过
12	随机数安全审计	-	通过
13	粉尘攻击安全审计	-	通过

项目结构

```
$ tree contracts
├── CMakeLists.txt
├── hello
│   ├── CMakeLists.txt
│   ├── hello.abi
│   ├── hello.cpp
│   ├── hello.hi_rc.md
│   └── hello_rc.md
├── libc++
│   ├── CMakeLists.txt
│   └── upstream
├── merkle_proof
│   ├── CMakeLists.txt
│   ├── merkle_proof.abi
│   └── merkle_proof.cpp
├── musl
│   ├── CMakeLists.txt
│   └── upstream
├── stltest
│   ├── CMakeLists.txt
│   ├── stltest.abi
│   └── stltest.cpp
├── ultrainio.bank
│   ├── #ultrainio.bank.cpp#
│   ├── CMakeLists.txt
│   ├── ultrainio.bank.abi
│   ├── ultrainio.bank.cpp
│   └── ultrainio.bank.hpp
├── ultrainio.msig
│   ├── CMakeLists.txt
│   ├── README.md
│   ├── ultrainio.msig.abi
│   ├── ultrainio.msig.cpp
│   └── ultrainio.msig.hpp
├── ultrainio.rand
│   ├── CMakeLists.txt
│   ├── lib
│   ├── ultrainio.rand.abi
│   ├── ultrainio.rand.ts
│   ├── ultrainio.rand.wasm.m
```

```
│   └── ultrainio.rand.wast.t
├── ultrainio.sudo
│   ├── CMakeLists.txt
│   ├── README.md
│   ├── ultrainio.sudo.abi
│   ├── ultrainio.sudo.cpp
│   └── ultrainio.sudo.hpp
├── ultrainio.system
│   ├── BlockHeaderExtKey.h
│   ├── CMakeLists.txt
│   ├── CheckPoint.h
│   ├── CommitteeDelta.h
│   ├── CommitteeInfo.h
│   ├── CommitteeSet.h
│   ├── ConfirmPoint.h
│   ├── delegate.cpp
│   ├── native.hpp
│   ├── producer.cpp
│   ├── reward.cpp
│   ├── scheduler.cpp
│   ├── synctransaction.cpp
│   ├── ultrainio.system.abi
│   ├── ultrainio.system.cpp
│   └── ultrainio.system.hpp
├── ultrainio.token
│   ├── CMakeLists.txt
│   ├── ultrainio.token.abi
│   ├── ultrainio.token.cpp
└────── ultrainio.token.hpp
```

(审计范围不包含ultrainlib库)

备注：审计意见及建议见代码注释 **//SlowMist//.....**

审计结果：**通过**

审计编号：0X001905280001

审计日期：2019 年 05 月 28 日

审计团队：慢雾安全团队

总结：此为系统合约，经反馈修正后，综合评估合约无已知风险。

以下针对部分代码问题进行详细分析，分析写于注释处。

delegate.cpp

```
  void system_contract::clearexpirecontract( account_name owner ){
    vector<permission_level> pem = { { owner, N(active) },
                                { N(ultrainio),    N(active) } };
    {
      // clear contract
      ultrainio::transaction trx;
      trx.actions.emplace_back(pem, _self, NEX(setcode), std::make_tuple(owner, 0, 0, bytes()) );
//clear contract account code
      trx.actions.emplace_back(pem, _self, NEX(setabi), std::make_tuple(owner, bytes()) );
//clear contract account abi
      trx.delay_sec = 0;

      uint128_t trxid = now() + owner + N(clrcontract); //SlowMist// trxid 算法可能被攻击者利
```

用，通过创建特殊 owner，在特定时间调用此函数可取消目标 owner 的延时交易

```
      cancel_deferred(trxid);
      trx.send( trxid, _self, true );
      print("checkresexpire  clear contract account name: ",name{owner}, " trxid:",trxid);
    }
    {
      //recycle resource
      ultrainio::transaction recyclerestrans;
      recyclerestrans.actions.emplace_back( permission_level{ _self, N(active) }, _self,
                                    NEX(recycleresource), std::make_tuple(owner) );
      recyclerestrans.delay_sec = 10;

      uint128_t trxid = now() + owner + N(recycleres); //SlowMist// trxid 算法可能被攻击者利
```

用，通过创建特殊 owner，在特定时间调用此函数可取消目标 owner 的延时交易

```
      cancel_deferred(trxid);
      recyclerestrans.send( trxid, _self, true );
      print("checkresexpire  recycle resource account name: ",name{owner}, " trxid:",trxid);
    }
  }
```

producer.cpp

```
  void system_contract::regproducer(const account_name producer,
```

```
                                const std::string& producer_key,
                                const std::string& bls_key,
                                account_name rewards_account,
                                const std::string& url,
                                name location ) {
      ultrainio_assert( url.size() < 512, "url too long" );
      // key is hex encoded
      ultrainio_assert( producer_key.size() == 64, "public key should be of size 64" );
      if(location != master_chain_name) {
         ultrainio_assert(location != N(master) , "wrong location");
         if(location != default_chain_name) {
            ultrainio_assert(_chains.find(location) != _chains.end(),
                        "wrong location, subchain is not existed");
            require_auth(_self);
         }
         else{
            ultrainio_assert(_chains.begin() != _chains.end(),
                        "no side chain is existed currently, registering to side chain is not
accepted");
            if (has_auth(_self)) {
               require_auth(_self);
            }
            else {
               require_auth(producer);
            }
         }
      }
      else {
         require_auth(_self);
      }
      uint64_t curblocknum = (uint64_t)head_block_number() + 1;
      auto briefprod = _briefproducers.find(producer);
      if(briefprod == _briefproducers.end()) {
         //new producer, add to disabled table for now
         ultrainio_assert( is_account( rewards_account ), "rewards account not exists" );
         disabled_producers_table dp_tbl(_self, _self);
         dp_tbl.emplace( [&]( disabled_producer& dis_prod ) {
            dis_prod.owner                = producer;
            dis_prod.producer_key          = producer_key;
            dis_prod.bls_key              = bls_key;
            dis_prod.url                  = url;
            dis_prod.last_operate_blocknum   = curblocknum;
            dis_prod.delegated_cons_blocknum = 0;
```

```
            dis_prod.claim_rewards_account   = rewards_account;
         });
         _briefproducers.emplace([&]( producer_brief& brief_prod ) {
            brief_prod.owner       = producer;
            brief_prod.location    = location;
            brief_prod.in_disable   = true;
         });
      } else {
         if(location == default_chain_name) {
            location = briefprod->location;
         }

         if (briefprod->in_disable) {
            disabled_producers_table dp_tbl(_self, _self);
            auto it_disable = dp_tbl.find(producer);
            ultrainio_assert(it_disable != dp_tbl.end(), "error: producer is not in disabled
table");
            if(it_disable->total_cons_staked >= _gstate.min_activated_stake) {
               producer_info new_en_prod;
               new_en_prod.owner                = producer;
               new_en_prod.producer_key          = producer_key;
               new_en_prod.bls_key              = bls_key;
               new_en_prod.total_cons_staked      = it_disable->total_cons_staked;
               new_en_prod.url                  = url;
               new_en_prod.total_produce_block    = it_disable->total_produce_block;
               new_en_prod.last_operate_blocknum   = curblocknum;
               new_en_prod.delegated_cons_blocknum = it_disable->delegated_cons_blocknum;
               new_en_prod.claim_rewards_account   = it_disable->claim_rewards_account;
               new_en_prod.unpaid_balance         = 0;
               new_en_prod.vote_number           = 0;
               new_en_prod.last_vote_blocknum     = 0;
               add_to_chain(location, new_en_prod, curblocknum);
               dp_tbl.erase(it_disable);
               _briefproducers.modify(briefprod, [&](producer_brief& producer_brf) {
                  producer_brf.in_disable = false;
                  producer_brf.location = location;
               });
            }
            else {
               //still disable
               dp_tbl.modify(it_disable, [&]( disabled_producer& dis_prod ) {
                  dis_prod.producer_key          = producer_key;
                  dis_prod.bls_key              = bls_key;
```

```
                    dis_prod.url                    = url;
                    dis_prod.last_operate_blocknum   = curblocknum;
                });
            }
        }
        else {
            if(location != master_chain_name) {
                ultrainio_assert(_chains.find(location) != _chains.end(),
                            "wrong location, subchain is not existed");
            }
            //if location changes
            producers_table _producers(_self, briefprod->location);
            auto prod = _producers.find(producer);
            ultrainio_assert(prod != _producers.end(), "producer not found");
            if(briefprod->location != location) {
                ultrainio_assert(!briefprod->is_on_master_chain(), "cannot move producers from
master chain");
                add_to_chain(location, *prod, curblocknum);
                remove_from_chain(briefprod->location, producer, curblocknum);
                _briefproducers.modify(briefprod, [&](producer_brief& producer_brf) {
                    producer_brf.location = location;
                });
            } else {
                _producers.modify( prod, [&]( producer_info& info ) {
                    info.producer_key = producer_key;
                    info.bls_key     = bls_key;
                    info.url         = url;
                });
            }
        }
    }

    if (has_auth(_self)) {
        require_auth(_self); //SlowMist// 重复校验
    } else {
        ultrainio_assert( _gstate.is_master_chain(), "only master chain allow regproducer" );
        asset cur_tokens =

ultrainio::token(N(utrio.token)).get_balance( producer,symbol_type(CORE_SYMBOL).name());
        ultrainio_assert( cur_tokens.amount >= 50000,
                    "The current action fee is 5 UGAS, please ensure that the account is
fully funded" );
```

```
            INLINE_ACTION_SENDER(ultrainio::token, safe_transfer)( N(utrio.token),
{producer,N(active)},
                { producer, N(utrio.fee), asset(50000), std::string("regproducer fee") } );
            for(auto ite_chain = _chains.begin(); ite_chain != _chains.end(); ++ite_chain) {
                if(ite_chain->chain_name == N(master))
                    continue;
                if(is_empowered(producer, ite_chain->chain_name))
                    continue;
                std::string errorlog = std::string("producer account must be synchronized to all the
subchains, \
                        so that we can schedule and secure the subchains. Please perform empoweruser
action \
                        producer:") + name{producer}.to_string() + std::string(" not synchronized to
subchain:") + name{ite_chain->chain_name}.to_string();
                ultrainio_assert( false, errorlog.c_str());
            }
        }
    }
```

慢雾科技
slow mist

官方网址

www.slowmist.com

电子邮箱

team@slowmist.com

微信公众号