密级 ★ | 项目组公开

[文档编号：ATHE-HYSJ-20190621]

智能合约审计报告

athenaexpect

主测人：

知道创宇

版本说明

修订人	修订内容	修订时间	版本号	审阅人
屈林成	编写文档	2019/06/21	V1.0	陈宇

文档信息

文档名称	文档版本	文档编号	保密级别
athenaexpect 智能合约审计报告	V1.0	【ATHE-HYSJ-20190621】	项目组公开

声明

创宇仅就本报告出具前已经发生或存在的事实出具本报告，并就此承担相应责任。对于出具以后发生或存在的事实，创宇无法判断其智能合约安全状况，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，仅基于信息提供者截至本报告出具时向创宇提供的文件和资料。创宇假设:已提供资料不存在缺失、被篡改、删减或隐瞒的情形。如已提供资料信息缺失、被篡改、删减、隐瞒或反映的情况与实际情况不符的，创宇对由此而导致的损失和不利影响不承担任何责任。

目录

1. 综述

本次报告有效测试时间是从 2019 年 06 月 21 日开始到 2019 年 06 月 2 1 日结束，在此期间针对 **athenaexpect 合约代码**的安全性和规范性进行审计并以此作为报告统计依据，合约审计详细内容见附录 A 注释。

此次测试中，知道创宇工程师对智能合约的常见漏洞（见第三章节）进行了全面的分析，未发现 athenaexpect 合约代码存在安全问题，故综合评定为 安全。

由于本次测试过程在非生产环境下进行，所有代码均为最新备份，测试过程均与相关接口人进行沟通，并在操作风险可控的情况下进行相关测试操作，以规避测试过程中的生产运营风险、代码安全风险。

本次测试的目标信息：

模块名称	
EOS 名称	athenaexpect
代码类型	EOS
代码语言	C++

本次项目测试人员信息：

姓名	邮箱/联系方式	职务
屈林成	qulc@knownsec.com	渗透测试工程师

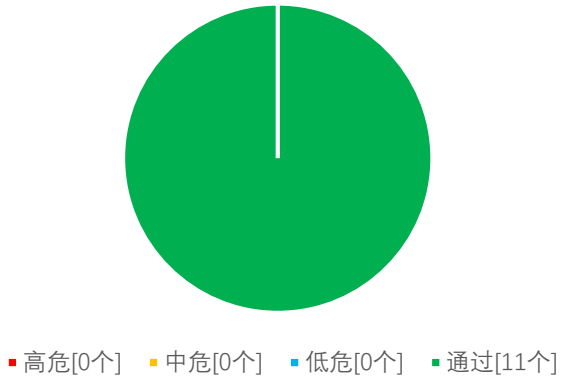
2. 代码漏洞分析

2.1. 漏洞等级分布

本次漏洞风险按等级统计:

漏洞风险等级个数统计表			
高危	中危	低危	通过
0	0	0	11

风险等级分布图



2.2. 审计结果汇总说明

审计结果			
测试项目	测试内容	状态	描述
智能合约	数值溢出检测	通过	经检测，不存在该安全问题。
	权限校验检测	通过	经检测，不存在该安全问题。
	Transfer 假通知检测	通过	经检测，不存在该安全问题。
	伪随机数安全	通过	经检测，不存在该安全问题。
	API 函数的不规范使用	通过	经检测，不存在该安全问题。
	返回值调用验证	通过	经检测，不存在该安全问题。
	状态表清除逻辑设计	通过	经检测，逻辑设计安全。
	开始共振逻辑设计	通过	经检测，逻辑设计安全。
	手动结束共振逻辑设计	通过	经检测，逻辑设计安全。。
	代币兑换逻辑设计	通过	经检测，逻辑设计安全。
	EOS 转换逻辑设计	通过	经检测，逻辑设计安全。

综合评定结果：通过

3. 代码审计结果分析

3.1. 数值溢出检测【安全】

智能合约中的算数问题是指整数溢出和整数下溢。

C++最多能处理 64 位的数字（2^64-1），最大数字增加 1 会溢出得到 0。同样，当数字为无符号类型时，0 减去 1 会下溢得到最大数字值。

整数溢出和下溢不是一种新类型的漏洞，但它们在智能合约中尤其危险。溢出情况会导致不正确的结果，特别是如果可能性未被预期，可能会影响程序的可靠性和安全性。

检测结果： 经检测，合约代码中使用了官方提供的 asset 数据结构描述代币，对于代币的算术运算同样利用 asset 完成，因为之前 asset 出现过“乘法溢出”问题，所以请项目方核查确认引用的是最新的 asset 文件。

安全建议：无。

3.2. 权限校验检测【安全】

权限校验缺陷是所有程序中都可能存在的安全风险，智能合约也同样会存在类似问题，在进行相关操作时，应严格判断函数入参和实际调用者是否一致，，可以使用 require_auth()进行校验。

检测结果：经检测，合约代码中不存在该安全问题。

安全建议：无。

3.3. Transfer 假通知检测【安全】

EOS 的合约可以通过 require_recipient 触发调用其他合约，设计这样的机制给合约的开发者提供了很大的便利性，但是也带了新的问题，如果在处理 transfer 通知时未校验 transfer 中的 to 是否为 self，将导致“假通知”漏洞。

检测结果：经检测，合约代码中不存在该安全问题。

```
void on_eos_transfer(name from, name to, asset quantity, std::string memo) {
    if (from == _self || to != _self) {
        return;
    }
    check(quantity.symbol == EOS_SYM, "we only need eos");  //knownsec // EOS symbol
    check(quantity.is_valid(), "invalid quantity" );
    check(quantity.amount > 0, "invalid quantity"); // knownsec //转账数量检测
    check(memo.size() <= 256, "memo has more than 256 bytes" );
    const auto sa = get_send_amount(from, quantity.amount);
    asset token_send(sa, TOK_SYM);
    action(permission_level{_self, "active"_n},
        TOKEN_CONTRACT, "transfer"_n,
        make_tuple(_self, from, token_send, string("diffraction"))
    ).send();
```

安全建议：无。

3.4. 伪随机数安全【安全】

EOS 合约中有可能需要使用到随机数，在使用随机数的生成算法生成随机数的过程中不应当引入可控或者可预测的随机数种子。

检测结果：经检测，合约中不涉及与随机数相关的代码。

安全建议：无。

3.5. API 函数的不规范使用【安全】

在调用 API 函数前应当严格检查传入的参数的类型以及大小是否正确，如 string_to_symbol(uint8_t，const char *)，第一个参数传入的整型变量需要小于

256，若使用 API 前未对输入进行检查，则有可能导致整型溢出等风险，带来严重后果。

检测结果：经检测，合约代码中不存在该安全问题。

安全建议：无。

3.6. 返回值调用验证【安全】

在 multi_index 中提供了 get 和 find 用于查询，其中 get 会检查数据是否查询成功，数据未找到则断言退出，而 find 不会检查数据查询情况，需要用户自行判断，如果缺少判断直接使用将会导致指针使用问题。

检测结果：经检测，合约代码中不存在该安全问题。

安全建议：无。

3.7. 状态表清除逻辑设计【安全】

合约中“状态表清除逻辑”设计需要满足以下需求：

1、检查当前 action 执行账户是否有权限，如果没有权限则终止当前 action。

2、判断状态表是否存在，如果存在，则清除状态表。

检测结果：经检测,逻辑设计无误。

```
void cleartable() {
    require_auth(permission_level{_self, "active"_n});  // knownsec //权限判断
    auto it = statetable.begin();
    while (it != statetable.end()) {
        it = statetable.erase(it);
    }
}
```

安全建议：无。

3.8. 开始共振逻辑设计【安全】

合约中“开始共振逻辑”设计需要满足以下需求：

1、检查当前 action 执行账户是否有权限，如果没有权限则终止当前 action。

2、获取状态表。

3、将状态表中的合约是否可用修改为 true。

检测结果：经检测,逻辑设计无误。

```
void start() {
    require_auth(permission_level{_self, "active"_n});
    const auto& stat = statetable.get(1);
    statetable.modify(stat, _self, [] (auto& row) {
        row.available = true;
    });
}
```

安全建议：无。

3.9. 手动结束共振逻辑设计【安全】

合约中“手动结束共振逻辑”设计需要满足以下需求：

1、检查当前 action 执行账户是否有权限，如果没有权限则终止当前 action。

2、获取状态表。

3、将状态表中的合约是否可用修改为 false。

检测结果：经检测，逻辑设计无误。

```
void stop() {
    require_auth(permission_level{_self, "active"_n});
    const auto& stat = statetable.get(1);
    statetable.modify(stat, _self, [] (auto& row) {
        row.available = false;
    });
}
```

安全建议：无。

3.10. 代币兑换逻辑设计【安全】

合约中“代币兑换逻辑”设计需要满足以下需求：

检查接收到的 eos 是否大于当前轮次剩余待接收 eos 数量 如果是则进入共振计算用户应该获取的 token 数量，如果不是则直接按照当前汇率兑换 token 数量。

检测结果：经检测，逻辑设计无误。

```
check(stat.available, "diffraction not yet started or has ended");
if (remain.amount > total_remain) {…}
while((int64_t)(eos_remain * rate) >= remain.amount) {  //knownsec //跳轮次
    eos_remain -= remain.amount / rate;
    sa += remain.amount;
    total_remain -=remain.amount;
    if (total_remain <= 0) {
        is_complete = true; //knownsec //共振完成
        break;
    }
    if (total_remain < LEVEL) {
        remain.amount = total_remain;
    } else {
        remain.amount = LEVEL;
    }
    round++;
    rate = rate / RATE_ACC;
}
if (is_complete) {…}
sa += eos_remain * rate;
remain.amount -= eos_remain * rate;
total_send.amount += sa;
total_received.amount += received;
check(total_send.amount <= SEND_LIMIT, "the amount to be sent exceeds the upper limit");
statetable.modify(stat, _self, [&](auto& row) {
    row.round_remain = remain;
    row.round = round;
    row.total_send = total_send;
    row.total_received = total_received;
    row.rate = rate;
});
return sa;
```

安全建议：无。

3.11. EOS 转换逻辑设计【安全】

合约中“EOS 转换逻辑”设计需要满足以下需求：

1、对传入的参数的合法性以及有效性进行严格的检查。

2、将 EOS 兑换为代币并发送给转入 EOS 的用户

检测结果：经检测，逻辑设计无误。

安全建议：无。

4. 附录 A：合约代码

本次测试代码来源:

diffraction_finalv2:

```
#include <eosio/eosio.hpp>
#include <eosio/asset.hpp>
#include <eosio/symbol.hpp>
#include <eosio/system.hpp>
#include <eosio/action.hpp>
using namespace eosio;
using namespace std;

class [[eosio::contract]] diffraction : public contract {
public:
    using contract::contract;
    diffraction(name receiver, name code, datastream<const char*> ds):contract(receiver, code, ds), statetable(_self, _self.value) {
        // each time the contract is called, check 'state' table first
        auto it = statetable.find(1);
        if (it == statetable.end()) {
            statetable.emplace(_self, [&]( auto& row ) {
                row.round_remain = asset(LEVEL, TOK_SYM);
                row.round = 1;
                row.total_send = asset(0, TOK_SYM);
                row.total_received = asset(0, EOS_SYM);
                row.rate = INIT_RATE;
                row.available = false;
            });
        }
    }
    // monitor all EOS sent to this contract
    [[eosio::on_notify("eosio.token::transfer")]]
    void on_eos_transfer(name from, name to, asset quantity, std::string memo) {
        if (from == _self || to != _self) {
            return;
        }
        check(quantity.symbol == EOS_SYM, "we only need eos");    //knownsec // EOS symbol
        check(quantity.is_valid(), "invalid quantity" );
        check(quantity.amount > 0, "invalid quantity");    //knownsec //转账数量检测
        check(memo.size() <= 256, "memo has more than 256 bytes" );
        const auto sa = get_send_amount(from, quantity.amount);
        asset token_send(sa, TOK_SYM);
        action(permission_level{_self, "active"_n},
            TOKEN_CONTRACT, "transfer"_n,
            make_tuple(_self, from, token_send, string("diffraction"))
        ).send();
    }
    // clear 'state' table
    [[eosio::action]]
    void cleartable() {
        require_auth(permission_level{_self, "active"_n});    //knownsec //权限判断
        auto it = statetable.begin();
        while (it != statetable.end()) {
            it = statetable.erase(it);
        }
    }
    // start diffraction
    [[eosio::action]]
    void start() {
        require_auth(permission_level{_self, "active"_n});
        const auto& stat = statetable.get(1);
        statetable.modify(stat, _self, [] (auto& row) {
            row.available = true;
        });
    }
    // stop diffraction
    [[eosio::action]]
    void stop() {
        require_auth(permission_level{_self, "active"_n});
        const auto& stat = statetable.get(1);
        statetable.modify(stat, _self, [] (auto& row) {
            row.available = false;
        });
    }

private:
    const symbol EOS_SYM = symbol(symbol_code("EOS"), 4); // EOS symbol
    const symbol TOK_SYM = symbol(symbol_code("ATHENA"), 4); // token symbol
    const name TOKEN_CONTRACT = "athenastoken"_n; // token contract
    const name EOS_CONTRACT = "eosio.token"_n; // eos contract
```

```
    const int64_t LEVEL = 3000000000;
    const double RATE_ACC = 1.015;
    const int64_t SEND_LIMIT = 790030000000;
    const double INIT_RATE = 100.0 / 3.0;

    struct [[eosio::table]] state {
        asset round_remain;
        int16_t round;
        asset total_send;
        asset total_received;
        double rate;
        bool available;
        uint64_t primary_key() const { return 1; }
    };
    typedef eosio::multi_index<"state"_n, state> state_index;
    state_index statetable;

    int64_t get_send_amount(name from, int64_t received) {
        const auto& stat = statetable.get(1);
        int64_t sa = 0; // send amount    //knownsec //待发送代币数量
        int64_t eos_remain = received;    //knownsec //待接收的 EOS 剩余数量
        auto rate = stat.rate;                //knownsec //当前轮次的兑换比例
        auto remain = stat.round_remain;//knownsec //当前轮次剩余待接收代币数量
        auto round = stat.round;              //knownsec //当前轮次，从 1 开始
        auto total_send = stat.total_send; // token send
        auto total_received = stat.total_received; // eos received
        auto total_remain = SEND_LIMIT - total_send.amount; //token remain
        bool is_complete = false;
        check(stat.available, "diffraction not yet started or has ended");
        if (remain.amount > total_remain) {
            remain.amount = total_remain;
        }
        while((int64_t)(eos_remain * rate) >= remain.amount) {    //knownsec //跳轮次
            eos_remain -= remain.amount / rate;
            sa += remain.amount;
            total_remain -=remain.amount;
            if (total_remain <= 0) {
                is_complete = true; //knownsec //共振完成
                break;
            }
            if (total_remain < LEVEL) {
                remain.amount = total_remain;
            } else {
                remain.amount = LEVEL;
            }
            round++;
            rate = rate / RATE_ACC;
        }
        if (is_complete) {
            //knownsec //将用户多余的 eos 返还给用户
            check(eos_remain < received, "refund eos must less then received");
            asset eos_send(eos_remain, EOS_SYM);
            action(permission_level{_self, "active"_n},
                EOS_CONTRACT, "transfer"_n,
                make_tuple(_self, from, eos_send, string("diffraction refund"))
            ).send();
            remain.amount = 0;
            total_send.amount += sa;
            total_received.amount += (received - eos_remain);
            check(total_send.amount == SEND_LIMIT, "total send amount overflow");
            statetable.modify(stat, _self, [&](auto& row) {
                row.round_remain = remain;
                row.round = round;
                row.total_send = total_send;
                row.rate = rate;
                row.available = false;
                row.total_received = total_received;
            });
            return sa;
        }
        int64_t s = eos_remain * rate;
        sa += s;
        remain.amount -= s;
        total_send.amount += sa;
        total_received.amount += received;
        check(total_send.amount <= SEND_LIMIT, "the amount to be sent exceeds the upper limit");
        statetable.modify(stat, _self, [&](auto& row) {
            row.round_remain = remain;
            row.round = round;
            row.total_send = total_send;
            row.total_received = total_received;
            row.rate = rate;
        });
        return sa;
    }
};
```

知道创宇

[咨询电话] +86(10)400 060 9587
[投诉电话] 13811527185
[邮　　箱] sec@knownsec.com
[网　　址] www.knownsec.com
[地　　址] 北京市朝阳区望京SOHO T3 A座15层

北京知道创宇信息技术有限公司