成都链安
BEOSIN

YOYOW

公链安全审计报告

审计编号：202106301800

审计公链名称：YOYOW

审计公链代码地址:

https://github.com/yoyow-org/yoyow-core/tree/yy-testnet-3.0（增量审计）

初始审计 commit：d4029f0b13a88236dde216f9c024cfe49a3d13cf

最终审计 commit：fa9b46e04c27a4ff3b2b89f9cccab6f70c322e75

审计开始日期：2021.05.31

审计完成日期：2021.06.30

审计结果：通过（优）

审计团队：成都链安科技有限公司

审计类型及结果：

序号	**审计类型**	**审计子项**	**审计结果**
1	合约虚拟机	合约部署	**通过**
		合约更新	**通过**
		合约调用	**通过**
2	资产安全	合约资产	**通过**
		交易手续费	**通过**

免责声明：本报告系针对项目代码而作出，本报告的任何描述、表达或措辞均不得被解释为对项目的认可、肯定或确认。本次审计仅针对本报告载明的审计类型及结果表中给定的审计类型范围进行审计，其他未知安全漏洞不在本次审计责任范围之内。成都链安科技仅根据本报告出具前已经存在或发生的攻击或漏洞出具本报告，对于出具以后存在或发生的新的攻击或漏洞，成都链安科技无法判断其对公链安全状况可能的影响，亦不对此承担责任。本报告所作的安全审计分析及其他内容，仅基于公链提供者在本报告出具前已向成都链安科技提供的文件和资料，且该部分文件和资料不存在任何缺失、被篡改、删减或隐瞒的前提下作出的；如提供的文件和资料存在信息缺失、被篡改、删减、隐瞒或反映的情况与实际情况不符等情况或提供文件和资料在本报告出具后发生任何变动的，成都链安科技对由此而导致的损失和不利影响不承担任何责任。成都链安科技出具的本审计报告系根据公链提供者提供的文件和资料依靠成都链安科技现掌握的技术而作出的，由于任何机构均存在技术的局限性，成都链安科技作出的本审计报告仍存在无法完整检测出全部风险的可能性，成都链安科技对由此产生的损失不承担任何责任。

本声明最终解释权归成都链安科技所有。

审计结果：

本公司采用模拟攻击的方式对公链 YOYOW 的 WASM 模块安全性以及业务逻辑安全性等方面进行多维度全面的黑盒安全审计。**经审计,YOYOW 公链的 WASM 模块通过所有黑盒攻击检测项，审计结果为通过(优)。**

1 合约虚拟机

YOYOW 3.0 新增了智能合约模块，智能合约虚拟机为 WASM，用户可以部署、升级、调用智能合约。

1.1 合约部署

使用 gxx 工具编译智能合约源码，生成 WASM 字节码文件和 ABI 接口文件。然后使用 deploy_contract 命令部署合约。

```
// 成都链安 // 源码路径：libraries/wallet/wallet.cpp
   signed_transaction deploy_contract(
                                     uint64_t contract_id,
                                     string vm_type,
                                     string vm_version,
                                     string contract_dir,
                                     bool csaf_fee =  true,
                                     bool broadcast = false)
   { try {
      FC_ASSERT(!self.is_locked());

      std::vector<uint8_t> wasm;
      variant abi_def_data;
      // 成都链安 // 加载智能合约的 WASM 字节码文件与 ABI 接口文档
      auto load_contract = [&]() {
          fc::path cpath(contract_dir);
          if (cpath.filename().generic_string() == ".") cpath = cpath.parent_path();

          fc::path wasm_path = cpath / (cpath.filename().generic_string() + ".wasm");
          fc::path abi_path = cpath / (cpath.filename().generic_string() + ".abi");

          bool wasm_exist = fc::exists(wasm_path);
          bool abi_exist = fc::exists(abi_path);

          FC_ASSERT(abi_exist, "no abi file exist");
          FC_ASSERT(wasm_exist, "no wasm file exist");

          abi_def_data = fc::json::from_file(abi_path);

          std::string wasm_string;
          fc::read_file_contents(wasm_path, wasm_string);
          // 成都链安 // 智能合约的 WASM 字节码文件必须以\x00\x61\x73\x6d 四个字节开头
          const string binary_wasm_header("\x00\x61\x73\x6d", 4);
          FC_ASSERT(wasm_string.size() > 4 && (wasm_string.compare(0, 4, binary_wasm_header) ==
0), "wasm invalid");

          for (auto it = wasm_string.begin(); it != wasm_string.end(); ++it) {
              wasm.push_back(*it); //TODO
          }
```

```
        };

        load_contract();
        // 成都链安 // 设置交易参数
        contract_deploy_operation op;
        op.contract_id = contract_id;
        op.vm_type = vm_type;
        op.vm_version = vm_version;
        op.code = bytes(wasm.begin(), wasm.end());
        op.abi = abi_def_data.as<abi_def>(GRAPHENE_MAX_NESTED_OBJECTS);

        signed_transaction tx;
        tx.operations.push_back(op);
        set_operation_fees(tx,
_remote_db->get_global_properties().parameters.get_current_fees(), csaf_fee);
        auto dyn_props = get_dynamic_global_properties();
        tx.set_reference_block( dyn_props.head_block_id );
        tx.set_expiration( dyn_props.time + fc::seconds(30) );

        tx.validate();
        // 成都链安 // 签名并广播交易
        return sign_transaction(tx, broadcast);
    }
FC_CAPTURE_AND_RETHROW( (contract_id)(vm_type)(vm_version)(contract_dir)(csaf_fee)(broadcast)
) }
```

节点在打包合约部署交易时，会先通过 do_evaluate 函数对合约部署交易进行校验，如果通过校验，会调用 do_apply 函数将合约信息写入到目标地址。

```
// 成都链安 // 源码路径：libraries/chain/contract_evaluator.cpp
// 成都链安 // 合约部署交易有效性检测
void_result contract_deploy_evaluator::do_evaluate(const contract_deploy_operation &op)
{ try {
    database &d = db();

    FC_ASSERT(d.head_block_time()>=HARDFORK_2_0_TIME,"contract is not enabled before
HARDFORK_2_0_TIME");

    contract_obj = &(d.get_account_by_uid(op.contract_id));
    FC_ASSERT(contract_obj->code.size() == 0, "account: ${a} already deployed contract", ("a",
op.contract_id));

    wasm_interface::validate(op.code);

    return void_result();
} FC_CAPTURE_AND_RETHROW((op)) }

// 成都链安 // 合约部署交易生效
```

```
void_result contract_deploy_evaluator::do_apply(const contract_deploy_operation &op)
{ try {
    db().modify(*contract_obj, [&](account_object &obj) {
        obj.vm_type = op.vm_type;
        obj.vm_version = op.vm_version;
        obj.code = op.code;
        obj.code_version = fc::sha256::hash(op.code);
        obj.abi = op.abi;
    });

    return void_result();
} FC_CAPTURE_AND_RETHROW((op)) }
```

注意：合约部署仅在对应目标地址无字节码时才能够部署到对应地址，如果已有字节码，则合约部署交易检测时会抛出异常，合约无法部署。合约部署后可以使用 update_contract 命令进行升级，目前未提供合约字节码删除的相关 RPC 命令。

1.2 合约升级

使用 gxx 工具编译智能合约源码，生成 WASM 字节码文件和 ABI 接口文件。然后使用 update_contract 命令对已有合约进行升级。

```
// 成都链安 // 源码路径：libraries/wallet/wallet.cpp
   signed_transaction update_contract(string contract,
                                      string contract_dir,
                                      bool csaf_fee,
                                      bool broadcast)
   { try {
      FC_ASSERT(!self.is_locked());


      std::vector<uint8_t> wasm;
      variant abi_def_data;
      // 成都链安 // 加载智能合约的 WASM 字节码文件与 ABI 接口文档
      auto load_contract = [&]() {
          fc::path cpath(contract_dir);
          if (cpath.filename().generic_string() == ".") cpath = cpath.parent_path();

          fc::path wasm_path = cpath / (cpath.filename().generic_string() + ".wasm");
          fc::path abi_path = cpath / (cpath.filename().generic_string() + ".abi");

          bool wasm_exist = fc::exists(wasm_path);
          bool abi_exist = fc::exists(abi_path);

          FC_ASSERT(abi_exist, "no abi file exist");
          FC_ASSERT(wasm_exist, "no wasm file exist");

          abi_def_data = fc::json::from_file(abi_path);
```

```
        std::string wasm_string;
        fc::read_file_contents(wasm_path, wasm_string);
        // 成都链安 // 智能合约的 WASM 字节码文件必须以\x00\x61\x73\x6d 四个字节开头
        const string binary_wasm_header("\x00\x61\x73\x6d", 4);
        FC_ASSERT(wasm_string.size() > 4 && (wasm_string.compare(0, 4, binary_wasm_header) ==
0), "wasm invalid");

        for (auto it = wasm_string.begin(); it != wasm_string.end(); ++it) {
            wasm.push_back(*it); //TODO
        }
    };

    load_contract();

    account_object contract_obj = this->get_account(contract);

    optional<account_uid_type> new_owner_account_id;
    // 成都链安 // 设置交易参数
    contract_update_operation op;
    op.contract_id = contract_obj.uid;
    op.code = bytes(wasm.begin(), wasm.end());
    op.abi = abi_def_data.as<abi_def>(GRAPHENE_MAX_NESTED_OBJECTS);

    signed_transaction tx;
    tx.operations.push_back(op);
    set_operation_fees(tx,
_remote_db->get_global_properties().parameters.get_current_fees(), csaf_fee);
    auto dyn_props = get_dynamic_global_properties();
    tx.set_reference_block( dyn_props.head_block_id );
    tx.set_expiration( dyn_props.time + fc::seconds(30) );
    tx.validate();
    // 成都链安 // 签名并广播交易
    return sign_transaction(tx, broadcast);
  } FC_CAPTURE_AND_RETHROW( (contract)(contract_dir)(csaf_fee)(broadcast)) }
```

节点在打包合约升级交易时，会先通过 do_evaluate 函数对合约升级交易进行校验，如果通过校验，会调用 do_apply 函数将合约信息更新到目标地址。

```
// 成都链安 // 源码路径：libraries/chain/contract_evaluator.cpp
// 成都链安 // 合约升级交易有效性检测
void_result contract_update_evaluator::do_evaluate(const contract_update_operation &op)
{ try {
    database &d = db();

    const account_object& contract_obj = d.get_account_by_uid(op.contract_id);
    FC_ASSERT(contract_obj.code.size() > 0, "can not update a contract not deployed: ${a}", ("a",
op.contract_id));
```

```
    code_hash = fc::sha256::hash(op.code);
    FC_ASSERT(code_hash != contract_obj.code_version, "code not updated");

    wasm_interface::validate(op.code);


    return void_result();
} FC_CAPTURE_AND_RETHROW((op)) }
// 成都链安 // 合约升级交易生效
void_result contract_update_evaluator::do_apply(const contract_update_operation &op)
{ try {
    database &d = db();
    const account_object& contract_obj = d.get_account_by_uid(op.contract_id);
    db().modify(contract_obj, [&](account_object &obj) {
        obj.code = op.code;
        obj.code_version = code_hash;
        obj.abi = op.abi;
    });

    return void_result();
} FC_CAPTURE_AND_RETHROW((op)) }
```

注意：合约升级仅在对应目标地址有字节码时才能够部署到对应地址，如果无字节码或者升级前后字节码一致，则合约升级交易检测时会抛出异常，这意味着合约无法单独升级 ABI。

1.3 合约调用

合约部署到链平台后，用户可使用 call_contract 命令调用对应合约函数。

```
// 成都链安 // 源码路径：libraries/wallet/wallet.cpp
   signed_transaction call_contract(string account,
                                    string contract,
                                    optional<asset> amount,
                                    string method,
                                    string args,
                                    bool csaf_fee,
                                    bool broadcast)
   { try {
        FC_ASSERT(!self.is_locked());

        account_object caller = get_account(account);
        account_object contract_obj = get_account(contract);
        // 成都链安 // 设置交易参数
        contract_call_operation contract_call_op;
        contract_call_op.account = caller.uid;
        contract_call_op.contract_id = contract_obj.uid;
        if (amount.valid()) {
            contract_call_op.amount = amount;
        }
        contract_call_op.method_name = string_to_name(method.c_str());
        fc::variant action_args_var = fc::json::from_string(args);

        abi_serializer abis(contract_obj.abi, fc::milliseconds(1000000));
        auto action_type = abis.get_action_type(method);
        GRAPHENE_ASSERT(!action_type.empty(), action_validate_exception, "Unknown action
${action} in contract ${contract}", ("action", method)("contract", contract));
        contract_call_op.data = abis.variant_to_binary(action_type, action_args_var,
fc::milliseconds(1000000));

        signed_transaction tx;
        tx.operations.push_back(contract_call_op);
        set_operation_fees(tx,
_remote_db->get_global_properties().parameters.get_current_fees(), csaf_fee);
       auto dyn_props = get_dynamic_global_properties();
      tx.set_reference_block( dyn_props.head_block_id );
      tx.set_expiration( dyn_props.time + fc::seconds(30) );
        tx.validate();
        // 成都链安 // 签名并广播交易
        return sign_transaction(tx, broadcast);
   } FC_CAPTURE_AND_RETHROW( (account)(contract)(amount)(method)(args)(csaf_fee)(broadcast)) }
```

节点在打包合约调用交易时，会先通过 do_evaluate 函数对合约调用交易进行校验，如果通过校验，会调用 do_apply 函数执行调用操作，并将执行结果写入区块链。

```
// 成都链安 // 源码路径：libraries/chain/contract_evaluator.cpp
// 成都链安 // 合约调用交易有效性检测
void_result contract_call_evaluator::do_evaluate(const contract_call_operation &op)
{ try {
    database& d = db();
    const account_object& contract_obj = d.get_account_by_uid(op.contract_id);
    FC_ASSERT(contract_obj.code.size() > 0, "contract has no code, contract_id ${n}", ("n",
op.contract_id));

    // check method_name
    const auto& actions = contract_obj.abi.actions;
    auto iter = std::find_if(actions.begin(), actions.end(),
            [&](const action_def& act) { return act.name == op.method_name; });
    FC_ASSERT(iter != actions.end(), "method_name ${m} not found in abi", ("m", op.method_name));
    if (op.amount.valid()) {
        // check method_name, must be payable
        FC_ASSERT(iter->payable, "method_name ${m} not payable", ("m", op.method_name));

        // check balance
        bool sufficient_balance = d.get_balance(op.account, op.amount->asset_id).amount >=
op.amount->amount;
```

```
        FC_ASSERT(sufficient_balance,
                  "insufficient balance: ${balance}, unable to deposit '${total_transfer}' from
account '${a}' to '${t}'",
                  ("a", op.account)("t", contract_obj.uid)("total_transfer",
d.to_pretty_string(*(op.amount)))
                  ("balance", d.to_pretty_string(d.get_balance(op.account,
op.amount->asset_id))));
    }

    if (op.fee.total.amount > 0) {
        // fee_from_account is calculated by evaluator::evaluate()
        // prepare_fee --> do_evaluate -> convert_fee -> pay_fee -> do_apply
        // if cpu_fee charged, this check may fail for cpu time may different for the same opertion
        FC_ASSERT(op.fee.total >= fee_from_account, "insufficient fee paid in trx, ${a} needed",
("a", d.to_pretty_string(fee_from_account)));
    }

    // ram-account must exists
    const auto &account_idx = d.get_index_type<account_index>().indices().get<by_name>();
    const auto &account_itr = account_idx.find("ramaccount");
    FC_ASSERT(account_itr != account_idx.end(), "ramaccount not exist");
    ram_account_id = account_itr->uid;

    return void_result();
} FC_CAPTURE_AND_RETHROW((op)) }
// 成都链安 // 合约调用交易生效
contract_receipt contract_call_evaluator::do_apply(const contract_call_operation &op)
{ try {
    // do apply:
    //  1. run contract code
    //  2. charge base fee
    //  3. charge ram fee by account

    // charge base fee:
    // 1. calculate base_fee (basic fee + cpu fee)
    // 2. convert base_fee to core asset
    //      2.1 call prepare_fee to calculate fee_from_account and core_fee_paid
    //      2.2 call convert_fee to adjust UIA fee_pool (UIA: user-issued assets)
    // 3. deposit cashback
    // 4. adjust fee_payer's balance

    auto &d = db();

    fc::microseconds max_trx_cpu_us = fc::seconds(3);
    if (_billed_cpu_time_us == 0)
        max_trx_cpu_us =
fc::microseconds(std::min(d.get_global_extension_params().trx_cpu_limit,
d.get_max_trx_cpu_time()));

    action act{op.account, op.contract_id, op.method_name, op.data};
    if (op.amount.valid()) {
        act.amount.amount = op.amount->amount.value;
        act.amount.asset_id = op.amount->asset_id;
    }

    // run contract code
    transaction_context trx_context(d, op.fee_payer_uid(), max_trx_cpu_us);
    apply_context ctx{d, trx_context, act};
    ctx.exec();

    fee_param = get_contract_call_fee_parameter(d);
    uint32_t cpu_time_us = _billed_cpu_time_us > 0? _billed_cpu_time_us :
trx_context.get_cpu_usage();
    if(cpu_time_us > 1000)
    {
        auto cpu_fee = uint64_t(cpu_time_us + 999) / 1000 * fee_param.price_per_ms_cpu;
        d.adjust_balance(op.account, - asset(cpu_fee,GRAPHENE_CORE_ASSET_AID));
    }

    contract_receipt receipt;
    receipt.billed_cpu_time_us = cpu_time_us;
    receipt.fee = fee_from_account;

    account_receipt r;

    auto ram_statistics = trx_context.get_ram_statistics();
    for (const auto &ram : ram_statistics) {
        // map<account, ram_bytes>
        r.account = account_uid_type(ram.first);
        r.ram_bytes = ram.second;

        // charge and set ram_fee
        charge_ram_fee_by_account(r, d, op);
        receipt.ram_receipts.push_back(r);
    }

    return receipt;
} FC_CAPTURE_AND_RETHROW((op)) }
```

如果调用的合约函数为 payable，那么在合约调用过程中可随交易附带资产。在合约执行过程中，合约可向另外一个合约发起跨合约调用交易以及内联转账，在跨合约调用中也可携带对应的资产。由于合约调用操作的有效性检测会查询调用的函数是否在 ABI 中，如果 ABI 中无法查询到被调用的函数接口，调用将抛出异常。

```
// 成都链安 // 源码路径：libraries/chain/wasm_interface.cpp
```

```
class transaction_api : public context_aware_api {
  public:
    using context_aware_api::context_aware_api;

    void send_inline(array_ptr<char> data, size_t data_len) {
      uint32_t max_inline_action_size =
context.trx_context.get_inter_contract_calling_params().max_inline_action_size;
      FC_ASSERT(data_len <= max_inline_action_size, "inline action too big, max size=${s}
bytes", ("s", max_inline_action_size));
      // 成都链安 // 调用次数检查，最大调用次数为 4
      context.trx_context.check_inter_contract_depth();

      action act;
      fc::raw::unpack<action>(data, data_len, act, 20);
      // 成都链安 // 调用者地址检查
      // check action sender
      FC_ASSERT(act.sender == context.receiver,
            "the sender must be current contract, actually act.sender=${s}, current
receiver=${r}", ("s", act.sender)("r", context.receiver));
      // check amount
      FC_ASSERT(act.amount.amount >=0, "action amount must >= 0, actual amount: ${a}", ("a",
act.amount.amount));
      // 成都链安 // 目标合约检查
      // check action contract code
      const account_object& contract_obj =
context._db->get_account_by_uid(act.contract_id);
      FC_ASSERT(contract_obj.code.size() > 0, "inline action's code account ${account} does
not exist", ("account", act.contract_id));
      // 成都链安 // 调用函数 ABI 检查
      // check method_name, must be payable
      const auto &actions = contract_obj.abi.actions;
      auto iter = std::find_if(actions.begin(), actions.end(),
            [&](const action_def &act_def) { return act_def.name == act.method_name; });
      FC_ASSERT(iter != actions.end(), "method_name ${m} not found in abi", ("m",
act.method_name));
      if (act.amount.amount > 0) {
        FC_ASSERT(iter->payable, "method_name ${m} not payable", ("m", act.method_name));
      }
      // 成都链安 // 生成 inter_contract_call_operation
      inter_contract_call_operation op;
      op.fee = asset{0, context._db->get_core_asset().asset_id};
      if(act.amount.amount > 0)
        op.amount = asset{act.amount.amount, asset_aid_type(act.amount.asset_id)};
      op.contract_id = account_uid_type(act.contract_id);
      op.data = act.data;
      op.method_name = act.method_name;
      op.sender_contract = account_uid_type(context.receiver);
```

```
          context.execute_inline(std::move(op));
      }
};
```

在此生成 inter_contract_call_operation 的过程中，需要注意的是单笔交易中，最多能生成 4 笔跨合约调用的 operation，超出 4 笔则交易执行抛出异常。生成 inter_contract_call_operation 后，会调用 execute_inline 函数将 operation 放入_inline_operations 队列的尾部，在当前函数执行完成后，再执行_inline_operations 队列中的 operation。

```
// 成都链安 // 源码路径：libraries/chain/contract_evaluator.cpp
// 成都链安 // 跨合约调用有效性检测
void_result inter_contract_call_evaluator::do_evaluate(const inter_contract_call_operation
&op)
{ try {
    database &d = db();
    FC_ASSERT(d.get_contract_transaction_ctx() != nullptr, "contract_transaction_ctx
invalid");
    return void_result();
} FC_CAPTURE_AND_RETHROW((op)) }
// 成都链安 // 跨合约调用生效
void_result inter_contract_call_evaluator::do_apply(const inter_contract_call_operation &op)
{ try {
    database &d = db();
    transaction_context* contract_transaction_ctx = d.get_contract_transaction_ctx();

    action act{op.sender_contract, op.contract_id, op.method_name, op.data};
    if (op.amount.valid()) {
        act.amount.amount = op.amount->amount.value;
        act.amount.asset_id = op.amount->asset_id;
    }

    apply_context ctx{d, *contract_transaction_ctx, act};
    ctx.exec();
    return void_result();
} FC_CAPTURE_AND_RETHROW((op)) }
```

注意：(1) 跨合约调用执行是异步的，即跨合约调用不会立即执行，而是将跨合约调用放入队列，然后继续执行当前函数剩余代码，之后再执行跨合约调用；

(2) 多个跨合约调用执行顺序可以参考树形结构的前序遍历方式，即：相同深度的跨合约调用依据先放入队列的先后顺序执行，每个跨合约调用以及其产生的跨合约调用执行完成之后再执行同深度的下一个跨合约调用；

(3) 所有的跨合约调用无论深度，在一次交易执行中，最多只能有 4 笔跨合约调用；

(4) 跨合约调用的第三个参数，即 sender 必须为当前合约账户。因此，需要注意，如果跨合约调用的目标函数里面设置的 RAM 支付账户为调用者，那么将会由本合约支付对应的 RAM。在智能合约开发过程中需要注意该特点可能被攻击者利用，导致恶意 RAM 消耗产生的合约账户资产损失。

```
// 成都链安 // 源码路径：contracts/graphenelib/action.hpp
      /**
       *  @tparam T - the type of the action data
       *  @param contract - name of the contract account
       *  @param method - name of the action
       *  @param args - will be serialized via pack into data
       *  @param sender - the contract caller
       *  @param amt - the amount of asset to transfer to target contract
       */
      template <typename T>
      action(std::string contract, action_name method, T &&args, uint64_t sender, const
contract_asset &amt = {0, 0})
          : sender(sender)
          , amount(amt)
          , method(method)
          , method_args_binary(pack(std::forward<T>(args)))
      {
          int64_t acc_id = get_account_id(contract.c_str(), contract.length());
          graphene_assert(-1 != acc_id, "account not found");
          contract_id = acc_id;
      }

      GRAPHENE_SERIALIZE(action, (sender)(contract_id)(amount)(method)(method_args_binary))

      void send() const
      {
          auto serialize = pack(*this);
          ::send_inline(serialize.data(), serialize.size());
      }
```

2 资产安全

2.1 合约资产安全

合约可使用合约账户的余额进行转账，但不可使用合约账户的积分以及零钱。WASM 合约虚拟机提供了两个函数接口用于转出合约账户转账：withdraw_asset 和 inline_transfer。这两个函数接口功能基本一致，inline_transfer 额外新增了 memo 参数，可在内部转账中附加 memo 信息。这两个函数接口在执行时会生成 inline_transfer_operation 用于实现转账操作。

```
// 成都链安 // 源码路径：libraries/chain/wasm_interface.cpp
    void withdraw_asset(int64_t from, int64_t to, int64_t asset_id, int64_t amount)
    {
        // 成都链安 // 转账条件检测
        FC_ASSERT(from == context.receiver, "can only withdraw from contract ${c}", ("c",
context.receiver));
        FC_ASSERT(from != to, "cannot transfer to self");
        FC_ASSERT(amount > 0, "withdraw  amount ${a} must > 0", ("a", amount));
        FC_ASSERT(from >= 0, "account id ${a} from must  >= 0", ("a", from));
        FC_ASSERT(to >= 0, "account id ${a} to must >= 0", ("a", to));
        FC_ASSERT(asset_id >= 0, "asset id ${a} must >= 0", ("a", asset_id));

        auto &d = context.db();
        asset a{amount, asset_aid_type(asset_id & GRAPHENE_DB_MAX_INSTANCE_ID)};
        account_uid_type from_account = account_uid_type(from & GRAPHENE_DB_MAX_INSTANCE_ID);
        account_uid_type to_account = account_uid_type(to & GRAPHENE_DB_MAX_INSTANCE_ID);
        FC_ASSERT(d.get_balance(from_account, a.asset_id).amount >= amount, "insufficient
balance ${b}, unable to withdraw ${a} from account ${c}", ("b",
d.to_pretty_string(d.get_balance(from_account, a.asset_id)))("a", amount)("c", from_account));

        // adjust balance
        transaction_evaluation_state op_context(&d);
        op_context.skip_fee_schedule_check = true;

        inline_transfer_operation op;
        op.amount = a;
        op.from = from_account;
        op.to = to_account;
        op.fee = asset{0, d.get_core_asset().asset_id};
        d.apply_operation(op_context, op);
    }

    void inline_transfer(int64_t from, int64_t to, int64_t asset_id, int64_t amount,
array_ptr<char> data, size_t datalen)
    {
        auto &d = context.db();

        FC_ASSERT(from == context.receiver, "can only transfer from contract ${c}", ("c",
context.receiver));
        FC_ASSERT(from >= 0, "account id ${a} from must  >= 0", ("a", from));
        FC_ASSERT(to >= 0, "account id ${a} to must >= 0", ("a", to));
        FC_ASSERT(from != to, "cannot transfer to self");
        FC_ASSERT(asset_id >= 0, "asset id ${a} must >= 0", ("a", asset_id));

        asset a{amount, asset_aid_type(asset_id & GRAPHENE_DB_MAX_INSTANCE_ID)};
        account_uid_type from_account = account_uid_type(from & GRAPHENE_DB_MAX_INSTANCE_ID);
        account_uid_type to_account = account_uid_type(to & GRAPHENE_DB_MAX_INSTANCE_ID);

        std::string memo(data, datalen);

        // apply operation
        transaction_evaluation_state op_context(&d);
        op_context.skip_fee_schedule_check = true;
        inline_transfer_operation op;
```

```
         op.amount = a;
         op.from = from_account;
         op.to = to_account;
         op.memo = memo;
         op.fee = asset{0, d.get_core_asset().asset_id};
         d.apply_operation(op_context, op);
    }
```

inline_transfer_operation 会立即执行。执行 inline_transfer_operation 时，会调用 do_evaluate 检查发送方(当前为合约账户)是否具有足够的转账资产，如果资产余额不足将抛出异常。如果通过检查，将调用 do_apply 修改转账双方余额。

```
// 成都链安 // 源码路径：libraries/chain/transfer_evaluator.cpp
// 成都链安 // inline_transfer_operation 有效性检测
void_result inline_transfer_evaluator::do_evaluate(const inline_transfer_operation &op)
{ try {
   const database& d = db();

   const account_object& from_account     = d.get_account_by_uid(op.from);
   const account_object& to_account       = d.get_account_by_uid(op.to);
   const asset_object&   asset_type       = d.get_asset_by_aid( op.amount.asset_id );

   bool insufficient_balance = d.get_balance(from_account, asset_type).amount >=
op.amount.amount;
   FC_ASSERT(insufficient_balance,
             "Insufficient Balance: ${balance}, unable to transfer '${total_transfer}' from
account '${a}' to '${t}'",
             ("a", from_account.name)("t", to_account.name)("total_transfer",
d.to_pretty_string(op.amount))("balance", d.to_pretty_string(d.get_balance(from_account,
asset_type))));

   return void_result();

} FC_CAPTURE_AND_RETHROW((op))}
// 成都链安 // inline_transfer_operation 生效
void_result inline_transfer_evaluator::do_apply(const inline_transfer_operation& op)
{ try {
    db().adjust_balance(op.from, -op.amount);
    db().adjust_balance(op.to, op.amount);
    return void_result();
} FC_CAPTURE_AND_RETHROW((op))}
```

2.2 交易手续费

为了避免资源消耗攻击，YOYOW 公链对每笔交易收取了手续费，除 RAM 手续费是发送到 ramaccount 账户，其余手续费直接从手续费支付者账户销毁对应数量的 YOYO。合约操作的相关手续费计算公式如下：

- 合约部署

- 基本手续费：1 YOYO
- 数据手续费：10 YOYO/KB （备注：包括 vm_type、vm_version、code、abi）
- 总手续费 = 基本手续费 + 数据手续费

```
// 成都链安 // 源码路径：libraries/chain/transfer_evaluator.cpp
struct contract_deploy_operation : public base_operation {
    // 成都链安 // 定义手续费类型以及默认值
    struct fee_parameters_type {
        uint64_t fee = 1 * GRAPHENE_BLOCKCHAIN_PRECISION;
        uint64_t min_real_fee     = 0;
        uint16_t min_rf_percent   = 0;
        uint32_t price_per_kbyte  = 10 * GRAPHENE_BLOCKCHAIN_PRECISION;
        extensions_type   extensions;
    };

    fee_type                        fee;
    account_uid_type                 contract_id;
    fc::string                      vm_type;
    fc::string                      vm_version;
    bytes                           code;
    abi_def                         abi;
    extensions_type                  extensions;

    account_uid_type fee_payer_uid() const
    {
        return contract_id;
    }

    void validate() const
    {
        validate_op_fee( fee, "contract_deploy" );
        validate_account_uid(contract_id,"contract_id ");
        FC_ASSERT(code.size() > 0, "contract code cannot be empty");
        FC_ASSERT(abi.actions.size() > 0, "contract has no actions");
    }
    // 成都链安 // 总手续费计算
    share_type calculate_fee(const fee_parameters_type &k) const
    {
        auto core_fee_required = k.fee;
        auto bSize = vm_type.size() + vm_version.size() + code.size()+
fc::raw::pack_size(abi);
        auto data_fee = calculate_data_fee(bSize, k.price_per_kbyte);
        core_fee_required += data_fee;
        return core_fee_required;
    }
};
```

- 合约升级

➢ 基本手续费：1 YOYO

➢ 数据手续费：10 YOYO/KB （备注：包括 code、abi）

➢ 总手续费 = 基本手续费 + 数据手续费

```
// 成都链安 // 源码路径：libraries/chain/transfer_evaluator.cpp
struct contract_update_operation : public base_operation {
    // 成都链安 // 定义手续费类型以及默认值
    struct fee_parameters_type {
        uint64_t fee = 1 * GRAPHENE_BLOCKCHAIN_PRECISION;
        uint64_t min_real_fee     = 0;
        uint16_t min_rf_percent   = 0;
        uint32_t price_per_kbyte  = 10 * GRAPHENE_BLOCKCHAIN_PRECISION;
        extensions_type   extensions;
    };

    fee_type                          fee;
    account_uid_type                   contract_id;
    bytes                             code;
    abi_def                           abi;
    extensions_type                    extensions;

    account_uid_type fee_payer_uid() const
    {
        return contract_id;
    }

    void validate() const
    {
        validate_op_fee( fee, "contract_update" );
        FC_ASSERT(code.size() > 0, "contract code cannot be empty");
        FC_ASSERT(abi.actions.size() > 0, "contract has no actions");


        validate_account_uid(contract_id,"contract_id ");
    }
    // 成都链安 // 总手续费计算
    share_type calculate_fee(const fee_parameters_type &k) const
    {
        auto core_fee_required = k.fee;
        auto bSize = code.size() + fc::raw::pack_size(abi);
        auto data_fee = calculate_data_fee(bSize, k.price_per_kbyte);
        core_fee_required += data_fee;
        return core_fee_required;
    }
};
```

- 合约调用

➢ 基本手续费：0.5 YOYO

➢ 合约执行手续费：5 YOYO/ms （备注：执行时间小于 1ms 不收取本项费用）

➢ RAM 手续费：10 YOYO/KB

➢ 总手续费 = 基本手续费 + 合约执行手续费 + RAM 手续费

```
// 成都链安 // 源码路径：libraries/chain/transfer_evaluator.cpp
struct contract_call_operation : public base_operation {
    // 成都链安 // 定义手续费类型以及默认值
    struct fee_parameters_type {
        uint64_t fee =  GRAPHENE_BLOCKCHAIN_PRECISION/2;
        uint64_t price_per_kbyte_ram =  GRAPHENE_BLOCKCHAIN_PRECISION * 10; //fee of ram
should pay from balance directly. not include in fee_type fee
        uint64_t price_per_ms_cpu = GRAPHENE_BLOCKCHAIN_PRECISION*5;
        uint64_t min_real_fee     = 0;
        uint16_t min_rf_percent   = 0;
        extensions_type   extensions;
    };

    fee_type                                 fee;
    account_uid_type                          account;
    account_uid_type                          contract_id;
    fc::optional<asset>                       amount;
    action_name                              method_name;
    bytes                                    data;
    extensions_type                           extensions;

    account_uid_type fee_payer_uid() const { return account; }

    void validate() const
    {
        validate_op_fee( fee, "contract_call" );
        validate_account_uid(contract_id,"contract_id ");
        FC_ASSERT(data.size() >= 0);
        if (amount.valid()) {
            FC_ASSERT(amount->amount > 0, "amount must > 0");
        }
    }
    // 成都链安 // 基本手续费计算
    share_type calculate_fee(const fee_parameters_type &k) const
    {
        // just return basic fee, real fee will be calculated after runing
        return k.fee;
    }
};
```

3 审计结果

3.1 合约部署

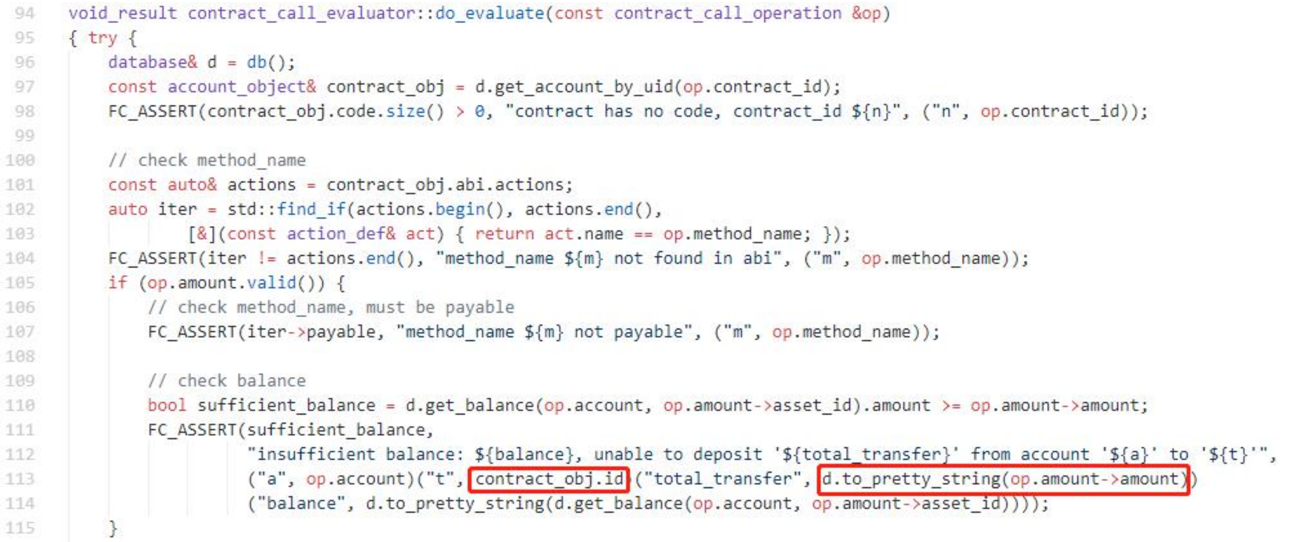
问题描述：合约部署时，vm_type 与 vm_version 未起到作用，节点也未对其进行检测。

修复结果：经项目方确认，vm_type 与 vm_version 只是纯记录，未赋予任何意义。

3.2 调用合约时携带资产

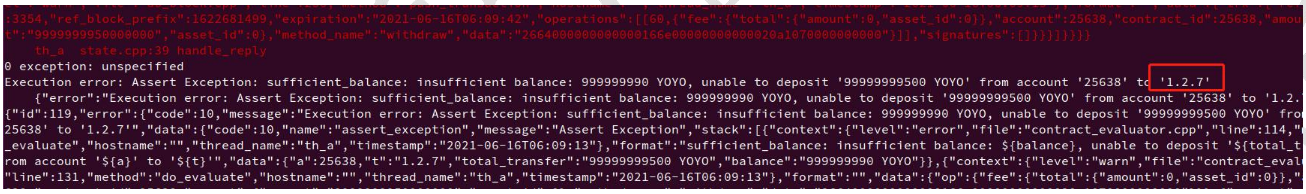
问题描述：调用合约时，如果用户的账户余额小于指定携带的资产数量，交易发送失败，并返回交易失败原因。在返回的交易失败原因中，发送者账户与合约账户显示格式不一致，如图 2 所示。此外，在发送的是非 YOYOW 资产时，该资产仍按照 YOYOW 资产进行显示，如图 3 所示。建议修改交易失败原因字符串。

修复结果：已修复。



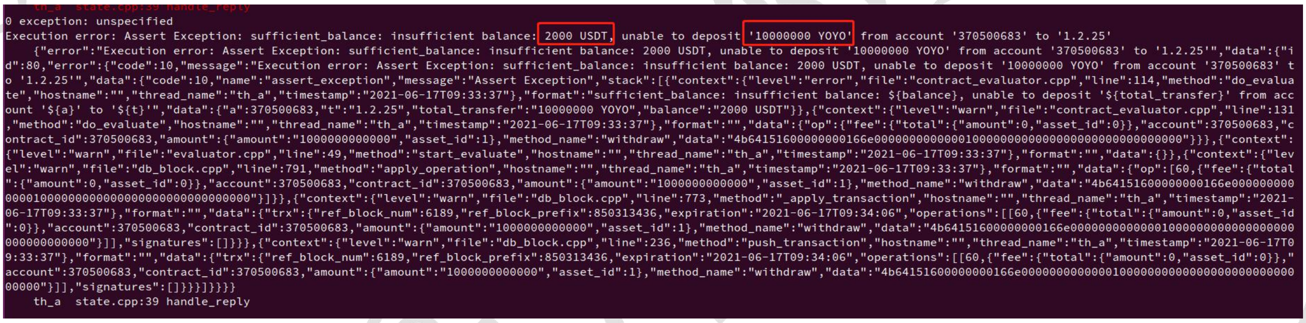
```
void_result contract_call_evaluator::do_evaluate(const contract_call_operation &op)
{ try {
    database& d = db();
    const account_object& contract_obj = d.get_account_by_uid(op.contract_id);
    FC_ASSERT(contract_obj.code.size() > 0, "contract has no code, contract_id ${n}", ("n", op.contract_id));

    // check method_name
    const auto& actions = contract_obj.abi.actions;
    auto iter = std::find_if(actions.begin(), actions.end(),
            [&](const action_def& act) { return act.name == op.method_name; });
    FC_ASSERT(iter != actions.end(), "method_name ${m} not found in abi", ("m", op.method_name));
    if (op.amount.valid()) {
        // check method_name, must be payable
        FC_ASSERT(iter->payable, "method_name ${m} not payable", ("m", op.method_name));

        // check balance
        bool sufficient_balance = d.get_balance(op.account, op.amount->asset_id).amount >= op.amount->amount;
        FC_ASSERT(sufficient_balance,
                  "insufficient balance: ${balance}, unable to deposit '${total_transfer}' from account '${a}' to '${t}'",
                  ("a", op.account)("t", contract_obj.id)("total_transfer", d.to_pretty_string(op.amount->amount))
                  ("balance", d.to_pretty_string(d.get_balance(op.account, op.amount->asset_id))));
    }
```

图 1 do_evaluate 函数部分源码截图



```
:3354,"ref_block_prefix":1622681499,"expiration":"2021-06-16T06:09:42","operations":[[60,{"fee":{"total":{"amount":0,"asset_id":0}},"account":25638,"contract_id":25638,"amou
t":"999999995000000","asset_id":0},"method_name":"withdraw","data":"2664000000000000166e000000000000020a1070000000000"}]],"signatures":[]}}]]}}}
    th_a  state.cpp:39 handle_reply
0 exception: unspecified
Execution error: Assert Exception: sufficient_balance: insufficient balance: 999999990 YOYO, unable to deposit '99999999500 YOYO' from account '25638' to '1.2.7'
    {"error":"Execution error: Assert Exception: sufficient_balance: insufficient balance: 999999990 YOYO, unable to deposit '99999999500 YOYO' from account '25638' to '1.2.
{"id":119,"error":{"code":10,"message":"Execution error: Assert Exception: sufficient_balance: insufficient balance: 999999990 YOYO, unable to deposit '99999999500 YOYO' fro
25638' to '1.2.7'","data":{"code":10,"name":"assert_exception","message":"Assert Exception","stack":[{"context":{"level":"error","file":"contract_evaluator.cpp","line":114,"
_evaluate","hostname":"","thread_name":"th_a","timestamp":"2021-06-16T06:09:13"},"format":"sufficient_balance: insufficient balance: ${balance}, unable to deposit '${total_t
rom account '${a}' to '${t}'","data":{"a":25638,"t":"1.2.7","total_transfer":"99999999500 YOYO","balance":"999999990 YOYO"}},{"context":{"level":"warn","file":"contract_eval
"line":131,"method":"do_evaluate","hostname":"","thread_name":"th_a","timestamp":"2021-06-16T06:09:13"},"format":"","data":{"op":{"fee":{"total":{"amount":0,"asset_id":0}},"
```

图 2 交易失败原因截图(1)



```
    th_a  state.cpp:39 handle_reply
0 exception: unspecified
Execution error: Assert Exception: sufficient_balance: insufficient balance: 2000 USDT, unable to deposit '10000000 YOYO' from account '370500683' to '1.2.25'
    {"error":"Execution error: Assert Exception: sufficient_balance: insufficient balance: 2000 USDT, unable to deposit '10000000 YOYO' from account '370500683' to '1.2.25'","data":{"i
d":80,"error":{"code":10,"message":"Execution error: Assert Exception: sufficient_balance: insufficient balance: 2000 USDT, unable to deposit '10000000 YOYO' from account '370500683' t
o '1.2.25'","data":{"code":10,"name":"assert_exception","message":"Assert Exception","stack":[{"context":{"level":"error","file":"contract_evaluator.cpp","line":114,"method":"do_evalua
te","hostname":"","thread_name":"th_a","timestamp":"2021-06-17T09:33:37"},"format":"sufficient_balance: insufficient balance: ${balance}, unable to deposit '${total_transfer}' from acc
ount '${a}' to '${t}'","data":{"a":370500683,"t":"1.2.25","total_transfer":"10000000 YOYO","balance":"2000 USDT"}},{"context":{"level":"warn","file":"contract_evaluator.cpp","line":131
,"method":"do_evaluate","hostname":"","thread_name":"th_a","timestamp":"2021-06-17T09:33:37"},"format":"","data":{"op":{"fee":{"total":{"amount":0,"asset_id":0}},"account":370500683,"c
ontract_id":370500683,"amount":{"amount":"1000000000000","asset_id":1},"method_name":"withdraw","data":"4b64151600000000166e00000000000001000000000000000000000000000000"}}},{"context":
{"level":"warn","file":"evaluator.cpp","line":49,"method":"start_evaluate","hostname":"","thread_name":"th_a","timestamp":"2021-06-17T09:33:37"},"format":"","data":{}},{"context":{"lev
el":"warn","file":"db_block.cpp","line":791,"method":"apply_operation","hostname":"","thread_name":"th_a","timestamp":"2021-06-17T09:33:37"},"format":"","data":{"op":[60,{"fee":{"total
":{"amount":0,"asset_id":0}},"account":370500683,"contract_id":370500683,"amount":{"amount":"1000000000000","asset_id":1},"method_name":"withdraw","data":"4b64151600000000166e000000000
0000100000000000000000000000000000"}]}},{"context":{"level":"warn","file":"db_block.cpp","line":773,"method":"_apply_transaction","hostname":"","thread_name":"th_a","timestamp":"2021-
06-17T09:33:37"},"format":"","data":{"trx":{"ref_block_num":6189,"ref_block_prefix":850313436,"expiration":"2021-06-17T09:34:06","operations":[[60,{"fee":{"total":{"amount":0,"asset_id
":0}},"account":370500683,"contract_id":370500683,"amount":{"amount":"1000000000000","asset_id":1},"method_name":"withdraw","data":"4b64151600000000166e00000000000001000000000000000000
000000000000"}]],"signatures":[]}}},{"context":{"level":"warn","file":"db_block.cpp","line":236,"method":"push_transaction","hostname":"","thread_name":"th_a","timestamp":"2021-06-17T0
9:33:37"},"format":"","data":{"trx":{"ref_block_num":6189,"ref_block_prefix":850313436,"expiration":"2021-06-17T09:34:06","operations":[[60,{"fee":{"total":{"amount":0,"asset_id":0}},"
account":370500683,"contract_id":370500683,"amount":{"amount":"1000000000000","asset_id":1},"method_name":"withdraw","data":"4b64151600000000166e00000000000001000000000000000000000000
00000"}]],"signatures":[]}}]}}}}
    th_a  state.cpp:39 handle_reply
```

图 3 交易失败原因截图(2)

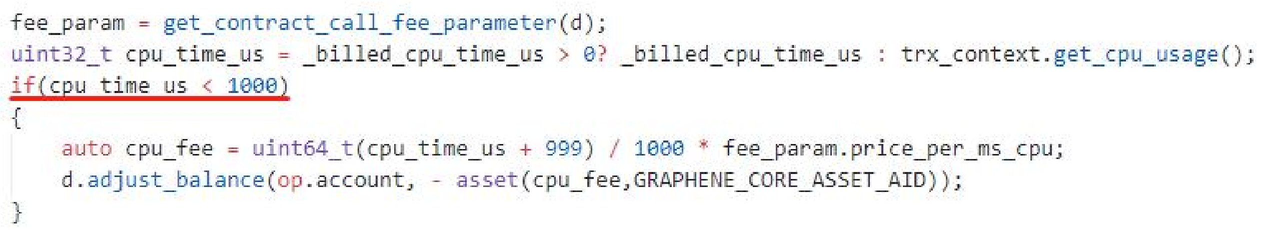
3.3 手续费

问题描述：(1) YOYOW 手续费相关配置错误，导致交易无手续费，可能导致节点资源消耗、DDoS 等问题；(2) 实现相似的转账功能的情况下，直接调用 transfer 相关命令转账的手续费超过了通过合约进行转账的手续费，建议修改合约手续费的经济模型；(3) 在合约升级过程中，建议修改数据手续费计算方式，数据手续费对增量进行收费。

修复结果：第(1)、(2)项已修复，第(3)项忽略。

3.4 合约调用手续费

问题描述： YOYOW 合约调用过程中，合约执行手续费相关逻辑实现有误，导致超过 1 ms 的交易不收取合约执行手续费。代码如图 4 所示：



```
fee_param = get_contract_call_fee_parameter(d);
uint32_t cpu_time_us = _billed_cpu_time_us > 0? _billed_cpu_time_us : trx_context.get_cpu_usage();
if(cpu_time_us < 1000)
{
    auto cpu_fee = uint64_t(cpu_time_us + 999) / 1000 * fee_param.price_per_ms_cpu;
    d.adjust_balance(op.account, - asset(cpu_fee,GRAPHENE_CORE_ASSET_AID));
}
```

图 4 contract_call_evaluator::do_apply 函数部分源码截图

修复结果：已修复。

4 审计总结

成都链安科技公司采用模拟攻击的方式对公链 YOYOW 的 WASM 模块安全性以及业务逻辑安全性等方面进行多维度全面的黑盒安全审计。所有在审计过程中发现的问题均已告知项目方进行修改。**经审计，YOYOW 公链的 WASM 模块通过所有黑盒攻击检测项，审计结果为通过(优)。**

成都链安
BEOSIN

官方网址

https://lianantech.com

电子邮箱

vaas@lianantech.com

微信公众号