KNOWNSEC
Blockchain Lab

Smart Contract Audit Report

Version description

The revision	Date	Revised	Version
Write documentation	20220110	KNOWNSEC Blockchain Lab	V1.1

Document information

Title	Version	Document Number	Type
Aboard Smart Contract Audit Report	V1.1	fe165f9a430d47d9a0a02024a45d8c17	Open to project team

Statement

KNOWNSEC Blockchain Lab only issues this report for facts that have occurred or existed before the issuance of this report, and assumes corresponding responsibilities for this. KNOWNSEC Blockchain Lab is unable to determine the security status of its smart contracts and is not responsible for the facts that will occur or exist in the future. The security audit analysis and other content made in this report are only based on the documents and information provided to us by the information provider as of the time this report is issued. KNOWNSEC Blockchain Lab 's assumption: There is no missing, tampered, deleted or concealed information. If the information provided is missing, tampered with, deleted, concealed or reflected in the actual situation, KNOWNSEC Blockchain Lab shall not be liable for any losses and adverse effects caused thereby.

Directory

1. Summarize

The effective test time of this report is **from December 13, 2021 to January 10, 2022**. During this period, the security and standardization of **the P1Admin, P1Margin, and P1Trade derivatives transaction codes of the Aboard smart contract** will be audited. This serves as the statistical basis for the report.

The scope of this smart contract security audit does not include external contract calls, new attack methods that may appear in the future, and code after contract upgrades or tampering. (With the development of the project, the smart contract may add a new pool , New functional modules, new external contract calls, etc.), does not include front-end security and server security.

In this audit report, engineers conducted a comprehensive analysis of the common vulnerabilities of smart contracts (Chapter 6). **The smart contract code of the Aboard** is comprehensively assessed as **PASS**.

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

KNOWNSEC Attest information:

classification	information
report number	fe165f9a430d47d9a0a02024a45d8c17
report query link	https://attest.im/attestation/searchResult?qurey=fe165f9a430d47d9a0a02024a45d8c17

2. Item information

2.1. Item description

Aboard is a Ethereum-based protocol designed to conduct decentralized derivatives trading.

2.2. The project's website

https：//aboard.exchange

2.3. White Paper

https：//aboard.exchange

2.4. Review version code

BSC Mainnet

PerpetualProxy：

0x7a08b29a7ad4a19a5eca0c82f5f082872488d135

PerpetualV1

0x055a53eddfa640d98a91345261ed3eafce5dc5ed

P1Orders：

0xc7871bc802a118f37367a54ccd18c7d5a531820e

P1Liquidation：

0xfab3479737122560670 7bc9328a2efc1ff2eb1b8

P1Deleveraging：

0xd7167a4de097c52a608755530af1d36b0d5fc45d

P1ChainlinkOracle：

0xea89f0b58079cdc323b70083535e29a28766fa5b

Abitrum Mainnet

PerpetualProxy：

0x7a08b29a7ad4a19a5eca0c82f5f082872488d135

PerpetualV1：

0x055a53eddfa640d98a91345261ed3eafce5dc5ed

P1Orders：

0xc7871bc802a118f37367a54ccd18c7d5a531820e

P1Liquidation：

0xfab347973712256067 07bc9328a2efc1ff2eb1b8

P1Deleveraging：

0xd7167a4de097c52a608755530af1d36b0d5fc45d

P1ChainlinkOracle：

0xea89f0b58079cdc323b70083535e29a28766fa5b

2.5. Contract file and Hash/contract deployment address

The contract documents	MD5
P1Operator.sol	afde2517ed0b8013b4e60e808f8d9aed
P1Settlement.sol	03de52ec56895506086fdb309a657d63
P1Margin.sol	c7e5e74bd272ac85e8ef8ee1bde79dce
P1Storage.sol	ad5336cfa26ed5243db5a7e33cd768fe
P1Trade.sol	d6c840cd505660aac2df4128d1d23030

P1Getters.sol	76294f5d92ef25cd6f0a4012d9833619
P1Admin.sol	7fc88744afe28619047480b306974369
PerpetualV1.sol	d74765fb67aa7d48370c37d91be1d2cd
P1ChainlinkOracle.sol	31f0a07d5d843e821e75e43450678cf4
P1Orders.sol	65918408c879bd3deef82f264c1d2e75
P1TraderConstants.sol	f9e81b318560ab6cacfab6eeeb792c55
P1Deleveraging.sol	4159836a693da419419cf3738a6b495c
P1Liquidation.sol	52c6827cb0535363ca490ad1cad47519
I_P1Funder.sol	d88140cbad01bedde7d53630d52b63d4
I_PerpetualV1.sol	9b27b6d6d5643457ccb2cac155229c78
I_P1Oracle.sol	37e6aa4942aa6410e542eb4502116725
I_P1Trader.sol	c9dc1a8bcb5ca35d3d3136e8d6828882
P1Types.sol	b654e9604760f8807893885b65d148b5
PerpetualProxy.sol	ea4eda37b6c9f96efbfd1450e53e6764
ReentrancyGuard.sol	561cdd5ac3ecc9dfeeeff11dc1ae04ce
BaseMath.sol	03853be972c77f6e236df02748d6c4c9
SignedMath.sol	5f891ae34c34c12dfd909d8ae8b376f2
Storage.sol	46a8c64e6d0c44094dd467d995ac2092
Math.sol	5b1df84dfb8f90c652c1068f66ac990d
SafeCast.sol	00154267400692a2436f430158cbc9b0

Adminable.sol	60e57604dd46c4385562cafc8b594a8d
AggregatorV3Interface.sol	7001c1ed44d5e1df43fce4f76f64b496

3. External visibility analysis

3.1. P1Admin contracts

P1Admin					
funcName	**visibility**	**state changes**	**decorator**	**payable reception**	**instructions**
setGlobalOperator	external	True	onlyAdmin, nonReentrant	---	---
setToken	external	True	onlyAdmin, nonReentrant	---	---
setOracle	external	True	onlyAdmin, nonReentrant	---	---
setTokenSymbolInitial	external	True	onlyAdmin, nonReentrant	---	---

3.2. P1Getters contracts

P1Getters					
funcName	**visibility**	**state changes**	**decorator**	**payable reception**	**instructions**
getAccountPosition	external	False	---	---	---
getAccountQvalue	external	False	---	---	---
getIsLocalOperator	external	False	---	---	---
getIsGlobalOpera	external	False	---	---	---

tor					
getTokenContract	external	False	---	---	---
getOracleContract	external	False	---	---	---
getOraclePrice	external	False	---	---	---
hasAccountPermissions	public	False	---	---	---

3.3. P1Margin contracts

P1Margin					
funcName	**visibility**	**state changes**	**decorator**	**payable reception**	**instructions**
setGlobalOperator	external	True	onlyAdmin, nonReentrant	---	---
setToken	external	True	onlyAdmin, nonReentrant	---	---
setOracle	external	True	onlyAdmin, nonReentrant	---	---
setTokenSymbolInitial	external	True	onlyAdmin, nonReentrant	---	---

3.4. P1Operator contracts

P1Operator					
funcName	**visibility**	**state changes**	**decorator**	**payable reception**	**instructions**
setLocalOperator	external	True	---	---	---

3.5. P1Settlement contracts

P1Settlement					
funcName	**visibility**	**state changes**	**decorator**	**payable reception**	**instructions**
getPosition	internal	False	---	---	---
setPosition	internal	True	---	---	---
addToPosition	internal	True	---	---	---
subFromPosition	internal	True	---	---	---
getMargin	internal	False	---	---	---
setMargin	internal	True	---	---	---
addToMargin	internal	True	---	---	---
subFromMargin	internal	True	---	---	---
toBytes32	internal	False	---	---	---
toBytes32_deposit_withdraw	internal	False	---	---	---
toBytes32_fee	internal	False	---	---	---
toBytes32_funding	internal	False	---	---	---

3.6. P1Trade contracts

P1Trade					
funcName	**visibility**	**state changes**	**decorator**	**payable reception**	**instructions**
trade	public	True	nonReentrant	---	---
_isOrder	private	False	---	---	---

_verifyAccounts	private	False	---	---	---
margin_position	private	True	---	---	---

4. Code vulnerability analysis

4.1. Summary description of the audit results

Audit results			
audit project	**audit content**	**condition**	**description**
Business security detection	Transaction clearing function	Pass	After testing, there is no security issue.
	Authority management function	Pass	After testing, there is no security issue.
	Get information function	Pass	After testing, there is no security issue.
	Deposit and withdrawal function	Pass	After testing, there is no security issue.
	Fund settlement function	Pass	After testing, there is no security issue.
	Transaction function	Pass	After testing, there is no security issue.
Code basic vulnerability detection	Compiler version security	Pass	After testing, there is no security issue.
	Redundant code	Pass	After testing, there is no security issue.
	Use of safe arithmetic library	Pass	After testing, there is no security issue.
	Not recommended encoding	Pass	After testing, there is no security issue.
	Reasonable use of require/assert	Pass	After testing, there is no security issue.
	fallback function safety	Pass	After testing, there is no security issue.
	tx.origin authentication	Pass	After testing, there is no security issue.

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

	Fake recharge vulnerability detection	Pass	After testing, there is no security issue.
	Reentry attack detection	Pass	After testing, there is no security issue.
	Replay attack detection	Pass	After testing, there is no security issue.
	Rearrangement attack detection	Pass	After testing, there is no security issue.

5. Business security detection

5.1. Transaction clearing function【Pass】

Audit analysis: The trade function of the P1Liquidation.sol contract is used to realize the settlement of transactions between accounts. The function permissions are correct, and no obvious security problems have been found.

```
function trade(
        address maker,
        address taker,
        bytes calldata data
    )
        external
        returns (P1Types.TradeResult memory)
    {
        address perpetual = _PERPETUAL_V1_;

        require(
            msg.sender == perpetual,
            "msg.sender must be PerpetualV1"
        ); //knownsec // The account must be a perpetual account

        TradeData memory tradeData = abi.decode(data, (TradeData));

        bool taker_isBuy = !tradeData.liquidatee_is_buy;
        emit LogLiquidated(
            maker,
            taker,
            tradeData.amount,
            taker_isBuy,
            tradeData.price_liq
        );
```

```
        uint256 margin_change = tradeData.amount.baseMul(tradeData.price_liq); //knownsec // Margin changes
        return P1Types.TradeResult({
            fee_maker: tradeData.fee_liquidatee,
            fee_taker: tradeData.fee_liquidator,
            funding_maker: tradeData.funding_liquidatee,
            funding_taker: tradeData.funding_liquidator,
            positionAmount: tradeData.amount,
            margin_change: margin_change,
            is_neg_fee: tradeData.neg_fee_liquidator,
            isBuy: taker_isBuy,
            traderFlags: TRADER_FLAG_LIQUIDATION
        });
    }
```

Security advice: None.

5.2. Authority management function 【Pass】

Audit analysis: Perform a security audit on the logic of the authority management function in the P1Admin.sol contract. Its purpose is to set related high authority functions, check whether the parameters are legally verified, the administrator adds or deletes the global operator address, and sets a new one Whether there are design flaws in the logical design of token contracts, setting up new oracle contracts, and whether there are reentry attacks, etc. The method use authority is: external administrator authority, which is a normal business requirement.

```
function setGlobalOperator(
        address operator,
        bool approved
```

```
    )
        external
        onlyAdmin
        nonReentrant
    { //knownsec// Add or delete global operator address
        _GLOBAL_OPERATORS_[operator] = approved; //knownsec// True if approved, false if not approved
        emit LogSetGlobalOperator(operator, approved);
    }

    /**
     * @notice Sets a new token contract.
     * @dev Must be called by the PerpetualV1 admin. Emits the LogSetToken event.
     *
     * @param  token_address  The address of the token smart contract.
     */
    function setToken(
        address token_address
    )
        external
        onlyAdmin
        nonReentrant
    { //knownsec// Set up a new token contract
        IERC20(token_address).totalSupply();
        _TOKEN_ = token_address; //knownsec// The address of the token smart contract
        emit LogSetToken(token_address);
    }

    /**
     * @notice Sets a new price oracle contract.
     * @dev Must be called by the PerpetualV1 admin. Emits the LogSetOracle event.
     *
     * @param  oracle  The address of the new price oracle contract.
```

```
    */
    function setOracle(
        address oracle
    )
        external
        onlyAdmin
        nonReentrant
    { //knownsec// Set up a new price oracle contract
        uint32 numTokens = uint32(_TOKEN_SYMBOL_.length);
        for (uint32 i = 0; i < numTokens; i++) {
            string memory token = _TOKEN_SYMBOL_[i];
            require(
                I_P1Oracle(oracle).getPrice(token) != 0, //knownsec// New oracle machine
cannot return 0 price
                "New oracle cannot return a zero price"
            );
        }
        _ORACLE_ = oracle; //knownsec// Set the oracle address
        emit LogSetOracle(oracle);
    }

    /**
     * @notice Initialize symbols array for adding new symbols.
     * @dev Must be called by the PerpetualV1 admin. Emits the LogSetTokenSymbolInitial
event.
     *
     * @param  symbol_array  array of trading tokens pair names for short.
     */
    function setTokenSymbolInitial(
        string[] calldata symbol_array
    )
        external
        onlyAdmin
```

```
        nonReentrant
    { //knownsec// Initialize the symbol array to add a new symbol
        _TOKEN_SYMBOL_ = new string[](symbol_array.length);
        for (uint256 i = 0; i < symbol_array.length; i++) {
            _TOKEN_SYMBOL_[i] = symbol_array[i]; //knownsec// Set the new symbol of the trading pair
        }
        emit LogSetTokenSymbolInitial(_TOKEN_SYMBOL_);
    }
```

Security advice: None.

5.3. Get information function 【Pass】

Audit analysis: Perform a security audit on the function logic for obtaining information in the P1Getters.sol contract. Its purpose is to obtain relevant useful information, check whether the parameters are legally verified, obtain the Q value of the account position, and specify the account operation status and authority status. Whether there are design flaws in the logical design of tokens and the contract address of the oracle, whether there are reentry attacks, etc. The method use permission is: external owner, which belongs to the normal business requirements.

```
function getAccountPosition(
        address account,
        string calldata symbol
    )
        external
        view
        returns (P1Types.PositionStruct memory)
    { //knownsec// Get the position of an account, regardless of index changes
        return _BALANCES_[account].tokenPosition[symbol]; //knownsec// The balance of the
```

```
specified account transaction pair
    }

    /**
     * @notice Get the Q value of an account, without accounting for changes in the index.
     *
     * @param  account   The address of the account to query the Qvalue of.
     * @return           The Qvalue of the account.
     */
    function getAccountQvalue(
        address account
    )
        external
        view
        returns (P1Types.MarginStruct memory)
    { //knownsec// Get the Q value of an account, regardless of index changes
        return P1Types.MarginStruct({
            marginIsPositive: _BALANCES_[account].marginIsPositive, //knownsec// Whether the margin is positive
            margin: _BALANCES_[account].margin //knownsec// Return the specified account margin
        });
    }

    /**
     * @notice Gets the local operator status of an operator for a particular account.
     *
     * @param  account    The account to query the operator for.
     * @param  operator   The address of the operator to query the status of.
     * @return            True if the operator is a local operator of the account, false otherwise.
     */
    function getIsLocalOperator(
```

```
        address account,
        address operator
    )
        external
        view
        returns (bool)
    { //knownsec// Get the local operator status of the operator of a specific account
        return _LOCAL_OPERATORS_[account][operator]; //knownsec// True if the operator is the local operator of the account, otherwise false
    }

    // ============ Global Getters ============

    /**
     * @notice Gets the global operator status of an address.
     *
     * @param  operator  The address of the operator to query the status of.
     * @return           True if the address is a global operator, false otherwise.
     */
    function getIsGlobalOperator(
        address operator
    )
        external
        view
        returns (bool)
    { //knownsec// Get the global operator status of the address
        return _GLOBAL_OPERATORS_[operator]; //knownsec// If the address is a global operator, it is true, otherwise it is false
    }

    /**
     * @notice Gets the address of the ERC20 margin contract used for margin deposits.
     *
```

```
  * @return The address of the ERC20 token.
  */
function getTokenContract()
    external
    view
    returns (address)
{ //knownsec// Get the ERC20 margin contract address for margin deposit
    return _TOKEN_; //knownsec// ERC20 token address
}

/**
  * @notice Gets the current address of the price oracle contract.
  *
  * @return The address of the price oracle contract.
  */
function getOracleContract()
    external
    view
    returns (address)
{ //knownsec// Get the current address of the price oracle contract
    return _ORACLE_; //knownsec// Price oracle contract address
}

/**
  * @notice Gets the symbols array.
  *
  * @return Array of trading tokens pair names for short.
  */
function getSymbolArray()
    external
    view
    returns (string [] memory)
{ //knownsec// Get the symbol array
```

```
        return _TOKEN_SYMBOL_; //knownsec// Trading pair symbol
    }

    // ============ Authorized External Getters ============

    /**
     * @notice Gets the price returned by the oracle.
     * @dev Only able to be called by global operators.
     *
     * @param  symbol    Trading tokens pair name for short.
     * @return           The price returned by the current price oracle.
     */
    function getOraclePrice(
        string calldata symbol
    )
        external
        view
        returns (uint256)
    { //knownsec// Get the price returned by oracle
        require(
            _GLOBAL_OPERATORS_[msg.sender], //knownsec// Can only be called by global operators
            "Oracle price requester not global operator"
        );
        return I_P1Oracle(_ORACLE_).getPrice(symbol); //knownsec// The price returned by the current price oracle
    }

    // ============ Public Getters ============

    /**
     * @notice Gets whether an address has permissions to operate an account.
     *
```

```
     * @param    account     The account to query.
     * @param    operator    The address to query.
     * @return               True if the operator has permission to operate the account,
     *                        and false otherwise.
     */
    function hasAccountPermissions(
        address account,
        address operator
    )
        public
        view
        returns (bool)
    { //knownsec// Whether to obtain the address has the authority to operate the account
        return account == operator
            || _GLOBAL_OPERATORS_[operator]
            || _LOCAL_OPERATORS_[account][operator]; //knownsec// True if the operator
has the authority to operate the account, otherwise false
    }
```

Security advice: None.

5.4. Deposit and withdrawal function 【Pass】

Audit analysis: Perform security audit on the deposit and withdrawal function logic in the P1Margin.sol contract. Its purpose is deposit and withdrawal tokens. Check whether the parameters are legally verified, and whether there are design flaws in the logic design of deposits and withdrawals. Re-entry attacks, etc. The method use permission is: external, which is a normal business requirement.

```
function deposit(
        address account,
        uint256 amount
```

```
    )
        external
        nonReentrant
    {
        SafeERC20.safeTransferFrom(
            IERC20(_TOKEN_),
            msg.sender,
            address(this),
            amount
        );

        addToMargin(account, amount);
        //addToMargin(account, amount.mul(DECIMAL_ADJ));    //Ethernet 2/3

        emit LogDeposit(
            account,
            amount,
            toBytes32_deposit_withdraw(account, SignedMath.Int({value:0, isPositive:false}))
        );
    }

    /**
     * @notice Withdraw apply. some amount of margin tokens from an account to a destination
address.
     * @dev Emits LogAccWithdrawRquest events.
     *
     * @param    account         The account for which to debit the withdrawal.
     * @param    destination     The address to which the tokens are transferred.
     * @param    amount          The amount of tokens to withdraw.
     */
    function withdraw_apply(
        address account,
        address destination,
```

```
        uint256 amount
    )
        external
        nonReentrant
    {
        require(
            hasAccountPermissions(account, msg.sender), //knownsec// Is there permission
            "withdraw_apply sender does not have permission to withdraw"
        );

        uint256 ts = block.timestamp;
        bytes32 applyhash = _getApplyHash(account, destination, amount, ts);

        require(
            !_WD_APPLY_[applyhash], //knownsec// No withdrawal request has been made
            "withdraw_apply same apply at the same time"
        );

        _WD_APPLY_[applyhash] = true;

        emit LogAccWithdrawRquest(account, destination, amount, ts);
    }

    /**
     * @notice Withdraw some amount of margin tokens from an account to a destination address.
     * @dev Only able to be called by gateway. Emits LogWithdraw event.
     *
     * @param   account         The account for which to debit the withdrawal.
     * @param   funding         The funding of the account
     * @param   destination   The address to which the tokens are transferred.
     * @param   amount          The amount of tokens to withdraw.
     * @param   timestamp       The timestamp of the withdraw apply
```

```
     * @param   r              signature r
     * @param   s              signature s
     * @param   v              signature v
     */
    function withdraw(
        address account,
        SignedMath.Int calldata funding,
        address destination,
        uint256 amount,
        uint256 timestamp,
        bytes32 r,
        bytes32 s,
        uint8 v
    )
        external
        nonReentrant
    {
        require(
            msg.sender == _GATEWAY_, //knownsec// Only the network management address can be called
            "Withdraw msg sender is not gateway"
        );

        //check hash
        bytes32 applyhash = _getApplyHash(account, destination, amount, timestamp);
        require(
            _WD_APPLY_[applyhash],
            "Withdraw hash mismatch"
        ); //knownsec// The hash can be withdrawn

        //check signature
        require(
            _SIGNER_ == ecrecover(keccak256(abi.encodePacked("\x19Ethereum Signed
```

```
Message:\n32", applyhash)), v, r, s),
                "Withdraw invalid signature"
            );

            _WD_APPLY_[applyhash] = false;

            SafeERC20.safeTransfer(
                IERC20(_TOKEN_),
                destination,
                amount
            );

            //SignedMath.Int memory signedChange = funding.sub(amount.mul(DECIMAL_ADJ));
//Ethernet 3/3
            SignedMath.Int memory signedChange = funding.sub(amount);

            if (signedChange.isPositive) {
                addToMargin(account, signedChange.value);
            }   else {
                subFromMargin(account, signedChange.value);
            }

            emit LogWithdraw(
                account,
                destination,
                amount,
                toBytes32_deposit_withdraw(account, funding)
            );
        }
```

Security advice: None.

5.5. Fund settlement function 【Pass】

Audit analysis: Perform a security audit on the fund settlement function logic in the P1Settlement.sol contract. Its purpose is to settle fund payments between accounts, check whether the parameters are legally verified, and whether the settlement fund payment logic between accounts is designed There are design flaws, whether there are reentry attacks, etc. The method use permission is: internal, which is a normal business requirement.

```
function getPosition(
        address account,
        string memory token
    )
        internal
        view
        returns (SignedMath.Int memory)
    { //knownsec// Returns the position of the token balance of the specified account
        return SignedMath.Int({
            value: _BALANCES_[account].tokenPosition[token].position,
            isPositive: _BALANCES_[account].tokenPosition[token].positionIsPositive
        });
    }

    /**
     * @dev In-place modify the signed position value of a balance.
     */
    function setPosition(
        address account,
        SignedMath.Int memory newPosition,
        string memory token
    )
```

```
        internal
    { //knownsec// Modify the new position of the specified token balance of the specified account
        _BALANCES_[account].tokenPosition[token].position =
newPosition.value.toUint120();
        _BALANCES_[account].tokenPosition[token].positionIsPositive =
newPosition.isPositive;
    }

    /**
     * @dev In-place add amount to balance.position.
     */
    function addToPosition(
        address account,
        uint256 amount,
        string memory token
    )
        internal
    { //knownsec// Increase position
        SignedMath.Int memory signedPosition = getPosition(account, token);
        signedPosition = signedPosition.add(amount);
        setPosition(account, signedPosition, token);
    }


    /**
     * @dev In-place subtract amount from balance.position.
     */
    function subFromPosition(
        address account,
        uint256 amount,
        string memory token
    )
        internal
```

```
{ //knownsec// Decrease positions
        SignedMath.Int memory signedPosition = getPosition(account, token);
        signedPosition = signedPosition.sub(amount);
        setPosition(account, signedPosition, token);
}

/**
  * @dev Returns a SignedMath.Int version of the margin in balance.
  */
function getMargin(
        address account
)
        internal
        view
        returns (SignedMath.Int memory)
{ //knownsec// Returns the margin balance of a SignedMath.Int version
        return SignedMath.Int({
                value: _BALANCES_[account].margin,
                isPositive: _BALANCES_[account].marginIsPositive
        });
}

/**
  * @dev In-place modify the signed margin value of a balance.
  */
function setMargin(
        address account,
        SignedMath.Int memory newMargin
)
        internal
{ //knownsec// Modify account balance position
        _BALANCES_[account].margin = newMargin.value.toUint120();
        _BALANCES_[account].marginIsPositive = newMargin.isPositive;
```

```
}


/**
 * @dev In-place add amount to balance.margin.
 */
function addToMargin(
    address account,
    uint256 amount
)
    internal
{ //knownsec// Add the amount to balance.margin
    SignedMath.Int memory signedMargin = getMargin(account);
    signedMargin = signedMargin.add(amount);
    setMargin(account, signedMargin);
}


/**
 * @dev In-place subtract amount from balance.margin.
 */
function subFromMargin(
    address account,
    uint256 amount
)
    internal
{ //knownsec// Subtract the amount from balance.margin
    SignedMath.Int memory signedMargin = getMargin(account);
    signedMargin = signedMargin.sub(amount);
    setMargin(account, signedMargin);
}
```

Security advice: None.

5.6. Transaction function【Pass】

Audit analysis: Perform a security audit on the logic of the withdrawal function (withdraw) in the P1Trade.sol contract. The extraction purpose is (withdrawing a specified share of the local currency). The amountMin and deadline parameters, as well as the price slippage and price impact limits, are added. Check whether there is a right The parameters are checked for legitimacy, and whether there are design flaws in the logic design for the withdrawal of the designated share of the local currency, and whether there is a reentry attack, etc. The method use permission is: external, which is a normal business requirement.

```
function trade(
        address[] memory accounts,
        TradeArg[] memory trades
    )
        public
        nonReentrant
    { //knownsec// Submit one or more transactions between any number of accounts
        require(
            _GLOBAL_OPERATORS_[msg.sender],
            "function trade: msg.sender is not global operator"
        );
        _verifyAccounts(accounts); //knownsec// Verify account

        for (uint256 i = 0; i < trades.length; i++) {
            TradeArg memory tradeArg = trades[i];

            require(
                _GLOBAL_OPERATORS_[tradeArg.trader],
                "trader is not global operator"
```

```
        );

        address maker = accounts[tradeArg.makerIndex];
        address taker = accounts[tradeArg.takerIndex];

        P1Types.TradeResult memory tradeResult = I_P1Trader(tradeArg.trader).trade(
            maker,
            taker,
            tradeArg.data
        );

        (
            bool maker_is_neg_fee,
            bool taker_is_neg_fee
        ) = margin_position(maker, taker, tradeResult, tradeArg.symbol);

        emit LogTrade(
            maker,
            taker,
            tradeArg.trader,
            tradeArg.symbol,
            toBytes32(maker, tradeArg.symbol),
            toBytes32(taker, tradeArg.symbol),
            toBytes32_funding(tradeResult.funding_maker),
            toBytes32_funding(tradeResult.funding_taker),
            toBytes32_fee(tradeResult.fee_maker, maker_is_neg_fee),
            toBytes32_fee(tradeResult.fee_taker, taker_is_neg_fee),
            tradeResult.margin_change,
            tradeResult.positionAmount,
            tradeResult.isBuy
        );
    }
```

```
}
```

Security advice: None.

6. Code basic vulnerability detection

6.1. Compiler version security 【Pass】

Check to see if a secure compiler version is used in the contract code implementation.

Detection results: After detection, the smart contract code has developed a compiler version of 0.8.9 or more, there is no security issue.

Security advice: None.

6.2. Redundant code 【Pass】

Check that the contract code implementation contains redundant code.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.3. Use of safe arithmetic library 【Pass】

Check to see if the SafeMath security abacus library is used in the contract code implementation.

Detection results: The SafeMath security abacus library has been detected in the smart contract code and there is no such security issue.

Security advice: None.

6.4. Not recommended encoding【Pass】

Check the contract code implementation for officially uns recommended or deprecated coding methods.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.5. Reasonable use of require/assert【Pass】

Check the reasonableness of the use of require and assert statements in contract code implementations.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.6. Fallback function safety【Pass】

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

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.7. tx.origin authentication 【Pass】

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

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.8. Owner permission control 【Pass】

Check that theowner in the contract code implementation has excessive permissions. For example, modify other account balances at will, and so on.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.9. Gas consumption detection 【Pass】

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

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.10. call injection attack 【Pass】

When a call function is called, strict permission control should be exercised, or the function called by call calls should be written directly to call calls.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.11. Low-level function safety 【Pass】

Check the contract code implementation for security vulnerabilities in the use of call/delegatecall

The execution context of the call function is in the contract being called, while the execution context of the delegatecall function is in the contract in which the function is currently called.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.12. Vulnerability of additional token issuance 【Pass】

Check to see if there are functions in the token contract that might increase the total token volume after the token total is initialized.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.13. Access control defect detection 【Pass】

Different functions in the contract should set reasonable permissions, check whether the functions in the contract correctly use pubic, private and other keywords for visibility modification, check whether the contract is properly defined and use modifier access restrictions on key functions, to avoid problems caused by overstepping the authority.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.14. Numerical overflow detection 【Pass】

The arithmetic problem in smart contracts is the integer overflow and integer overflow, with Solidity able to handle up to 256 digits (2^256-1), and a maximum number increase of 1 will overflow to get 0. Similarly, when the number is an unsigned type, 0 minus 1 overflows to get the maximum numeric value.

Integer overflows and underflows are not a new type of vulnerability, but they are particularly dangerous in smart contracts. Overflow conditions can lead to incorrect results, especially if the likelihood is not anticipated, which can affect the reliability and safety of the program.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.15. Arithmetic accuracy error 【Pass】

Solidity has a data structure design similar to that of a normal programming language, such as variables, constants, arrays, functions, structures, and so on, and there is a big difference between Solidity and a normal programming language - Solidity does not have floating-point patterns, and all of Solidity's numerical operations result in integers, without the occurrence of decimals, and without allowing the definition of decimal type data. Numerical operations in contracts are essential, and numerical operations are designed to cause relative errors, such as sibling operations: 5/2 x 10 x 20, and 5 x 10/2 x 25, resulting in errors, which can be greater and more obvious when the data is larger.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.16. Incorrect use of random numbers 【Pass】

Random numbers may be required in smart contracts, and while the functions and variables provided by Solidity can access significantly unpredictable values, such as block.number and block.timestamp, they are usually either more public than they seem, or are influenced by miners, i.e. these random numbers are somewhat predictable, so malicious users can often copy it and rely on its unpredictability to attack the feature.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.17. Unsafe interface usage 【Pass】

Check the contract code implementation for unsafe external interfaces, which can be controlled, which can cause the execution environment to be switched and control contract execution arbitrary code.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.18. Variable coverage 【Pass】

Check the contract code implementation for security issues caused by variable overrides.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.19. Uninitialized storage pointer 【Pass】

A special data structure is allowed in solidity as a strut structure, while local variables within the function are stored by default using stage or memory.

The existence of store (memory) and memory (memory) is two different concepts, solidity allows pointers to point to an uninitialized reference, while uninitialized local stage causes variables to point to other stored variables, resulting in variable overrides, and even more serious consequences, and should avoid initializing the task variable in the function during development.

Detection results: After detection, the smart contract code does not have the problem.

Security advice: None.

6.20. Return value call verification 【Pass】

This issue occurs mostly in smart contracts related to currency transfers, so it is also known as silent failed sending or unchecked sending.

In Solidity, there are transfer methods such as transfer(), send(), call.value(), which can be used to send tokens to an address, the difference being: transfer send failure will be throw, and state rollback; Call.value returns false when it fails to send, and passing all available gas calls (which can be restricted by incoming gas_value parameters) does not effectively prevent reentration attacks.

If the return values of the send and call.value transfer functions above are not checked in the code, the contract continues to execute the subsequent code, possibly with unexpected results due to token delivery failures.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.21. Transaction order dependency【Pass】

Because miners always get gas fees through code that represents an externally owned address (EOA), users can specify higher fees to trade faster. Since blockchain is public, everyone can see the contents 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 transactions at a higher cost to preempt the original solution.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.22. Timestamp dependency attack【Pass】

Block timestamps typically use miners' local time, which can fluctuate over a range of about 900 seconds, and when other nodes accept a new chunk, they only need to verify that the timestamp is later than the previous chunk and has a local time error of less than 900 seconds. A miner can profit from setting the timestamp of a block to meet as much of his condition as possible.

Check the contract code implementation for key timestamp-dependent features.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.23. Denial of service attack【Pass】

Smart contracts that are subject to this type of attack may never return to normal operation. There can be many reasons for smart contract denial of service, including malicious behavior as a transaction receiver, the exhaustion of gas caused by the artificial addition of the gas required for computing functionality, the misuse of access control to access the private component of smart contracts, the exploitation of confusion and negligence, and so on.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.24. Fake recharge vulnerability【Pass】

The transfer function of the token contract checks the balance of the transfer initiator (msg.sender) in the if way, when the balances < value enters the else logic part and return false, and ultimately does not throw an exception, we think that only if/else is a gentle way of judging in a sensitive function scenario such as transfer is a less rigorous way of coding.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.25. Reentry attack detection 【Pass】

The call.value() function in Solidity consumes all the gas it receives when it is used to send tokens, and there is a risk of re-entry attacks when the call to the call tokens occurs before the balance of the sender's account is actually reduced.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.26. Replay attack detection 【Pass】

If the requirements of delegate management are involved in the contract, attention should be paid to the non-reusability of validation to avoid replay attacks

In the asset management system, there are often cases of entrustment management, the principal will be the assets to the trustee management, the principal to pay a certain fee to the trustee. This business scenario is also common in smart contracts.

Detection results: The security issue is not present in the smart contract code after detection.

Security advice: None.

6.27. Rearrangement attack detection 【Pass】

A reflow attack is an attempt by a miner or other party to "compete" with a smart contract participant by inserting their information into a list or mapping, giving an attacker the opportunity to store their information in a contract.

Detection results: After detection, there are no related vulnerabilities in the smart contract code.

Security advice: None.

7. Appendix A: Security Assessment of Contract Fund Management

Contract fund management		
The type of asset in the contract	**The function is involved**	**Security risks**
User deposit token assets	deposit、withdraw	**SAFE**

Check the security of the management of **digital currency assets** transferred by users in the business logic of the contract. Observe whether there are security risks that may cause the loss of customer funds, such as **incorrect recording, incorrect transfer, and backdoor** withdrawal of the **digital currency assets** transferred into the contract.

KNOWNSEC
Blockchain Lab

Official Website
www.knownseclab.com

E-mail
blockchain@knownsec.com

WeChat Official Account