Security Classification	Open in the Project Group

SENIOR

Smart Contract Audit Report

athenaexpect

STATUS

SAFE

★ ★ ★ ★ ★

Test Engineer: 马林成

KNOWNSEC

Release notes

Revised one	Revise the	The revision	The version	review
Lincheng Qu	Written document	2019/06/21	V1.0	Yu Chen

Document information

The document name	Document	The document number	Level of
athenaexpect smart contract audit reports	V1.0	【ATHE－HYSJ－20190621】	Project team disclosure

The statement

Knownsec shall only issue this report with respect to the facts that have occurred or existed before the issuance of this report and assume the corresponding responsibilities. Knownsec cannot judge the security status of its intelligent contract and is not responsible for the fact that occurs or exists after the issuance. The security audit analysis and other contents of this report are only based on the documents and materials provided by information providers to Knownsec as of the issuance of this report. Knownsec assumes that no information provided is missing, altered, deleted or withheld. If the information provided is missing, altered, deleted, concealed or reflected in a way inconsistent with the actual situation, Knownsec shall not be liable for the loss and adverse effects caused thereby.

directory

1. review

The effective testing period of this report is from June 21, 2019 to June 21, 2019. During this period, an audit of the security and normality of **athenaexpect contract code** is conducted and used as the statistical basis for the report. Details of contract audit are shown in appendix A.

In this test, Knownsec engineers conducted a comprehensive analysis of common vulnerabilities in intelligent contracts (see chapter 3). They did not find any security problems in **athenaexpect contract code**, so they comprehensively rated it as **safe**.

Since this test is conducted in a non-production environment, all codes are the latest backup, and the test process is communicated with relevant interface personnel, and relevant test operations are conducted under controllable operational risks to avoid production and operation risks and code safety risks in the test process.

Target information of this test:

The name of the module	
The name of the EOS	athenaexpect
Code type	EOS
Code language	C + +

Information of testers for this project:

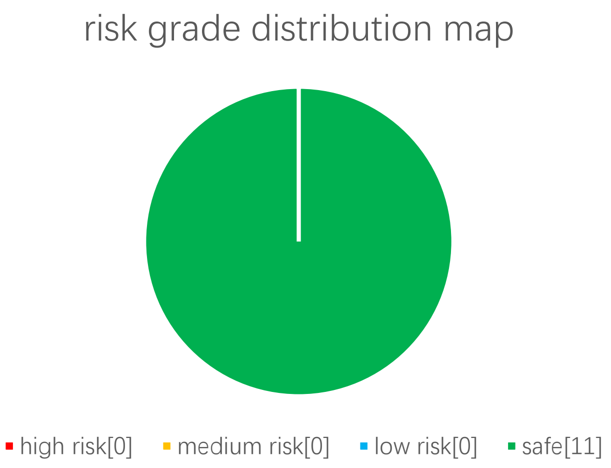
The name	Email/contact information	position
Lincheng Qu	qulc@knownsec.com	Penetration test engineer

2. Code vulnerability analysis

2.1. Vulnerability grade distribution

The vulnerability risk is counted by grade:

Statistical table of vulnerability risk levels			
High risk	Medium risk	Low risk	Safe
0	0	0	11



2.2. Summary of audit results

The audit results			
Test project	The test content	state	describe
Intelligent contract	Numerical overflow detection	pass	After testing, there is no such safety problem.
	Permission check check	pass	After testing, there is no such safety problem.
	Transfer false notice detection	pass	After testing, there is no such safety problem.
	Pseudo-random Numbers are safe	pass	After testing, there is no such safety problem.
	Nonstandard use of API functions	pass	After testing, there is no such safety problem.
	The return value invokes validation	pass	After testing, there is no such safety problem.
	The state table clears the logical design	pass	After testing, the logic design is safe.
	Start resonance logic design	pass	After testing, the logic design is safe.
	Manually end the resonance logic design	pass	After testing, the logic design is safe.
	Token exchange logic design	pass	After testing, the logic design is safe.
	EOS transformation logic design	pass	After testing, the logic design is safe.

Comprehensive evaluation result: pass

3. Code audit results analysis

3.1. Numerical overflow detection [safe]

The arithmetic problems in smart contracts are integer overflow and integer underflow.

C++ can handle up to 64 digits (2^64-1). Increasing the maximum number by 1 will overflow to 0.Similarly, when a number is of unsigned type, 0 minus 1 will overflow to get the maximum numeric value.

Integer overflow and underflow are not new types of vulnerabilities, but they are particularly dangerous in intelligent contracts. Overflow situations can lead to incorrect results, especially if the possibility is not anticipated, which can affect the reliability and security of the program.

Test results: according to the test, the official asset data structure description token is used in the contract code, and the arithmetic operation of token is also completed by using asset. Because "multiplication overflow" has occurred in asset before, the project party is requested to check and confirm that the reference is the latest asset file.

Safety advice: none.

3.2. Authority check and test [safe]

Permission checking defects are a security risk that can exist in all programs, as well as in intelligent contracts, and should be strictly determined to match the actual callers, which can be verified using require_auth().

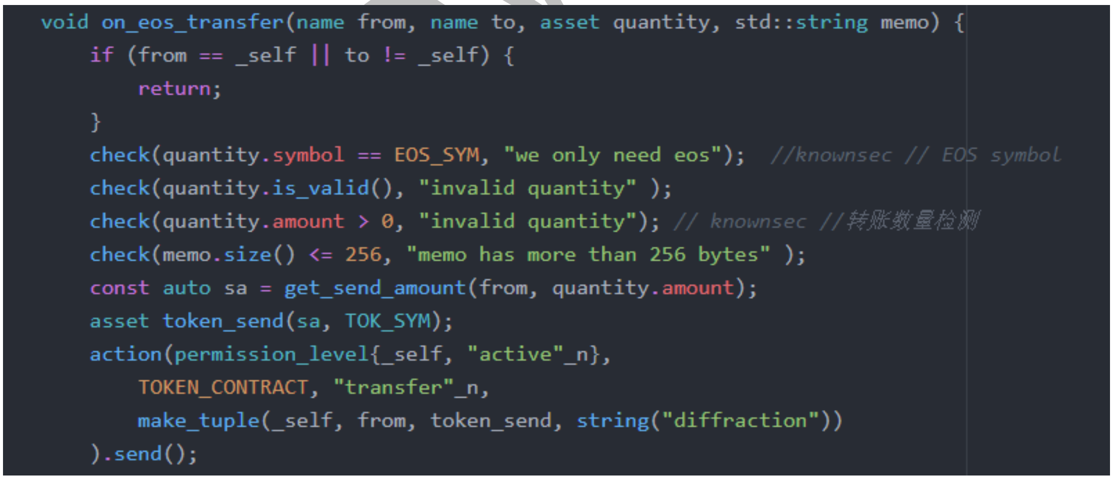
Test results: after testing, there is no such security problem in the contract code.

Safety advice: none.

3.3. Detection of Transfer false notice[safe]

Designing the mechanism that EOS contracts can invoke other contracts via require_recipient provides a great convenience to contract developers, but it also introduces a new problem: if the to in the transfer notification is not verified to be self, the "fake notification" vulnerability will result.

Test results: after testing, there is no such security problem in the contract code.



```
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();
```

Safety advice: none.

3.4. Pseudo-random number security[safe]

Random Numbers may be required in the EOS contract, and controllable or predictable random number seeds should not be introduced in the process of generating random Numbers using random number generation algorithm.

Test results: after testing, the contract does not involve random number related code.

Safety advice: none.

3.5. Non-standard use of API function [safe]

Before calling the API function, the type and size of the passed parameter should be strictly checked. For example, string_to_symbol(uint8_t, const char *), the input variable of the first parameter should be less than 256. If the input is not checked before using the API, the risk of integer overflow may be caused, bringing serious consequences.

Test results: after testing, there is no such security problem in the contract code.

Safety advice: none.

3.6. Return value call verification [safe]

In multi_index, get and find are provided for query, where get will check whether the data is successfully queried, and if the data is not found, the assertion exits, while find will not check the data query, which requires users to determine by themselves. Direct use of the pointer will cause problems if there is no judgment.

Test results: after testing, there is no such security problem in the contract code.

Safety advice: none.

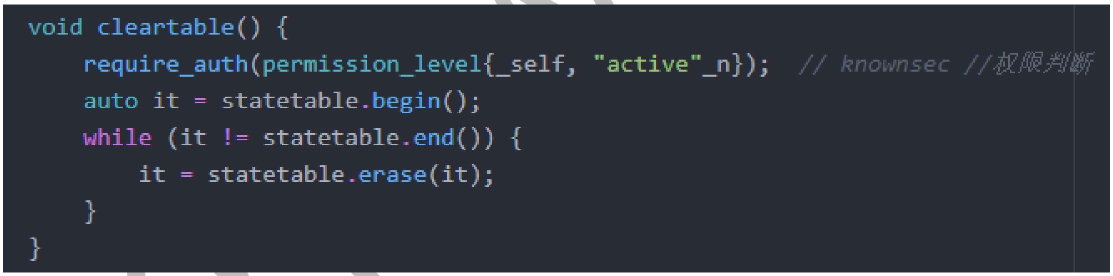
3.7. State table clear logical design [safe]

The design of "state table cleaning logic" in the contract shall meet the following requirements:

1. Check whether the current action execution account has permissions, if not, terminate the current action.

2. Determine whether the state table exists, and if so, clear the state table.

Test result: the logic design is correct.



```
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);
    }
}
```

Safety advice: none.

3.8. Starting resonance logic design [safe]

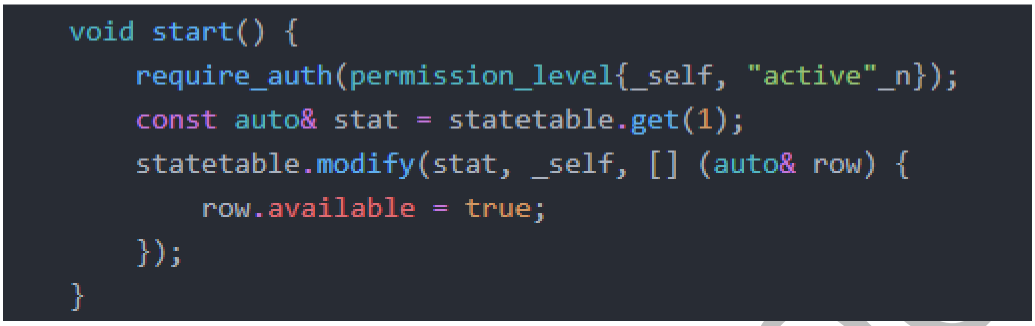
The design of "start resonance logic" in the contract shall meet the following requirements:

1. Check whether the current action execution account has permissions, if not, terminate the current action.

2. Obtain the status table.

3. Whether the contract in the status table can be modified to true.

Test result: the logic design is correct.



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

Safety advice: none.

3.9. Manually end resonance logic design [safe]

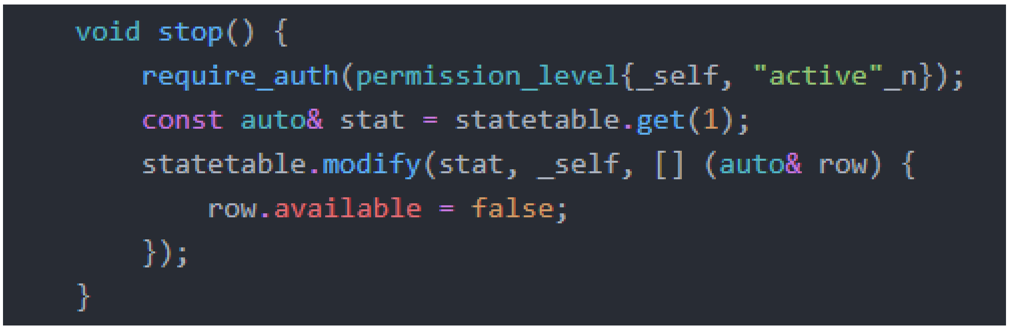
The design of "manually ending resonance logic" in the contract shall meet the following requirements:

1. Check whether the current action execution account has permissions, if not, terminate the current action.

2. Obtain the status table.

3. Whether the contract in the status table can be modified to false.

Test result: the logic design is correct.



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

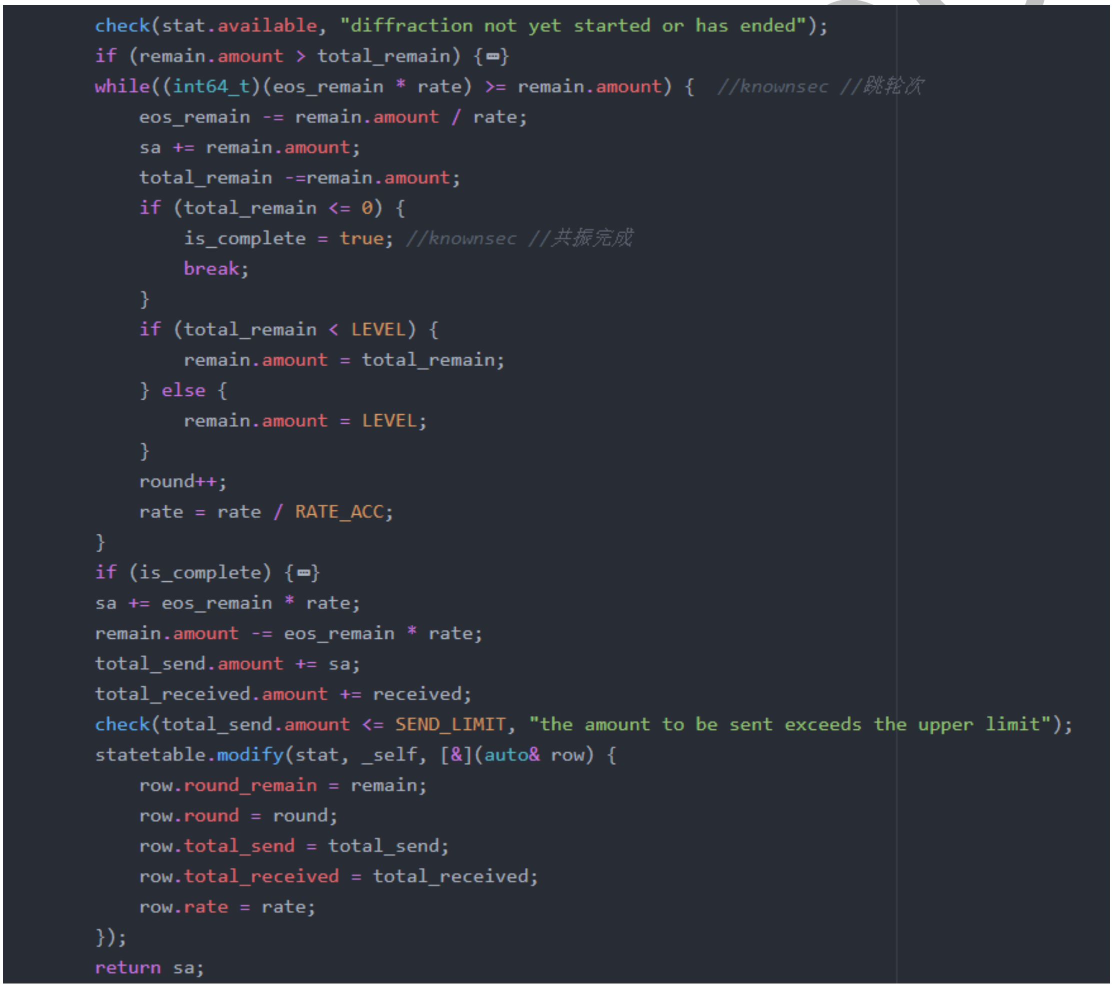
Safety advice: none.

3.10. Logic design of token exchange [safe]

The design of "token exchange logic" in the contract shall meet the following requirements:

Check whether the number of eos received is greater than the number of eos to be received in the current round.

Test result: the logic design is correct.



```
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;
```

Safety advice: none.

3.11. EOS transformation logic design [safe]

The design of "EOS transformation logic" in the contract shall meet the following requirements:

1. Strictly check the validity and validity of the incoming parameters.
2. Exchange EOS for tokens and send them to users who have transferred EOS

Test result: the logic design is correct.

Safety advice: none.

4. Appendix A: contract code

Source of this test code:

Diffraction_finalv2:

```
# include < eosio/eosio HPP >
# include < eosio/asset. HPP >
# include < eosio/symbol. HPP >
# include < eosio/system. HPP >
# include < eosio/action. HPP >
Using the namespace eosio;
Using namespace STD.

Diffraction class [[eosio: : contract]] : public contract {
Public:
        Using contract: contract;
        Diffraction (name receiver, name code, datastream < const char * > ds) : contract (receiver, code, ds),
statetable (value) of love, love. {
                // 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. The round = 1;
                                Row. Total_send = asset (0, TOK_SYM);
                                Row. Total_received = asset (0, EOS_SYM);
                                Row. Rate = INIT_RATE;
                                Row. The 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! = {love)
                        The return;
                }
                Check (quantity. Symbol == EOS_SYM, "we only need eos");       / / knownsec / / EOS symbol
                Check (quantity. Is_valid (), "invalid quantity");
                Check (quantity. Amount >, "invalid quantity");         //knownsec // transfer quantity detection
                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);
                The action (permission_level {love, "active" _n},
                        "Transfer" _n TOKEN_CONTRACT,
                        Make_tuple (love, from token_send, string (" diffraction "))
                ). The send ();
        }
        / / the clear 'state' table
        [[eosio: : action]]
        Void cleartable () {
                Require_auth (permission_level {love, "active" _n});       //knownsec // authority judgment
                Auto it = statetable. The begin ();
                While (it! = statetable. End ()) {
                        It = statetable. Erase (it);
                }
        }
        / / start diffraction
        [[eosio: : action]]
        Void the start () {
                Require_auth (permission_level {love, "active" _n});
                Const auto& stat = statetable.get(1);
                Modify (stat, _self, [] (auto& row) {
                        Row. The available = true;
                });
        }
        / / stop diffraction
        [[eosio: : action]]
        Void the stop () {
                Require_auth (permission_level {love, "active" _n});
                Const auto& stat = statetable.get(1);
                Modify (stat, _self, [] (auto& row) {
                        Row. The 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 = 35;

        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 < _n "state", the 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 // the amount of tokens to be sent
                Int64_t eos_remain = received;        //knownsec // the remaining amount of EOS to be received
                Auto rate = stat. Rate;                  //knownsec // the exchange ratio of the current round
                Auto remain = stat. Round_remain; //knownsec // the amount of remaining tokens to be received in the
current round
                Auto round = stat. Round;                  //knownsec // current round, starting from 1
                Auto total_send = stat. Total_send;      / / token to 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 (remains. Amount > total_remains) {
                        Remain. Amount = total_remain;
                }
                While ((int64_t)(eos_remain * rate) >= remain. Amount) {//knownsec // hop cycles
                        Eos_retain -= retain. Amount/rate;
                        Sa + = remain. Amount;
                        Total_remain - = remain. Amount;
                        If (total_remain <= 0) {
                                Is_complete = true;      //knownsec // resonance completed
                                Break;
                        }
                        If (total_remain < LEVEL) {
                                Remain. Amount = total_remain;
                        } else {
                                Remain. Amount = LEVEL;
                        }
                        Round++;
                        Rate = rate/RATE_ACC;
                }
                If (is_complete) {
                        //knownsec // return excess eos to users
                        Check (eos_remain < received, "refund eos must less then received");
                        Asset eos_send (eos_remain EOS_SYM);
                        The action (permission_level {love, "active" _n},
                                "Transfer" _n EOS_CONTRACT,
                                Make_tuple (love, from eos_send, string (" diffraction refund "))
                        ). The send ();
                        Remain. The amount = 0;
                        Total_send. Amount + = sa;
                        Total_received. Amount += (received - eos_remain);
                        Check (total_send. Amount == SEND_LIMIT, "total send amount overflow");
                        Modify (stat, _self, [&](auto& row) {
                                Row. Round_remain = remain;
                                Row. Round = round;
                                Row. Total_send = total_send;
                                Row. Rate = rate;
                                Row. The available = false;
                                Row. Total_received = total_received;
                        });
                        Return the 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");
                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 the sa;
        }
};
```

KNOWNSEC

【 Advisory telephone 】+86(10)400 060 9587
【 Complaints Hotline 】13811527185
【 E-mail 】sec@knownsec.com
【 Website 】www.knownsec.com
【 Address 】wangjing soho T3-A15,Chaoyang District, Beijing

Beijing KnownSec Information Technology Co., Ltd.