Security Classification	Open in the Project Group

[Doc Number:ZYNECOIN-DMSJ-20191028]

SENIOR

Smart Contract Audit Report

ZYNECOIN Token

Test Engineer: JiTao Sun

KNOWNSEC

Release notes

Revised one	Revise the content	The revision	The version	review
Ji-tao sun	Written document	2019/10/28	V1.0	Drawn to

Document information

The document name	Document	The document number	Level of
ZYNECOIN intelligent contract audit report	V1.0	【ZYNECOIN-DMSJ-20191028】	Project team disclosure

The statement

Knownsec shall only issue this report in respect of the facts that have occurred or existed prior to the issuance of this report, and shall assume corresponding responsibilities for such facts.Knownsec is not able to judge the security status of its smart contract for the fact that it happened or existed after the issuance, nor is it responsible for it.The security audit analysis and other contents of this report are only based on the documents and materials provided to Knownsec by the information provider as of the issuance of this report.Knownsec assumes that the provided information is not missing, tampered with, deleted or concealed.If the information provided is missing, tampered with, deleted, concealed or reflected is inconsistent with the actual situation, Knownsec shall not bear any responsibility for the losses and adverse effects caused thereby.

directory

1. review

The effective test period of this report is from October 27, 2019 to October 28, 2019, during which the security and standardization of ZYNECOIN intelligent contract code is audited and used as the statistical basis of the report.

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

Objective information of this test:

The name of the module	
The name of the Token	ZYNECOIN Token
Code type	Token code
Contract address	*
The link address	*
Code language	solidity

Information of the project tester:

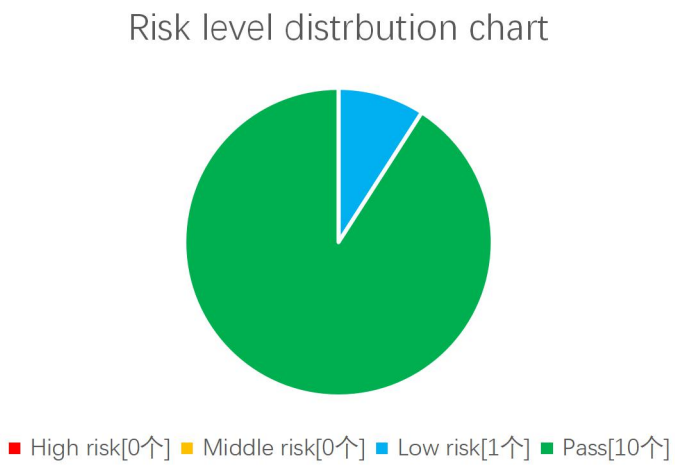
The name	Email/contact information	position
Ji-tao sun	sunjt@knownsec.com	Senior engineer

2. Code vulnerability analysis

2.1. Vulnerability grade distribution

The risk of this vulnerability is calculated by grade:

Statistical table of the number of vulnerability risk levels			
High risk	Middle risk	Low risk	Pass
0	0	1	10



2.2. Summary statement of audit results

The audit results			
Test project	The test content	state	describe
Intelligent contract	Reentry attack detection	Pass	After testing, there is no such safety problem.
	Numerical overflow detection	Pass	After testing, there is no such safety problem.
	Access control defect detection	Pass	After testing, there is no such safety problem.
	The call did not validate the return value	Pass	After testing, there is no such safety problem.
	Error using random number detection	Pass	After testing, there is no such safety problem.
	Transaction order dependent detection	Low risk (pass)	It has been detected that there is a risk of transaction sequence dependence in the code, but it is too difficult to make use of it, so the comprehensive evaluation is passed.
	Denial of service attack detection	Pass	After testing, there is no such safety problem.
	Logical design defect detection	Pass	After testing, there is no such safety problem.
	False recharge vulnerability detection	Pass	After testing, there is no such safety problem.
	Additional token vulnerability detection	Pass	After testing, there is no such safety problem.

	Frozen accounts bypass detection	Pass	After testing, there is no such safety problem.

Comprehensive assessment results: passed

3. Code audit result analysis

3.1. Reentry attack detection [safety]

The reentry hole is The most famous ethereum smart contract hole that led to The DAO hack.

The call.value() function in solexpenditure consumes all the gas it receives when it is used to send Ether, and there is a risk of reentry attacks when the operation of sending Ether by calling the call.value() function takes place before the actual reduction in the sender's account balance.

Detection result: after detection, there is no relevant call external contract call in the intelligent contract code.

Safety advice: none.

3.2. Numerical overflow detection [safety]

Arithmetic problems in smart contracts refer to integer overflow and integer underflow.

Solemotion is capable of processing 256 digits at most (2^256-1), and increasing the maximum number by 1 will overflow to 0.Similarly, when the number is unsigned, 0 minus 1 will overflow to the maximum numeric value.

Integer overflows and underflows are not a new type of vulnerability, but they are particularly dangerous in smart contracts. Overflow situations can lead to incorrect results, especially if the possibility is not expected, which may affect

the reliability and security of the program.

Test result: the security problem is not found in the intelligent contract code.

Safety advice: none.

3.3. Access control detection [safety]

Access control defects are a security risk that can occur in all programs, and similar problems can occur with smart contracts, including the famous Parity Wallet smart contract.

Test result: the security problem is not found in the intelligent contract code.

Safety advice: none.

3.4. Return value call validation [safety]

This problem often occurs in smart contracts related to the transfer of COINS, so it is also known as silent failure to send or unchecked send.

Transfer (), send(), call.value() and other transfer methods exist in solemotion, which can be used to send Ether to a certain address.Only 2300gas is passed for invocation to prevent reentry attacks. False if send fails; Only 2300gas is passed for invocation to prevent reentry attacks. False on failure to send call.value; Passing all available gas calls (which can be restricted by passing in the gas_value parameter) does not effectively prevent reentry attacks.

If the return values of the above send and call.value transliteration functions are not checked in the code, the contract will continue to execute the following code, possibly resulting in an unexpected result due to Ether sending failure.

Test results: after testing, there is no relevant vulnerability in the intelligent contract code.

Safety advice: none.

3.5. Wrong use of random Numbers [safety]

In intelligent contracts may need to use a random number, although the Solidity of functions and variables can access the value of the unpredictable obviously such as block. The number and block. The timestamp, but they usually or more open than it looks, or is affected by the miners, that is, to some extent, these random Numbers is predictable, so a malicious user can copy it and usually rely on its unpredictability to attack the function.

Test result: the problem does not exist in the intelligent contract code.

Safety advice: none.

3.6. Transaction order dependence [low risk]

Because miners always get gas fees through a code that represents an externally owned address (EOA), users can specify higher fees for faster transactions.Since the ethereum blockchain is public, everyone can see the content of other pending transactions. This means that if a user submits a

valuable solution, a malicious user can steal the solution and copy the transaction at a higher cost to preempt the original solution.

Test results: after tests, FixedBurnableTokenWithOrcalize. Sol in the file the approve function exists in the transaction order depend on the risk of attack, the code is as follows:

```
149     function approve(address spender, uint tokens) public returns (bool success) {
150         allowed[msg.sender][spender] = tokens;
151         emit Approval(msg.sender, spender, tokens);
152         return true;
153     }
```

The possible security risks are described as follows:

1. The number of transfers that user A allows user B to make on his behalf by calling the approve function is N (N>0);
2. After A period of time, user A decides to change N to M (M>, 0), so the approve function is called again.
3. User B quickly calls transferFrom function to transfer token of N quantity before the second call is processed by miner;
4. After user A's second successful call to approve, user B can obtain the transfer limit of M again, that is, user B obtained the transfer limit of N+M through transaction sequence attack.

Safety advice:

1. Front-end limit. When user A changes the limit from N to M, it can first change the limit from N to 0 and then from 0 to M.
2. Add the following code at the beginning of the approve function:

   Require ((_value = = 0) | | (allowed [MSG. Sender] [spender] = = 0));

3.7. Denial of service attack [safety]

In ethereum's world, denial of service is lethal, and smart contracts that suffer from this type of attack may never return to working order. There are many possible reasons for the intelligent contract denial of service, including malicious behavior as the recipient of the transaction, gas exhaustion caused by artificially increasing the need for computing capabilities, abuse of access control to access private components of the intelligent contract, exploitation of confusion and negligence, and so on.

Test results: after testing, there is no relevant vulnerability in the intelligent contract code.

Safety advice: none.

3.8. Logical design defects [safety]

Detect security issues related to business design in intelligent contract code.

Test results: after testing, there is no relevant vulnerability in the intelligent contract code.

Safety advice: none.

3.9. False recharge hole [safety]

In tokens, the transfer function of the contract to transfer the originator (MSG) sender) balance check is if judgment way, when the balances (MSG) sender) < value into the else logic part and return false, eventually did not throw

an exception, we believe that only the if/else in this gentle way of judging transfer such sensitive function scenario is a rigorous way of coding.

Test results: after testing, there is no relevant vulnerability in the intelligent contract code.

Safety advice: none.

3.10. Additional tokens loophole [safety]

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

Test result: the problem does not exist in the intelligent contract code.

Safety advice: none.

3.11. Frozen account bypass [safety]

Check whether there is an operation that the origin account, the originating account and the target account are frozen when the token is transferred in the token contract.

Test result: the problem does not exist in the intelligent contract code.

Safety advice: none.

4. Appendix A: contract code

Source of this test code: *

FixedBurnableTokenWithOrcalize. Sol

```
Pragma solidity ^ 0.5.0;       //knownsec// developed the compiler version in accordance with the recommended practice

// ----------------------------------------------------------------------
/ / Safe maths
// ----------------------------------------------------------------------
The library SafeMath {
        Function add(uint a, uint b) internal pure returns (uint c) {
                C is equal to a plus b;
                Require (c > = a);
        }
        Function sub(uint a, uint b) internal pure returns (uint c) {
                Require (b < = a);
                C = a - b;
        }
        Function mul(uint a, uint b) internal pure returns (uint c) {
                C is equal to a times b;
                Require (a == 0 || c/a == b);
        }
        Function div(uint a, uint b) internal pure returns (uint c) {
                Require (b > 0);
                C = a/b;
        }
}


// ----------------------------------------------------------------------
// ERC Token Standard #20 Interface
/ / https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
// ----------------------------------------------------------------------
Contract ERC20Interface {
        Function totalSupply() public view returns (uint);
        Function balanceOf(address tokenOwner) public view returns (uint balance);
        Function allowance(address tokenOwner, address spender) public view returns (uint remaining);
        Function transfer(address to, uint tokens) public returns (bool success);
        Function approve(address spender, uint tokens) public returns (bool success);
        Function transferFrom(address from, address to, uint tokens) public returns (bool success);

        Event Transfer(address indexed from, address indexed to, uint tokens);
        Event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}


// ----------------------------------------------------------------------
Contract function to receive approval and execute function in one call
//
/ / doesn from MiniMeToken
// ----------------------------------------------------------------------
Contract ApproveAndCallFallBack {
        ReceiveApproval function receiveApproval(address from, uint256 tokens, address token, bytes memory data) public;
}


// ----------------------------------------------------------------------
/ / Owned contract
// ----------------------------------------------------------------------
Contract Owned {
        Address public owner;
        Address public newOwner;

        Event OwnershipTransferred(address indexed _from, address indexed _to);

        Public constructor () {
                The owner = MSG. Sender;
        }

        Modifier onlyOwner {
                Require (MSG. Sender = = owner);
                _;
        }

        Function transferOwnership(address _newOwner) public onlyOwner {
                NewOwner = _newOwner;
        }
        The function acceptOwnership (public) {
```

```
        Require (MSG. Sender = = newOwner);
        Emit OwnershipTransferred (owner, newOwner);
        The owner = newOwner;
        NewOwner = address (0);
    }
}


// ----------------------------------------------------------------------
// ERC20 Token, with the addition of symbol, name and decimals and a
/ / fixed supply
// ----------------------------------------------------------------------
Contract FixedSupplyBurnableToken is ERC20Interface, Owned {
    Using SafeMath for uint;        //knownsec// references SafeMath function to prevent overflow

    String public symbol;
    String public name;
    Uint8 public decimals.
    Uint _totalSupply;

    The mapping (address = > uint) balances;
    Mapping (address => mapping(address => uint)) allowed;


    // ----------------------------------------------------------------------
    / / Constructor
    // ----------------------------------------------------------------------
    Public constructor () {
        Symbol = "FIXED";
        Name = "Example Fixed Supply Token";
        Decimals = 18;
        _totalSupply = 170000000 * 10**uint(decimals);
        Balances [owner] = _totalSupply;
        Emit Transfer (address (0), the owner, _totalSupply);
    }


    // ----------------------------------------------------------------------
    / / Total supply
    // ----------------------------------------------------------------------
    Function totalSupply() public view returns (uint) {
        Return _totalSupply. Sub (balances [address] (0));
    }


    // ----------------------------------------------------------------------
    // Get the token balance for account 'tokenOwner'
    // ----------------------------------------------------------------------
    Function balanceOf(address tokenOwner) public view returns (uint balance) {
        Return balances [tokenOwner];
    }


    // ----------------------------------------------------------------------
    // Transfer the balance from token owner's account to 'to' account
    // -owner's account must have sufficient balance to transfer
    // -0 value transfers are allowed
    // ----------------------------------------------------------------------
    Function transfer(address to, uint tokens) public returns (bool success) {/// tokens public returns (bool success) {/// tokens public returns (bool success) {/// tokens public returns (bool success) {/// tokens public returns (bool success) {/// tokens public returns (bool success) {/// tokens public returns (bool success)
        Balances [MSG. Sender] = balances [MSG. Sender]. Sub (tokens);      //knownsec// subtract first and add later, which is in line with the recommended practice
        Balances [to] = balances [to] the add (tokens);
        Emit Transfer (MSG) sender, to, tokens);
        Return true;
    }


    // ----------------------------------------------------------------------
    // Token owner can approve for 'spender' to transferFrom(... ) `tokens `
    // from the token owner's account
    //
    / / https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md
    // recommends that there are no checks for the approval double-spend attack
    Implemented in user interfaces as this should be
    // ----------------------------------------------------------------------
    Function approve(address spender, uint tokens) public returns (bool success) {/// tokens public returns (bool success) {/// tokens public returns (bool success) {/// tokens public returns (bool success) {/// tokens public returns (bool success) {/// tokens public returns (bool success) {/// tokens public returns (bool success)
    / / the require ((_value = = 0) | | (most egregious nest-feathering [MSG. Sender] [spender] = = 0));
        Allowed [MSG. Sender] [spender] = tokens;
        Emit Approval (MSG) sender, spender, tokens);
        Return true;
    }
```

```
        // ------------------------------------------------------------------------
        Tokens' from the 'from' account to the 'to' account
        //
        // The calling account must already have sufficient tokens approve(... D) -
        // for spending from the 'from' account and
        // - From account must have sufficient balance to transfer
        // -spender must have sufficient allowance to transfer
        // -0 value transfers are allowed
        // ------------------------------------------------------------------------
        Function transferFrom(address from, address to, uint tokens) public returns (bool success)
            Balances [from] = balances [from]. Sub (tokens);
            Allowed [from] [MSG. Sender] = allowed [from] [MSG. Sender]. Sub (tokens);
            Balances [to] = balances [to] the add (tokens);
            Emit Transfer (from, to, tokens);
            Return true;
        }


        // ------------------------------------------------------------------------
        Tokens tokens approved by the owner that can be
        // transferred to the spender's account
        // ------------------------------------------------------------------------
        Function allowance(address tokenOwner, address spender) public view returns (uint remaining) {
            Return charges [tokenOwner] [spender];
        }


        // ------------------------------------------------------------------------
        // Token owner can approve for 'spender' to transferFrom(... ) ` tokens `
        // from the token owner's account. the 'spender' contract function
        / / ` receiveApproval (... ) ` is then executed
        // ------------------------------------------------------------------------
        Function approveAndCall(address spender, uint tokens, bytes memory data) public returns (bool success) {
            Allowed [MSG. Sender] [spender] = tokens;
            Emit Approval (MSG) sender, spender, tokens);
            ApproveAndCallFallBack (spender.) receiveApproval (MSG) sender, tokens, the address (this), data);
            Return true;
        }

        // ------------------------------------------------------------------------
        Burns an amount of the token of a given account
        // ------------------------------------------------------------------------
        Function _burn(address account, uint256 value) internal {
            Require (account! = address (0));
            Require (value > 0);
            Require (_totalSupply > = value);
            Require (balances [account] > = value);

            _totalSupply = _totalSupply. Sub (value);
            Balances [account] = balances [account]. Sub (value);
            Emit Transfer (the account, the address (0), value);
        }

        // ------------------------------------------------------------------------
        // Burns an amount of the token of sender's account
        // ------------------------------------------------------------------------
        Function burn(uint256 value) public {
            _burn (MSG) sender, value);
        }

        // ------------------------------------------------------------------------
        Burns an amount of the token of a given
        Deducting from the sender's allowance for said account.
        // ------------------------------------------------------------------------
        Function burnFrom(address account, uint256 value) public {
            Require (allowed [account] [MSG. Sender] > = value);
            Allowed [account] [MSG. Sender] = allowed [account] [MSG. Sender]. Sub (value);
              burn (account, value);
            Emit Approval (account, MSG. Sender, allowed [account] [MSG. Sender]);

        }

        // ------------------------------------------------------------------------
        // Owner can transfer out any accidentally sent ERC20 tokens
        // ------------------------------------------------------------------------
        Function transferAnyERC20Token(address tokenAddress, uint tokens) public onlyOwner returns (bool
success) {
            Return ERC20Interface (tokenAddress). Transfer (the owner, tokens);
        }
}
```

ICOContract. Sol

```
Pragma solidity ^ 0.4.18;      //knownsec// specifies the compiler version, as recommended

The import "github.com/oraclize/ethereum-api/oraclizeAPI_0.4.sol";

// ----------------------------------------------------------------------
/ / Safe maths
// ----------------------------------------------------------------------
The library SafeMath {
        Function add(uint a, uint b) internal pure returns (uint c) {
                C is equal to a plus b;
                Require (c > = a);
        }
        Function sub(uint a, uint b) internal pure returns (uint c) {
                Require (b < = a);
                C = a - b;
        }
        Function mul(uint a, uint b) internal pure returns (uint c) {
                C is equal to a times b;
                Require (a == 0 || c/a == b);
        }
        Function div(uint a, uint b) internal pure returns (uint c) {
                Require (b > 0);
                C = a/b;
        }
}

// ----------------------------------------------------------------------
// ERC Token Standard #20 Interface
/ / https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
// ----------------------------------------------------------------------
Contract ERC20Interface {
        Function totalSupply() public view returns (uint);
        Function balanceOf(address tokenOwner) public view returns (uint balance);
        Function allowance(address tokenOwner, address spender) public view returns (uint remaining);
        Function transfer(address to, uint tokens) public returns (bool success);
        The function burn (uint tokens) public;
        Function approve(address spender, uint tokens) public returns (bool success);
        Function transferFrom(address from, address to, uint tokens) public returns (bool success);

        Event Transfer(address indexed from, address indexed to, uint tokens);
        Event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}

// ----------------------------------------------------------------------
/ / Owned contract
// ----------------------------------------------------------------------
Contract Owned {
        Address public owner;
        Address public newOwner;

        Event OwnershipTransferred(address indexed _from, address indexed _to);

        The function Owned (public) {
                The owner = MSG. Sender;
        }

        Modifier onlyOwner {
                Require (MSG. Sender = = owner);
                _;
        }

        Function transferOwnership(address _newOwner) public onlyOwner {///knownsec// suggest that the address
is 0
                NewOwner = _newOwner;
        }
        The function acceptOwnership (public) {
                Require (MSG. Sender = = newOwner);
                OwnershipTransferred (owner, newOwner);
                The owner = newOwner;
                NewOwner = address (0);
        }
}

/ *
   * @title String & slice utility library for solstays.
   * @author Nick Johnson <arachnid@notdot.net>
   *
   * @ dev Functionality in this library is largely implemented using the an
   * abstraction called a 'slice'. A slice represents a part of a string
   * anything from the entire string to a single character, or even no
```

```
    * characters at all (a 0-length slice). Since a slice only has to specify
    * an offset and a length, copying and manipulating slices is a lot less
    * expensive than copying and manipulating the strings they reference.
    *
    * To further reduce gas costs, most functions on slice that needs To return
    * a slice the modify the what one home allocating a new one;      The for
    * instance, 's.plit (".")' will return the text up to the first '.',
    * modifying s to only contain the remainder of the string after the '. '.
    * In situations where you do not want to modify the original slice, you
    * can make a copy first with '. Copy () ', for example:
    * 's.loops ().split(".")'. Try and avoid using this idiom in loops;      since
    * Solidity has no memory management, it will result in allocating the things
    * short - lived slices that are later discarded.
    *
    * Functions provides that return two slices in two versions: a non - allocating
    * version that takes the second slice as an argument, modifying it in
    * place, and the an allocating version that allocates the and returns to the second
    * slice;      See `nextRune `for example.
    *
    * Functions that have to copy string data will return strings rather than
    * slices;      These can be cast back to slices for further processing if
    * required.
    *
    * For convenience, some functions provides are provided with non - modifying
    * the factory will be producing a new slice and return both.      For instance,
    * 's.plitnew ('.')' leaves s unmodified, and returns two values
    The corresponding word corresponding to the left and right parts of the string.
    * /
The library strings {
        Struct slice {
            Uint _len;
            Uint _ptr;
        }

        Function memcpy(uint dest, uint SRC, uint len) private {
            Copy word-length chunks while possible
            For (;      Len > = 32;      Len - = 32) {
                The assembly {
                    Mstore (dest, mload (SRC))
                }
                Dest + = 32;
                The SRC + = 32;
            }

            / / Copy remaining bytes
            Uint mask = 256 ** (32-len) -1;
            The assembly {
                Let srcpart := and(mload(SRC), not(mask))
                Let destpart := and(mload(dest), mask)
                Mstore (dest, or (destpart srcpart))
            }
        }

        / *
          * @dev Returns a slice containing the entire string.
          * @param self The string to make a slice from.
          * @ return A newly allocated slice containing the - string.
          * /
        Function toSlice(string self) internal returns (slice) {
            Uint PTR.
            The assembly {
                PTR: = add (self, 0 x20)
            }
            Return slice (bytes (self). Length, PTR);
        }

        / *
          * @dev Returns the length of a null-terminated bytes32 string.
          * @param self The value to find The length of.
          * @return The length of The string, from 0 to 32.
          * /
        Function len(bytes32 self) internal returns (uint) {
            Uint ret.
            If (self = = 0)
                Return 0;
            If (self & 0 XFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF = = 0) {
                Ret + = 16;
                The self = bytes32 (uint (self) / 0 x100000000000000000000000000000000);
            }
            If (self & 0xffffffffffffffff == 0) {
                Ret + = 8;
                Self = bytes32(uint(self) / 0x10000000000000000);
            }
            If (self & 0xffffffff == 0) {
                Ret + = 4;
                Self = bytes32(uint(self) / 0x100000000);
            }
```

```
        If (self & 0xffff == 0) {
            Ret + = 2;
            Self = bytes32(uint(self) / 0x10000);
        }
        If (self & 0xff == 0) {
            Ret + = 1;
        }
        The return of 32 - ret;
    }

    / *
     * @dev Returns a slice containing the entire bytes32, interpreted as a
     * null - termintaed utf-8 string.
     * @param self The bytes32 value to convert to a slice.
     * @return A new slice containing the value of the input argument up to the
     * first null.
     * /
    Function toSliceB32(bytes32 self) internal returns (slice ret) {
        Allocate space for 'self' in memory, copy it there, and point ret at it
        The assembly {
            Let the PTR: = mload (0 x40)
            Mstore (0 x40, add (PTR, 0 x20))
            Mstore (PTR, self)
            Mstore (add (ret, 0 x20), PTR)
        }
        Ret. _len = len (self);
    }

    / *
     * @dev Returns a new slice containing the same data as the current slice
     * @param self The slice to copy.
     * @return A new slice containing the same data as' self '
     * /
    Function copy(slice self) internal returns (slice) {
        Return slice (self _len, self _ptr);
    }

    / *
     * @dev Copies a slice to a new string
     * @param self The slice to copy.
     * @ return A newly allocated string containing the slice 's text.
     * /
    Function toString(slice self) internal returns (string) {
        Var ret = new string(self._len);
        Uint retptr;
        Assembly {retptr := add(ret, 32)}

        Memcpy (retptr, self _ptr, self _len);
        Return ret.
    }

    / *
     * @dev Returns the length in runes of the slice.note that this operation
     * takes time to the length of the slice;      Get the using of it
     * in loops, and call 'slice. Empty ()' if you only need to know whether
     * the slice is empty or not.
     * @param self The slice to operate on.
     * @return The length of The slice in runes
     * /
    Function len(slice self) internal returns (uint) {
        // Starting at ptr-31 means the LSB will be the byte we care about
        Var PTR = self. _ptr-31;
        Var end = PTR + self. _len;
        For (uint len = 0;      PTR < end;      Len++) {
            Uint8 b;
            Assembly {b := and(mload(PTR), 0xFF)}
            If (b < 0x80) {
                PTR + = 1;
            Else if(b < 0xE0) {
                PTR + = 2;
            Else if(b < 0xF0) {
                PTR + = 3;
            Else if(b < 0xF8) {
                PTR + = 4;
            Else if(b < 0xFC) {
                PTR + = 5;
            } else {
                PTR + = 6;
            }
        }
        Return len.
    }

    / *
     * @dev Returns true if the slice is empty (has a length of 0).
     * @param self The slice to operate on.
     * @return True if the slice is empty, False otherwise.
```

```
 * /
Function empty(slice self) internal returns (bool) {
    Return the self. _len = = 0;
}

/ *
 * @ dev Returns a positive number if ` other ` comes lexicographically after
 * 'self', a negative number if it comes before, or zero if the
 * contents of the two slices are equal. Comparison is done per-rune.
 * on unicode codepoints.
 * @param self The first slice to compare.
 * @param other The second slice to compare.
 * @return The result of The comparison.
 * /
Function compare(slice self, slice other) internal returns (int) {
    Uint shortest = self. _len;
    If (other _len < self. _len)
        Shortest = other. _len;

    Var selfptr = self. _ptr;
    Var otherptr = other. _ptr;
    For (uint idx = 0;    Independence idx < shortest;    Independence idx + = 32) {
        Uint a;
        Uint b;
        The assembly {
            A: = mload (selfptr)
            B: = mload (otherptr)
        }
        If (a! = b) {
            // Mask out irrelevant bytes and check again
            Uint mask = ~(2 ** (8 * (32-buffy + idx)) - 1);
            Var diff = (a & mask) - (b & mask);
            If (diff! = 0)
                Return int (diff);
        }
        Selfptr + = 32;
        Otherptr + = 32;
    }
    Return int (self _len) - int (other. _len);
}

/ *
 * @dev Returns true if the two slices contain the same text.
 * @param self The first slice to compare.
 * @param self The second slice to compare.
 * @return True if the slices are equal, false otherwise.
 * /
Function equals(slice self, slice other) internal returns (bool) {
    Return compare(self, other) == 0;
}

/ *
 * @dev Extracts the first rune in the slice into 'rune', advancing the
 * slice to point to the next rune and returning 'self'.
 * @param self The slice to operate on.
 * @param rune The slice that will contain The first rune.
 * @ return ` rune `.
 * /
Function nextRune(slice self, slice rune) internal returns (slice) {
    Rune. _ptr = self. _ptr;

    If (self. _len == 0) {
        Rune. _len = 0;
        Return rune.
    }

    Uint len.
    Uint b;
    // Load the first byte of the rune into the LSBs of b
    Assembly {b := and(mload(sub(mload(add(self, 32)), 31)), 0xFF)}
    If (b < 0x80) {
        Len = 1;
    Else if(b < 0xE0) {
        Len = 2;
    Else if(b < 0xF0) {
        Len = 3;
    } else {
        Len = 4;
    }

    Check for truncated codepoints
    If (len > self. _len) {
        Rune. _len = self. _len;
        Self. _ptr + = self. _len;
        Self. _len = 0;
        Return rune.
    }
```

```
        Self._ptr + = len;
        Self._len - = len;
        Rune._len = len;
        Return rune.
    }

    / *
     * @dev Returns the first rune in the slice, advancing the slice to point
     * to the next rune.
     * @param self The slice to operate on.
     * @return A slice containing only the first rune from 'self'
     * /
    Function nextRune(slice self) internal returns (slice ret) {
        NextRune (self, ret);
    }

    / *
     * @dev Returns the number of the first codepoint in the slice.
     * @param self The slice to operate on.
     * @return The number of The first codepoint in The slice.
     * /
    Function ord(slice self) internal returns (uint ret) {
        If (self._len == 0) {
            Return 0;
        }

        Uint word;
        Uint len.
        Uint div = 2 ** 248;

        // Load the rune into the MSBs of b
        Assembly {word:= mload(mload(add(self, 32)))}
        Var b = word/div;
        If (b < 0x80) {
            Ret = b;
            Len = 1;
        Else if(b < 0xE0) {
            Ret = b & 0x1F;
            Len = 2;
        Else if(b < 0xF0) {
            Ret = b & 0x0F;
            Len = 3;
        } else {
            Ret = b & 0x07;
            Len = 4;
        }

        Check for truncated codepoints
        If (len > self._len) {
            Return 0;
        }

        For (uint I = 1;      I < len.      I++) {
            Div = div / 256;
            B = (word/div) & 0xFF;
            If (b & 0xC0! = 0 x80) {
                / / Invalid utf-8 sequence
                Return 0;
            }
            Ret = (ret * 64) | (b & 0x3F);
        }

        Return ret.
    }

    / *
     * @dev Returns the kecak-256 hash of the slice.
     * @param self The slice to hash.
     * @return The hash of The slice.
     * /
    Function keccak(slice self) internal returns (bytes32 ret) {
        The assembly {
            Ret := sha3(mload(add(self, 32)), mload(self))
        }
    }

    / *
     * @dev Returns true if 'self' starts with 'needle'.
     * @param self The slice to operate on.
     * @param needle The slice to search for.
     * @return True if the slice starts with the provided text, false otherwise.
     * /
    Function startsWith(slice self, slice needle) internal returns (bool) {
        If (self._len < needle._len) {
            Return false;
        }
```

```
        If (self._ptr == needle._ptr) {
            Return true;
        }

        Bool equal;
        The assembly {
            Let len: = mload (needle)
            Let selfptr := mload(add(self, 0x20))
            Let needleptr := mload(add(needle, 0x20))
            Equal := eq(sha3(selfptr, len), sha3(needleptr, len))
        }
        Return equal;
    }

    / *
     * @dev If 'self' starts with 'needle', 'needle' is removed from the
     * beginning of 'self'. Otherwise, 'self' is unmodified.
     * @param self The slice to operate on.
     * @param needle The slice to search for.
     * @ return `self`
     * /
    Function beyond(slice self, slice needle) internal returns (slice) {
        If (self._len < needle._len) {
            Return the self;
        }

        Bool equal = true;
        If (self._ptr! = needle._ptr) {
            The assembly {
                Let len: = mload (needle)
                Let selfptr := mload(add(self, 0x20))
                Let needleptr := mload(add(needle, 0x20))
                Equal := eq(sha3(selfptr, len), sha3(needleptr, len))
            }
        }

        If (equal) {
            Self._len - = needle._len;
            Self._ptr + = needle._len;
        }

        Return the self;
    }

    / *
     * @dev Returns true if the slice ends with 'needle'.
     * @param self The slice to operate on.
     * @param needle The slice to search for.
     * @return True if the slice starts with the provided text, false otherwise.
     * /
    Function endsWith(slice self, slice needle) internal returns (bool) {
        If (self._len < needle._len) {
            Return false;
        }

        Var selfptr = self._ptr + self._len - needle.

        If (selfptr == needle._ptr) {
            Return true;
        }

        Bool equal;
        The assembly {
            Let len: = mload (needle)
            Let needleptr := mload(add(needle, 0x20))
            Equal := eq(sha3(selfptr, len), sha3(needleptr, len))
        }

        Return equal;
    }

    / *
     * @dev If 'self' ends with 'needle', 'needle' is removed from the
     * end of 'self'. Otherwise, 'self' is unmodified.
     * @param self The slice to operate on.
     * @param needle The slice to search for.
     * @ return `self`
     * /
    Function until(slice self, slice needle) internal returns (slice) {
        If (self._len < needle._len) {
            Return the self;
        }

        Var selfptr = self._ptr + self._len - needle.
        Bool equal = true;
        If (selfptr! = needle._ptr) {
```

```
            The assembly {
                Let len: = mload (needle)
                Let needleptr := mload(add(needle, 0x20))
                Equal := eq(sha3(selfptr, len), sha3(needleptr, len))
            }
        }

        If (equal) {
            Self. _len - = needle. _len;
        }

        Return the self;
    }

    // Returns the memory address of the first byte of the first occurrence of
    // 'needle' in 'self', or the first byte after 'self' if not found.
    The function findPtr (uint selflen, uint selfptr, uint needlelen, uint needleptr) private returns (uint) {
        Uint PTR.
        Uint independence idx;

        If (needlelen < = selflen) {
            If (needlelen <= 32) {
                Optimized assembly for 68 gas per byte on short strings
                The assembly {
                    Let mask := not(sub(exp(2, mul(8, sub(32, needlelen)), 1))
                    Let needledata := and(mload(needleptr), mask)
                    Let the end: = add (selfptr, sub (selflen needlelen))
                    PTR: = selfptr
                    Loop:
                    Jumpi (exit, eq (and (mload (PTR), mask), needledata))
                    PTR: = add (PTR, 1)
                    Jumpi (loop, lt (sub (PTR, 1), end))
                    PTR: = add (selfptr selflen)
                    Exit:
                }
                Return PTR.
            } else {
                // For long needles, use hashing
                Bytes32 hash.
                Assembly {hash := sha3(needleptr, needlelen)}
                PTR = selfptr;
                For (independence idx = 0;      Independence idx < = selflen - needlelen;      Idx++) {
                    Bytes32 testHash;
                    Assembly {testHash := sha3(PTR, needlelen)}
                    If (hash = = testHash)
                        Return PTR.
                    PTR + = 1;
                }
            }
        }
        Return selfptr + selflen;
    }

    // Returns the memory address of the first byte after the last occurrence of
    // 'needle' in 'self', or the address of 'self' if not found.
    The function rfindPtr (uint selflen, uint selfptr, uint needlelen, uint needleptr) private returns (uint) {
        Uint PTR.

        If (needlelen < = selflen) {
            If (needlelen <= 32) {
                Optimized assembly for 69 gas per byte on short strings
                The assembly {
                    Let mask := not(sub(exp(2, mul(8, sub(32, needlelen)), 1))
                    Let needledata := and(mload(needleptr), mask)
                    PTR: = add (selfptr, sub (selflen needlelen))
                    Loop:
                    Jumpi (ret, eq (and (mload (PTR), mask), needledata))
                    The PTR: sub = (PTR, 1)
                    Jumpi (loop, gt (add (PTR, 1), selfptr))
                    PTR: = selfptr
                    Jump (exit)
                    Ret:
                    PTR: = add (PTR, needlelen)
                    Exit:
                }
                Return PTR.
            } else {
                // For long needles, use hashing
                Bytes32 hash.
                Assembly {hash := sha3(needleptr, needlelen)}
                PTR = selfptr + (selflen - needlelen);
                While (PTR >= selfptr) {
                    Bytes32 testHash;
                    Assembly {testHash := sha3(PTR, needlelen)}
                    If (hash = = testHash)
                        The return of PTR + needlelen;
                    PTR - = 1;
```

```
                }
            }
        }
        Return selfptr;
    }

    / *
     * @dev Modifies' self 'to contain everything from the first occurrence of
     * 'needle' to the end of the slice. 'self' is set to the empty slice
     * if 'needle' is not found.
     * @param self The slice to search and modify.
     * @param needle The text to search for.
     * @ return ` self `.
     * /
    Function find(slice self, slice needle) internal returns (slice) {
        Uint PTR = findPtr(self._len, self._ptr, needle.
        Self._len -= PTR - self.
        Self._ptr = PTR;
        Return the self;
    }

    / *
     * @dev Modifies' self 'to contain the part of the string from the start of
     * 'self' to the end of the first occurrence of 'needle'. If 'needle'
     * is not found, 'self' is set to the empty slice.
     * @param self The slice to search and modify.
     * @param needle The text to search for.
     * @ return ` self `.
     * /
    Function rfind(slice self, slice needle) internal returns (slice) {
        Uint PTR = rfindPtr(self._len, self._ptr, needle.
        Self._len = PTR - self.
        Return the self;
    }

    / *
     * @ dev Splits the slice, setting ` self ` to everything after the first
     * occurrence of 'needle', and 'token' to everything before it.if
     * 'needle' does not occur in 'self', 'self' is set to the empty slice,
     * and ` token ` is set to the entirety of ` self `.
     * @param self The slice to split.
     * @param needle The text to search for in 'self'.
     * @param token An output parameter to which the first token is written.
     * @ return ` token `.
     * /
    Function split(slice self, slice needle, slice token) internal returns (slice) {
        Uint PTR = findPtr(self._len, self._ptr, needle.
        Token._ptr = self._ptr;
        Token._len = ptr-self._ptr;
        If (PTR == self._ptr + self._len) {
            / / Not found
            Self._len = 0;
        } else {
            Self._len -= token._len + needle.
            Self._ptr = PTR + needle.
        }
        Return the token.
    }
    / *
     * @ dev Splits the slice, setting ` self ` to everything after the first
     * occurrence of 'needle', and returning everything before it.if
     * 'needle' does not occur in 'self', 'self' is set to the empty slice,
     * and the entirety of ` self ` is returned.
     * @param self The slice to split.
     * @param needle The text to search for in 'self'.
     * return The part of 'self' up to The first occurrence of 'delim'.
     * /
    Function split(slice self, slice needle) internal returns (slice token) {
        The split (self, needle, token);
    }

    / *
     * @ dev Splits the slice, setting ` self ` to everything before the last
     * occurrence of 'needle', and 'token' to everything after it.if
     * 'needle' does not occur in 'self', 'self' is set to the empty slice,
     * and ` token ` is set to the entirety of ` self `.
     * @param self The slice to split.
     * @param needle The text to search for in 'self'.
     * @param token An output parameter to which the first token is written.
     * @ return ` token `.
     * /
    Function rsplit(slice self, slice needle, slice token) internal returns (slice) {
        Uint PTR = rfindPtr(self._len, self._ptr, needle.
        Token._ptr = PTR;
        Token._len = self._len - (PTR - self._ptr);
        If (PTR == self._ptr) {
```

```
            // Not found
            self._len = 0;
        } else {
            self._len -= token._len + needle.
        }
        return the token.
}

/*
 * @dev Splits the slice, setting `self` to everything before the last
 * occurrence of 'needle', and returning everything after it.if
 * 'needle' does not occur in 'self', 'self' is set to the empty slice,
 * and the entirety of `self` is returned.
 * @param self The slice to split.
 * @param needle The text to search for in 'self'.
 * return The part of 'self' after The last occurrence of 'delim'.
 */
function rsplit(slice self, slice needle) internal returns (slice token) {
    Rsplit (self, needle, token);
}

/*
 * @dev Counts the number of nonoverlapping occurrences of `needle ``in self`.
 * @param self The slice to search.
 * @param needle The text to search for in 'self'.
 * return The number of occurrences of 'needle' found in 'self'.
 */
function count(slice self, slice needle) internal returns (uint count) {
    uint PTR = findPtr(self._len, self._ptr, needle.
    while (PTR <= self._ptr + self._len) {
        count++;
        PTR = findPtr(self._len - (PTR - self._ptr), PTR, needle.
    }
}

/*
 * @dev Returns True if 'self' contains' needle '.
 * @param self The slice to search.
 * @param needle The text to search for in 'self'.
 * @return True if 'needle' is found in 'self', false otherwise.
 */
function contains(slice self, slice needle) internal returns (bool) {
    return rfindPtr(self._len, self._ptr, needle. = self._ptr;
}

/*
 * @dev Returns a newly allocated string containing the concatenation of
 * `self` and `other`.
 * @param self The first slice to concatenate.
 * @param other The second slice to concatenate
 * @return The concatenation of The two strings.
 */
function concat(slice self, slice other) internal returns (string) {
    var ret = new string(self._len + other._len);
    uint retptr;
    assembly {retptr := add(ret, 32)}
    memcpy (retptr, self _ptr, self _len);
    memcpy (retptr + self._len, other._ptr, other._len);
    return ret.
}

/*
 * @dev Joins an array of slices, using 'self' as a delimiter, returning a
 * newly allocated string.
 * @param self The delimiter to use.
 * @param parts A list of slices to join.
 * @ return A newly allocated string containing all the slices in `parts`,
 * joined with `self`.
 */
function join(slice self, slice[] parts) internal returns (string) {
    if (parts. The length = = 0)
        return "";

    uint len = self._len * (parts.length-1);
    for (uint I = 0;     I < parts. Length;     I++)
        len + = parts [I] _len;

    var ret = new string(len);
    uint retptr;
    assembly {retptr := add(ret, 32)}

    for (I = 0;     I < parts. Length;     I++) {
        memcpy (retptr, parts [I] _ptr, parts [I] _len);
        retptr + = parts [I] _len;
        if (I < parts.length-I) {
            memcpy (retptr, self _ptr, self _len);
            retptr + = self. _len;
```

```
                }
            }

            Return ret.
        }
}


Contract ICOContract is usingOraclize, Owned {
        Using SafeMath for uint256;      //knownsec// references SafeMath function to prevent overflow

        Uint256 public usdPerETH;
        Uint256 public tokenPerETH;

        Address public tokenContractAddress;

        The event newOraclizeQuery (string description);
        Tokens; event TokensBought(uint256 tokens, uint256 price);
        The event PriceUpdated (uint256 ethPerToken);

        The function ICOContract (address _tokenContractAddress) {
            Oraclize_setProof (proofType_TLSNotary | proofStorage_IPFS);
            TokenContractAddress = _tokenContractAddress;
            The update ();
        }

        / / Oraclize callback
        Function arbitration (bytes32 myid, string result, bytes proof) {
            If (MSG) sender! = oraclize_cbAddress throw ());
            String memory priceFromAPI = result;
            String memory sep = ".";
            Strings. Slice memory sepSlice = strings. ToSlice (sep);
            Strings. Slice memory wholeNumberSlice = strings. ToSlice (priceFromAPI);
            String. Slice memory decimalValSlice = strings. Rsplit (wholeNumberSlice, sepSlice);

            Uint256 wholeUint = stringToUint (strings. ToString (wholeNumberSlice));
            Uint256 decimalUint = stringToUint (strings. ToString (decimalValSlice));

            Uint256 actualValue = (wholeUint * 1 ether) + ((decimalUint * 1 ether)/(10**(strings. Len
(decimalValSlice)));
            UsdPerETH = actualValue;

            TokenPerETH = (usdPerETH * 100 ether) / 43 ether;

            If (isCrowdsaleOpen() == true) {
                The update ();
            }
        }

        // Make a call to Kraken API to get current ETH/USD market price
        The function update () payable {
            If (oraclize. GetPrice ("URL") > this.balance) {
                NewOraclizeQuery ("Oraclize query was NOT sent, please add some ETH to cover for the query
fee");
            } else {
                NewOraclizeQuery ("Oraclize query was sent, standing by for the answer.. ");
                Oraclize_query (60, "URL", "json (https://api.kraken.com/0/public/Ticker? Pair = ETHUSD).
Result. XETHZUSD. C. 0 ");
            }
        }

        Bool public isCrowdsaleOpen = false;

        // // Marks ICO as ended. Investors cannot buy tokens after crowdsale has ended
        Function closeICO() public onlyOwner {
        / / isCrowdsaleOpen = false;
        / /}

        // Returns discount value for investors based on block.timestamp
        Function getDiscountForUser() internal view returns (uint256) {//knownsec// returns investor discount
according to block.timestamp
            If (block.timestamp >= 1548374400 && block.timestamp < 1550275200) {
                / / For Round 1
                Return 0.25 * 1 ether;
            } else if (block.timestamp >= 1550275200 && block.timestamp < 1552176000) {
                / / For Round 2
                Return 0.20 * 1 ether;
            } else if (block.timestamp >= 1552176000 && block.timestamp < 1554076800) {
                / / For Round 3
                Return 0.15 * 1 ether;
            } else if (block.timestamp >= 1554076800 && block.timestamp < 1555977600) {
                / / For Round 4
                Return 0.10 * 1 ether;
            } else if (block.timestamp >= 1555977600 && block.timestamp < 1557878400) {
                / / For Round 5
                Return 0.07 * 1 ether;
            } else if (block.timestamp >= 1557878400 && block.timestamp < 1559779200) {
```

```
                // For Round 6
                Return 0.05 * 1 ether;
            }

            // For Round 7
            Return 0;
        }

        Timestamp is between Jan 25, 2019 and Jun 27, 2019
        Function isCrowdsaleOpen() public view returns (bool) {
            Return (block.timestamp >= 1548374400 && block.timestamp < 1561679999);
        }

        // Returns an amount of tokens equivalent to ETH value paid
        Function buyToken() public payable {//knownsec// purchase token
            // the require (isCrowdsaleOpen ());
            Require (MSG) value > 0);

            Uint256 discountedTokenPerETH = (tokenPerETH * 43 ether)/(43 * (1 ether - getDiscountForUser()));
//knownsec// discount after token price
            Tokens = (msg.value * discountedTokenPerETH) / 1 ether;
            TokensBought (tokens, discountedTokenPerETH);
            ERC20Interface tokenContract = ERC20Interface (tokenContractAddress);
            TokenContract. Transfer (MSG) sender, tokens);
        }

        Payable function () {
            BuyToken ();
        }

        Function burnICOTokens(uint256 tokens) public onlyOwner {
            ERC20Interface tokenContract = ERC20Interface (tokenContractAddress);
            TokenContract. Burn (tokens);
        }

        // Return all the contracts balance to owner's address
        Function getInvestmentsToWallet() public onlyOwner {
            Owner. Transfer (enclosing the balance);
        }

        Function stringToUint(string s) constant returns (uint result) {
            Bytes memory b = bytes(s);
            Uint I;
            Result = 0;
            For (I = 0;      I < b.l ength;      I++) {
                Uint c = uint [I] (b);
                If (c >= 48 && c <= 57) {
                    Result = result * 10 + (c-48);
                }
            }
        }

}
```

5. Appendix B: vulnerability risk rating criteria

Smart contract vulnerability rating standard	
Vulnerability rating	**Description of vulnerability rating**
High risk vulnerabilities	Can directly cause the token contract or the user's fund loss loopholes, such as: can cause the token value to zero numerical overflow holes, can cause the exchange loss token false recharge holes, can cause the contract account loss ETH or token re-entry holes; Can cause the leakage of token contract ownership loss, such as: key function access control defects, call injection leading to key function access control bypass, etc. Loopholes that can cause token contracts to fail to work properly, such as the denial of service vulnerability caused by sending ETH to malicious addresses, denial of service vulnerability caused by gas exhaustion.
In the dangerous holes	High-risk vulnerabilities that need specific addresses to trigger, such as numeric overflow vulnerabilities that can be triggered by the owner of token contracts; Access control defects of non-critical functions, logical design defects that cannot cause direct loss of funds, etc.
Low latent loophole	Vulnerabilities that are difficult to be triggered, vulnerabilities that do limited harm after triggering, such as numeric overflow vulnerabilities that require large amounts of ETH or tokens to trigger, vulnerabilities that can't directly benefit attackers after triggering numeric overflow, and risks that depend on transaction sequence triggered by specifying high gas.

6. Appendix C: introduction to vulnerability testing tools

6.1. Manticore

Manticore is a symbol execution tool for analyzing binary files and intelligent contracts. Manticore contains a symbol ethereum virtual machine (EVM), an EVM disassembler/assembler, and a convenient interface for automatic compilation and analysis of solemotion. It also integrates Ethersplay, a visual disassembler of Bit of Traits of Bits for EVM bytecode, for visual analysis. Like binaries, Manticore provides a simple command-line interface and a Python API for parsing EVM bytecode.

6.2. Oyente

Oyente is a smart contract analysis tool that can be used to detect common bugs in smart contracts, such as reentrancy, transaction sort dependencies, and so on. More conveniently, Oyente's design is modular, so this allows advanced users to implement and insert their own detection logic to check custom properties in their contracts.

6.3. Securify. Sh

Securify can validate common security problems of ethereal smart contracts, such as transaction chaos and lack of input validation, it analyzes all possible execution paths of the program simultaneously in full automation, and it also has a specific language for specifying vulnerabilities, which enables Securify to focus on current security and other reliability issues at any time.

6.4. Echidna

Echidna is a Haskell library designed for fuzzing EVM code.

6.5. MAIAN

MAIAN, an automated tool for finding holes in ethereum's smart contracts, processes the bytecode of the contracts and tries to set up a series of transactions to find and confirm errors.

6.6. ethersplay

Ethersplay is an EVM disassembler that includes analysis tools.

6.7. IDA - evm entry

IDA -evm is an IDA processor module for the ethereum virtual machine (evm).

6.8. Want - ide

Remix is a browser-based compiler and IDE that allows users to build ethereum contracts and debug transactions using solemotion.

6.9. Knownsec penetration tester kit

Knownsec penetration tester kit, developed, collected and used by Knownsec penetration test engineers, includes batch automated testing tools for testers, independently developed tools, scripts or utilization tools, etc.

KNOWNSEC

[Advisory telephone]13366385246
[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.