智能合约安全审计报告

慢雾安全团队于 2020-08-24 日，收到 GXChain 团队对 GXC-relay 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：

合约哈希：

SHA256(dbswap.cpp)=b048fd1c9c5d3081683a35ce11a497bc6835c4f3bfab1f9502f96f0e99e961c2

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	回滚攻击审计	-	通过
8	重放攻击审计	-	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
14	微分叉安全审计	-	通过
15	排挤攻击安全审计	-	通过

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：0X002008310003

审计日期：2020 年 08 月 31 日

审计团队：慢雾安全团队

总结：此为 GXChain 中继合约，经反馈修正后，综合评估合约无已知风险。

以下针对合约代码进行详细分析，分析写于注释处。

```
#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// 无需校验交易接受人为合约，因为 GXChain 中不存在事件通知，不存在假通知问题

```
    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// 校验了 asset_id，避免了假币问题

        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// 校验了 asset_id，避免了假币问
题

        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// 权限校验

        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// 权限校验

    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// 由于 GXChain 机制问题，不存在转账通知，转账接收者无法拒绝交易导致 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))
```

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