KNOWNSEC

Degree of secrets ★ | Project team open

Smart contract audit report

payaso

Security status

Security

★ ★ ★ ★ ★

Chief test Officer: *Knownsec blockchain security team*

Version Summary

Content	Date	Version
Editing Document	20201012	V1.0

Report Information

Title	Version	Document Number	Type
payaso contract audit report	V1.0	【PAYASO-ZNHY-20201012】	Open to project team

Copyright Notice

Table of Contents

1. Introduction

The effective test time of this report is from October 12, 2020 to October 12, 2020. During this period, the security and standardization of **the smart contract code of the payaso** will be audited and used as the statistical basis for the report.

In this audit report, engineers conducted a comprehensive analysis of the common vulnerabilities of smart contracts (Chapter 3).No related vulnerabilities found, **the smart contract code of the paysao** is comprehensively assessed as **SAFE**.

Results of this smart contract security audit：SAFE

Since the testing is under non-production environment, all codes are the latest version. In addition, the testing process is communicated with the relevant engineer, and testing operations are carried out under the controllable operational risk to avoid production during the testing process, such as: Operational risk, code security risk.

Target information of the payaso audit:

Target information	
Token name	payaso
Code type	Eth smart contract code
Code language	solidity

Contract documents and hash:

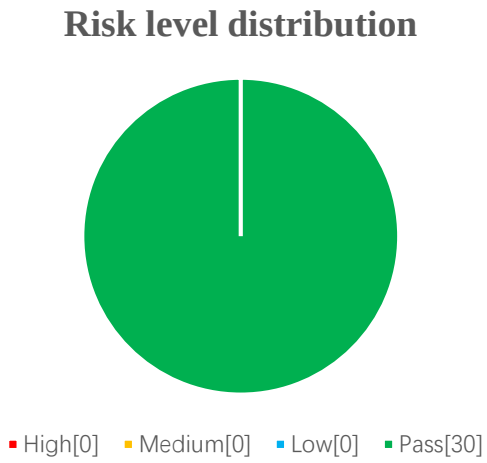
Contract documents	MD5
payaso.sol	bdec036a2080f4a7b9a59c1d4a8852a3

2. Code vulnerability analysis

Vulnerability Level Distribution

Vulnerability risk statistics by level：

Vulnerability risk level statistics table			
High	Medium	Low	Pass
0	0	0	30



Audit Result

Result of audit			
Audit Target	**Audit**	**Status**	**Audit Description**
Business security testing	**Cost calculation function**	Pass	After testing, there is no such safety vulnerability.
	Policy purchase function	Pass	After testing, there is no such safety vulnerability.
	Waive policy function	Pass	After testing, there is no such safety vulnerability.
Basic code vulnerability detection	**Compiler version security**	Pass	After testing, there is no such safety vulnerability.
	Redundant code	Pass	After testing, there is no such safety vulnerability.
	Use of safe arithmetic library	Pass	After testing, there is no such safety vulnerability.
	Not recommended encoding	Pass	After testing, there is no such safety vulnerability.
	Reasonable use of require/assert	Pass	After testing, there is no such safety vulnerability.
	fallback function safety	Pass	After testing, there is no such safety vulnerability.
	tx.orgin authentication	Pass	After testing, there is no such safety vulnerability.
	Owner permission control	Pass	After testing, there is no such safety vulnerability.

	Gas consumption detection	Pass	After testing, there is no such safety vulnerability.
	call injection attack	Pass	After testing, there is no such safety vulnerability.
	Low-level function safety	Pass	After testing, there is no such safety vulnerability.
	Vulnerability of additional token issuance	Pass	After testing, there is no such safety vulnerability.
	Access control defect detection	Pass	After testing, there is no such safety vulnerability.
	Numerical overflow detection	Pass	After testing, there is no such safety vulnerability.
	Arithmetic accuracy error	Pass	After testing, there is no such safety vulnerability.
	Wrong use of random number detection	Pass	After testing, there is no such safety vulnerability.
	Unsafe interface use	Pass	After testing, there is no such safety vulnerability.
	Variable coverage	Pass	After testing, there is no such safety vulnerability.
	Uninitialized storage pointer	Pass	After testing, there is no such safety vulnerability.
	Return value call verification	Pass	After testing, there is no such safety vulnerability.
	Transaction order dependency detection	Pass	After testing, there is no such safety vulnerability.

	Timestamp dependent attack	Pass	After testing, there is no such safety vulnerability.
	Denial of service attack detection	Pass	After testing, there is no such safety vulnerability.
	Fake recharge vulnerability detection	Pass	After testing, there is no such safety vulnerability.
	Reentry attack detection	Pass	After testing, there is no such safety vulnerability.
	Replay attack detection	Pass	After testing, there is no such safety vulnerability.
	Rearrangement attack detection	Pass	After testing, there is no such safety vulnerability.

3. Analysis of code audit results

3.1. Expense calculation function【PASS】

Audit analysis: The payaso contract has a fee calculation function, and the design is reasonable. Lines 1758 to 1766 of the payaso.sol file are used correctly.

```
function calcFee(uint volume) public view returns (address recipient, uint fee) {
        uint feeRate = OptionFactory(factory).getConfig(_feeRate_);
        recipient = address(OptionFactory(factory).getConfig(_feeRecipient_));

        if(feeRate != 0 && recipient != address(0))
            fee = volume.mul(feeRate).div(1 ether);
        else
            fee = 0;
    }
```

Recommendation： nothing.

3.2. Policy purchase function【PASS】

Audit analysis: There is a policy purchase function in the payaso contract to ensure the normal purchase of the policy and the design is reasonable. Lines 1970 to 1991 of the payaso.sol file are used correctly.

```
function buy(uint askID, uint volume) virtual public returns (uint bidID, uint vol, uint amt) {
        require(asks[askID].seller != address(0), 'Nonexistent ask order');
        vol = volume;
        if(vol > asks[askID].remain)
            vol = asks[askID].remain;

        amt = vol.mul(asks[askID].price).div(1 ether);
        address                                    weth                                    =
```

```
IUniswapV2Router01(OptionFactory(factory).getConfig(_uniswapRounter_)).WETH();
        address settleToken = asks[askID].settleToken;
        if(settleToken != weth || address(this).balance < amt)
            IERC20(settleToken).safeTransferFrom(msg.sender, asks[askID].seller, amt);
        else
            payable(asks[askID].seller).transfer(amt);

        asks[askID].remain = asks[askID].remain.sub(vol);
        IERC20(asks[askID].long).safeTransfer(msg.sender, vol);

        bidID = bidsN++;
        bids[bidID] = Bid(bidID, askID, msg.sender, vol, amt, vol);

        emit Buy(bidID, askID, msg.sender, vol, amt);
    }
```

Recommendation： nothing.

3.3. Waiver of policy function【PASS】

Audit analysis: There is a policy purchase function in the payaso contract to ensure the normal purchase of the policy and the design is reasonable. Lines 1970 to 1991 of the payaso.sol file are used correctly.

```
function waive(uint bidID, uint volume) virtual public returns (uint vol) {
        vol = volume;
        if(vol > bids[bidID].remain)
            vol = bids[bidID].remain;
        bids[bidID].remain = bids[bidID].remain.sub(vol);
```

```
        address long = asks[bids[bidID].askID].long;
        IERC20(long).safeTransferFrom(msg.sender, address(this), vol);
        LongOption(long).burn(vol);

        _settleReward(msg.sender, LongOption(long).collateral(),
LongOption(long).underlying(), vol);
        emit Waive(bidID, msg.sender, vol);
    }
```

Recommendation： nothing.

4. Basic code vulnerability detection

4.1. Compiler version security【PASS】

Check whether a safe compiler version is used in the contract code implementation.

Audit result: After testing, the smart contract code has a compiler version 0.5.15 or higher, and there is no such security problem.

```
pragma solidity ^0.6.0;
```

Recommendation： nothing.

4.2. Redundant code【PASS】

Check whether the contract code implementation contains redundant code.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.3. Use of safe arithmetic library【PASS】

Check whether the SafeMath safe arithmetic library is used in the contract code implementation.

Audit result: After testing, the SafeMath safe arithmetic library has been used in the smart contract code, and there is no such security problem.

Recommendation： nothing.

4.4. Not recommended encoding【PASS】

Check whether there is an encoding method that is not officially recommended or abandoned in the contract code implementation

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.5. Reasonable use of require/assert【PASS】

Check the rationality of the use of require and assert statements in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.6. Fallback function safety【PASS】

Check whether the fallback function is used correctly in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.7. tx.origin authentication【PASS】

tx.origin is a global variable of Solidity that traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication in a smart contract makes the contract vulnerable to attacks like phishing.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.8. Owner permission control【PASS】

Check whether the owner in the contract code implementation has excessive authority. For example, arbitrarily modify other account balances, etc.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.9. Gas consumption detection【PASS】

Check whether the consumption of gas exceeds the maximum block limit.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.10. call injection attack【PASS】

When the call function is called, strict permission control should be done, or the function called by the call should be written dead.

Audit result: After testing, the smart contract does not use the call function, and this vulnerability does not exist.

Recommendation： nothing.

4.11. Low-level function safety【PASS】

Check whether there are security vulnerabilities in the use of low-level functions (call/delegatecall) in the contract code implementation

The execution context of the call function is in the called contract; the execution context of the delegatecall function is in the contract that currently calls the function.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.12. Vulnerability of additional token issuance【PASS】

Check whether there is a function that may increase the total amount of tokens in the token contract after initializing the total amount of tokens.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.13. Access control defect detection【PASS】

Different functions in the contract should set reasonable permissions.

Check whether each function in the contract correctly uses keywords such as public and private for visibility modification, check whether the contract is correctly defined and use modifier to restrict access to key functions to avoid problems caused by unauthorized access.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.14. Numerical overflow detection【PASS】

The arithmetic problems in smart contracts refer to integer overflow and integer underflow.

Solidity can handle up to 256-bit numbers (2^256-1). If the maximum number increases by 1, it will overflow to 0. Similarly, when the number is an unsigned type, 0 minus 1 will underflow to get the maximum digital value.

Integer overflow and underflow are not a new type of vulnerability, but they are especially dangerous in smart contracts. Overflow conditions can lead to incorrect results, especially if the possibility is not expected, which may affect the reliability and safety of the program.

Audit result: After testing, the security problem does not exist in the smart

contract code.

Recommendation： nothing.

4.15. Arithmetic accuracy error【PASS】

As a programming language, Solidity has data structure design similar to ordinary programming languages, such as variables, constants, functions, arrays, functions, structures, etc. There is also a big difference between Solidity and ordinary programming languages-Solidity does not float Point type, and all the numerical calculation results of Solidity will only be integers, there will be no decimals, and it is not allowed to define decimal type data. Numerical calculations in the contract are indispensable, and the design of numerical calculations may cause relative errors. For example, the same level of calculations: 5/2*10=20, and 5*10/2=25, resulting in errors, which are larger in data The error will be larger and more obvious.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.16. Incorrect use of random numbers【PASS】

Smart contracts may need to use random numbers. Although the functions and variables provided by Solidity can access values that are obviously unpredictable, such as block.number and block.timestamp, they are usually more public than they appear or are affected by miners. These random numbers are predictable to a certain extent, so

malicious users can usually copy it and rely on its unpredictability to attack the function.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.17. Unsafe interface usage【PASS】

Check whether unsafe interfaces are used in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.18. Variable coverage【PASS】

Check whether there are security issues caused by variable coverage in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.19. Uninitialized storage pointer【PASS】

In solidity, a special data structure is allowed to be a struct structure, and the local variables in the function are stored in storage or memory by default.

The existence of storage (memory) and memory (memory) are two different

concepts. Solidity allows pointers to point to an uninitialized reference, while uninitialized local storage will cause variables to point to other storage variables, leading to variable coverage, or even more serious As a consequence, you should avoid initializing struct variables in functions during development.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.20. Return value call verification【PASS】

This problem mostly occurs in smart contracts related to currency transfer, so it is also called silent failed delivery or unchecked delivery.

In Solidity, there are transfer(), send(), call.value() and other currency transfer methods, which can all be used to send Ether to an address. The difference is: When the transfer fails, it will be thrown and the state will be rolled back; Only 2300gas will be passed for calling to prevent reentry attacks; false will be returned when send fails; only 2300gas will be passed for calling to prevent reentry attacks; false will be returned when call.value fails to be sent; all available gas will be passed for calling (can be Limit by passing in gas_value parameters), which cannot effectively prevent reentry attacks.

If the return value of the above send and call.value transfer functions is not checked in the code, the contract will continue to execute the following code, which may lead to unexpected results due to Ether sending failure.

Audit result: After testing, the security problem does not exist in the smart

contract code.

Recommendation： nothing.

4.21. Transaction order dependency【PASS】

Since miners always get gas fees through codes that represent externally owned addresses (EOA), users can specify higher fees for faster transactions. Since the Ethereum blockchain is public, everyone can see the content of other people's pending transactions. This means that if a user submits a valuable solution, a malicious user can steal the solution and copy its transaction at a higher fee to preempt the original solution.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.22. Timestamp dependency attack【PASS】

The timestamp of the data block usually uses the local time of the miner, and this time can fluctuate in the range of about 900 seconds. When other nodes accept a new block, it only needs to verify whether the timestamp is later than the previous block and The error with local time is within 900 seconds. A miner can profit from it by setting the timestamp of the block to satisfy the conditions that are beneficial to him as much as possible.

Check whether there are key functions that depend on the timestamp in the contract code implementation.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.23. Denial of service attack【PASS】

In the world of Ethereum, denial of service is fatal, and a smart contract that has suffered this type of attack may never be able to return to its normal working state. There may be many reasons for the denial of service of the smart contract, including malicious behavior as the transaction recipient, artificially increasing the gas required for computing functions to cause gas exhaustion, abusing access control to access the private component of the smart contract, using confusion and negligence, etc. Wait.

Audit result: After testing, the security problem does not exist in the smart contract code.

Recommendation： nothing.

4.24. Fake recharge vulnerability【PASS】

The transfer function of the token contract uses the if judgment method to check the balance of the transfer initiator (msg.sender). When balances[msg.sender] <value, enter the else logic part and return false, and finally no exception is thrown. We believe that only if/else this kind of gentle judgment method is an imprecise coding method in sensitive function scenarios such as transfer.

Audit result: After testing, the security problem does not exist in the smart

contract code.

Recommendation： nothing.

4.25. Reentry attack detection【PASS】

Re-entry vulnerability is the most famous Ethereum smart contract vulnerability, which caused the fork of Ethereum(The DAO hack).

The **call.value()** function in Solidity consumes all the gas it receives when it is used to send Ether. When the **call.value()** function to send Ether occurs before the actual reduction of the sender's account balance, There is a risk of reentry attacks.

Audit results： After auditing, the vulnerability does not exist in the smart contract code.

Recommendation： nothing.

4.26. Replay attack detection【PASS】

If the contract involves the need for entrusted management, attention should be paid to the non-reusability of verification to avoid replay attacks

In the asset management system, there are often cases of entrusted management. The principal assigns assets to the trustee for management, and the principal pays a certain fee to the trustee. This business scenario is also common in smart contracts.

Audit results： After testing, the smart contract does not use the call function, and this vulnerability does not exist.

Recommendation： nothing.

4.27. Rearrangement attack detection【PASS】

A rearrangement attack refers to a miner or other party trying to "compete" with smart contract participants by inserting their own information into a list or mapping, so that the attacker has the opportunity to store their own information in the contract. in.

Audit results： After auditing, the vulnerability does not exist in the smart contract code.

Recommendation： nothing.

5. Appendix A：Contract code

Source code:

```
contract Constants {
    bytes32 internal constant _LongOption_          = 'LongOption';
    bytes32 internal constant _ShortOption_         = 'ShortOption';
    bytes32 internal constant _feeRate_             = 'feeRate';
    bytes32 internal constant _feeRecipient_        = 'feeRecipient';
    bytes32 internal constant _uniswapRounter_      = 'uniswapRounter';
}

contract OptionFactory is Configurable, Constants {
    using SafeERC20 for IERC20;
    using SafeMath for uint;

    mapping(bytes32 => address) public productImplementations;
    mapping(address => mapping(address => mapping(address => mapping(uint => mapping(uint => address))))) public longs;
    mapping(address => mapping(address => mapping(address => mapping(uint => mapping(uint => address))))) public shorts;
    address[] public allLongs;
    address[] public allShorts;

    function length() public view returns (uint) {
        return allLongs.length;
    }

    function initialize(address _governor, address _implLongOption, address _implShortOption, address _feeRecipient) public initializer {
        super.initialize(_governor);
        productImplementations[_LongOption_]      = _implLongOption;
        productImplementations[_ShortOption_]     = _implShortOption;
        config[_feeRate_]                         = 0.002 ether;            // 0.2%
        config[_feeRecipient_]                    = uint(_feeRecipient);
        config[_uniswapRounter_]                  = uint(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);
    }

    function upgradeProductImplementationsTo(address _implLongOption, address _implShortOption) external governance {
        productImplementations[_LongOption_] = _implLongOption;
        productImplementations[_ShortOption_] = _implShortOption;
    }

    function createOption(bool _private, address _collateral, address _underlying, uint _strikePrice, uint _expiry) public returns (address long, address short) {
        require(_collateral != _underlying, 'Payaso: IDENTICAL_ADDRESSES');
        require(_collateral != address(0) && _underlying != address(0), 'Payaso: ZERO_ADDRESS');
        require(_strikePrice != 0, 'Payaso: ZERO_STRIKE_PRICE');
        require(_expiry > now, 'Cannot create an expired option');

        address creator = _private ? tx.origin : address(0);
        require(longs[creator][_collateral][_underlying][_strikePrice][_expiry] == address(0), 'Payaso: SHORT_PROXY_EXISTS');        // single check is sufficient

        bytes32 salt = keccak256(abi.encodePacked(creator, _collateral, _underlying, _strikePrice, _expiry));

        bytes memory bytecode = type(LongProxy).creationCode;
        assembly {
            long := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        InitializableProductProxy(payable(long)).initialize(address(this), abi.encodeWithSignature('initialize(address,address,address,uint256,uint256)', creator, _collateral, _underlying, _strikePrice, _expiry));

        bytecode = type(ShortProxy).creationCode;
        assembly {
            short := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        InitializableProductProxy(payable(short)).initialize(address(this), abi.encodeWithSignature('initialize(address,address,address,uint256,uint256)', creator, _collateral, _underlying, _strikePrice, _expiry));

        longs [creator][_collateral][_underlying][_strikePrice][_expiry] = long;
        shorts[creator][_collateral][_underlying][_strikePrice][_expiry] = short;
        allLongs.push(long);
        allShorts.push(short);
        emit OptionCreated(creator, _collateral, _underlying, _strikePrice, _expiry, long, short, allLongs.length);
    }
    event OptionCreated(address indexed creator, address indexed _collateral, address indexed _underlying, uint
```

```
_strikePrice, uint _expiry, address long, address short, uint count);

    function mint(bool _private, address _collateral, address _underlying, uint _strikePrice, uint _expiry, uint volume) public {
        address creator = _private ? tx.origin : address(0);
        address long  = longs [creator][_collateral][_underlying][_strikePrice][_expiry];
        address short = shorts[creator][_collateral][_underlying][_strikePrice][_expiry];
        if(short == address(0)) // single check is sufficient
            (long, short) = createOption(_private, _collateral, _underlying, _strikePrice, _expiry);

        IERC20(_collateral).safeTransferFrom(msg.sender, short, volume);
        ShortOption(short).mint_(msg.sender, volume);
        LongOption(long).mint_(msg.sender, volume);

        emit Mint(msg.sender, _private, _collateral, _underlying, _strikePrice, _expiry, volume);
    }
    event Mint(address indexed seller, bool _private, address indexed _collateral, address indexed _underlying, uint _strikePrice, uint _expiry, uint volume);

    function mint(address longOrShort, uint volume) external {
        LongOption long = LongOption(longOrShort);
        mint(long.creator()!=address(0), long.collateral(), long.underlying(), long.strikePrice(), long.expiry(), volume);
    }

    function burn(address _creator, address _collateral, address _underlying, uint _strikePrice, uint _expiry, uint volume) public {
        address long  = longs [_creator][_collateral][_underlying][_strikePrice][_expiry];
        address short = shorts[_creator][_collateral][_underlying][_strikePrice][_expiry];
        require(short != address(0), 'Payaso: ZERO_ADDRESS'); // single check is sufficient

        LongOption(long).burn_(msg.sender, volume);
        ShortOption(short).burn_(msg.sender, volume);

        emit Burn(msg.sender, _creator, _collateral, _underlying, _strikePrice, _expiry, volume);
    }
    event Burn(address indexed seller, address _creator, address indexed _collateral, address indexed _underlying, uint _strikePrice, uint _expiry, uint volume);

    function burn(address longOrShort, uint volume) external {
        LongOption long = LongOption(longOrShort);
        burn(long.creator(), long.collateral(), long.underlying(), long.strikePrice(), long.expiry(), volume);
    }

    function calcExerciseAmount(address _long, uint volume) public view returns (uint) {
        return calcExerciseAmount(volume, LongOption(_long).strikePrice());
    }
    function calcExerciseAmount(uint volume, uint _strikePrice) public pure returns (uint) {
        return volume.mul(_strikePrice).div(1 ether);
    }

    function _exercise(address buyer, address _creator, address _collateral, address _underlying, uint _strikePrice, uint _expiry, uint volume, address[] memory path) internal returns (uint vol, uint fee, uint amt) {
        require(now <= _expiry, 'Payaso: Expired');

        address long  = longs[_creator][_collateral][_underlying][_strikePrice][_expiry];
        LongOption(long).burn_(buyer, volume);

        address short = shorts[_creator][_collateral][_underlying][_strikePrice][_expiry];
        amt = calcExerciseAmount(volume, _strikePrice);
        if(path.length == 0) {
            IERC20(_underlying).safeTransferFrom(buyer, short, amt);
            (vol, fee) = ShortOption(short).exercise_(buyer, volume);
        } else {
            (vol, fee) = ShortOption(short).exercise_(address(this), volume);
            IERC20(_collateral).safeApprove(address(config[_uniswapRounter_]), vol);
            uint[] memory amounts = IUniswapV2Router01(config[_uniswapRounter_]).swapTokensForExactTokens(amt, vol, path, short, now);
            vol = vol.sub(amounts[0]);
            IERC20(_collateral).safeTransfer(buyer, vol);
            amt = 0;
        }
        emit Exercise(buyer, _collateral, _underlying, _strikePrice, _expiry, volume, vol, fee, amt);
    }
    event Exercise(address indexed buyer, address indexed _collateral, address indexed _underlying, uint _strikePrice, uint _expiry, uint volume, uint vol, uint fee, uint amt);

    function exercise_(address buyer, address _creator, address _collateral, address _underlying, uint _strikePrice, uint _expiry, uint volume, address[] calldata path) external returns (uint vol, uint fee, uint amt) {
        address long  = longs[_creator][_collateral][_underlying][_strikePrice][_expiry];
        require(msg.sender == long, 'Payaso: Only LongOption');

        return _exercise(buyer, _creator, _collateral, _underlying, _strikePrice, _expiry, volume, path);
    }
```

```
    function exercise(address _creator, address _collateral, address _underlying, uint _strikePrice, uint _expiry, uint volume, address[] calldata path) external returns (uint vol, uint fee, uint amt) {
        return _exercise(msg.sender, _creator, _collateral, _underlying, _strikePrice, _expiry, volume, path);
    }

    function exercise(address _long, uint volume, address[] memory path) public returns (uint vol, uint fee, uint amt) {
        LongOption long = LongOption(_long);
        return _exercise(msg.sender, long.creator(), long.collateral(), long.underlying(), long.strikePrice(), long.expiry(), volume, path);
    }

    function exercise(address _long, uint volume) public returns (uint vol, uint fee, uint amt) {
        LongOption long = LongOption(_long);
        return _exercise(msg.sender, long.creator(), long.collateral(), long.underlying(), long.strikePrice(), long.expiry(), volume, new address[](0));
    }

    function exercise(address long, address[] calldata path) external returns (uint vol, uint fee, uint amt) {
        return exercise(long, LongOption(long).balanceOf(msg.sender), path);
    }

    function exercise(address long) external returns (uint vol, uint fee, uint amt) {
        return exercise(long, LongOption(long).balanceOf(msg.sender), new address[](0));
    }

    function settleable(address _creator, address _collateral, address _underlying, uint _strikePrice, uint _expiry, uint volume) public view returns (uint col, uint fee, uint und) {
        address short = shorts[_creator][_collateral][_underlying][_strikePrice][_expiry];
        return ShortOption(short).settleable(volume);
    }
    function settleable(address short, uint volume) public view returns (uint col, uint fee, uint und) {
        return ShortOption(short).settleable(volume);
    }
    function settleable(address seller, address short) public view returns (uint col, uint fee, uint und) {
        return ShortOption(short).settleable(seller);
    }

    function settle(address _creator, address _collateral, address _underlying, uint _strikePrice, uint _expiry, uint volume) external {
        address short = shorts[_creator][_collateral][_underlying][_strikePrice][_expiry];
        settle(short, volume);
    }
    function settle(address short, uint volume) public {
        ShortOption(short).settle_(msg.sender, volume);
    }
    function settle(address short) external {
        settle(short, ShortOption(short).balanceOf(msg.sender));
    }

    function emitSettle(address seller, address _creator, address _collateral, address _underlying, uint _strikePrice, uint _expiry, uint volume, uint col, uint fee, uint und) external {
        address short  = shorts[_creator][_collateral][_underlying][_strikePrice][_expiry];
        require(msg.sender == short, 'Payaso: Only ShortOption');
        emit Settle(seller, _creator, _collateral, _underlying, _strikePrice, _expiry, volume, col, fee, und);
    }
    event Settle(address indexed seller, address _creator, address indexed _collateral, address indexed _underlying, uint _strikePrice, uint _expiry, uint volume, uint col, uint fee, uint und);
}

contract LongProxy is InitializableProductProxy, Constants {
    function productName() override public pure returns (bytes32) {
        return _LongOption_;
    }
}

contract ShortProxy is InitializableProductProxy, Constants {
    function productName() override public pure returns (bytes32) {
        return _ShortOption_;
    }
}


contract LongOption is ERC20UpgradeSafe {
    using SafeMath for uint;

    address public factory;
    address public creator;
    address public collateral;
    address public underlying;
    uint public strikePrice;
    uint public expiry;

    function initialize(address _creator, address _collateral, address _underlying, uint _strikePrice, uint _expiry) external initializer {
        (string memory name, string memory symbol) = spellNameAndSymbol(_collateral, _underlying, _strikePrice, _expiry);
```

```
        __ERC20_init(name, symbol);
        _setupDecimals(ERC20UpgradeSafe(_collateral).decimals());

        factory = msg.sender;
        creator = _creator;
        collateral = _collateral;
        underlying = _underlying;
        strikePrice = _strikePrice;
        expiry = _expiry;
    }

    function spellNameAndSymbol(address _collateral, address _underlying, uint _strikePrice, uint _expiry) public
view returns (string memory name, string memory symbol) {
        //return ('Payaso.io ETH long put option strike 500 USDC or USDC long call option strike 0.002 ETH
expiry 2020/10/10', 'USDC(0.002ETH)201010');
        return('Payaso Long Option Token', 'Long');
    }

    modifier onlyFactory {
        require(msg.sender == factory, 'Payaso: Only Factory');
        _;
    }

    function mint_(address _to, uint volume) external onlyFactory {
        _mint(_to, volume);
    }

    function burn_(address _from, uint volume) external onlyFactory {
        _burn(_from, volume);
    }

    function burn(uint volume) external {
        _burn(msg.sender, volume);
    }
    function burn() external {
        _burn(msg.sender, balanceOf(msg.sender));
    }

    function exercise(uint volume, address[] memory path) public {
        OptionFactory(factory).exercise_(msg.sender, creator, collateral, underlying, strikePrice, expiry, volume,
path);
    }

    function exercise(uint volume) public {
        exercise(volume, new address[](0));
    }

    function exercise(address[] calldata path) external {
        exercise(balanceOf(msg.sender), path);
    }

    function exercise() external {
        exercise(balanceOf(msg.sender), new address[](0));
    }
}

contract ShortOption is ERC20UpgradeSafe, Constants {
    using SafeERC20 for IERC20;
    using SafeMath for uint;

    address public factory;
    address public creator;
    address public collateral;
    address public underlying;
    uint public strikePrice;
    uint public expiry;

    function initialize(address _creator, address _collateral, address _underlying, uint _strikePrice, uint _expiry)
external initializer {
        (string memory name, string memory symbol) = spellNameAndSymbol(_collateral, _underlying,
_strikePrice, _expiry);
        __ERC20_init(name, symbol);
        _setupDecimals(ERC20UpgradeSafe(_collateral).decimals());

        factory = msg.sender;
        creator = _creator;
        collateral = _collateral;
        underlying = _underlying;
        strikePrice = _strikePrice;
        expiry = _expiry;
    }

    function spellNameAndSymbol(address _collateral, address _underlying, uint _strikePrice, uint _expiry) public
view returns (string memory name, string memory symbol) {
        //return ('Payaso.io ETH short put option strike 500 USDC or USDC short call option strike 0.002 ETH
expiry 2020/10/10', 'USDC(0.002ETH)201010s');
        return('Payaso Short Option Token', 'Short');
```

```
    }

    modifier onlyFactory {
        require(msg.sender == factory, 'Payaso: Only Factory');
        _;
    }

    function mint_(address _to, uint volume) external onlyFactory {
        _mint(_to, volume);
    }

    function burn_(address _from, uint volume) external onlyFactory {
        _burn(_from, volume);
        IERC20(collateral).safeTransfer(_from, volume);
    }

    function calcFee(uint volume) public view returns (address recipient, uint fee) {
        uint feeRate = OptionFactory(factory).getConfig(_feeRate_);
        recipient = address(OptionFactory(factory).getConfig(_feeRecipient_));

        if(feeRate != 0 && recipient != address(0))
            fee = volume.mul(feeRate).div(1 ether);
        else
            fee = 0;
    }

    function _payFee(uint volume) internal returns (uint) {
        (address recipient, uint fee) = calcFee(volume);
        if(recipient != address(0) && fee > 0)
            IERC20(collateral).safeTransfer(recipient, fee);
        return fee;
    }

    function exercise_(address buyer, uint volume) external onlyFactory returns (uint vol, uint fee) {
        fee = _payFee(volume);
        vol = volume.sub(fee);
        IERC20(collateral).safeTransfer(buyer, vol);
    }

    function settle_(address seller, uint volume) external onlyFactory {
        _settle(seller, volume);
    }

    function settleable(address seller) public view returns (uint col, uint fee, uint und) {
        return settleable(balanceOf(seller));
    }

    function settleable(uint volume) public view returns (uint col, uint fee, uint und) {
        //require(now > expiry, 'Payaso: Unmatured');
        if(now <= expiry) {
            address long    = OptionFactory(factory).longs(creator, collateral, underlying, strikePrice, expiry);
            uint waived = IERC20(collateral).balanceOf(address(this)).sub(IERC20(long).totalSupply());
            uint exercised = totalSupply().sub(IERC20(collateral).balanceOf(address(this)));
            if(waived.add(exercised) == 0)
                return (0, 0, 0);
            if(volume > waived.add(exercised))
                volume = waived.add(exercised);
            col = waived.mul(volume).div(waived.add(exercised));
            und = IERC20(underlying).balanceOf(address(this)).mul(volume).div(waived.add(exercised));
        } else {
            col = IERC20(collateral).balanceOf(address(this)).mul(volume).div(totalSupply());
            und = IERC20(underlying).balanceOf(address(this)).mul(volume).div(totalSupply());
        }
        (, fee) = calcFee(col);
        col = col.sub(fee);
    }

    function _settle(address seller, uint volume) internal returns (uint col, uint fee, uint und) {
        (col, fee, und) = settleable(volume);
        _burn(seller, volume);
        _payFee(col.add(fee));
        IERC20(collateral).safeTransfer(seller, col);
        IERC20(underlying).safeTransfer(seller, und);
        OptionFactory(factory).emitSettle(seller, creator, collateral, underlying, strikePrice, expiry, volume, col,
fee, und);
    }

    function settle(uint volume) external {
        _settle(msg.sender, volume);
    }

    function settle() external {
        _settle(msg.sender, balanceOf(msg.sender));
    }
}
```

```
contract Constants2 {
    address internal constant _DAI =
0x6B175474E89094C44Da98b954EedeAC495271d0F;
    bytes32 internal constant _ecoAddr_ = 'ecoAddr';
    bytes32 internal constant _ecoRatio_ = 'ecoRatio';
}

struct Ask {
    uint askID;
    address seller;
    address long;
    uint volume;
    address settleToken;
    uint price;
    uint remain;
}

struct Bid {
    uint bidID;
    uint askID;
    address buyer;
    uint volume;
    uint amount;
    uint remain;
}

contract OptionOrder is Configurable, Constants, Constants2 {
    using SafeERC20 for IERC20;
    using SafeMath for uint;

    address public factory;
    mapping(uint => Ask) public asks;
    mapping(uint => Bid) public bids;
    uint public asksN;
    uint public bidsN;

    address public farm;
    address public reward;
    mapping(address => uint) settledRewards;
    mapping(address => uint) claimedRewards;
     uint public begin;
     uint public span;
     uint public end;
     uint public times;
     uint public period;
     uint public frequency;
     uint public lasttime;

     mapping(address => mapping(address => uint)) public rewardThreshold;

    function initialize(address _governor, address _factory, address _farm, address _reward, address _ecoAddr)
public initializer {
        super.initialize(_governor);
        factory = _factory;
        farm    = _farm;
        reward  = _reward;
        config[_ecoAddr_]  = uint(_ecoAddr);
        config[_ecoRatio_] = 0.1 ether;             // 10%

        //IFarm(farm).crop();                           // just check
        IERC20(_reward).totalSupply();              // just check

        address weth = IUniswapV2Router01(OptionFactory(factory).getConfig(_uniswapRounter_)).WETH();
        setRewardThreshold(_DAI_, weth, 500 ether);
    }

    function setRewardThreshold(address _collateral, address _underlying, uint volume) public governance {
        rewardThreshold[_collateral][_underlying] = volume;
    }

    function setSpan(uint _span, bool isLinear, uint _period) virtual external governance {
        span = _span;
        if(isLinear)
            end = now + _span;
        else
            end = 0;

        if(begin == 0)
            begin = now;

        if(lasttime == 0)
            lasttime = now;

        period = _period;
    }

    function sellOnETH(bool _private, address _underlying, uint _strikePrice, uint _expiry, address settleToken,
```

```
uint price) virtual external payable returns (uint askID) {
        address weth = IUniswapV2Router01(OptionFactory(factory).getConfig(_uniswapRounter_)).WETH();
        IWETH(weth).deposit{value: msg.value}();
        return sell(_private, weth, _underlying, _strikePrice, _expiry, msg.value, settleToken, price);
    }

    function sell(bool _private, address _collateral, address _underlying, uint _strikePrice, uint _expiry, uint volume, address settleToken, uint price) virtual public returns (uint askID) {
        address weth = IUniswapV2Router01(OptionFactory(factory).getConfig(_uniswapRounter_)).WETH();
        if(_collateral != weth || IERC20(_collateral).balanceOf(address(this)) < volume)
            IERC20(_collateral).safeTransferFrom(msg.sender, address(this), volume);
        IERC20(_collateral).safeApprove(factory, volume);
        OptionFactory(factory).mint(_private, _collateral, _underlying, _strikePrice, _expiry, volume);

        address creator = _private ? tx.origin : address(0);
        address short = OptionFactory(factory).shorts(creator, _collateral, _underlying, _strikePrice, _expiry);
        IERC20(short).safeTransfer(msg.sender, volume);

        address long = OptionFactory(factory).longs(creator, _collateral, _underlying, _strikePrice, _expiry);
        return _sell(long, volume, settleToken, price);
    }

    function sell(address long, uint volume, address settleToken, uint price) virtual public returns (uint askID) {
        IERC20(long).safeTransferFrom(msg.sender, address(this), volume);
        return _sell(long, volume, settleToken, price);
    }

    function _sell(address long, uint volume, address settleToken, uint price) virtual internal returns (uint askID) {
        askID = asksN++;
        asks[askID] = Ask(askID, msg.sender, long, volume, settleToken, price, volume);

        emit Sell(askID, msg.sender, long, volume, settleToken, price);
    }
    event Sell(uint askID, address indexed seller, address indexed long, uint volume, address indexed settleToken, uint price);

    function reprice(uint askID, uint newPrice) virtual external returns (uint newAskID) {
        require(asks[askID].seller != address(0), 'Nonexistent ask order');
        require(asks[askID].seller == msg.sender, 'Not yours ask Order');

        newAskID = asksN++;
        asks[newAskID] = Ask(newAskID, asks[askID].seller, asks[askID].long, asks[askID].remain, asks[askID].settleToken, newPrice, asks[askID].remain);
        asks[askID].remain = 0;

        emit Reprice(askID, newAskID, asks[newAskID].seller, asks[newAskID].long, asks[newAskID].volume, asks[newAskID].settleToken, asks[askID].price, newPrice);
    }
    event Reprice(uint askID, uint newAskID, address indexed seller, address indexed long, uint volume, address indexed settleToken, uint price, uint newPrice);

    function cancel(uint askID) virtual external returns (uint vol) {
        require(asks[askID].seller != address(0), 'Nonexistent ask order');
        require(asks[askID].seller == msg.sender, 'Not yours ask Order');

        vol = asks[askID].remain;
        IERC20(asks[askID].long).safeTransfer(msg.sender, vol);
        asks[asksN].remain = 0;

        emit Cancel(askID, msg.sender, asks[askID].long, vol);
    }
    event Cancel(uint askID, address indexed seller, address indexed long, uint vol);

    function buy(uint askID, uint volume) virtual public returns (uint bidID, uint vol, uint amt) {
        require(asks[askID].seller != address(0), 'Nonexistent ask order');
        vol = volume;
        if(vol > asks[askID].remain)
            vol = asks[askID].remain;

        amt = vol.mul(asks[askID].price).div(1 ether);
        address weth = IUniswapV2Router01(OptionFactory(factory).getConfig(_uniswapRounter_)).WETH();
        address settleToken = asks[askID].settleToken;
        if(settleToken != weth || address(this).balance < amt)
            IERC20(settleToken).safeTransferFrom(msg.sender, asks[askID].seller, amt);
        else
            payable(asks[askID].seller).transfer(amt);

        asks[askID].remain = asks[askID].remain.sub(vol);
        IERC20(asks[askID].long).safeTransfer(msg.sender, vol);

        bidID = bidsN++;
        bids[bidID] = Bid(bidID, askID, msg.sender, vol, amt, vol);

        emit Buy(bidID, askID, msg.sender, vol, amt);
    }
    event Buy(uint bidID, uint askID, address indexed buyer, uint vol, uint amt);
```

```
    function buyInETH(uint askID, uint volume) virtual external payable returns (uint bidID, uint vol, uint amt) {
        require(asks[askID].seller != address(0), 'Nonexistent ask order');
        address weth = IUniswapV2Router01(OptionFactory(factory).getConfig(_uniswapRounter_)).WETH();
        require(asks[askID].settleToken == weth, 'settleToken is NOT WETH');

        vol = volume;
        if(vol > asks[askID].remain)
            vol = asks[askID].remain;

        amt = vol.mul(asks[askID].price).div(1 ether);
        require(msg.value >= amt, 'value is too low');

        (bidID, vol, amt) = buy(askID, vol);

        if(msg.value > amt)
            msg.sender.transfer(msg.value.sub(amt));
    }

    function exercise(uint bidID) virtual external returns (uint vol, uint fee, uint amt) {
        return exercise(bidID, bids[bidID].remain, new address[](0));
    }
    function exercise(uint bidID, address[] calldata path) virtual external returns (uint vol, uint fee, uint amt) {
        return exercise(bidID, bids[bidID].remain, path);
    }
    function exercise(uint bidID, uint volume) virtual public returns (uint vol, uint fee, uint amt) {
        return exercise(bidID, volume, new address[](0));
    }
    function exercise(uint bidID, uint volume, address[] memory path) virtual public returns (uint vol, uint fee, uint
amt) {
        require(bids[bidID].buyer != address(0), 'Nonexistent bid order');
        if(volume > bids[bidID].remain)
            volume = bids[bidID].remain;
        bids[bidID].remain = bids[bidID].remain.sub(volume);

        address long = asks[bids[bidID].askID].long;
        IERC20(long).safeTransferFrom(msg.sender, address(this), volume);

        if(path.length == 0) {
            amt = OptionFactory(factory).calcExerciseAmount(long, volume);
            address                                                    weth                                                    =
IUniswapV2Router01(OptionFactory(factory).getConfig(_uniswapRounter_)).WETH();
            address underlying = LongOption(long).underlying();
            if(underlying != weth || IERC20(underlying).balanceOf(address(this)) < amt)
                IERC20(underlying).safeTransferFrom(msg.sender, address(this), amt);
            IERC20(underlying).safeApprove(factory, amt);
        }

        (vol, fee, amt) = OptionFactory(factory).exercise(long, volume, path);
        IERC20(LongOption(long).collateral()).safeTransfer(msg.sender, vol);

        _settleReward(msg.sender, LongOption(long).collateral(), LongOption(long).underlying(), volume);
        emit Exercise(bidID, msg.sender, vol, fee, amt);
    }
    event Exercise(uint bidID, address indexed buyer, uint vol, uint fee, uint amt);

    function exerciseETH(uint bidID, uint volume) virtual public payable returns (uint vol, uint fee, uint amt) {
        require(bids[bidID].buyer != address(0), 'Nonexistent bid order');
        address long = asks[bids[bidID].askID].long;
        address underlying = LongOption(long).underlying();
        address weth = IUniswapV2Router01(OptionFactory(factory).getConfig(_uniswapRounter_)).WETH();
        require(underlying == weth, 'underlying is NOT WETH');

        if(volume > bids[bidID].remain)
            volume = bids[bidID].remain;
        amt = OptionFactory(factory).calcExerciseAmount(long, volume);
        require(msg.value >= amt, 'value is too low');

        IWETH(weth).deposit{value: amt}();
        (vol, fee, amt) = exercise(bidID, volume, new address[](0));

        if(msg.value > amt)
            msg.sender.transfer(msg.value.sub(amt));
    }
    function exerciseETH(uint bidID) virtual external payable returns (uint vol, uint fee, uint amt) {
        return exerciseETH(bidID, bids[bidID].remain);
    }

    function waive(uint bidID) virtual external {
        waive(bidID, bids[bidID].remain);
    }
    function waive(uint bidID, uint volume) virtual public returns (uint vol) {
        vol = volume;
        if(vol > bids[bidID].remain)
            vol = bids[bidID].remain;
        bids[bidID].remain = bids[bidID].remain.sub(vol);

        address long = asks[bids[bidID].askID].long;
```

```
        IERC20(long).safeTransferFrom(msg.sender, address(this), vol);
        LongOption(long).burn(vol);

        _settleReward(msg.sender, LongOption(long).collateral(), LongOption(long).underlying(), vol);
        emit Waive(bidID, msg.sender, vol);
    }
    event Waive(uint bidID, address indexed buyer, uint vol);

    function _settleReward(address buyer, address _collateral, address _underlying, uint volume) virtual internal
returns (uint amt) {
        if(span == 0)
            return 0;

        amt = settleableReward(_collateral, _underlying, volume);

        _updateFrequency();

        if(amt == 0)
            return 0;

        settledRewards[buyer] = settledRewards[buyer].add(amt);

        uint a = 0;
        address addr = address(config[_ecoAddr_]);
        uint ratio = config[_ecoRatio_];
        if(addr != address(0) && ratio != 0) {
            a = amt.mul(ratio).div(1 ether);
            settledRewards[addr] = settledRewards[addr].add(a);
        }

        settledRewards[address(0)] = settledRewards[address(0)].add(amt).add(a);

        emit SettleReward(buyer, _collateral, _underlying, volume, amt, settledRewards[buyer]);
    }
    event SettleReward(address indexed buyer, address indexed _collateral, address indexed _underlying, uint
volume, uint amt, uint settled);

    function settleableReward(address _collateral, address _underlying, uint volume) public view returns (uint) {
        uint threshold = rewardThreshold[_collateral][_underlying];
        if(threshold == 0 || volume < threshold)
            return 0;
        else
            return settleableReward();
    }
    function settleableReward() public view returns (uint amt) {
        if(span == 0)
            return 0;

        amt                                    =                                    IERC20(reward).allowance(farm,
address(this)).add(claimedRewards[address(0)]).sub(settledRewards[address(0)]);

        // calc settleable in period
        if(end == 0) {                                                                                    // isNonLinear,
endless
            if(period < span)
                amt = amt.mul(period).div(span);
        }else if(now.add(period) < end)
            amt = amt.mul(period).div(end.sub(now));
        else if(now >= end)
            amt = 0;

        amt = amt.mul(1 ether).div(calcFrequency());
    }

    function calcFrequency() public view returns (uint f) {
        if(now < begin.add(period))
            if(now > begin)
                f = times.add(1 ether).mul(period).div(now.sub(begin));
            else
                f = uint(-1);
        else
            if(lasttime.add(period) > now)
                f = lasttime.add(period).sub(now).mul(frequency).div(period).add(1 ether);
            else
                f = 1 ether;
    }

    function _updateFrequency() internal returns(uint) {
        frequency = calcFrequency();
        times = times.add(1 ether);
        lasttime = now;
    }

    function claim() virtual external returns (uint amt) {
        amt = claimable(msg.sender);
        IERC20(reward).safeTransferFrom(farm, msg.sender, amt);
        claimedRewards[msg.sender] = settledRewards[msg.sender];
```

```
        claimedRewards[address(0)] = claimedRewards[address(0)].add(amt);

        emit Claim(msg.sender, amt, claimedRewards[msg.sender]);
    }
    event Claim(address indexed seller, uint amt, uint claimed);

    function claimable(address buyer) public view returns (uint) {
        return settledRewards[buyer].sub(claimedRewards[buyer]);
    }

}
```

6. Appendix B：Vulnerability rating standard

Smart contract vulnerability rating standards	
Level	**Level Description**
High	Vulnerabilities that can directly cause the loss of token contracts or user funds, such as: value overflow loopholes that can cause the value of tokens to zero, fake recharge loopholes that can cause exchanges to lose tokens, and can cause contract accounts to lose ETH or tokens. Access loopholes, etc.; Vulnerabilities that can cause loss of ownership of token contracts, such as: access control defects of key functions, call injection leading to bypassing of access control of key functions, etc.; Vulnerabilities that can cause the token contract to not work properly, such as: denial of service vulnerability caused by sending ETH to malicious addresses, and denial of service vulnerability caused by exhaustion of gas.
Medium	High-risk vulnerabilities that require specific addresses to trigger, such as value overflow vulnerabilities that can be triggered by token contract owners; access control defects for non-critical functions, and logical design defects that cannot cause direct capital losses, etc.
Low	Vulnerabilities that are difficult to be triggered, vulnerabilities with limited damage after triggering, such as value overflow vulnerabilities that require a large amount of ETH or tokens to trigger, vulnerabilities where attackers cannot directly profit after triggering value overflow, and the transaction sequence triggered by specifying high gas depends on the risk Wait.

7. Appendix C：Introduction to auditing tools

Manticore

Manticore is a symbolic execution tool for analyzing binary files and smart contracts. Manticore includes a symbolic Ethereum Virtual Machine (EVM), an EVM disassembler/assembler and a convenient interface for automatic compilation and analysis of Solidity. It also integrates Ethersplay, Bit of Traits of Bits visual disassembler for EVM bytecode, used for visual analysis. Like binary files, Manticore provides a simple command line interface and a Python for analyzing EVM bytecode API.

Oyente

Oyente is a smart contract analysis tool. Oyente can be used to detect common bugs in smart contracts, such as reentrancy, transaction sequencing dependencies, etc. More convenient, Oyente's design is modular, so this allows advanced users to implement and Insert their own detection logic to check the custom attributes in their contract.

securify.sh

Securify can verify common security issues of Ethereum smart contracts, such as disordered transactions and lack of input verification. It analyzes all possible execution paths of the program while fully automated. In addition, Securify also has a specific

language for specifying vulnerabilities, which makes Securify can keep an eye on current security and other reliability issues at any time.

Echidna

Echidna is a Haskell library designed for fuzzing EVM code.

MAIAN

MAIAN is an automated tool for finding vulnerabilities in Ethereum smart contracts. Maian processes the bytecode of the contract and tries to establish a series of transactions to find and confirm the error.

ethersplay

ethersplay is an EVM disassembler, which contains relevant analysis tools.

ida-evm

ida-evm is an IDA processor module for the Ethereum Virtual Machine (EVM).

Remix-ide

ida-evm is an IDA processor module for the Ethereum Virtual Machine (EVM).

Knownsec Penetration Tester Special Toolkit

Pen-Tester tools collection is created by KnownSec team. It contains plenty of

Pen-Testing tools such as automatic testing tool, scripting tool, Self-developed tools etc.

KNOWNSEC

Beijing KnownSec Information Technology Co., Ltd.

Advisory telephone	+86(10)400 060 9587
E-mail	sec@knownsec.com
Website	www.knownsec.com
Address	wangjing soho T2-B2509,Chaoyang District, Beijing