知道创宇

密级 ★ | 项目组公开

Smart Contract Audit Report

STATUS

SAFE

Release notes

Revise the content	The revision of time	The version number
Written document	2020/9/3	V1.0

Document information

The document name	Document	The document number	Level of
FOUR Smart Contract Audit Report	V1.0	FOUR- ZNHY - 20200903	Open Project Team

The statement

Chuangyu shall only issue this report for the facts that have occurred or existed before the issuance of this report, and shall assume corresponding responsibilities accordingly.Chuangyu is not in a position to judge the security status of its intelligent contract and is not responsible for the occurrence or existence of such facts after issuance.The security audit analysis and other contents in this report are based only on the documents and materials provided by the information provider to Chuangyu as of the issuance of this report.Creation hypothesis: there is no absence, modification, deletion or concealment of the provided information.In the event that the information provided is missing, altered, deleted, concealed or reflected does not conform to the actual situation, Chuangyu shall not be liable for any loss or adverse effect caused thereby.

Knownsec

directory

1. review - 5 -

2. Code vulnerability analysis - 6 -

2.1. Vulnerability grade distribution - 6 -

2.2. Summary of audit results - 7 -

3. Code audit results analysis - 8 -

3.1. Reentry attack detection [pass] - 8 -

3.2. Numerical overflow detection [pass] - 8 -

3.3. Access control detection [pass] - 9 -

3.4. Return value call validation [pass] - 9 -

3.5. Error using random number [pass] - 10 -

3.6. Transaction order dependence [Low risk] - 10 -

3.7. Denial of service attack [pass] - 12 -

3.8. Logical design flaw [pass] - 12 -

3.9. False recharge vulnerability [pass] - 13 -

3.10. Issue of token loopholes [pass] - 13 -

3.11. Freeze account bypass [pass] - 13 -

4. Appendix A: Contract code - 15 -

5. Appendix B: Vulnerability risk rating criteria - 23 -

6. Appendix C: Introduction to vulnerability testing tools - 24 -

6.1. Manticore - 24 -

Knownsec

6.2. Oyente...- 24 -

6.3. securify.sh..- 24 -

6.4. Echidna..- 24 -

6.5. MAIAN..- 25 -

6.6. ethersplay...- 25 -

6.7. ida-evm..- 25 -

6.8. Remix-ide...- 25 -

6.9. Know chuangyu penetration tester kit...- 25 -

Knownsec

1. review

This report will be effective from September 3, 2020 to September 3, 2020, during which the security and standardization of FOUR's smart contract code will be audited and used as the statistical basis for the report.

In this test, it is known that Chuangyu engineers made a comprehensive analysis of common vulnerabilities of intelligent contract (see the third chapter), and found that there is a dependence on the order of things. However, due to the difficulty in exploiting the vulnerability, the comprehensive assessment was passed.

The smart contract security audit results: pass

Since this test is conducted in a non-production environment, all codes are updated, the test process is communicated with the relevant interface personnel, and relevant test operations are carried out under the control of operational risks, so as to avoid production and operation risks and code security risks in the test process.

The target information of this test:

The name of the module	
The name of the Token	FOUR
Code type	Token code
Contract address	0x4730fB1463A6F1F44AEB45F6c5c422427f37F4D0
Contract link	https://cn.etherscan.com/address/0x4730fB1463A6F1F44AEB45F6c5c422427f37F4D0#code
Code language	solidity

Knownsec

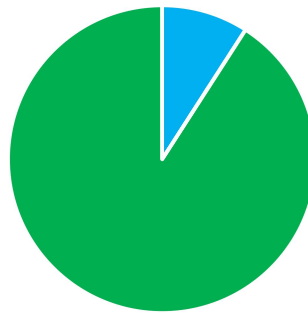
2. Code vulnerability analysis

2.1. Vulnerability grade distribution

This vulnerability risk is calculated by level：

Statistical table of vulnerability risk levels			
At high risk of	In a crisis	Low risk	pass
0	0	1	10

Risk level distribution map



■ High Risk[0个] ■ Middle Risk[0个] ■ Low Risk[1个] ■ Pass[10个]

Know

2.2. Summary of audit results

The audit results			
Test project	The test content	state	describe
Smart 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 dependency detection	Low risk (approved)	According to the test, there is a risk of transaction order dependence in the code, but it is too difficult to use, 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.
	Issue token vulnerability detection	pass	After testing, there is no such safety problem.
	Freezing accounts bypasses detection	pass	After testing, there is no such safety problem.

3. Code audit results analysis

3.1. Reentry attack detection [pass]

Re—entry holes are The most famous ethereum intelligent contract holes that have led to The DAO hack of Ethereum.

The call.value() function in Soldesert consumes all the gas it receives when it is used to send Ether, and there is a risk of a reentrant attack if the operation to send Ether is called to the call.value() function before it actually reduces the balance in the sender's account.

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

3.2. Numerical overflow detection [pass]

The arithmetic problem in intelligent contract refers to integer overflow and integer underflow.

Instead of trying to contain something deep inside the body, which is capable of processing a maximum of 256 digits (2^256—1), a maximum increase of 1 would allow the body to drain down to zero. Similarly, when the number is unsigned, 0 minus 1 overflows to get the maximum number value.

Integer overflow and underflow are not a new type of vulnerability, but they are particularly dangerous in smart contracts. Overflow scenarios can lead to incorrect

Knownsec

results, especially if the possibility is not anticipated, and can affect the reliability and security of the program.

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

3.3. Access control detection [pass]

Access control defect is a security risk that can exist in all programs. Smart contract also has similar problem. The famous Parity Wallet smart contract was affected by this problem.

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

3.4. Return value call validation [pass]

This problem occurs mostly in smart contracts associated with currency transfers, so it is also known as silent failed or unchecked send.

A transfer method, such as transfer(), send(), or call.value(), could all be used to send Ether to an address, with the difference between throw and state rollback if the transfer fails;Only 2300GAS will be passed for invocation to prevent reentrant attack; Send returns false on failure; Only 2300GAS will be passed for invocation to prevent reentrant attack; Call.value returns false on failure; Passing all available

Knownsec

gas for invocation (which can be restricted by passing in the GAS_value parameter) does not effectively prevent a reentrant attack.

If the return value of the send and call.value transfer function above is not checked in the code, the contract will continue to execute the following code, possibly causing unexpected results due to the failure of Ether to send.

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

3.5. Error using random number [pass]

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.

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

3.6. Transaction order dependence [Low risk]

Since miners always get gas fees through a code that represents an externally

Knownsec

owned address (EOA), users can specify higher fees for faster transactions.Because 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 cost to preempt the original solution.

Detection results: After detection, there is a risk of transaction order dependency attack in the approve function of FOUR token contract. The code is as follows:

```
function approve(address _spender, uint256 _value) public returns (bool) {
    require(_spender != address(0));

    allowed[msg.sender][_spender] = _value;
    Approval(msg.sender, _spender, _value);

    return true;
}
```

The possible security risks are described as follows:

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

Safety advice:

Knownsec

1. Front-end restriction: when user A changes the amount from N to M, it can be modified from N to 0 and then from 0 to M.

2. Add the following code at the beginning of approve:

```
require((value == 0) || (allowance[msg.sender][spender] == 0));
```

3.7. Denial of service attack [pass]

In ethereum's world, denial of service is deadly, and smart contracts that suffer from this type of attack may never return to normal functioning. The reasons for intelligent contract denial of service can be many, including malicious behavior while on the receiving end of a transaction, gas depletion due to the artificial addition of needed gas for computing functions, abuse of access control to access private components of intelligent contracts, exploitation of obtuse and negligence, and so on.

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

3.8. Logical design flaw [pass]

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

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

Knownsec

3.9. False recharge vulnerability [pass]

In the transfer function of the token contract, the balance check over the originator (MSG. sender) becomes an if judgment method. When the veto [MSg. sender] < value, enter the else logic part and return false, no exception will become available. We believe that the if/else gentle judgment method is an unrigorous coding method in the transfer sensitive function scene.

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

3.10. Issue of token loopholes [pass]

Checks if there is a function in the scrip contract that is likely to increase the scrip total after initializing the scrip total.

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

3.11. Freeze account bypass [pass]

Check whether the source account, the originating account and the target account are frozen when the token is transferred in the token contract.

Detection results: After detection, there is no security problem in the intelligent contract code.

Knownsec

Safety advice：None.

Knownsec

4. Appendix A: Contract code

Source code for this test：

```
/**
 *Submitted for verification at Etherscan.io on 2018-04-23
*/

pragma solidity ^0.4.17;

// File: contracts/helpers/Ownable.sol

/**
 * @title Ownable
 * @dev The Ownable contract has an owner address, and provides basic authorization control
 * functions, this simplifies the implementation of "user permissions".
 */
contract Ownable {
  address public owner;

  event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

  /**
   * @dev The Constructor sets the original owner of the contract to the
   * sender account.
   */
  function Ownable() public {
    owner = msg.sender;
  }

  /**
   * @dev Throws if called by any other account other than owner.
   */
  modifier onlyOwner() {
    require(msg.sender == owner);
    _;
  }

  /**
    * @dev Allows the current owner to transfer control of the contract to a newOwner.
    * @param newOwner The address to transfer ownership to.
    */
  function transferOwnership(address newOwner) public onlyOwner {
    require(newOwner != address(0));
    OwnershipTransferred(owner, newOwner);
    owner = newOwner;
  }

}

// File: contracts/helpers/SafeMath.sol

/**
 * @title SafeMath
 * @dev Math operations with safety checks that throw on error
 */
library SafeMath {

  /**
  * @dev Multiplies two numbers, throws on overflow.
  */
  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    if (a == 0) {
      return 0;
    }
    uint256 c = a * b;
    assert(c / a == b);
    return c;
  }

  /**
  * @dev Integer division of two numbers, truncating the quotient.
  */
  function div(uint256 a, uint256 b) internal pure returns (uint256) {
    // assert(b > 0); // Solidity automatically throws when dividing by 0
    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
    return c;
  }

  /**
```

Knownsec

```
     * @dev Substracts two numbers, throws on overflow (i.e. if subtrahend is greater than minuend).
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        assert(b <= a);
        return a - b;
    }

    /**
     * @dev Adds two numbers, throws on overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        assert(c >= a);
        return c;
    }
}

// File: contracts/token/ERC20Interface.sol

contract ERC20Interface {

    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);

    function totalSupply() public view returns (uint256);
    function balanceOf(address who) public view returns (uint256);
    function transfer(address to, uint256 value) public returns (bool);

    function approve(address spender, uint256 value) public returns (bool);
    function transferFrom(address from, address to, uint256 value) public returns (bool);
    function allowance(address owner, address spender) public view returns (uint256);

}

// File: contracts/token/BaseToken.sol

contract BaseToken is ERC20Interface {
    using SafeMath for uint256;

    mapping(address => uint256) balances;
    mapping (address => mapping (address => uint256)) internal allowed;

    uint256 totalSupply_;

    /**
     * @dev Obtain total number of tokens in existence
     */
    function totalSupply() public view returns (uint256) {
        return totalSupply_;
    }

    /**
     * @dev Gets the balance of the specified address.
     * @param _owner The address to query the the balance of.
     * @return An uint256 representing the amount owned by the passed address.
     */
    function balanceOf(address _owner) public view returns (uint256 balance) {
        return balances[_owner];
    }

    /**
     * @dev Transfer token for a specified address
     * @param _to The address to transfer to.
     * @param _value The amount to be transferred.
     */
    function transfer(address _to, uint256 _value) public returns (bool) {
        require(_to != address(0));
        require(_value <= balances[msg.sender]);

        // SafeMath.sub will throw if there is not enough balance
        balances[msg.sender] = balances[msg.sender].sub(_value);
        balances[_to] = balances[_to].add(_value);
        Transfer(msg.sender, _to, _value);

        return true;
    }

    /**
      * @dev Approve the passed address to spend the specified amount of tokens on behalf of msg.sender.
      *
      * Beware that changing an allowance with this method brings the risk that someone may use both the old
      * and the new allowance by unfortunate transaction ordering. One possible solution to mitigate this
      * race condition is to first reduce the spender's allowance to 0 and set the desired value afterwards:
      * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
      * @param _spender The address which will spend the funds.
      * @param _value The amount of tokens to be spent.
```

Knownsec

```
    */
  function approve(address _spender, uint256 _value) public returns (bool) {
      require(_spender != address(0));

      allowed[msg.sender][_spender] = _value;
      Approval(msg.sender, _spender, _value);

      return true;
  }

  /**
    * @dev Transfer tokens from one address to another
    * @param _from address The address which you want to send tokens from
    * @param _to address The address which you want to transfer to
    * @param _value uint256 the amount of tokens to be transferred
    */
  function transferFrom(address _from, address _to, uint256 _value) public returns (bool) {
      require(_to != address(0));
      require(_value <= balances[_from]);
      require(_value <= allowed[_from][msg.sender]);

      balances[_from] = balances[_from].sub(_value);
      balances[_to] = balances[_to].add(_value);
      allowed[_from][msg.sender] = allowed[_from][msg.sender].sub(_value);

      Transfer(_from, _to, _value);

      return true;
  }

  /**
    * @dev Function to check the amount of tokens that an owner allowed to a spender.
    * @param _owner address The address which owns the funds.
    * @param _spender address The address which will spend the funds.
    * @return A uint256 specifying the amount of tokens still available for the spender.
    */
  function allowance(address _owner, address _spender) public view returns (uint256) {
      return allowed[_owner][_spender];
  }

  /**
    * @dev Increase the amount of tokens that an owner allowed to a spender.
    *
    * approve should be called when allowed[_spender] == 0. To increment
    * allowed value is better to use this function to avoid 2 calls (and wait until
    * the first transaction is mined)
    * From MonolithDAO Token.sol
    * @param _spender The address which will spend the funds.
    * @param _addedValue The amount of tokens to increase the allowance by.
    */
  function increaseApproval(address _spender, uint256 _addedValue) public returns (bool) {
      allowed[msg.sender][_spender] = allowed[msg.sender][_spender].add(_addedValue);
      Approval(msg.sender, _spender, allowed[msg.sender][_spender]);

      return true;
  }

  /**
    * @dev Decrease the amount of tokens that an owner allowed to a spender.
    *
    * approve should be called when allowed[_spender] == 0. To decrement
    * allowed value is better to use this function to avoid 2 calls (and wait until
    * the first transaction is mined)
    * From MonolithDAO Token.sol
    * @param _spender The address which will spend the funds.
    * @param _subtractedValue The amount of tokens to decrease the allowance by.
    */
  function decreaseApproval(address _spender, uint256 _subtractedValue) public returns (bool) {
      uint oldValue = allowed[msg.sender][_spender];
      if (_subtractedValue > oldValue) {
          allowed[msg.sender][_spender] = 0;
      } else {
          allowed[msg.sender][_spender] = oldValue.sub(_subtractedValue);
      }
      Approval(msg.sender, _spender, allowed[msg.sender][_spender]);
      return true;
  }

}

// File: contracts/token/MintableToken.sol

/**
 * @title Mintable token
 * @dev Simple ERC20 Token example, with mintable token creation
 * @dev Issue: * https://github.com/OpenZeppelin/zeppelin-solidity/issues/120
 *              Based              on              code              by              TokenMarketNet:
```

knownsec

```
https://github.com/TokenMarketNet/ico/blob/master/contracts/MintableToken.sol
 */
contract MintableToken is BaseToken, Ownable {

  event Mint(address indexed to, uint256 amount);
  event MintFinished();

  bool public mintingFinished = false;

  modifier canMint() {
    require(!mintingFinished);
    _;
  }

  /**
   * @dev Function to mint tokens
   * @param _to The address that will receive the minted tokens.
   * @param _amount The amount of tokens to mint.
   * @return A boolean that indicates if the operation was successful.
   */
  function mint(address _to, uint256 _amount) onlyOwner canMint public returns (bool) {
    require(_to != address(0));

    totalSupply_ = totalSupply_.add(_amount);
    balances[_to] = balances[_to].add(_amount);
    Mint(_to, _amount);
    Transfer(address(0), _to, _amount);
    return true;
  }

  /**
   * @dev Function to stop minting new tokens.
   * @return True if the operation was successful.
   */
  function finishMinting() onlyOwner canMint public returns (bool) {
    mintingFinished = true;
    MintFinished();
    return true;
  }

}

// File: contracts/token/CappedToken.sol

contract CappedToken is MintableToken {

  uint256 public cap;

  function CappedToken(uint256 _cap) public {
    require(_cap > 0);
    cap = _cap;
  }

  function mint(address _to, uint256 _amount) onlyOwner canMint public returns (bool) {
    require(totalSupply_.add(_amount) <= cap);

    return super.mint(_to, _amount);
  }

}

// File: contracts/helpers/Pausable.sol

/**
 * @title Pausable
 * @dev Base contract which allows children to implement an emergency stop mechanism.
 */
contract Pausable is Ownable {
  event Pause();
  event Unpause();

  bool public paused = false;

  /**
   * @dev Modifier to make a function callable only when the contract is not paused.
   */
  modifier whenNotPaused() {
    require(!paused);
    _;
  }

  /**
   * @dev Modifier to make a function callable only when the contract is paused.
   */
  modifier whenPaused() {
    require(paused);
    _;
```

Knownsec

```
  }

  /**
   * @dev called by the owner to pause, triggers stopped state
   */
  function pause() onlyOwner whenNotPaused public {
    paused = true;
    Pause();
  }

  /**
   * @dev called by the owner to unpause, returns to normal state
   */
  function unpause() onlyOwner whenPaused public {
    paused = false;
    Unpause();
  }
}

// File: contracts/token/PausableToken.sol

/**
 * @title Pausable token
 * @dev BaseToken modified with pausable transfers.
 **/
contract PausableToken is BaseToken, Pausable {

  function transfer(address _to, uint256 _value) public whenNotPaused returns (bool) {
    return super.transfer(_to, _value);
  }

  function transferFrom(address _from, address _to, uint256 _value) public whenNotPaused returns (bool) {
    return super.transferFrom(_from, _to, _value);
  }

  function approve(address _spender, uint256 _value) public whenNotPaused returns (bool) {
    return super.approve(_spender, _value);
  }

  function increaseApproval(address _spender, uint _addedValue) public whenNotPaused returns (bool success)
{
    return super.increaseApproval(_spender, _addedValue);
  }

  function decreaseApproval(address _spender, uint _subtractedValue) public whenNotPaused returns (bool
success) {
    return super.decreaseApproval(_spender, _subtractedValue);
  }
}

// File: contracts/token/SignedTransferToken.sol

/**
* @title SignedTransferToken
* @dev The SignedTransferToken enables collection of fees for token transfers
* in native token currency. User will provide a signature that allows the third
* party to settle the transaction in his name and collect fee for provided
* serivce.
*/
contract SignedTransferToken is BaseToken {

  event TransferPreSigned(
    address indexed from,
    address indexed to,
    address indexed settler,
    uint256 value,
    uint256 fee
  );

  event TransferPreSignedMany(
    address indexed from,
    address indexed settler,
    uint256 value,
    uint256 fee
  );


  // Mapping of already executed settlements for a given address
  mapping(address => mapping(bytes32 => bool)) executedSettlements;

  /**
  * @dev Will settle a pre-signed transfer
  */
  function transferPreSigned(address _from,
                             address _to,
                             uint256 _value,
```

Knownsec

```
                                    uint256 _fee,
                                    uint256 _nonce,
                                    uint8 _v,
                                    bytes32 _r,
                                    bytes32 _s) public returns (bool) {
    uint256 total = _value.add(_fee);
    bytes32 calcHash = calculateHash(_from, _to, _value, _fee, _nonce);

    require(_to != address(0));
    require(isValidSignature(_from, calcHash, _v, _r, _s));
    require(balances[_from] >= total);
    require(!executedSettlements[_from][calcHash]);

    executedSettlements[_from][calcHash] = true;

    // Move tokens
    balances[_from] = balances[_from].sub(_value);
    balances[_to] = balances[_to].add(_value);
    Transfer(_from, _to, _value);

    // Move fee
    balances[_from] = balances[_from].sub(_fee);
    balances[msg.sender] = balances[msg.sender].add(_fee);
    Transfer(_from, msg.sender, _fee);

    TransferPreSigned(_from, _to, msg.sender, _value, _fee);

    return true;
}

/**
* @dev Settle multiple transactions in a single call. Please note that
* should a single one fail the full state will be reverted. Your client
* implementation should always first check for balances, correct signatures
* and any other conditions that might result in failed transaction.
*/
function transferPreSignedBulk(address[] _from,
                                    address[] _to,
                                    uint256[] _values,
                                    uint256[] _fees,
                                    uint256[] _nonces,
                                    uint8[] _v,
                                    bytes32[] _r,
                                    bytes32[] _s) public returns (bool) {
    // Make sure all the arrays are of the same length
    require(_from.length == _to.length &&
            _to.length == _values.length &&
            _values.length == _fees.length &&
            _fees.length == _nonces.length &&
            _nonces.length == _v.length &&
            _v.length == _r.length &&
            _r.length == _s.length);

    for(uint i; i < _from.length; i++) {
        transferPreSigned(_from[i],
                          _to[i],
                          _values[i],
                          _fees[i],
                          _nonces[i],
                          _v[i],
                          _r[i],
                          _s[i]);
    }

    return true;
}


function transferPreSignedMany(address _from,
                                    address[] _tos,
                                    uint256[] _values,
                                    uint256 _fee,
                                    uint256 _nonce,
                                    uint8 _v,
                                    bytes32 _r,
                                    bytes32 _s) public returns (bool) {
   require(_tos.length == _values.length);
   uint256 total = getTotal(_tos, _values, _fee);

   bytes32 calcHash = calculateManyHash(_from, _tos, _values, _fee, _nonce);

   require(isValidSignature(_from, calcHash, _v, _r, _s));
   require(balances[_from] >= total);
   require(!executedSettlements[_from][calcHash]);

   executedSettlements[_from][calcHash] = true;
```

Knownsec

```
    // transfer to each recipient and take fee at the end
    for(uint i; i < _tos.length; i++) {
        // Move tokens
        balances[_from] = balances[_from].sub(_values[i]);
        balances[_tos[i]] = balances[_tos[i]].add(_values[i]);
        Transfer(_from, _tos[i], _values[i]);
    }

    // Move fee
    balances[_from] = balances[_from].sub(_fee);
    balances[msg.sender] = balances[msg.sender].add(_fee);
    Transfer(_from, msg.sender, _fee);

    TransferPreSignedMany(_from, msg.sender, total, _fee);

    return true;
  }

  function getTotal(address[] _tos, uint256[] _values, uint256 _fee) private view returns (uint256)    {
      uint256 total = _fee;

      for(uint i; i < _tos.length; i++) {
          total = total.add(_values[i]); // sum of all the values + fee
          require(_tos[i] != address(0)); // check that the recipient is a valid address
      }

      return total;
  }

  /**
   * @dev Calculates transfer hash for transferPreSignedMany
   */
  function calculateManyHash(address _from, address[] _tos, uint256[] _values, uint256 _fee, uint256 _nonce)
public view returns (bytes32) {
      return keccak256(uint256(1), address(this), _from, _tos, _values, _fee, _nonce);
  }

  /**
   * @dev Calculates transfer hash.
   */
  function calculateHash(address _from, address _to, uint256 _value, uint256 _fee, uint256 _nonce) public view
returns (bytes32) {
      return keccak256(uint256(0), address(this), _from, _to, _value, _fee, _nonce);
  }

  /**
   * @dev Validates the signature
   */
  function isValidSignature(address _signer, bytes32 _hash, uint8 _v, bytes32 _r, bytes32 _s) public pure returns
(bool) {
      return _signer == ecrecover(
              keccak256("\x19Ethereum Signed Message:\n32", _hash),
              _v,
              _r,
              _s
          );
  }

  /**
   * @dev Allows you to check whether a certain transaction has been already
   * settled or not.
   */
  function isTransactionAlreadySettled(address _from, bytes32 _calcHash) public view returns (bool) {
      return executedSettlements[_from][_calcHash];
  }

}

// File: contracts/token/PausableSignedTransferToken.sol

contract PausableSignedTransferToken is SignedTransferToken, PausableToken {

  function transferPreSigned(address _from,
                             address _to,
                             uint256 _value,
                             uint256 _fee,
                             uint256 _nonce,
                             uint8 _v,
                             bytes32 _r,
                             bytes32 _s) public whenNotPaused returns (bool) {
      return super.transferPreSigned(_from, _to, _value, _fee, _nonce, _v, _r, _s);
  }

  function transferPreSignedBulk(address[] _from,
                                 address[] _to,
                                 uint256[] _values,
                                 uint256[] _fees,
```

```
                                        uint256[] _nonces,
                                        uint8[] _v,
                                        bytes32[] _r,
                                        bytes32[] _s) public whenNotPaused returns (bool) {
        return super.transferPreSignedBulk(_from, _to, _values, _fees, _nonces, _v, _r, _s);
    }

    function transferPreSignedMany(address _from,
                                        address[] _tos,
                                        uint256[] _values,
                                        uint256 _fee,
                                        uint256 _nonce,
                                        uint8 _v,
                                        bytes32 _r,
                                        bytes32 _s) public whenNotPaused returns (bool) {
        return super.transferPreSignedMany(_from, _tos, _values, _fee, _nonce, _v, _r, _s);
    }
}

// File: contracts/FourToken.sol

contract FourToken is CappedToken, PausableSignedTransferToken  {
    string public name = 'The 4th Pillar Token';
    string public symbol = 'FOUR';
    uint256 public decimals = 18;

    // Max supply of 400 million
    uint256 public maxSupply = 400000000 * 10**decimals;

    function FourToken()
        CappedToken(maxSupply) public {
            paused = true;
    }

    // @dev Recover any mistakenly sent ERC20 tokens to the Token address
    function recoverERC20Tokens(address _erc20, uint256 _amount) public onlyOwner {
        ERC20Interface(_erc20).transfer(msg.sender, _amount);
    }

}
```

Knownsec

5. Appendix B: Vulnerability risk rating criteria

Smart contract vulnerability rating criteria	
Vulnerability rating	Vulnerability rating description
High risk vulnerabilities	Vulnerabilities that can directly cause losses of token contracts or users' funds, such as: numerical overflow loopholes that can cause the value of token to return to zero, false charging loopholes that can cause losses of tokens in exchanges, or ETH or re-entry loopholes in contract accounts; Vulnerabilities that can cause the loss of escrow rights of token contracts, such as access control defects of key functions, access control bypass of key functions caused by call injection, etc. Vulnerabilities that cause token contracts to not work properly, such as the denial of service vulnerability caused by sending the ETH to a malicious address, or the denial of service vulnerability caused by running out of gas.
In the dangerous holes	High-risk vulnerabilities that require a specific address to trigger, such as numerical overflow vulnerabilities that can only be triggered by the owner of a token contract; Access control defects of non-critical functions, logical design defects that cannot cause direct capital loss, etc.
Low latent loophole	Vulnerabilities that are difficult to be triggered, vulnerabilities that have limited harm after being triggered, such as numerical overflow vulnerabilities that require a large number of ETH or tokens to be triggered, vulnerabilities that the attacker cannot directly profit after triggering numerical overflow, and transaction sequence dependence risks triggered by specifying high gas, etc.

Knownsec

6. Appendix C: Introduction to vulnerability testing tools

6.1. Manticore

A Manticore is a symbolic execution tool for analyzing binary files and intelligent contracts. A Manticore consists of a symbolic Ethereum virtual machine (EVM), an EVM disassembler/assembler, and a convenient interface for automatic compilation and analysis of the Solarium body. It also incorporates Ethersplay, a Bit of Traits of Bits visual disassembler for EVM bytecode, for visual analysis. Like binaries, Manticore provides a simple command-line interface and a Python API for analyzing 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 ordering dependencies, and so on. More conveniently, Oyente's design is modular, so this allows power users to implement and insert their own inspection logic to check the custom properties in their contracts.

6.3. securify.sh

Securify verifies the security issues common to Ethereum's smart contracts, such as unpredictability of trades and lack of input verification, while fully automated and analyzing all possible execution paths, and Securify has a specific language for identifying vulnerabilities that enables the securities to focus on current security and other reliability issues at all times.

6.4. Echidna

Echidna is a Haskell library designed for fuzzy testing EVM code.

6.5. MAIAN

MAIAN is an automated tool used to find holes in Ethereum's intelligent contracts. MAIAN processes the bytecode of the contract and tries to set up a series of transactions to find and confirm errors.

6.6. ethersplay

Ethersplay is an EVM disassembler that includes correlation analysis tools.

6.7. ida-evm

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

6.8. Remix-ide

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

6.9. Know chuangyu penetration tester kit

The tool kit, developed, collected and used by The penetration testing engineer of Knowchuangyu, contains batch automated testing tools, independently developed tools, scripts or utilization tools, etc. for testers.

Knownsec

北京知道创宇信息技术股份有限公司

咨询电话 +86(10)400 060 9587
邮　　箱 sec@knownsec.com
官　　网 www.knownsec.com
地　　址 北京市 朝阳区 望京 SOHO T2-B座-2509