Electra Smart Contract

SMART CONTRACT AUDIT REPORT

January 2025

ExVul

www.exvul.com

Table of Contents

1. EXECUTIVE SUMMARY

Exvul Web3 Security was engaged by Electra 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	*Informational*	Medium	High	Critical
Medium	*Informational*	Low	Medium	High
Low	*Informational*	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
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: Electra

Audit Time: January 13, 2025 – January 20, 2025

Language: solidity

File Name	Link
Electra	https://github.com/ElectraFinance/electra-smart-contracts
Commit Hash	aea5d0fe51406c34afdd5d17253f74f4b92b92ac

2.2 Summary

Severity	Found	
Critical	0	
High	3	
Medium	1	
Low	2	
Informational	0	

2.3 Key Findings

ID	Severity	Findings Title	Status	Confirm
NVE-001	High	Unverified signatures lead to arbitrary liquidation	Fixed	Confirmed
NVE-002	High	Incorrect liquidation accumulation	Fixed	Confirmed
NVE-003	High	Possible bypass order expiration check	Fixed	Confirmed
NVE-004	Medium	Function selector collision	Fixed	Confirmed
NVE-005	Low	liquidate limit condition too simple	Fixed	Confirmed
NVE-005	Low	Wrong revert Msg	Fixed	Confirmed

Table 2.3: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 Unverified signatures lead to arbitrary liquidation

ID:	NVE-001	**Location:**	CFDLiquidationFacet.sol
Severity:	High	**Category:**	Business Issues
Likelihood:	Medium	**Impact:**	High

Description:

The liquidatePositionsV2 method in the CFDLiquidationFacet contract does not validate the signature included in the MultiLiquidationOrderV2 struct. Specifically, the signature field is not checked to ensure that the liquidation order was authorized by the rightful owner of the accountToLiquidate. As a result, malicious actors can forge and submit unauthorized liquidation orders.

Impact:Malicious actors can impersonate a liquidator and submit forged liquidation orders to manipulate account balances or liquidate positions without proper authorization.Unauthorized liquidation can lead to severe financial losses for the affected accounts, as their positions might be liquidated at manipulated prices or fees.

```
/**
 * @notice Liquidates positions for an account
 * @param liquidationOrder The liquidation order details.
 */
function liquidatePositionsV2(
    CFDStructs.MultiLiquidationOrderV2 calldata liquidationOrder,
    uint64 executionTimestamp
) external override setOrderTimestamp(executionTimestamp) {
    CFDStructs.CFDStorage storage ds = CFDStorageLib.cfdStorage();
    if (!(ds.allowedMatchers[msg.sender] || msg.sender == liquidationOrder.liquidator))
        revert CFDEventsAndErrors.IncorrectSenderAddress();

```

Recommendations:

Introduce signature verification logic to verify whether the liquidation parameters are within a reasonable range.

Result: Confirmed

Fix Result: Fixed

Updated judgment.

3.2 Incorrect liquidation accumulation

ID:	NVE-002	Location:	CFDLiquidationFacet.sol
Severity:	High	Category:	Business Issues
Likelihood:	High	Impact:	Medium

Description:

In the contract, the logic of balancesAccumulator = _liquidatePositionV2(...) is to process the liquidation of a single position through the _liquidatePositionV2 method and return an updated BalancesAccumulator structure, which contains the accumulated values of the user balance, liquidator balance, and funding fee balance.

The problem is:In the _liquidatePositionV2 method, the value of balancesAccumulator.accUserBalanceSummand is overwritten after the calculation results of _getPNL and _getAccountFR. This overwriting operation means that even in multiple liquidations, subsequent liquidations will completely replace the previous accumulated values instead of accumulating them correctly.

Impact:User balances are not calculated correctly, the result of balances[accountToLiquidate] += balancesAccumulator.accUserBalanceSummand may be wrong because accUserBalanceSummand is not accumulated correctly.

```
function _liquidatePositionV2(
    uint256 index,
    address accountToLiquidate,
    address liquidator,
    int112 liquidationPrice,
    uint96 liquidationFee,
    BalancesAccumulator memory balancesAccumulator
) private returns (BalancesAccumulator memory) {
    CFDStructs.CFDStorage storage ds = CFDStorageLib.cfdStorage();
    CFDStructs.PositionInfo storage accountToLiquidatePosition = ds.positionInfo[index][accountToLiquidate];

    int112 atlPositionSize = accountToLiquidatePosition.position;

    if (atlPositionSize == 0) revert CFDEventsAndErrors.ZeroPositionLiquidation();

    balancesAccumulator.accUserBalanceSummand = _getPNL(accountToLiquidate, index, liquidationPrice);

    balancesAccumulator.accUserBalanceSummand +=
        (_getAccountFR(accountToLiquidate, index) * liquidationPrice) / CFDConstants.CFD_1e8_i112;

    {
```

Recommendations:

Accumulate correct Lyon each liquidation, balance is accumulator.accuser balance summand and other fields will accumulate all liquidation results.

Result: Confirmed

Fix Result: Fixed

The cumulative method has been used.

3.3 Possible bypass order expiration check

ID:	NVE-003	Location:	CFDFillOrdersFacet2.sol
Severity:	High	Category:	Business Issues
Likelihood:	High	Impact:	Medium

Description:

if Weiwei execute fill order via fill orders V2temporal, we have strict timestamp check:

```
function fillOrdersV2Temporal(
    CFDStructs.OrderV2 calldata buyOrder,
    CFDStructs.OrderV2 calldata sellOrder,
    uint80 filledPrice,
    uint96 filledAmount,
    uint64 executionTimestamp
) external override onlyMatcher_(msg.sender) setOrderTimestamp(executionTimestamp) {
    _fillOrdersV2(buyOrder, sellOrder, filledPrice, filledAmount);
}

function _fillOrdersV2(
    CFDStructs.OrderV2 calldata buyOrder,
    CFDStructs.OrderV2 calldata sellOrder,
    uint80 filledPrice,
    uint96 filledAmount
) private {
    CFDStructs.CFDStorage storage ds = CFDStorageLib.cfdStorage();

    // Check orders and get digests
    if (buyOrder.buySide != 1 && sellOrder.buySide != 0) revert CFDEventsAndErrors.IncorrectBuySide();

    // Check instrument indexes
    if (buyOrder.instrumentIndex != sellOrder.instrumentIndex)
        revert CFDEventsAndErrors.DifferentInstrumentIndexes();
    if (ds.instrumentsLength <= buyOrder.instrumentIndex) revert CFDEventsAndErrors.InstrumentDoesNotExist();

    // Check Price values
    if (filledPrice > buyOrder.price || filledPrice < sellOrder.price)
        revert CFDEventsAndErrors.IncorrectFilledPrice();

    // Check Expiration Time. Convert to seconds first
    uint64 orderTimestamp = _getOrderTimestamp();

    if (buyOrder.expiration / 1000 < orderTimestamp || sellOrder.expiration / 1000 < orderTimestamp)
        revert CFDEventsAndErrors.OrderExpired();
```

The issue is, once we execute order via fill order , we don't have setOrderTimestamp , so the value `_getOrderTimestamp()` return 0 , expiration check will always pass.

```
    /**
     * @inheritdoc ICFDFillOrdersFacet
     */
    function fillOrdersV2(
        CFDStructs.OrderV2 calldata buyOrder,
        CFDStructs.OrderV2 calldata sellOrder,
        uint80 filledPrice,
        uint96 filledAmount
    ) external override onlyMatcher_(msg.sender) {
        _fillOrdersV2(buyOrder, sellOrder, filledPrice, filledAmount);
    }
```

Impact: bypass order expiration check.

Recommendations:

It is recommended to delete the fillOrdersV2 method or add checks.

Result: Confirmed

Fix Result: Fixed

Removed fillOrdersV2 method.

3.4 Function selector collision

ID:	NVE-004	Location:	DiamondCFD.sol
Severity:	Medium	Category:	Business Issues
Likelihood:	Low	Impact:	High

Description:

In the DiamondCFD contract, if two or more facetAddress_ (Facet contracts) have methods with the same method selector (derived from the first 4 bytes of the keccak256 hash of the function signature), the contract cannot distinguish between them.

Impact: An attacker can exploit the selector collision problem to execute malicious methods with the same selector, resulting in unauthorized method execution.

```
function diamondCut(
    FacetCut[] calldata diamondCut_,
    address init_,
    bytes calldata calldata_
) external override onlyOwner(msg.sender) {
    for (uint256 i; i < diamondCut_.length; ++i) {
        FacetCut memory cut = diamondCut_[i];
        if (cut.action == FacetCutAction.Add) {
            _addFunctions(cut.facetAddress, cut.functionSelectors);
        } else if (cut.action == FacetCutAction.Replace) {
            _replaceFunctions(cut.facetAddress, cut.functionSelectors);
        } else if (cut.action == FacetCutAction.Remove) {
            _removeFunctions(cut.facetAddress, cut.functionSelectors);
        }
    }

    if (init_ != address(0)) {
        init_.functionDelegateCall(calldata_);
    }
    emit DiamondCut(diamondCut_, init_, calldata_);
}

function _addFunctions(address facetAddress_, bytes4[] memory functionSelectors_) internal {
    DiamondStorage storage ds = diamondStorage();
    require(facetAddress_ != address(0), "Diamond: Facet address cannot be zero");

    for (uint256 i; i < functionSelectors_.length; ++i) {
        bytes4 selector = functionSelectors_[i];
        if (ds.facetAddressAndSelectorPosition[selector] != address(0)) {
            revert FunctionAlreadyExists(selector);
        }
        ds.facetAddressAndSelectorPosition[selector] = facetAddress_;
        ds.selectors.add(bytes32(selector));
        ds.facetFunctionSelectors[facetAddress_].add(bytes32(selector));
    }
}
```

Recommendations:

before deploying A facet, ensure all function selector are unique across all facets int and diamond contract.

Result: Confirmed

Fix Result: Fixed

Customer response: Helper function added.

3.5 liquidate limit condition too simple

ID:	NVE-005	**Location:**	CFDLiquidationFacet.sol
Severity:	Low	**Category:**	Business Issues
Likelihood:	Low	**Impact:**	Low

Description:

Current liquidate logic only checks matcher call or liquidator call, some health position could possibly get liquidated, some other defi protocol have strict liquidation limitations, should have more strict limit to protect user funds.

Healthy position could possibly get liquidated.

```
/**
 * @notice Liquidates positions for an account
 * @param liquidationOrder The liquidation order details.
 */
function liquidatePositionsV2(
    CFDStructs.MultiLiquidationOrderV2 calldata liquidationOrder,
    uint64 executionTimestamp
) external override setOrderTimestamp(executionTimestamp) {
    CFDStructs.CFDStorage storage ds = CFDStorageLib.cfdStorage();
    if (!(ds.allowedMatchers[msg.sender] || msg.sender == liquidationOrder.liquidator))
        revert CFDEventsAndErrors.IncorrectSenderAddress();
```

Recommendations:

Should have more strict limitations.

Result: Confirmed

Fix Result: Fixed

Customer response: Additional health check added.

3.6 Wrong revert Msg

ID:	NVE-006	**Location:**	CFDLiquidationFacet.sol
Severity:	Low	**Category:**	Business Issues
Likelihood:	Low	**Impact:**	Low

Description:

Stop Profit order(TP) with error Msg

CFDEventsAndErrors.WrongStopLossOrderAmount(sltpOrder.amount);

Impact:Wrong error msg.

```
    function _sltpOrdersV2(
        CFDStructs.OrderV2 memory sltpOrder,
        CFDStructs.OrderV2 memory regularOrder,
        uint80 filledPrice,
        uint96 filledAmount
    ) private {
        CFDStructs.CFDStorage storage ds = CFDStorageLib.cfdStorage();

        if (sltpOrder.amount != 0) {
            if (_isTP(sltpOrder.orderType)) revert CFDEventsAndErrors.WrongStopLossOrderAmount(sltpOrder.amount);
            else revert CFDEventsAndErrors.WrongTakeProfitOrderAmount(sltpOrder.amount);
        }
```

Recommendations:

Modify the error to:

CFDEventsAndErrors.WrongStopLossOrderAmount(sltpOrder.amount).

Result: Confirmed

Fix Result: Fixed

Customer response: Updated error.

4. CONCLUSION

In this audit, we thoroughly analyzed **Electra** 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.
- Results: Not Found
- Severity: High

5.2.2 Account Permission Control

- Description: Examine permission control issue in the contract
- Results: Not Found
- Severity: Medium

5.2.3 Malicious Code Behavior

- Description: Examine whether sensitive behavior present in the code
- Results: 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
- Results: 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

www.exvul.com
contact@exvul.com
@EXVULSEC
github.com/EXVUL-Sec

ExVul