Smart Contract Security Audit Report

The SlowMist Security Team received the GXChain team's application for smart contract security audit of the GXC-relay contract on Aug. 24, 2020. The following are the details and results of these smart contracts security audit:

Token Name :

GXC

合约哈希:

SHA256(dbswap.cpp)=b048fd1c9c5d3081683a35ce11a497bc6835c4f3bfab1f9502f96f0e99e961c2

The audit items and results :

(Other unknown security vulnerabilities are not included in the audit responsibility scope)

No.	Audit Items	Audit Subclass	Audit Subclass Result
1	Overflow Audit	-	Passed
2	Authority Control Audit	Authority Vulnerability Audit	Passed
		Authority Excessive Audit	Passed
3	Safety Design Audit	Hard-coded Audit	Passed
		Show coding Audit	Passed
		Abnormal check Audit	Passed
		Type safety Audit	Passed
4	Performance Optimization Audit	-	Passed
5	Design Logic Audit	-	Passed
6	Denial of Service Audit	-	Passed
7	Rollback Attack Audit	-	Passed
8	Replay Attack Audit	-	Passed
9	False Error Notification Audit	-	Passed
10	False Notice Audit	-	Passed
11	Counterfeit Token Audit	-	Passed
12	Random Number Security Audit	-	Passed

13	Dusting Attack Audit	-	Passed
14	Micro Fork Audit	-	Passed
15	Tx-Exclude Attack Audit	-	Passed

Audit Result : **Passed**

Audit Number : 0X002008310003

Audit Date : Aug. 31, 2020

Audit Team : SlowMist Security Team

(Statement : SlowMist only issues this report based on the fact that has occurred or existed before the report is issued, and bears the corresponding responsibility in this regard. For the facts occur or exist later after the report, SlowMist cannot judge the security status of its smart contract. SlowMist is not responsible for it. The security audit analysis and other contents of this report are based on the documents and materials provided by the information provider to SlowMist as of the date of this report (referred to as "the provided information"). SlowMist assumes that: there has been no information missing, tampered, deleted, or concealed. If the information provided has been missed, modified, deleted, concealed or reflected and is inconsistent with the actual situation, SlowMist will not bear any responsibility for the resulting loss and adverse effects. SlowMist will not bear any responsibility for the background or other circumstances of the project.)

Summary:

```
#include <graphenelib/asset.h>
#include <graphenelib/contract.hpp>
#include <graphenelib/contract_asset.hpp>
#include <graphenelib/global.h>
#include <graphenelib/multi_index.hpp>
#include <graphenelib/system.h>
#include <graphenelib/dispatcher.hpp>
#include <vector>
using namespace graphene;
class relay : public contract
{
public:
    relay(uint64_t account_id)
        : contract(account_id), fund_in_table(_self, _self), eth_confirm_table(_self, _self), eth_withdraw_table(_self, _self),
fund_out_table(_self, _self)
    {
    }

    // @abi action
    // @abi payable
```

//SlowMist// There is no need to verify that the transaction recipient is contract itself, because there is no event notification in GXChain and there is no false notification problem

```
void deposit(std::string target, std::string addr)
{
    int64_t asset_amount = get_action_asset_amount();
    uint64_t asset_id = get_action_asset_id();
    //SlowMist// Asset_id is verified to avoid the problem of counterfeit token
    graphene_assert(asset_id == 1, "Only support GXC ");
    graphene_assert(asset_amount >= MIN_DEPOSIT, "Must greater than minnumber ");
    contract_asset amount{asset_amount, asset_id};
    uint64_t id_number = fund_in_table.available_primary_key();
    auto coin_kind = find(TARGETS.begin(), TARGETS.end(), target);
    graphene_assert(coin_kind != TARGETS.end(), "Invalid chain name");
    uint64_t sender = get_trx_sender();
    fund_in_table.emplace(sender, [&](auto &o) {
        o.id = id_number;
        o.from = sender;
        o.asset_id = asset_id;
        o.amount = asset_amount;
        o.target = target;
        o.to = addr;
        o.state = 0;
    });
}

//@abi action
void withdraw(std::string to_account, contract_asset amount, std::string from_target, std::string txid, std::string from_account)
{
    int64_t account_id = get_account_id(to_account.c_str(), to_account.size());
    uint64_t sender = get_trx_sender();
    auto coin_kind = find(TARGETS.begin(), TARGETS.end(), from_target);
    graphene_assert(amount.asset_id == 1, "Only support GXC"); //SlowMist// Asset_id is verified to avoid the problem of counterfeit token
    graphene_assert(amount.amount >= MIN_WITHDRAW, "Must greater than min number");
    graphene_assert(coin_kind != TARGETS.end(), "Invalid target");
```

```
    graphene_assert(sender == adminAccount, "No authority"); //SlowMist// Auth check

    graphene_assert(account_id >= 0, "Invalid account_name to_account");
    graphene_assert(amount.amount > 0, "Invalid amount");
    if (from_target == "ETH")
    {
        for(auto id_begin = eth_withdraw_table.begin(); id_begin != eth_withdraw_table.end(); id_begin++){
            graphene_assert((*id_begin).txid != txid, "The txid is existed, be honest");
        }
        auto id_number = eth_withdraw_table.available_primary_key();
        eth_withdraw_table.emplace(sender, [&](auto &o) {
            o.id = id_number;
            o.txid = txid;
        });
        auto begin_iterator = eth_withdraw_table.begin();
        if (id_number - (*begin_iterator).id > TXID_LIST_LIMIT)
        {
            eth_withdraw_table.erase(begin_iterator);
        }
        auto contract_id = current_receiver();
        auto contract_balance = get_balance(contract_id, amount.asset_id);
        graphene_assert(contract_balance > amount.amount, "Balance not enough");
        //withdraw_asset(_self, account_id, amount.asset_id, amount.amount);
        auto id_number2 = fund_out_table.available_primary_key();
        int64_t block_time = get_head_block_time();
        fund_out_table.emplace(sender, [&](auto &o){
            o.id = id_number2;
            o.to_account = account_id;
            o.asset_id = amount.asset_id;
            o.amount = amount.amount;
            o.from_target = from_target;
            o.txid = txid;
            o.from_account = from_account;
            o.block_time = block_time;
        });
    }
}

//@abi action
void confirmd(uint64_t order_id, std::string target, std::string addr, contract_asset amount, std::string txid)
{
```

```
    uint64_t sender = get_trx_sender();

    graphene_assert(sender == adminAccount, "You have no authority"); //SlowMist// Auth check

    auto idx = fund_in_table.find(order_id);
    graphene_assert(idx != fund_in_table.end(), "There is no that order_id");
    graphene_assert((*idx).target == target, "Unmatched chain name");
    graphene_assert((*idx).asset_id == amount.asset_id, "Unmatched assert id");
    graphene_assert((*idx).amount == amount.amount, "Unmatched assert amount");
    if (target == "ETH")
    {
        for(auto id_begin = eth_confirm_table.begin(); id_begin != eth_confirm_table.end(); id_begin++){
            graphene_assert((*id_begin).txid != txid, "The txid is existed, be honest");
        }
        auto id_number = eth_confirm_table.available_primary_key();
        eth_confirm_table.emplace(sender, [&](auto &o) {
            o.id = id_number;
            o.txid = txid;
        });
        auto begin_iterator = eth_confirm_table.begin();
        if (id_number - (*begin_iterator).id > TXID_LIST_LIMIT)
        {
            eth_confirm_table.erase(begin_iterator);
        }
        fund_in_table.modify(idx, sender, [&](auto &o) {
            o.state = 1;
        });
        fund_in_table.erase(idx);
    }
}

//@abi action
void confirmw()
{
    uint64_t sender = get_trx_sender();
    graphene_assert(sender == adminAccount, "You have no authority");
    int64_t block_time_now = get_head_block_time();
    auto idx = fund_out_table.begin();
    auto number_index = 0;
    graphene_assert(idx != fund_out_table.end(), "There id nothing to withdraw");
    while((idx != fund_out_table.end()) && number_index < NUMBER_LIMIT){
        if(((*idx).block_time + TIME_GAP) > block_time_now){
```

```
                break;
            }
            //SlowMist// Due to the problem of GXChain mechanism, there is no transfer notification,
and the transfer recipient cannot reject the transaction, leading to DoS
            withdraw_asset(_self, (*idx).to_account, (*idx).asset_id, (*idx).amount);
            idx = fund_out_table.erase(idx);
            number_index++;
        }

    }

private:
    const uint64_t adminAccount = 4707;
    const std::vector<std::string> TARGETS = {"ETH"};
    const uint64_t MIN_DEPOSIT = 50000;
    const uint64_t MIN_WITHDRAW = 50000;
    const uint64_t TXID_LIST_LIMIT = 10000;
    const int64_t TIME_GAP = 86400;
    const uint64_t NUMBER_LIMIT = 10;

    //@abi table ctxids i64
    struct ctxids
    {
        uint64_t id;
        std::string txid;

        uint64_t primary_key() const { return id; }
        GRAPHENE_SERIALIZE(ctxids, (id)(txid))
    };
    typedef multi_index<N(ctxids), ctxids> ctxids_index;

    //@abi table wtxids i64
    struct wtxids
    {
        uint64_t id;
        std::string txid;

        uint64_t primary_key() const { return id; }
        GRAPHENE_SERIALIZE(wtxids, (id)(txid))
```

```
};
typedef multi_index<N(wtxids), wtxids> wtxids_index;

//@abi table fundin i64
struct fundin
{
    uint64_t id;
    uint64_t from;
    uint64_t asset_id;
    int64_t amount;
    std::string target;
    std::string to;
    uint64_t state;

    uint64_t primary_key() const { return id; }
    uint64_t by_sender() const { return from; }

    GRAPHENE_SERIALIZE(fundin, (id)(from)(asset_id)(amount)(target)(to)(state))
};

typedef multi_index<N(fundin), fundin,
                    indexed_by<N(sender), const_mem_fun<fundin, uint64_t, &fundin::by_sender>>>
    fund_in_index;

//@abi table fundout i64
struct fundout{
    uint64_t id;
    uint64_t to_account;
    uint64_t asset_id;
    int64_t amount;
    std::string from_target;
    std::string txid;
    std::string from_account;
    int64_t block_time;

    uint64_t primary_key() const { return id; }

    GRAPHENE_SERIALIZE(fundout, (id)(to_account)(asset_id)(amount)(from_target)(txid)(from_account)(block_time))
};
typedef multi_index<N(fundout), fundout> fund_out_index;
```

```
    fund_in_index fund_in_table;
    ctxids_index eth_confirm_table;
    wtxids_index eth_withdraw_table;
    fund_out_index fund_out_table;

};

GRAPHENE_ABI(relay, (deposit)(withdraw)(confirmd)(confirmw))
```

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account