ExVul

SMART CONTRACT AUDIT REPORT

Artura Smart Contract

January 2025

Contents

1. EXECUTIVE SUMMARY

Exvul Web3 Security was engaged by **Artura** to review smart contract implementation. The assessment was conducted in accordance with our systematic approach to evaluate potential security issues based upon customer requirement. The report provides detailed recommendations to resolve the issue and provide additional suggestions or recommendations for improvement.

The outcome of the assessment outlined in chapter 3 provides the system's owners a full description of the vulnerabilities identified, the associated risk rating for each vulnerability, and detailed recommendations that will resolve the underlying technical issue.

1.1 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [10] which is the gold standard in risk assessment using the following risk models:

- Likelihood: represents how likely a particular vulnerability is to be uncovered and exploited in the wild.
- Impact: measures the technical loss and business damage of a successful attack.
- Severity: determine the overall criticality of the risk.

Likelihood can be: High, Medium and Low and impact are categorized into for: High, Medium, Low, Informational. Severity is determined by likelihood and impact and can be classified into five categories accordingly, Critical, High, Medium, Low, Informational shown in table 1.1.

Likelihood \ IMPACT	*Informational*	*Low*	*Medium*	*High*
High	INFO	MEDIUM	HIGH	CRITICAL
Medium	INFO	LOW	MEDIUM	HIGH
Low	INFO	LOW	LOW	MEDIUM

Table 1.1 Overall Risk Severity

To evaluate the risk, we will be going through a list of items, and each would be labelled with a severity category. The audit was performed with a systematic approach guided by a comprehensive assessment list carefully designed to identify known and impactful security issues. If our tool or analysis does not identify any issue, the contract can be considered safe regarding the assessed item. For any discovered issue, we might further deploy contracts on our private test environment and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.2.

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.
- Code and business security testing: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

Category	Assessment Item
Basic Coding Assessment	Apply Verification Control
	Authorization Access Control
	Forged Transfer Vulnerability
	Forged Transfer Notification
	Numeric Overflow
	Transaction Rollback Attack
	Transaction Block Stuffing Attack
	Soft Fail Attack
	Hard Fail Attack
	Abnormal Memo
	Abnormal Resource Consumption
	Secure Random Number
Advanced Source Code Scrutiny	Asset Security
	Cryptography Security
	Business Logic Review
	Source Code Functional Verification
	Account Authorization Control
	Sensitive Information Disclosure
	Circuit Breaker
	Blacklist Control
	System API Call Analysis
	Contract Deployment Consistency Check
	Abnormal Resource Consumption

Additional Recommendations	Semantic Consistency Checks
	Following Other Best Practices

Table 1.2: The Full List of Assessment Items

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [14], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development.

2. FINDINGS OVERVIEW

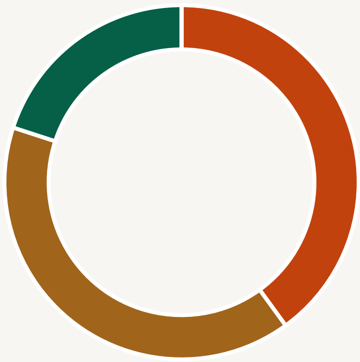
2.1 Project Info And Contract Address

Project Name	Audit Time	Language
Artura	2024.12.25 - 2025.1.10	solidity

Soure code	Link
Artura	https://github.com/bitperp/bitperp-contracts/tree/v7_update
Commit Hash	e02b6d8cfca76472fec24e2fc1fd1ea0e9fb1e7f

2.2 Summary

Severity	Found
CRITICAL	0
HIGH	2
MEDIUM	2
LOW	1
INFO	0



2.3 Key Findings

Severity	Findings Title	Status
HIGH	checkTp will revert when direction is sell	Fixed
HIGH	possible leaving uncleared pending order	Fixed
MEDIUM	non-legacy order should not have pending order limit	Fixed
MEDIUM	Excessive trade.leverage or pairOpenFeeP(pairIndex) values can cause openTradeMarketCallback to fail	Fixed
LOW	ReferralFee never charged	Fixed

Table 2.3: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 Missing Token Balance Validation

Location	Severity	Category
	HIGH	Business Logic

Description:

In the TradingStorageV7.sol contract, the checkTp function performs a validation on the takeProfit (TP) parameter when updating or opening an order. The function uses the value of maxGainP (default 900) to calculate the maxTpDist. This mechanism works fine for buy orders, but causes issues for sell orders. The maxTpDist is calculated as P * 9, leading to the following check for sell orders. When open order order updateTp , we have checkTp to check take profit value set. The issue is when order direction is selll and maxGainP default value is 900.Suppose leverage is 1 , the maxTpDist will be 9* price (tp < price && tp >= price - maxTpDist will revert.This will dos openOrder function when sell.

TradingCallbacksV7.sol | TradingStorageV7.sol | TradingV7.sol | PairInfosV7.sol

```
    function initialize(address _gov, address _dev, address _executor, TokenInterfaceV7 _token, TokenInterfaceV7 _
        public
        initializer
    {
        require(
            _gov != address(0) && _dev != address(0) && _executor != address(0)
            && address(_token) != address(0) && address(_dai) != address(0),
            "WRONG_ADDRESS"
        );
        gov = _gov;
        dev = _dev;
        executor = _executor;
        token = _token;
        dai = _dai;
        maxTradesPerPair = 3;
        maxTradesPerBlock = 5;
        maxPendingMarketOrders = 5;
        maxGainP = 900;
        maxSlP = 80;
        defaultLeverageUnlocked = 50;
    }

    // Modifiers
```

Recommendations:

Refactor checkTp logic to consider direction (buy/sell) separately.

Apply different maxGainP or maxTpDist logic for sell orders to allow valid TP range.

Consider symmetrizing TP checks for long/short positions or making the TP limit explicitly configurable for both directions.

Result	FixResult
Confirmed	Fixed

3.2 possible leaving uncleared pending order

Location	Severity	Category
	HIGH	Business Logic

Description:

Some user operations with oracle both have pending status.

For exmaple , legacy order trades open :

openTrade -> storePendingMarketOrder

oracle -> openTradeMarketCallback -> unregisterPendingMarketOrde

One invariant value is pendingOrder count. The issue is oracle retrun call back execution order is not executed sequentially. Suppose a situation, user position is unhealthy, at same time, executor execute liquidation, user execute closeTrade. This will create a PendingMarketOrder in storage. executeExecutorCloseOrderCallback will be executed first because it doesn't depends on oralce call back, due to executer's callback don't clear pending status and do unregister order, closeTradeMarketCallback will fail and permanently leave an unclosed pending order.When opening order, we have a pending order count check, and this will dos user open trade or close trade.

```
function closeTradeMarketCallback(AggregatorAnswer memory a) external onlyPriceAggregator notDone {
...

storageT.unregisterPendingMarketOrder(a.orderId, false);


}



function executeExecutorCloseOrderCallback(
...

}
```

Recommendations:

Ensure all code paths that finalize an order also perform unregisterPendingMarketOrder, including executor callbacks.

Add an additional safety check or force-clean mechanism in closeTradeMarketCallback to ignore already-cleared states gracefully.

Consider implementing an idempotent pending order clearing routine or reconciler to maintain storage invariants in asynchronous execution contexts.

Result	FixResult
Confirmed	Fixed

3.3 non-legacy order should not have pending order limit

Location	Severity	Category
	MEDIUM	Business Logic

Description:

When open trade, we have pending order limit. This check is both used in non-legacy order and pending order. Only pending order will storePendingMarketOrder , but non pending order still have this check.

```
function openTrade(
...
require(storageT.pendingOrderIdsCount(sender) < storageT.maxPendingMarketOrders(), "MAX_PENDING_ORDERS");

if (orderType != StorageInterfaceV7.OpenLimitOrderType.LEGACY) {
..

} else {
storageT.storePendingMarketOrder(
```

Recommendations:

Move the pendingOrderIdsCount check inside the LEGACY branch to ensure only actual pending orders are subject to the limit:

Result	FixResult
Confirmed	Fixed

3.4 Excessive leverage or pairOpenFeeP Can Cause openTradeMarketCallback to Fail

Location	Severity	Category
	MEDIUM	Input Validation

Description:

When the registerTrade method is called, the following logic calculates and deducts v.reward2 (i.e., trading fees):

```
v.levPosDai = trade.positionSizeDai * trade.leverage;
v.tokenPriceDai = aggregator.tokenPriceDai();

// 2. Charge opening fee - referral fee (if applicable)
v.reward2 = storageT.handleDevGovFees(trade.pairIndex, v.levPosDai, true, true);

trade.positionSizeDai -= v.reward2;
```

The trading fee is specifically calculated through the storageT.handleDevGovFees() function.

```
// Manage dev & gov fees
function handleDevGovFees(uint256 _pairIndex, uint256 _leveragedPositionSize, bool _dai, bool _fullFe
    external
    onlyTrading
    returns (uint256 fee)
{
    fee = _leveragedPositionSize * priceAggregator.openFeeP(_pairIndex) / PRECISION / 100;
    if (!_fullFee) fee /= 2;

    if (_dai) {
        govFeesDai += fee;
        devFeesDai += fee;
    } else {
        govFeesToken += fee;
        devFeesToken += fee;
    }

    fee *= 2;
}
```

Problem:If either trade.leverage or pairOpenFeeP(pairIndex) is excessively large, it will result in an abnormally high v.reward2 (fee). This can cause the operation trade.positionSizeDai -= v.reward2 to overflow, resulting in a failure of the registerTrade() method, which in turn causes the openTradeMarketCallback function to fail.

Current Constraints:

trade.leverage: Set by privileged roles, with a maximum limit of 1000.

pairOpenFeeP(pairIndex): Also set by privileged roles, but currently has no defined upper limit.

```
    modifier feeOk(Fee calldata _fee) {
        require(
            _fee.openFeeP > 0 && _fee.closeFeeP > 0 && _fee.referralFeeP > 0 && _fee.minLevPosDai > 0,
            "WRONG_FEES"
        );
        _;
    }
```

Example:

- trade.leverage = 1000

- trade.positionSizeDai = 10 DAI

- PRECISION = 1e10

- If priceAggregator.openFeeP(_pairIndex) exceeds 1,000,000, the operation trade.positionSizeDai -= v.reward2 will overflow, leading to an error and causing openTradeMarketCallback to fail.

Recommendations:

Enforce Upper Bound on pairOpenFeeP(pairIndex):

When setting the fee via governance/admin functions, ensure it is capped to a safe value that prevents v.reward2 from exceeding positionSizeDai.

Result	FixResult
Confirmed	Fixed

3.5 ReferralFee never charged

Location	Severity	Category
	LOW	Business Logic

Description:

The codebase defines an event ReferralFeeCharged(address indexed trader, uint256 valueDai); along with inline comments suggesting that referral fees should be collected during certain trading operations.

```
event ReferralFeeCharged(address indexed trader, uint256 valueDai);
```

```
    // Shared code between market & limit callbacks
    function registerTrade(StorageInterfaceV7.Trade memory trade)
        private
        returns (StorageInterfaceV7.Trade memory, uint256)
    {
        AggregatorInterfaceV7 aggregator = storageT.priceAggregator();
        PairsStorageInterfaceV7 pairsStored = aggregator.pairsStorage();

        Values memory v;

        v.levPosDai = trade.positionSizeDai * trade.leverage;
        v.tokenPriceDai = aggregator.tokenPriceDai();

//        event ReferralFeeCharged(address indexed trader, uint256 valueDai);
//      :qa we have  ReferralFeeCharged event,and comment, but never charged  ,event not used
        // 2. Charge opening fee - referral fee (if applicable)
        v.reward2 = storageT.handleDevGovFees( uint256: trade.pairIndex,  uint256: v.levPosDai,  bool: true,  bool: true);

```

Recommendations:

If referral fee logic is intended to be active, implement the appropriate deduction logic during trade execution (e.g., in registerTrade or fee handling modules) and emit the ReferralFeeCharged event.

Result	FixResult
Confirmed	Fixed

4. CONCLUSION

In this audit, we thoroughly analyzed **Artura** smart contract implementation. The problems found are described and explained in detail in Section 3. The problems found in the audit have been communicated to the project leader. We therefore consider the audit result to be **PASSED**. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

5. APPENDIX

5.1 Basic Coding Assessment

5.1.1 Apply Verification Control

Description	The security of apply verification
Result	Not found
Severity	CRITICAL

5.1.2 Authorization Access Control

Description	Permission checks for external integral functions
Result	Not found
Severity	CRITICAL

5.1.3 Forged Transfer Vulnerability

Description	Assess whether there is a forged transfer notification vulnerability in the contract
Result	Not found
Severity	CRITICAL

5.1.4 Transaction Rollback Attack

Description	Assess whether there is transaction rollback attack vulnerability in the contract
Result	Not found
Severity	CRITICAL

5.1.5 Transaction Block Stuffing Attack

Description	Assess whether there is transaction blocking attack vulnerability
Result	Not found
Severity	CRITICAL

5.1.6 Soft Fail Attack Assessment

Description	Assess whether there is soft fail attack vulnerability
Result	Not found
Severity	CRITICAL

5.1.7 Hard Fail Attack Assessment

Description	Examine for hard fail attack vulnerability
Result	Not found
Severity	CRITICAL

5.1.8 Abnormal Memo Assessment

Description	Assess whether there is abnormal memo vulnerability in the contract
Result	Not found
Severity	CRITICAL

5.1.9 Abnormal Resource Consumption

Description	Examine whether abnormal resource consumption in contract processing
Result	Not found
Severity	CRITICAL

5.1.10 Random Number Security

Description	Examine whether the code uses insecure random number
Result	Not found
Severity	CRITICAL

5.2 Advanced Code Scrutiny

5.2.1 Cryptography Security

Description	Examine for weakness in cryptograph implementation
Result	Not found
Severity	HIGH

5.2.2 Account Permission Control

Description	Examine permission control issue in the contract
Result	Not found
Severity	MEDIUM

5.2.3 Malicious Code Behavior

Description	Examine whether sensitive behavior present in the code
Result	Not found
Severity	MEDIUM

5.2.4 Sensitive Information Disclosure

Description	Examine whether sensitive information disclosure issue present in the code
Result	Not found
Severity	MEDIUM

5.2.5 System API

Description	Examine whether system API application issue present in the code
Result	Not found
Severity	LOW

6. DISCLAIMER

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without ExVul's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts ExVul to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. ExVul's position is that each company and individual are responsible for their own due diligence and continuous security. ExVul's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

7. REFERENCES

[1] MITRE. CWE- 191: Integer Underflow (Wrap or Wraparound).

https://cwe.mitre.org/data/ definitions/191.html.

[2] MITRE. CWE- 197: Numeric Truncation Error.

https://cwe.mitre.org/data/definitions/197. html.

[3] MITRE. CWE-400: Uncontrolled Resource Consumption.

https://cwe.mitre.org/data/ definitions/400.html.

[4] MITRE. CWE-440: Expected Behavior Violation.

https://cwe.mitre.org/data/definitions/440. html.

[5] MITRE. CWE-684: Protection Mechanism Failure.

https://cwe.mitre.org/data/definitions/ 693.html.

[6] MITRE. CWE CATEGORY: 7PK - Security Features.

https://cwe.mitre.org/data/definitions/ 254.html.

[7] MITRE. CWE CATEGORY: Behavioral Problems.

https://cwe.mitre.org/data/definitions/438. html.

[8] MITRE. CWE CATEGORY: Numeric Errors.

https://cwe.mitre.org/data/definitions/189.html.

[9] MITRE. CWE CATEGORY: Resource Management Errors.

https://cwe.mitre.org/data/ definitions/399.html.

[10] OWASP. Risk Rating Methodology.

https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology

Contact

Website

www.exvul.com

Email

contact@exvul.com

Twitter

@EXVULSEC

Github

github.com/EXVUL-Sec

ExVul