KNOWNSEC

Degree of secrets ★ | Project team open

Smart contract audit report

Security status

Security

★ ★ ★ ★ ★

Chief test Officer :

Release Notes

Amendment	Revised date	Report Version
Document	2020/9/15	V1.0

Document Information

Title	Version	ID	Classification
Smart Contract Audit Report	V1.0	【JGN-20200914】	Project Team Open

Disclaimer

This audit is only applied to the type of auditing specified in this report and the scope of given in the results table. Other unknown security vulnerabilities are beyond auditing responsibility. Knownsec Inc. (ZhidaoChuangyu) only issues this report based on the attacks or vulnerabilities that already existed or occurred before the issuance of this report. For the emergence of new attacks or vulnerabilities that exist or occur in the future, Knownsec Inc. (ZhidaoChuangyu) lacks the capability to judge its possible impact on the security status of smart contracts, thus taking no responsibility for them. The security audit analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to Knownsec Inc. (ZhidaoChuangyu) before the issuance of this report, and the contract provider warrants that there are no missing, tampered, deleted materials; if the documents and materials provided by the contract provider are missing, tampered, deleted, concealed or reflected

in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, Knownsec Inc. (ZhidaoChuangyu) assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Knownsec Inc. (ZhidaoChuangyu) is based on the documents and materials provided by the contract provider and relies on the technology currently possessed by Knownsec Inc. (ZhidaoChuangyu). Due to the technical limitations of any organization, this report conducted by Knownsec Inc. (ZhidaoChuangyu) still has the possibility that the entire risk cannot be completely detected. Knownsec Inc. (ZhidaoChuangyu) disclaims any liability for the resulting losses.

Table of Contents

1. Executive Summary

The audit was carried out from September 15, 2020, to September 16, 2020. During this period, two major aspects of **smart contract** JGN were audited, including Coding Standards and Security; they are used as the report's statistical basis.

In this analysis, we know that Knownsec's engineers have conducted a comprehensive audit of common vulnerabilities of smart contracts (see Chapter 3). JGN contract passed all audit items. The overall result is **Pass (Distinction)**.

Result of the Smart Contract Audit： PASS

All codes are the latest backup since the test process is conducted in a non-production environment. The test process is communicated with relevant personnel, and relevant test operation is performed under the condition of controllable risk, so as to avoid the operational risk and code security risk in the test process.

Basic Information of JGN:

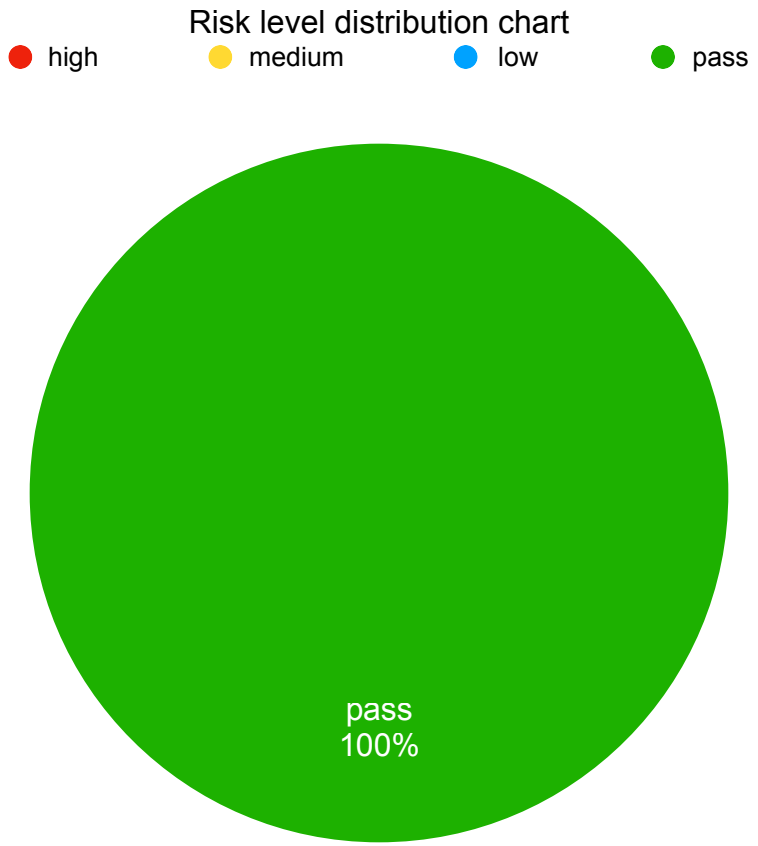
Items	
Code Type	Token
Language	Solidity
Smart Contract Address	0x56f95662e71f30b333b456439248c6de589082a4
Smart Contract Address Link	https://etherscan.io/address/0x56f95662e71f30b333b456439248c6de589082a4

2. Vulnerability Analysis

2.1 Vulnerability Severity

Test Results			
High	Medium	Low	Pass
0	0	0	11

Vulnerability issues found based on severity



2.2 Summary of the Results

Audit Results Overview			
Audit Subject	Check Items	Results	Comments
Smart Contract	Reentrancy	PASS	No issue found.
	Integer Overflow/Underflow	PASS	No issue found.
	Access Control of Owner	PASS	No issue found.
	Returned Value Security	PASS	No issue found.
	Pseudo-random Number Generator (PRNG)	PASS	No issue found.
	Transaction-Ordering Dependence	PASS	No issue found.
	DoS (Denial of Service)	PASS	No issue found.
	Business Logics	PASS	No issue found.
	Fake Deposit	PASS	No issue found.
	Token Vesting Implementation	PASS	No issue found.
	Frozen Account Bypass	PASS	No issue found.

3. Detailed Results

3.1 Reentrancy 【pass】

Reentrancy is the most infamous smart contract vulnerability in Ethereum. It once resulted in The Dao Hack where a coder abused a loophole in The Dao's split function.

When call.value() function in Solidity is used to send Ether, it consumes all the gas it receives. If call.value() function is invoked to sends Ether before the balance of the sender's account is actually reduced, there is a risk of Reentrancy.

Evaluation Result: This security risk does not exist in the tested smart contract code.

Recommendation: None.

3.2 Integer Overflow/Underflow 【pass】

The arithmetic problem in smart contracts refers to integer overflow and underflow.

Solidity can handle 256-bit numbers at most. Increasing the maximum number by 1 will overflow to get 0. Similarly, when the number is an unsigned type, 0 minus 1 will underflow to get the maximum number value.

Integer overflow and underflow are not new types of vulnerabilities, but they are particularly dangerous in smart contracts. Overflow can lead to data misrepresentation, especially if the possibility is not anticipated, it may affect the program's reliability and security.

Evaluation Result: This security risk does not exist in the tested smart contract code.

Recommendation: None.

3.3 Access Control of Owner 【pass】

Access security risk is common to all computer programs, including smart contracts. A hacker once exploited this vulnerability in Parity Wallet and stole over 150,000 ETH (~30M USD).

Evaluation Result: This security risk does not exist in the tested smart contract code.

Recommendation: None.

3.4 Returned Value Security 【pass】

This problem often occurs in smart contracts related to digital currency transfer, so it is also called silent failed-send or unchecked-send.

In Solidity, there are transfer(), send(), and call.value () and other currency transfer methods can be used to send ether to an address. The difference is: It will throw and rollback when the transfer fails. There will only be 2300 gas sent for calling to prevent reentrancy. If sending fails, it will return false. Also, 2300 gas will be passed for calling to prevent reentrancy. call.value () will also return false if the transfer fails. It will pass all available gas to call (the amount can also be limited by gas_ Value parameter), which will not effectively prevent reentrancy.

Suppose the return value of the previous send and call.value currency transfer function is not checked in the codes. In that case, the contract will continue to execute the following codes, which may cause unintentional results due to the failure of ether sending.

Evaluation Result: This security risk does not exist in the tested smart contract code.

Recommendation: None.

3.5 Pseudo-random Number Generator (PRNG) 【pass】

It may be necessary to use random numbers in smart contracts. Although functions and variables provided by Solidity can access values that are obviously difficult to predict, such as block.number and block.timestamp, they are usually either more public than they seem or influenced by miners, which means that these random numbers are, to some extent predictable. Therefore, malicious users can usually copy it and attack this function by relying on its unpredictability.

Evaluation Result: This security risk does not exist in the tested smart contract code.

Recommendation: None.

3.6 Transaction-Ordering Dependence 【pass】

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

Evaluation Result: This security risk does not exist in the tested smart contract code.

Recommendation: None.

3.7 DoS (Denial of Service) 【pass】

In the Ethereum world, denial of service is fatal, and a smart contract that suffers this type of attack may never return to regular operation. There are many reasons for the denial of service of smart contracts, including malicious behaviour when acting as the transaction receiver, gas depletion caused by the artificial increase of computing function, abuse of access control to access private components of smart contract, confusion and negligence, etc.

Evaluation Result: This security risk does not exist in the tested smart contract code.

Recommendation: None.

3.8 Business Logics 【pass】

Detect security issues related to business design in smart contracts.

Evaluation Result: This security risk does not exist in tested smart contract code.

Recommendation: None.

3.9 Fake Deposit 【pass】

Suppose the smart contract function uses the “if” method to check the balance of the transfer initiator (msg.sender). When balances [msg.sender] < value, it enters “else” and return false, and doesn’t throw an exception. We don’t think it is rigorous coding to use such a mild mechanism like “if/else” in this kind of sensitive scenario like “transfer”.

Evaluation Result: This security risk does not exist in the tested smart contract code.

Recommendation: None.

3.10 Token Vesting Implementation 【pass】

Detect after initializing the total amount of tokens, whether there is a function in the smart contract that may increase the total amount of tokens.

Evaluation Result: This security risk does not exist in the tested smart contract code.

Recommendation: None.

3.11 Frozen Account Bypass 【pass】

Check whether the token source account, initiating account and target account are frozen during token transfer in smart contracts.

Evaluation Result: This security risk does not exist in the tested smart contract code.

Recommendation: None.

4. Appendix A — Contract Codes

```
/**
 *Submitted for verification at Etherscan.io on 2020-09-14
*/

/*
   ____            __   __        __   _
  / __/__ __ ___  / /_ / /  ___  / /_ (_)__ __
 _\ \ / // // _ \/ __// _ \/ -_)/ __// / \ \ /
/___/ \_, //_//_/\__//_//_/\__/ \__//_/ /_\_\
     /___/

* Synthetix: JGNRewards.sol
*
* Docs: https://docs.synthetix.io/
*
*
* MIT License
* ===========
*
* Copyright (c) 2020 Synthetix
*
* Permission is hereby granted, free of charge, to any person obtaining a copy
* of this software and associated documentation files (the "Software"), to deal
* in the Software without restriction, including without limitation the rights
* to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
* copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
*
* The above copyright notice and this permission notice shall be included in all
* copies or substantial portions of the Software.
*
* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
* IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
* FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
* AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
* LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
* OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
*/

// File: @openzeppelin/contracts/math/Math.sol

pragma solidity ^0.5.0;

/**
 * @dev Standard math utilities missing in the Solidity language.
 */
library Math {
    /**
     * @dev Returns the largest of two numbers.
     */
    function max(uint256 a, uint256 b) internal pure returns (uint256) {
        return a >= b ? a : b;
    }

    /**
     * @dev Returns the smallest of two numbers.
     */
    function min(uint256 a, uint256 b) internal pure returns (uint256) {
        return a < b ? a : b;
    }

    /**
     * @dev Returns the average of two numbers. The result is rounded towards
     * zero.
     */
    function average(uint256 a, uint256 b) internal pure returns (uint256) {
        // (a + b) / 2 can overflow, so we distribute
        return (a / 2) + (b / 2) + ((a % 2 + b % 2) / 2);
    }
}
```

```
// File: @openzeppelin/contracts/math/SafeMath.sol

pragma solidity ^0.5.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom me
ssage on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     * - Subtraction cannot overflow.
     *
     * _Available since v2.4.0._
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure retur
ns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
```

```
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but th
e
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom
 message on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure retur
ns (uint256) {
        // Solidity only automatically asserts when dividing by 0
        require(b > 0, errorMessage);
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer
 modulo),
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer
 modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
```

```
     *
     * Requirements:
     * - The divisor cannot be zero.
     *
     * _Available since v2.4.0._
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure retur
ns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

// File: @openzeppelin/contracts/GSN/Context.sol

pragma solidity ^0.5.0;

/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
contract Context {
    // Empty internal constructor, to prevent people from mistakenly deploying
    // an instance of this contract, which should be used via inheritance.
    constructor () internal { }
    // solhint-disable-previous-line no-empty-blocks

    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see ht
tps://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

// File: @openzeppelin/contracts/ownership/Ownable.sol

pragma solidity ^0.5.0;

/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
contract Ownable is Context {
    address private _owner;

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

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        _owner = _msgSender();
        emit OwnershipTransferred(address(0), _owner);
    }

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

```
    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner");
        _;
    }

    /**
     * @dev Returns true if the caller is the current owner.
     */
    function isOwner() public view returns (bool) {
        return _msgSender() == _owner;
    }

    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

// File: @openzeppelin/contracts/token/ERC20/IERC20.sol

pragma solidity ^0.5.0;

/**
 * @dev Interface of the ERC20 standard as defined in the EIP. Does not include
 * the optional functions; to access them see {ERC20Detailed}.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
```

```
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256)
;

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: 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
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external
returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

// File: @openzeppelin/contracts/utils/Address.sol

pragma solidity ^0.5.5;

/**
 * @dev Collection of functions related to the address type
 */
library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * This test is non-exhaustive, and there may be false-negatives: during the
     * execution of a contract's constructor, its address will be reported as
     * not containing a contract.
     *
     * IMPORTANT: It is unsafe to assume that an address for which this
     * function returns false is an externally-owned account (EOA) and not a
     * contract.
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        // According to EIP-1052, 0x0 is the value returned for not-
yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is r
```

```
eturned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8
045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != 0x0 && codehash != accountHash);
    }

    /**
     * @dev Converts an `address` into `address payable`. Note that this is
     * simply a type cast: the actual underlying value is not changed.
     *
     * _Available since v2.4.0._
     */
    function toPayable(address account) internal pure returns (address payable) {
        return address(uint160(account));
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-
now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-
the-checks-effects-interactions-pattern[checks-effects-interactions pattern].
     *
     * _Available since v2.4.0._
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-call-value
        (bool success, ) = recipient.call.value(amount)("");
        require(success, "Address: unable to send value, recipient may have reverted")
;
    }
}

// File: @openzeppelin/contracts/token/ERC20/SafeERC20.sol

pragma solidity ^0.5.0;



/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for ERC20;` statement to your co
ntract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to,
value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) i
nternal {
```

```
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector,
from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        // safeApprove should only be called when setting an initial allowance,
        // or when resetting it to zero. To increase and decrease it, use
        // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
        // solhint-disable-next-line max-line-length
        require((value == 0) || (token.allowance(address(this), spender) == 0),
            "SafeERC20: approve from non-zero to non-zero allowance"
        );
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spend
er, value));
    }

    function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) inter
nal {
        uint256 newAllowance = token.allowance(address(this), spender).add(value);
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spend
er, newAllowance));
    }

    function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) inter
nal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(value, "Saf
eERC20: decreased allowance below zero");
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spend
er, newAllowance));
    }

    /**
     * @dev Imitates a Solidity high-
level call (i.e. a regular function call to a contract), relaxing the requirement
     * on the return value: the return value is optional (but if data is returned, it
must not be false).
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data
size checking mechanism, since
        // we're implementing it ourselves.

        // A Solidity high level call has three parts:
        //  1. The target address is checked to verify it contains contract code
        //  2. The call itself is made, and success asserted
        //  3. The return value is decoded, which in turn checks the size of the retur
ned data.
        // solhint-disable-next-line max-line-length
        require(address(token).isContract(), "SafeERC20: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did no
t succeed");
        }
    }
}

// File: contracts/IRewardDistributionRecipient.sol

pragma solidity ^0.5.0;


contract IRewardDistributionRecipient is Ownable {
    address rewardDistribution;

    function notifyRewardAmount(uint256 reward) external;

    modifier onlyRewardDistribution() {
        require(_msgSender() == rewardDistribution, "Caller is not reward distribution
");
```

```
        _;
    }

    function setRewardDistribution(address _rewardDistribution)
        external
        onlyOwner
    {
        rewardDistribution = _rewardDistribution;
    }
}

// File: contracts/CurveRewards.sol

pragma solidity ^0.5.0;




contract LPTokenWrapper {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    IERC20 public y = IERC20(0x048Fe49BE32adfC9ED68C37D32B5ec9Df17b3603);

    uint256 private _totalSupply;
    mapping(address => uint256) private _balances;

    function totalSupply() public view returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address account) public view returns (uint256) {
        return _balances[account];
    }

    function stake(uint256 amount) public {
        _totalSupply = _totalSupply.add(amount);
        _balances[msg.sender] = _balances[msg.sender].add(amount);
        y.safeTransferFrom(msg.sender, address(this), amount);
    }

    function withdraw(uint256 amount) public {
        _totalSupply = _totalSupply.sub(amount);
        _balances[msg.sender] = _balances[msg.sender].sub(amount);
        y.safeTransfer(msg.sender, amount);
    }
}

contract JGNRewards is LPTokenWrapper, IRewardDistributionRecipient {
    IERC20 public jgn = IERC20(0x73374Ea518De7adDD4c2B624C0e8B113955ee041);
    uint256 public constant DURATION = 7 days;

    uint256 public initreward = 10000*1e18;
    uint256 public starttime = 1600185600; //utc+8 2020 09-16 00:00:00
    uint256 public periodFinish = 0;
    uint256 public rewardRate = 0;
    uint256 public lastUpdateTime;
    uint256 public rewardPerTokenStored;
    mapping(address => uint256) public userRewardPerTokenPaid;
    mapping(address => uint256) public rewards;

    event RewardAdded(uint256 reward);
    event Staked(address indexed user, uint256 amount);
    event Withdrawn(address indexed user, uint256 amount);
    event RewardPaid(address indexed user, uint256 reward);

    modifier updateReward(address account) {
        rewardPerTokenStored = rewardPerToken();
        lastUpdateTime = lastTimeRewardApplicable();
        if (account != address(0)) {
            rewards[account] = earned(account);
            userRewardPerTokenPaid[account] = rewardPerTokenStored;
        }
        _;
    }
```

```
    function lastTimeRewardApplicable() public view returns (uint256) {
        return Math.min(block.timestamp, periodFinish);
    }

    function rewardPerToken() public view returns (uint256) {
        if (totalSupply() == 0) {
            return rewardPerTokenStored;
        }
        return
            rewardPerTokenStored.add(
                lastTimeRewardApplicable()
                    .sub(lastUpdateTime)
                    .mul(rewardRate)
                    .mul(1e18)
                    .div(totalSupply())
            );
    }

    function earned(address account) public view returns (uint256) {
        return
            balanceOf(account)
                .mul(rewardPerToken().sub(userRewardPerTokenPaid[account]))
                .div(1e18)
                .add(rewards[account]);
    }

    // stake visibility is public as overriding LPTokenWrapper's stake() function
    function stake(uint256 amount) public updateReward(msg.sender) checkStart{
        require(amount > 0, "Cannot stake 0");
        super.stake(amount);
        emit Staked(msg.sender, amount);
    }

    function withdraw(uint256 amount) public updateReward(msg.sender) checkStart{
        require(amount > 0, "Cannot withdraw 0");
        super.withdraw(amount);
        emit Withdrawn(msg.sender, amount);
    }

    function exit() external {
        withdraw(balanceOf(msg.sender));
        getReward();
    }

    function getReward() public updateReward(msg.sender) checkStart{
        uint256 reward = earned(msg.sender);
        if (reward > 0) {
            rewards[msg.sender] = 0;
            jgn.safeTransfer(msg.sender, reward);
            emit RewardPaid(msg.sender, reward);
        }
    }

    modifier checkStart(){
        require(block.timestamp > starttime,"not start");
        _;
    }

    function notifyRewardAmount(uint256 reward)
        external
        onlyRewardDistribution
        updateReward(address(0))
    {
        if (block.timestamp >= periodFinish) {
            rewardRate = reward.div(DURATION);
        } else {
            uint256 remaining = periodFinish.sub(block.timestamp);
            uint256 leftover = remaining.mul(rewardRate);
            rewardRate = reward.add(leftover).div(DURATION);
        }
        lastUpdateTime = block.timestamp;
        periodFinish = block.timestamp.add(DURATION);
        emit RewardAdded(reward);
    }
}
```

5. Appendix B — Vulnerability Severity Classification

Smart Contract Vulnerability Severity Classification	
Severity	Description
HIGH	The loopholes that can directly cause the loss of token contracts or users' funds, such as the integer overflow/underflow vulnerability that can cause token value return to zero, the fake deposit vulnerability that can cause token loss by the exchange, and reentrancy that can cause contract account to lose ETH or other tokens; The vulnerabilities that can cause the loss of ownership of token contracts, such as the access control defects of key functions, the bypass of key function access control caused by call injection, etc.; Vulnerabilities that can cause token contracts to fail to work properly, such as the denial of service vulnerabilities caused by sending ETH to malicious addresses and denial of service vulnerabilities caused by gas exhaustion.
MEDIUM	High-risk vulnerabilities that require a specific address to trigger, such as integer overflow/underflow vulnerability that only token contract owners can trigger; In addition, access control defects of non-key functions, business logics defects that do not cause direct financial losses, etc.

LOW	The vulnerability that is difficult to trigger and with little damage after triggering, such as the integer overflow/underflow vulnerability that needs a large number of ETH or other tokens to trigger, the vulnerability that the attacker can't make a direct profit after triggering the integer overflow/ underflow, and the transaction-ordering dependence started by high gas.

6. Appendix C — Methodology

6.1 Manticore

Manticore is a symbol execution tool for analyzing binary files and smart contracts. Manticore includes a symbolic Ethereum virtual machine (EVM), an EVM disassembler/assembler, and a convenient interface for automatically compiling and analyzing solidness. It also integrates Ethersplay, a bit of trait of bits visual disassembler for EVM bytecode, which is used 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. Oyente can detect common bugs in smart contracts, such as reentrancy, transaction-ordering dependence, and so on. More conveniently, Oyente's design is modular, allowing advanced users to implement and insert their detection logic to check for custom properties in their contracts.

6.3 Securify.sh

Securify can verify common security problems of Ethereum smart contracts, such as transaction disorder and lack of input validation. It analyzes all possible execution paths of the program while fully automated. Besides, Securify also has

a specific language for specific vulnerabilities, which enables Securify to pay attention to security and other reliability issues at any time.

6.4 Echidna

Echidna is a Haskell library designed for fuzzing of EVM codes.

6.5 MAIAN

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

6.6 Ethersplay

Etherplay is an EVM anti assembler, which contains relevant analysis tools.

6.7 Ida-Evm

Ida-Evm is an IDA processor module for 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 the Solidity language.

6.9 Toolkit developed by Knownsec’s penetration test team

Knownsec's unique toolkit for penetration testers is developed, collected and used by Knownsec penetration testing engineers, including batch automatic testing tools for testers, self-developed tools, scripts or utilization tools, etc.

Beijing Knownsec Information Tech. CO., LTD.

Telephone number	+86(10)400 060 9587
E-mail	sec@knownsec.com
Official website	www.knowsec.com
Address	2509, block T2-B, Wangjing SOHO, Chaoyang District, Beijing