JohnWick Sec

Guard the internet of value

ETH SMART CONTRACT AUDIT REPORT

JOHNWICK SECURITY LAB

WWW.JOHNWICK.IO

John Wick Security Lab received the **XTT** (company/team) **Xin Time Token (XTT)** project smart contract code audit requirements on **2019/04/07.**

Project Name：Xin Time Token (XTT)

Smart Contract Address：

https://etherscan.io/address/0xcb9a14d68cd0690b3696f42dcfdf609a67824736#code

Audit Number：20190403

Audit Date：20190407

Audit Category and Result:

Main Category	Subcategory	Result
Contract programming	Solidity version not specified	Pass
	Solidity version too old	Pass
	Integer overflow/underflow	Pass
	Function input parameters lack of check	Pass
	Function input parameters check bypass	Pass
	Function access control lacks management	Pass
	Critical operation lacks event log	Pass
	human/contract checks bypass	Pass
	Random number generation/use vulnerability	Pass
	Fallback function misuse	Pass
	Race condition	Not Pass
	Logical vulnerability	Pass
	Other programming issues	Not Pass
Code specification	Function visibility not explicitly declared	Pass
	Var. storage location not explicitly declared	Pass
	Use keywords/functions to be deprecated	Pass
	Other code specification issues	Pass
GAS optimization	assert() misuse	Pass
	High consumption `for/while` loop	Pass
	High consumption `storage` storage	Pass
	“Out of Gas” Attack	Pass
Business Risk	Evil mint/burn	Pass
	The maximum limit for mintage not sets	Pass
	"Fake Charge" Attack	Pass
	"Short Address" Attack	Pass
	"Double Spend" Attack	Pass
Auto fuzzing		Pass

(Other unknown security vulnerabilities and Ethereum design flaws are not included in this audit responsibility)

Audit Result：PASS

Auditor：John Wick Security Lab

(**Disclaimer:** The John Wick Security Lab issues this report based on the facts that have occurred or existed before the issuance of this report and assumes corresponding responsibility in this regard. For the facts that occur or exist after the issuance of this report, the John Wick Security Lab cannot judge the security status of its smart contracts and does not assume any responsibility for it. The safety audit analysis and other contents of this report are based on the relevant materials and documents provided by the information provider to the John Wick Security Lab when the report is issued (referred to as the information provided). The John Wick Security Lab assumes that there is no missing, falsified, deleted, or concealed information provided. If the information provided is missing, falsified, deleted, concealed, or the information provider's response is inconsistent with the actual situation, the John Wick Security Lab shall not bear any responsibility for the resulting loss and adverse effects.)

Audit Details：

//JohnWick: This smart contract uses the SafeMath library to avoid potential integer overflow issues, which is in line with the recommended practice.

//JohnWick: **[Low Risk] 296L 312L**

The two internal functions _burn(address account, uint256 value) and _burnFrom(address account, uint256 value) are not referenced by other public functions of this smart contract. We recommend deleting them.

//JonhWick: **[Low Risk] 202L**

The approve(address spender, uint256 value) function has a race condition problem.

To avoid this problem, We recommend using the increaseAllowance(address spender, uint256 addedValue) and decreaseAllowance(address spender, uint256 subtractedValue) of this smart contract to achieve atomic increase or decrease _allowed.

Smart Contract Source Code:

```
pragma solidity ^0.5.0;
```

```
// File: node_modules/openzeppelin-solidity/contracts/token/ERC20/IERC20.sol

/**
 * @title ERC20 interface
 * @dev see https://github.com/ethereum/EIPs/issues/20
 */
interface IERC20 {
    function transfer(address to, uint256 value) external returns (bool);

    function approve(address spender, uint256 value) external returns (bool);

    function transferFrom(address from, address to, uint256 value) external
returns (bool);

    function totalSupply() external view returns (uint256);

    function balanceOf(address who) external view returns (uint256);

    function allowance(address owner, address spender) external view returns
(uint256);

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

    event Approval(address indexed owner, address indexed spender, uint256
value);
}

//                                                                    File:
node_modules/openzeppelin-solidity/contracts/token/ERC20/ERC20Detailed.sol

/**
 * @title ERC20Detailed token
 * @dev The decimals are only for visualization purposes.
 * All the operations are done using the smallest and indivisible token unit,
 * just as on Ethereum all the operations are done in wei.
 */
contract ERC20Detailed is IERC20 {
    string private _name;
    string private _symbol;
    uint8 private _decimals;

    constructor (string memory name, string memory symbol, uint8 decimals)
public {
        _name = name;
```

```
        _symbol = symbol;
        _decimals = decimals;
    }

    /**
     * @return the name of the token.
     */
    function name() public view returns (string memory) {
        return _name;
    }

    /**
     * @return the symbol of the token.
     */
    function symbol() public view returns (string memory) {
        return _symbol;
    }

    /**
     * @return the number of decimals of the token.
     */
    function decimals() public view returns (uint8) {
        return _decimals;
    }
}

// File: node_modules/openzeppelin-solidity/contracts/math/SafeMath.sol

/**
 * @title SafeMath
 * @dev Unsigned math operations with safety checks that revert on error
 */
library SafeMath {
    /**
    * @dev Multiplies two unsigned integers, reverts on overflow.
    */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero,
but the
        // benefit is lost if 'b' is also tested.
        //                                                                See:
https://github.com/OpenZeppelin/openzeppelin-solidity/pull/522
        if (a == 0) {
            return 0;
        }
```

```
        uint256 c = a * b;
        require(c / a == b);

        return c;
    }

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

        return c;
    }

    /**
    * @dev Subtracts two unsigned integers, reverts on overflow (i.e. if
subtrahend is greater than minuend).
    */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a);
        uint256 c = a - b;

        return c;
    }

    /**
    * @dev Adds two unsigned integers, reverts on overflow.
    */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a);

        return c;
    }

    /**
    * @dev Divides two unsigned integers and returns the remainder (unsigned
integer modulo),
```

```
    * reverts when dividing by zero.
    */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0);
        return a % b;
    }
}

// File: node_modules/openzeppelin-solidity/contracts/token/ERC20/ERC20.sol

/**
 * @title Standard ERC20 token
 *
 * @dev Implementation of the basic standard token.
 * https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
 * Originally based on code by FirstBlood:
 *
https://github.com/Firstbloodio/token/blob/master/smart_contract/FirstBlood
Token.sol
 *
 * This implementation emits additional Approval events, allowing applications
to reconstruct the allowance status for
 * all accounts just by listening to said events. Note that this isn't required
by the specification, and other
 * compliant implementations may not do it.
 */
contract ERC20 is IERC20 {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;

    mapping (address => mapping (address => uint256)) private _allowed;

    uint256 private _totalSupply;

    /**
    * @dev 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 balance of.
```

```
    * @return An uint256 representing the amount owned by the passed address.
    */
    function balanceOf(address owner) public view returns (uint256) {
        return _balances[owner];
    }

    /**
     * @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 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) {
        _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.
     */
    function approve(address spender, uint256 value) public returns (bool) {
        require(spender != address(0));
```

```
        _allowed[msg.sender][spender] = value;
        emit Approval(msg.sender, spender, value);
        return true;
    }

    /**
     * @dev Transfer tokens from one address to another.
     * Note that while this function emits an Approval event, this is not required
as per the specification,
     * and other compliant implementations may not emit the event.
     * @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) {
        _allowed[from][msg.sender] = _allowed[from][msg.sender].sub(value);
        _transfer(from, to, value);
        emit Approval(from, msg.sender, _allowed[from][msg.sender]);
        return true;
    }

    /**
     * @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
     * Emits an Approval event.
     * @param spender The address which will spend the funds.
     * @param addedValue The amount of tokens to increase the allowance by.
     */
    function increaseAllowance(address spender, uint256 addedValue) public
returns (bool) {
        require(spender != address(0));

        _allowed[msg.sender][spender]                                          =
_allowed[msg.sender][spender].add(addedValue);
        emit 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
    * Emits an Approval event.
    * @param spender The address which will spend the funds.
    * @param subtractedValue The amount of tokens to decrease the allowance
by.
    */
   function  decreaseAllowance(address  spender,  uint256  subtractedValue)
public returns (bool) {
       require(spender != address(0));

       _allowed[msg.sender][spender]                                          =
_allowed[msg.sender][spender].sub(subtractedValue);
       emit Approval(msg.sender, spender, _allowed[msg.sender][spender]);
       return true;
   }

   /**
   * @dev Transfer token for a specified addresses
   * @param from The address to transfer from.
   * @param to The address to transfer to.
   * @param value The amount to be transferred.
   */
   function _transfer(address from, address to, uint256 value) internal {
       require(to != address(0));

       _balances[from] = _balances[from].sub(value);
       _balances[to] = _balances[to].add(value);
       emit Transfer(from, to, value);
   }

   /**
    * @dev Internal function that mints an amount of the token and assigns it
to
    * an account. This encapsulates the modification of balances such that the
    * proper events are emitted.
    * @param account The account that will receive the created tokens.
    * @param value The amount that will be created.
    */
   function _mint(address account, uint256 value) internal {
       require(account != address(0));
```

```
        _totalSupply = _totalSupply.add(value);
        _balances[account] = _balances[account].add(value);
        emit Transfer(address(0), account, value);
    }

    /**
     * @dev Internal function that burns an amount of the token of a given
     * account.
     * @param account The account whose tokens will be burnt.
     * @param value The amount that will be burnt.
     */
    function _burn(address account, uint256 value) internal {
        require(account != address(0));

        _totalSupply = _totalSupply.sub(value);
        _balances[account] = _balances[account].sub(value);
        emit Transfer(account, address(0), value);
    }

    /**
     * @dev Internal function that burns an amount of the token of a given
     * account, deducting from the sender's allowance for said account. Uses
the
     * internal burn function.
     * Emits an Approval event (reflecting the reduced allowance).
     * @param account The account whose tokens will be burnt.
     * @param value The amount that will be burnt.
     */
    function _burnFrom(address account, uint256 value) internal {
        _allowed[account][msg.sender]                                    =
_allowed[account][msg.sender].sub(value);
        _burn(account, value);
        emit Approval(account, msg.sender, _allowed[account][msg.sender]);
    }
}

// File: node_modules/openzeppelin-solidity/contracts/ownership/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 private _owner;

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

    /**
     * @dev The Ownable constructor sets the original `owner` of the contract
to the sender
     * account.
     */
    constructor () internal {
        _owner = msg.sender;
        emit OwnershipTransferred(address(0), _owner);
    }

    /**
     * @return the address of the owner.
     */
    function owner() public view returns (address) {
        return _owner;
    }

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

    /**
     * @return true if `msg.sender` is the owner of the contract.
     */
    function isOwner() public view returns (bool) {
        return msg.sender == _owner;
    }

    /**
     * @dev Allows the current owner to relinquish control of the contract.
     * @notice Renouncing to ownership will leave the contract without an owner.
     * It will not be possible to call the functions with the `onlyOwner`
     * modifier anymore.
     */
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
```

```
        _owner = address(0);
    }

    /**
     * @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 {
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers control of the contract to a newOwner.
     * @param newOwner The address to transfer ownership to.
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0));
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

// File: contracts/LockedPosition.sol

// 锁仓认购期的账户
contract LockedPosition is ERC20, Ownable {

    mapping (address => uint256) private _partners;
    mapping (address => uint256) private _release;

    bool public publish = false;
    uint256 public released = 0;

    /**
    * @dev update account to partner list
    * @return
    */
    function partner(address from, address to, uint256 value) internal {
        require(from != address(0), "The from address is empty");
        require(to != address(0), "The to address is empty");

        if(publish){
            // 已发行
            _release[from] = _release[from].add(value);
```

```
        }else{
            // 未发行
            if(owner() != from){
                _partners[from] = _partners[from].sub(value);
            }
            if(owner() != to){
                _partners[to] = _partners[to].add(value);
            }
        }
    }
    /**
     * @dev check an account position
     * @return bool
     */
    function checkPosition(address account, uint256 value) internal view
returns (bool) {
        require(account != address(0), "The account address is empty");
        // 发行人，不限制
        if (isOwner()){
            return true;
        }
        // 未发行，未锁仓，不限制
        if (!publish){
            return true;
        }
        // 放开率 100%，不限制
        if (released >= 100) {
            return true;
        }
        // 锁仓之后的交易地址，不限制
        if(_partners[account]==0){
            return true;
        }
        // 已经发行，锁定
        return ((_partners[account]/100) * released) >= _release[account] +
value;
    }

    /**
     * @dev locked partners account
     * @return
     */
    function locked() external onlyOwner {
        publish = true;
    }
```

```
    /**
     * @dev release position
     * @return
     */
    function release(uint256 percent) external onlyOwner {
        require(percent <= 100 && percent > 0, "The released must be between 0
and 100");
        released = percent;
    }
     /**
     * @dev get account position
     * @return bool
     */
    function getPosition() external view returns(uint256) {
        return _partners[msg.sender];
    }

    /**
     * @dev get account release
     * @return bool
     */
    function getRelease() external view returns(uint256) {
        return _release[msg.sender];
    }

    /**
     * @dev get account position
     * @return bool
     */
    function positionOf(address account) external onlyOwner view
returns(uint256) {
        require(account != address(0), "The account address is empty");
        return _partners[account];
    }

    /**
     * @dev get account release
     * @return bool
     */
    function releaseOf(address account) external onlyOwner view returns(uint256)
{
        require(account != address(0), "The account address is empty");
        return _release[account];
    }
```

```
    function transfer(address to, uint256 value) public returns (bool) {
        require(checkPosition(msg.sender, value), "Insufficient positions");

        partner(msg.sender, to, value);

        return super.transfer(to, value);
    }

    function transferFrom(address from,address to, uint256 value) public
returns (bool) {
        require(checkPosition(from, value), "Insufficient positions");

        partner(from, to, value);

        return super.transferFrom(from, to, value);
    }
}

// File: contracts/XinTimeToken.sol

contract XinTimeToken is ERC20Detailed, LockedPosition {
    uint256 private constant INITIAL_SUPPLY = 2 * (10**8) * (10**18);

    constructor () public ERC20Detailed("Xin Time Token", "XTT", 18){
        _mint(msg.sender, INITIAL_SUPPLY);
        emit Transfer(address(0), msg.sender, totalSupply());
    }

}
```