MoveFlow-Edu Smart Contract

SMART CONTRACT AUDIT REPORT

March 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	*High*	*Informational*	Medium	High	Critical
	Medium	*Informational*	Low	Medium	High
	Low	*Informational*	Low	Low	Medium
		Informational	*Low*	*Medium*	*High*
			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: Moveflow

Audit Time: March16, 2025 – March 26, 2025

Language: solidity

Soure code	Link
Moveflow-edu	https://github.com/move-flow/moveflow-edu
Commit Hash	2d6a6b58a4874143b3ab1a40d6cdfe39849a1901

2.2 Summary

Severity	Found	
Critical	0	
High	1	🟥
Medium	1	🟨
Low	2	🟦🟦
Informational	0	

2.3 Key Findings

ID	Severity	Findings Title	Status	Confirm
NVE-001	High	Missing Check for Stream Pause Status in closeStream Function	Fixed	Confirmed
NVE-002	Medium	No Check for New Stream Stop Time in Function Extend	Fixed	Confirmed
NVE-003	Low	Missing Token Removal Functionality	Fixed	Confirmed
NVE-004	Low	Lack of Validation for Cliff Amount in the create function	Fixed	Confirmed

Table 2.3: Key Audit Findings

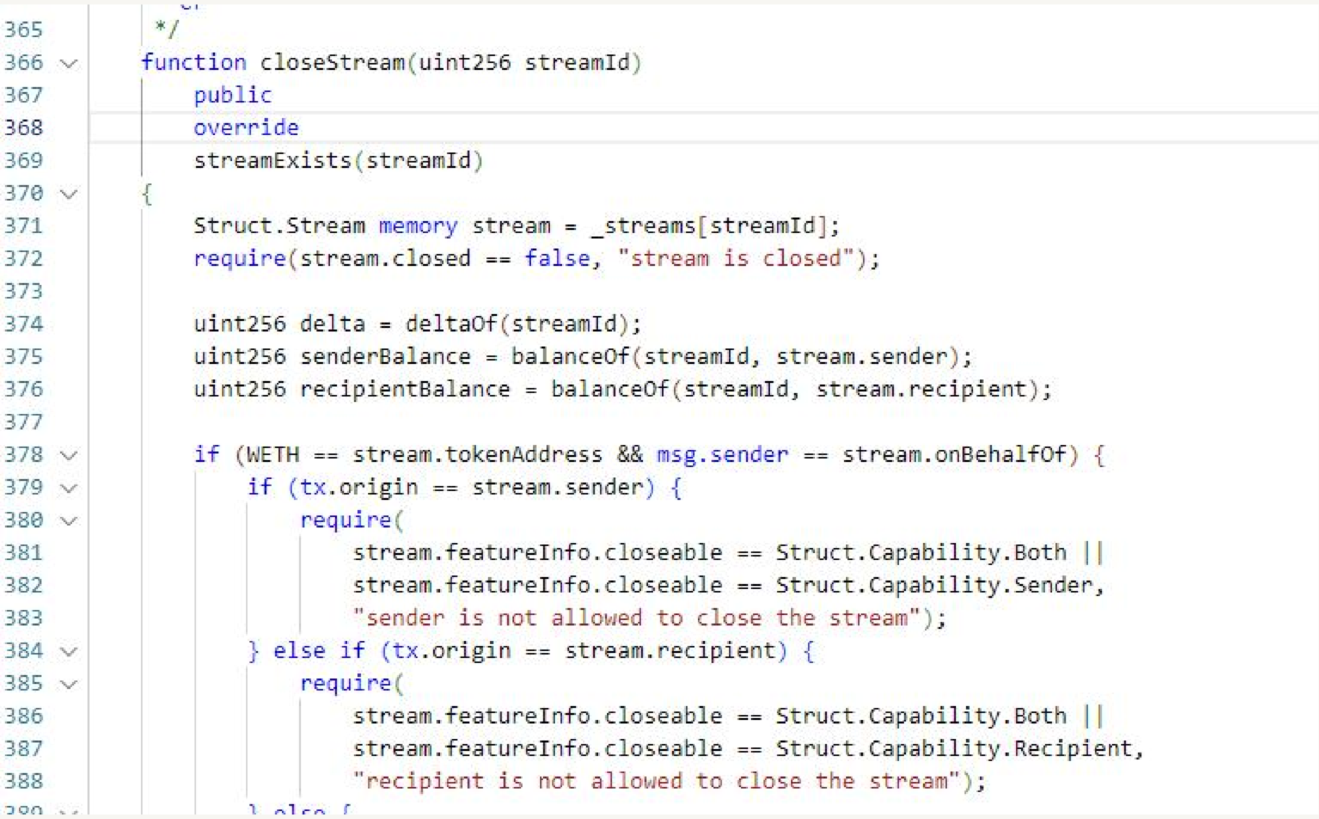
3. DETAILED DESCRIPTION OF FINDINGS

3.1 Missing Check for Stream Pause Status in closeStream Function

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

Description:

The Stream:closeStream function fails to check whether the stream is not in a paused state (stream.pauseInfo.isPaused == false) before permitting closure. This could enable users to force-close a stream when it paused, bypassing intended restrictions and potentially causing unintended financial outcomes or logical errors in the system.



```
     */
    function closeStream(uint256 streamId)
        public
        override
        streamExists(streamId)
    {
        Struct.Stream memory stream = _streams[streamId];
        require(stream.closed == false, "stream is closed");

        uint256 delta = deltaOf(streamId);
        uint256 senderBalance = balanceOf(streamId, stream.sender);
        uint256 recipientBalance = balanceOf(streamId, stream.recipient);

        if (WETH == stream.tokenAddress && msg.sender == stream.onBehalfOf) {
            if (tx.origin == stream.sender) {
                require(
                    stream.featureInfo.closeable == Struct.Capability.Both ||
                    stream.featureInfo.closeable == Struct.Capability.Sender,
                    "sender is not allowed to close the stream");
            } else if (tx.origin == stream.recipient) {
                require(
                    stream.featureInfo.closeable == Struct.Capability.Both ||
                    stream.featureInfo.closeable == Struct.Capability.Recipient,
                    "recipient is not allowed to close the stream");
            } else {
```

Recommendations:

Add a check at the beginning of the closeStream function to ensure the stream is not in a paused state before allowing closure.

Result: Confirmed

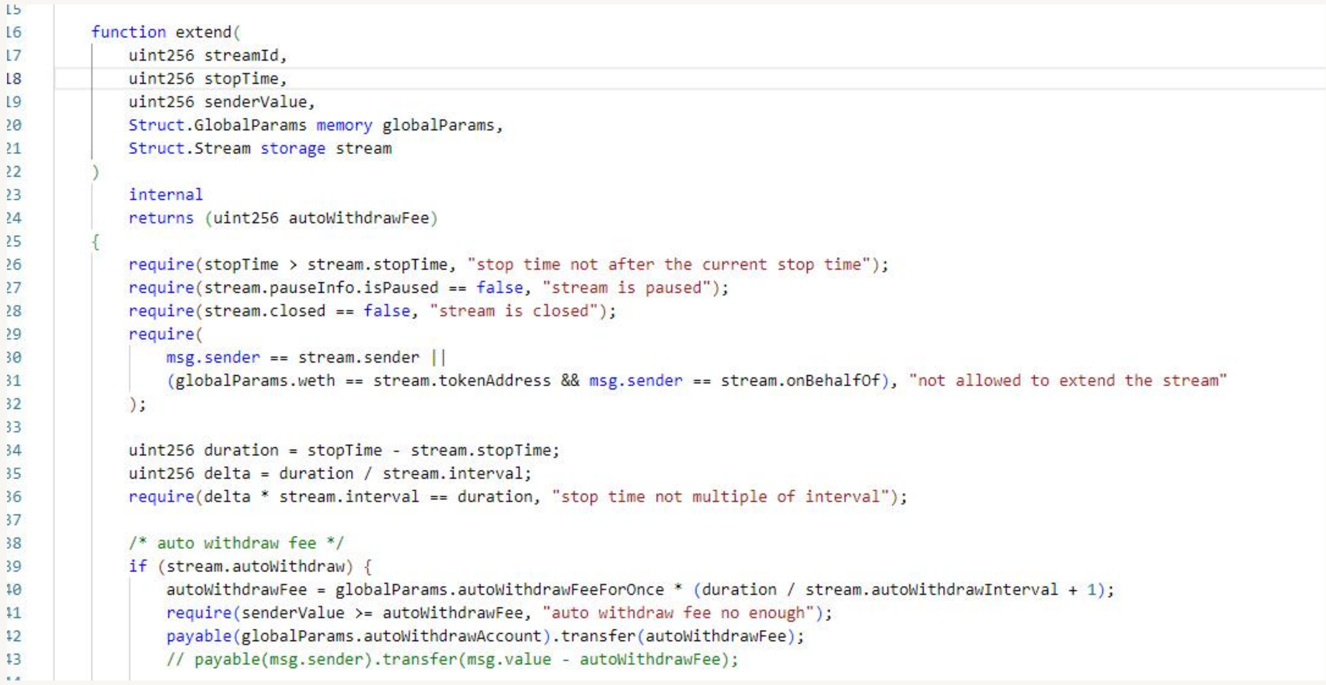
Fix Result: Fixed in commit ca9875c

3.2 No Check for New Stream Stop Time in Function

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

Description:

The ExtendLogic:extend function only checks if the new stopTime is greater than the current stream's stopTime, but does not verify whether the current stream's stopTime has already passed. This can lead to issues when attempting to extend a stream whose stopTime is already in the past.



```
    function extend(
        uint256 streamId,
        uint256 stopTime,
        uint256 senderValue,
        Struct.GlobalParams memory globalParams,
        Struct.Stream storage stream
    )
        internal
        returns (uint256 autoWithdrawFee)
    {
        require(stopTime > stream.stopTime, "stop time not after the current stop time");
        require(stream.pauseInfo.isPaused == false, "stream is paused");
        require(stream.closed == false, "stream is closed");
        require(
            msg.sender == stream.sender ||
            (globalParams.weth == stream.tokenAddress && msg.sender == stream.onBehalfOf), "not allowed to extend the stream"
        );

        uint256 duration = stopTime - stream.stopTime;
        uint256 delta = duration / stream.interval;
        require(delta * stream.interval == duration, "stop time not multiple of interval");

        /* auto withdraw fee */
        if (stream.autoWithdraw) {
            autoWithdrawFee = globalParams.autoWithdrawFeeForOnce * (duration / stream.autoWithdrawInterval + 1);
            require(senderValue >= autoWithdrawFee, "auto withdraw fee no enough");
            payable(globalParams.autoWithdrawAccount).transfer(autoWithdrawFee);
            // payable(msg.sender).transfer(msg.value - autoWithdrawFee);
```

Recommendations:

Add a check in the extend function to ensure the current stream's stopTime is greater than block.timestamp before allowing the extension.

Result: Confirmed

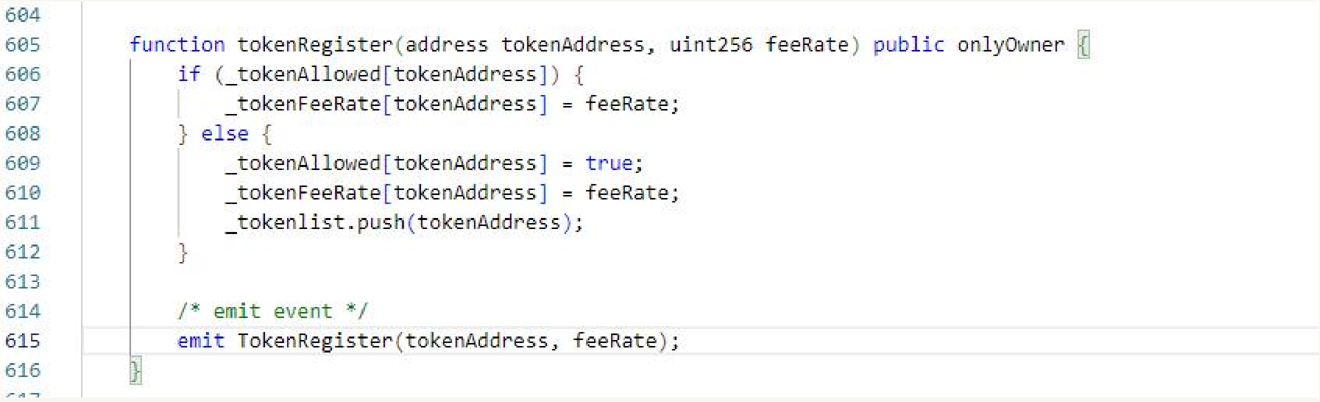
Fix Result: Fixed in commit ca9875c

3.3 Missing Token Removal Functionality

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

Description:

The current implementation only includes a Stream:tokenRegister function, which allows registering new tokens. However, there is no function to remove or unregister tokens. This creates a significant risk because if a token is compromised, hacked, or found to have issues, the system has no way to stop new streams from being created with that token. This lack of functionality could lead to security vulnerabilities, financial losses, or other problems if problematic tokens remain in use.



```
    function tokenRegister(address tokenAddress, uint256 feeRate) public onlyOwner {
        if (_tokenAllowed[tokenAddress]) {
            _tokenFeeRate[tokenAddress] = feeRate;
        } else {
            _tokenAllowed[tokenAddress] = true;
            _tokenFeeRate[tokenAddress] = feeRate;
            _tokenlist.push(tokenAddress);
        }

        /* emit event */
        emit TokenRegister(tokenAddress, feeRate);
    }
```

Recommendations:

Add a tokenunregister function that allows authorized parties (such as the contract owner) to remove or unregister compromised tokens.

Result: Confirmed

Fix Result: Fixed in commit ca9875c

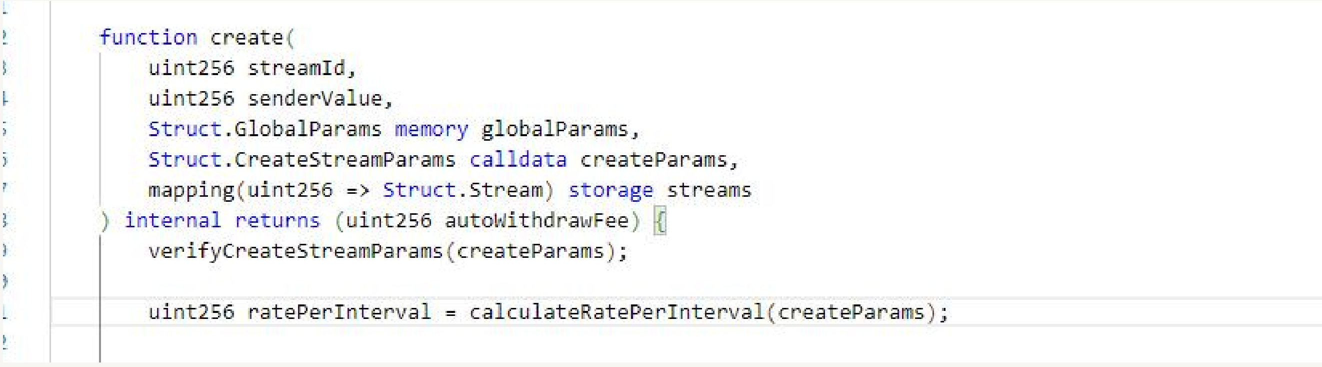
3.4 Lack of Validation for Cliff Amount in the create function

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

Description:

The CreateLogic:create function lacks an explicit check to ensure that cliffAmount does not

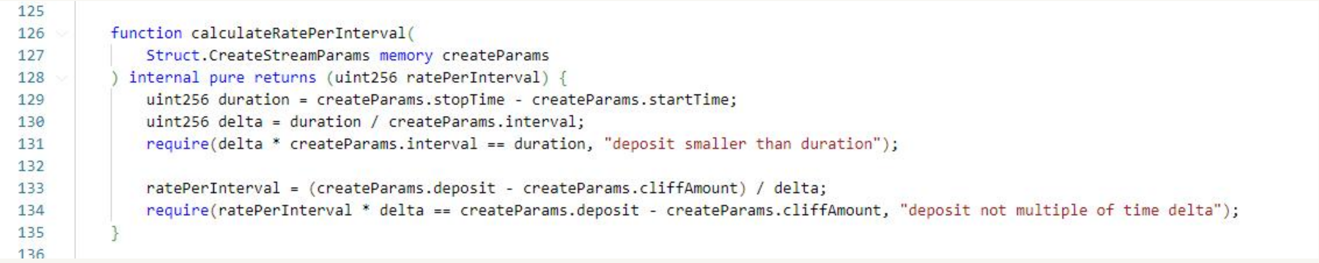
Exceed deposit. While the calculateRatePerInterval function indirectly verifies this through its calculations, a confusing error message would be thrown if cliffAmount were greater than deposit, as the root cause would not be clearly indicated.



```
    function create(
        uint256 streamId,
        uint256 senderValue,
        Struct.GlobalParams memory globalParams,
        Struct.CreateStreamParams calldata createParams,
        mapping(uint256 => Struct.Stream) storage streams
    ) internal returns (uint256 autoWithdrawFee) {
        verifyCreateStreamParams(createParams);

        uint256 ratePerInterval = calculateRatePerInterval(createParams);

```



```

    function calculateRatePerInterval(
        Struct.CreateStreamParams memory createParams
    ) internal pure returns (uint256 ratePerInterval) {
        uint256 duration = createParams.stopTime - createParams.startTime;
        uint256 delta = duration / createParams.interval;
        require(delta * createParams.interval == duration, "deposit smaller than duration");

        ratePerInterval = (createParams.deposit - createParams.cliffAmount) / delta;
        require(ratePerInterval * delta == createParams.deposit - createParams.cliffAmount, "deposit not multiple of time delta");
    }

```

Recommendations:

Add an explicit check in the create function to ensure cliffAmount is less than or equal to deposit.

Result: Confirmed

Fix Result: Fixed in commit ca9875c

4. CONCLUSION

In this audit, we thoroughly analyzed **Moveflow-edu** 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