Fortuna Smart Contract

SMART CONTRACT AUDIT REPORT

March 2025

ExVul

www.exvul.com

Table of Contents

1. EXECUTIVE SUMMARY

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

IMPACT

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: Fortuna

Audit Time: March 20, 2025 – March 26, 2025

Language: solidity

Soure code	Link
Fortuna	https://github.com/YouNeedWork/fortuna-evm/tree/master
Commit Hash	1ab40717a404b4d65d1b591d366fd7f46bb18c90

2.2 Summary

Severity	Found	
Critical	3	🟥🟥🟥
High	0	
Medium	1	🟨
Low	0	
Informational	0	

2.3 Key Findings

ID	Severity	Findings Title	Status	Confirm
NVE-001	Critical	Lack of Amount Check in Withdrawal Confirmation	Fixed	Confirmed
NVE-002	Critical	No Check For Return Value	Fixed	Confirmed
NVE-003	Critical	Centralization and Fund Security Risks	Acknowledge	Confirmed
NVE-004	Medium	Inadequate Withdrawal Validation	Acknowledge	Confirmed

Table 2.3: Key Audit Findings

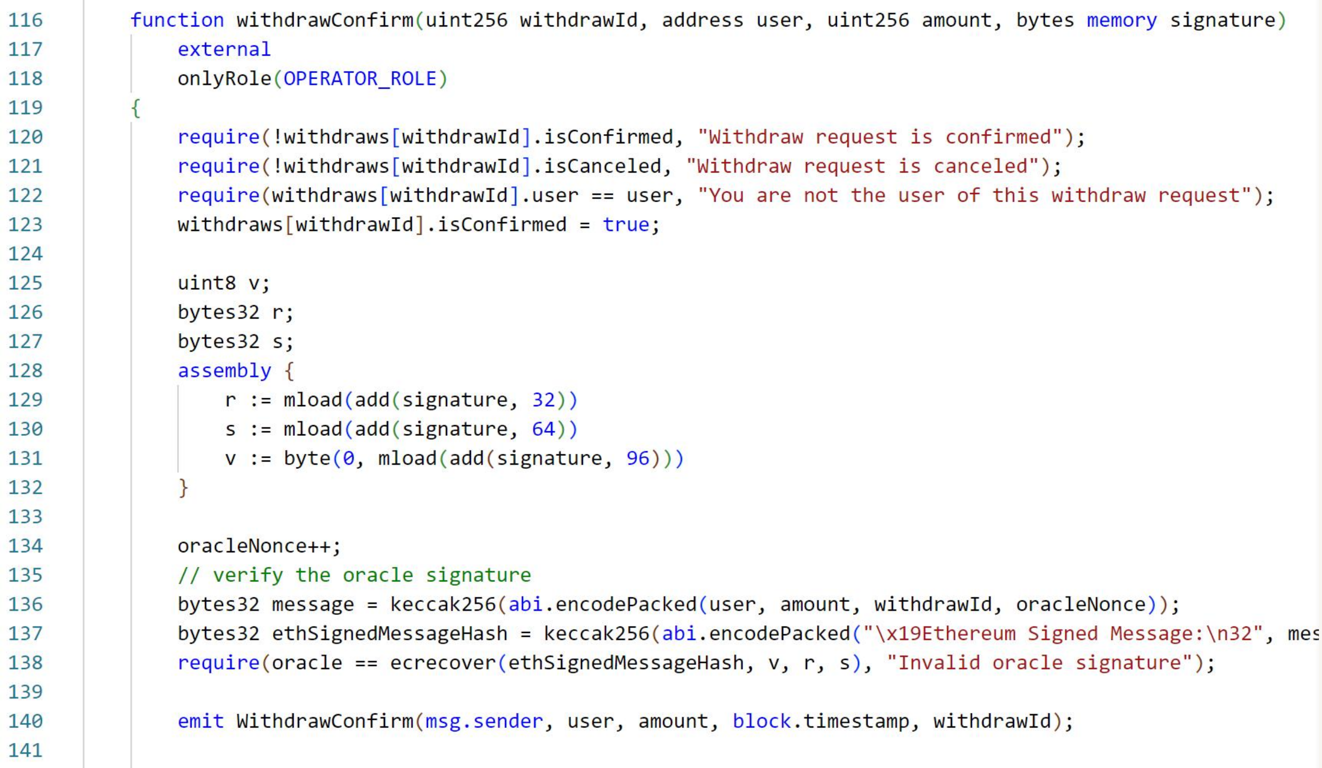
3. DETAILED DESCRIPTION OF FINDINGS

3.1 Lack of Amount Check in Withdrawal Confirmation

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

Description:

The withdrawConfirm function does not verify whether the amount parameter matches the amount stored in withdraws[withdrawId].amount. This omission creates a potential vulnerability that could allow an operator to manipulate the withdrawal amount, resulting in unintended fund transfers and potential financial losses for users.



```
    function withdrawConfirm(uint256 withdrawId, address user, uint256 amount, bytes memory signature)
        external
        onlyRole(OPERATOR_ROLE)
    {
        require(!withdraws[withdrawId].isConfirmed, "Withdraw request is confirmed");
        require(!withdraws[withdrawId].isCanceled, "Withdraw request is canceled");
        require(withdraws[withdrawId].user == user, "You are not the user of this withdraw request");
        withdraws[withdrawId].isConfirmed = true;

        uint8 v;
        bytes32 r;
        bytes32 s;
        assembly {
            r := mload(add(signature, 32))
            s := mload(add(signature, 64))
            v := byte(0, mload(add(signature, 96)))
        }

        oracleNonce++;
        // verify the oracle signature
        bytes32 message = keccak256(abi.encodePacked(user, amount, withdrawId, oracleNonce));
        bytes32 ethSignedMessageHash = keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32", mes
        require(oracle == ecrecover(ethSignedMessageHash, v, r, s), "Invalid oracle signature");

        emit WithdrawConfirm(msg.sender, user, amount, block.timestamp, withdrawId);

```

Recommendations:

Add amount validation to ensure the amount parameter matches withdraws[withdrawId].amount for accurate withdrawals.

Result: Confirmed

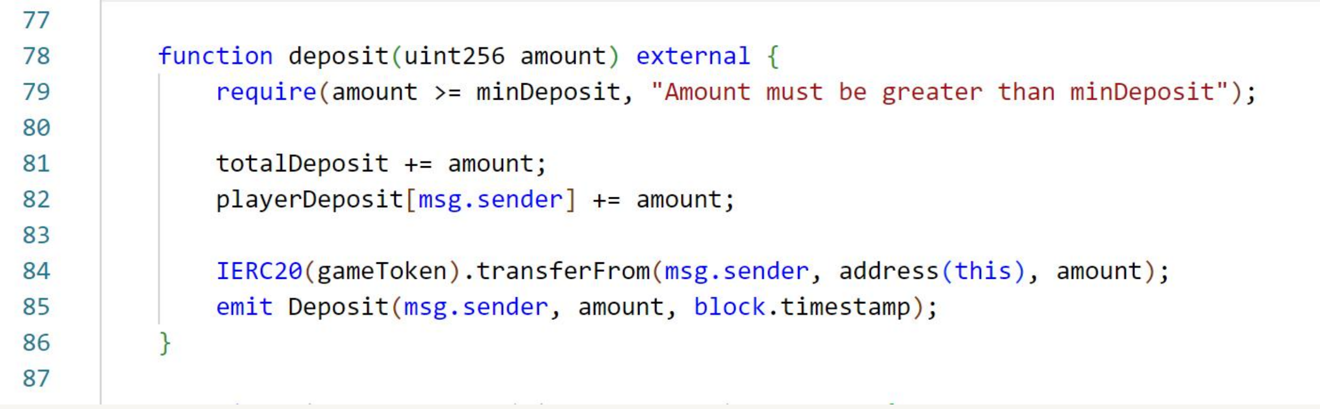
Fix Result: Fixed in commit 96e7d7f

3.2 No Check For Return Value

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

Description:

The transfer and transferFrom operations in the smart contract do not check the return value, which can lead to unexpected behavior and potential security vulnerabilities. Since ERC20 tokens specify that these functions should return a boolean indicating success or failure, ignoring this return value can result in silent failures where the transfer might fail without proper notification, potentially causing loss of funds and compromising the reliability of the contract's operations.



```

    function deposit(uint256 amount) external {
        require(amount >= minDeposit, "Amount must be greater than minDeposit");

        totalDeposit += amount;
        playerDeposit[msg.sender] += amount;

        IERC20(gameToken).transferFrom(msg.sender, address(this), amount);
        emit Deposit(msg.sender, amount, block.timestamp);
    }

```

Recommendations:

Use OpenZeppelin's SafeERC20 library functions safeTransfer and safeTransferFrom to check return values, ensuring transfer success and preventing silent failures.

Result: Confirmed

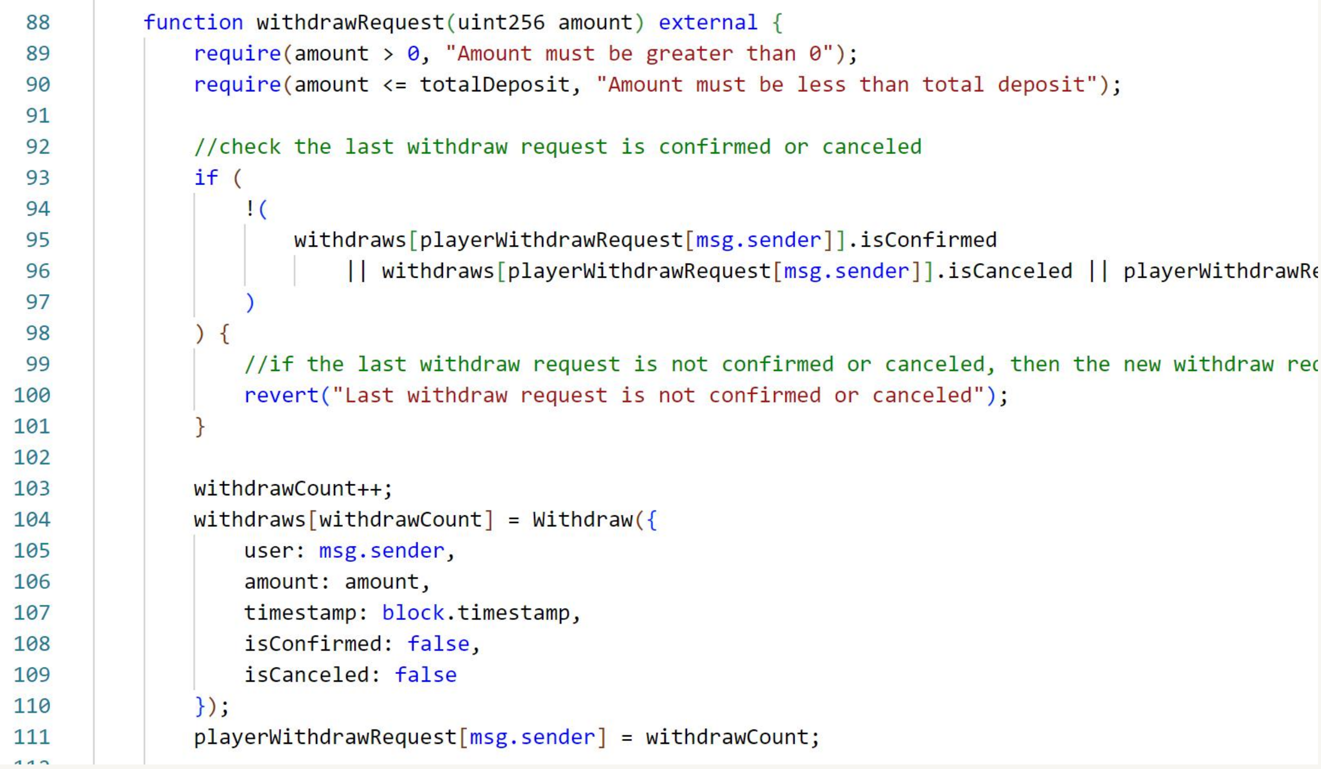
Fix Result: Fixed in commit f5a7b79

3.3 Centralization and Fund Security Risks

ID:	NVE-003	**Location:**	Fortuna.sol
Severity:	Cirtical	**Category:**	Business Issues
Likelihood:	High	**Impact:**	High

Description:

The project's excessive centralization creates multiple risks: the owner can change gameToken, potentially locking user funds; feePercent lacks restrictions, risking user losses; distributeFee allows unchecked fund extraction by admins; and users need approval to withdraw, blocking their access without it.



```
function withdrawRequest(uint256 amount) external {
    require(amount > 0, "Amount must be greater than 0");
    require(amount <= totalDeposit, "Amount must be less than total deposit");

    //check the last withdraw request is confirmed or canceled
    if (
        !(
            withdraws[playerWithdrawRequest[msg.sender]].isConfirmed
                || withdraws[playerWithdrawRequest[msg.sender]].isCanceled || playerWithdrawRe
        )
    ) {
        //if the last withdraw request is not confirmed or canceled, then the new withdraw rec
        revert("Last withdraw request is not confirmed or canceled");
    }

    withdrawCount++;
    withdraws[withdrawCount] = Withdraw({
        user: msg.sender,
        amount: amount,
        timestamp: block.timestamp,
        isConfirmed: false,
        isCanceled: false
    });
    playerWithdrawRequest[msg.sender] = withdrawCount;
```

Recommendations:

Implement multi-signature or governance for critical changes, set fee limits, restrict distributeFee, allow direct user withdrawals after conditions are met, and consider decentralized governance for transparency.

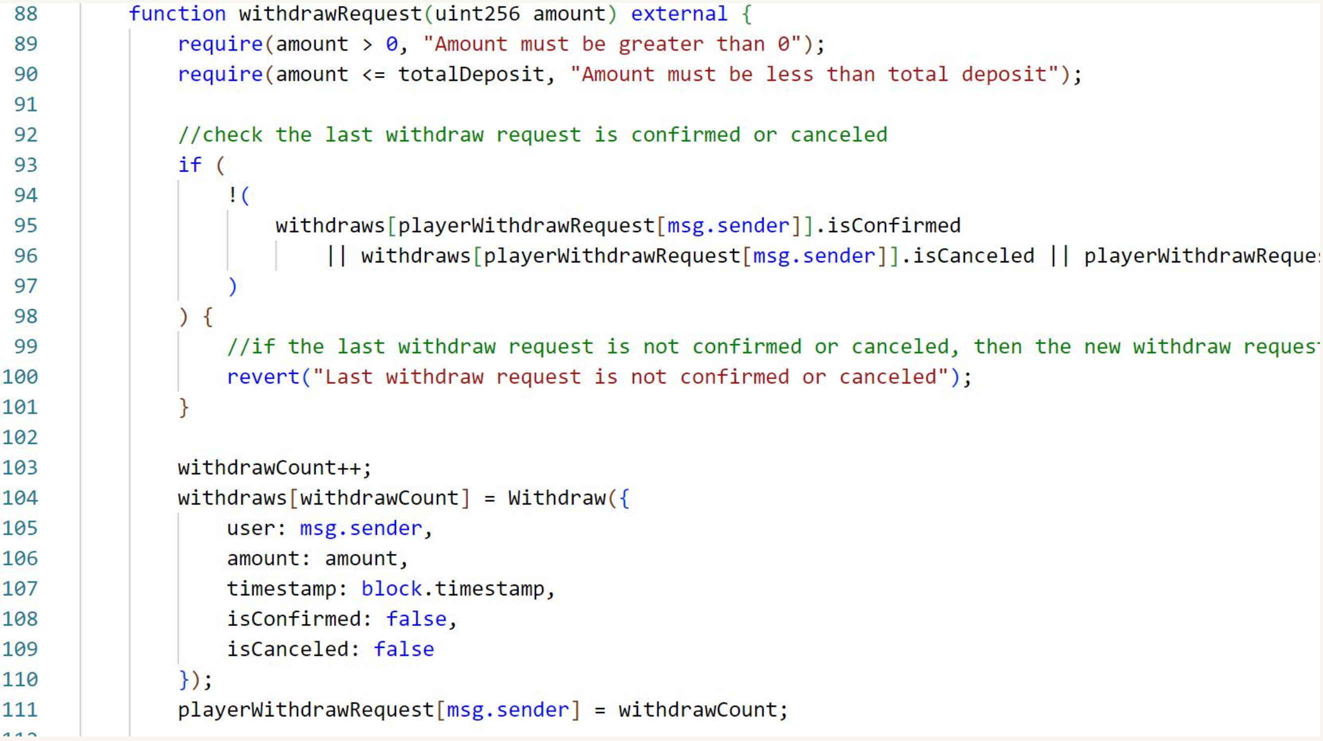
Result: Confirmed

3.4 Inadequate Withdrawal Validation

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

Description:

The withdrawRequest function incorrectly checks if the withdrawal amount is less than or equal to totalDeposit instead of verifying against the user's actual balance. This flawed validation could allow users to submit withdrawal requests exceeding their available balance, potentially leading to overwithdrawal attempts and fund discrepancies.



```
    function withdrawRequest(uint256 amount) external {
        require(amount > 0, "Amount must be greater than 0");
        require(amount <= totalDeposit, "Amount must be less than total deposit");

        //check the last withdraw request is confirmed or canceled
        if (
            !(
                withdraws[playerWithdrawRequest[msg.sender]].isConfirmed
                    || withdraws[playerWithdrawRequest[msg.sender]].isCanceled || playerWithdrawReque
            )
        ) {
            //if the last withdraw request is not confirmed or canceled, then the new withdraw reques
            revert("Last withdraw request is not confirmed or canceled");
        }

        withdrawCount++;
        withdraws[withdrawCount] = Withdraw({
            user: msg.sender,
            amount: amount,
            timestamp: block.timestamp,
            isConfirmed: false,
            isCanceled: false
        });
        playerWithdrawRequest[msg.sender] = withdrawCount;
```

Recommendations:

Modify the check to compare the withdrawal amount against the user's balance rather than the total deposit to ensure accurate withdrawal validation.

Result: Confirmed

4. CONCLUSION

In this audit, we thoroughly analyzed **Fortuna** 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