Morph Smart Contract

March 2024

SMART CONTRACT AUDIT REPORT

ExVul

www.exvul.com

Table of Contents

1. EXECUTIVE SUMMARY

Exvul Web3 Security was engaged by **Morph** 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.

Critical risk finding is primarily related to the process of consensus verification.

High risk findings are primarily related to the permission control and votes etc.

Medium risk findings are primarily related to the cross-chain message replay, parameter modifying and gas exceed limits, etc.

Low risk findings are primarily related to the fund allocation, address checking, parameter checking, etc.

Informational risk findings are primarily related to the Gas optimization and unused variables etc.

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

Category	Assessment Item
	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: Morph

Audit Time: March7nd, 2024 – March27th, 2024

Language: solidity

GitHub Link	commit
https://github.com/morph-l2/morph	523eaf60a40ed0f6ea689101aadb9c911bb712dd

File Name	Path
L1CrossDomainMessenger.sol	contracts/contracts/L1/L1CrossDomainMessenger.sol
Rollup.sol	contracts/contracts/L1/rollup/Rollup.sol
Staking.sol	contracts/contracts/L1/staking/Staking.sol
L1Sequencer.sol	contracts/contracts/L1/staking/L1Sequencer.sol
Gov.sol	contracts/contracts/L2/staking/Gov.sol
L2Sequencer.sol	contracts/contracts/L2/staking/L2Sequencer.sol
GasPriceOracle.so	contracts/contracts/L2/system/GasPriceOracle.sol
L2ToL1MessagePasser.sol	contracts/contracts/L2/system/L2ToL1MessagePasser.sol
L2CrossDomainMessenger.sol	contracts/contracts/L2/L2CrossDomainMessenger.sol
Tree.sol	contracts/contracts/libraries/common/Tree.sol
Sequencer.sol	contracts/contracts/libraries/sequencer/Sequencer.sol
CrossDomainMessenger.sol	contracts/contracts/libraries/CrossDomainMessenger.sol

2.2 Summary

Severity	Found	
Critical	1	🟥
High	3	🟥🟥🟥
Medium	4	🟨🟨🟨🟨
Low	6	🟦🟦🟦🟦🟦🟦
Informational	4	🟩🟩🟩🟩

2.3 Key Findings

Critical risk finding is primarily related to the process of consensus verification.

High risk findings are primarily related to the permission control and votes etc.

Medium risk findings are primarily related to the cross-chain message replay, parameter modifying and gas exceed limits, etc.

Low risk findings are primarily related to the fund allocation, address checking, parameter checking, etc.

Informational risk findings are primarily related to the Gas optimization and unused variables etc.

ID	Severity	Findings Title	Status	Confirm
NVE-001	Critical	Sequencers could submit batches that are not consensus-determined	Ignored	Confirmed
NVE-002	High	Missing permission control	Ignored	Confirmed
NVE-003	High	revertBatch function locks funds	Ignored	Confirmed
NVE-004	High	Check the number of verifiers	Ignored	Confirmed
NVE-005	Medium	The updateProofWindow function affects the results of prove	Ignored	Confirmed
NVE-006	Medium	Cross-chain message replay	Ignored	Confirmed
NVE-007	Medium	The _fee condition may be bypassed	Ignored	Confirmed
NVE-008	Medium	Gas may exceed limits	Ignored	Confirmed
NVE-009	Low	importGenesisBatch function init	Ignored	Confirmed
NVE-0010	Low	inChallenge is always false	Ignored	Confirmed
NVE-011	Low	Funds allocation does not match documentation	Ignored	Confirmed
NVE-012	Low	updateMaxGasLimit function	Ignored	Confirmed
NVE-013	Low	add and remove list checks	Ignored	Confirmed
NVE-014	Low	Code logic redundancy	Ignored	Confirmed
NVE-015	Informational	Error log message	Ignored	Confirmed
NVE-	Informational	Gas optimization	Ignored	Confirmed

ID	Severity	Findings Title	Status	Confirm
016				
NVE-017	Informational	Unused variables	Ignored	Confirmed
NVE-018	Informational	claimEth function does not follow the CEI pattern	Ignored	Confirmed

Table 2.1: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 Sequencers could submit batches that are not consensus-determined

ID:	NVE-001	**Location:**	Rollup.sol,L1Sequencer.sol
Severity:	High	**Category:**	Business Issues
Likelihood:	High	**Impact:**	High

Description:

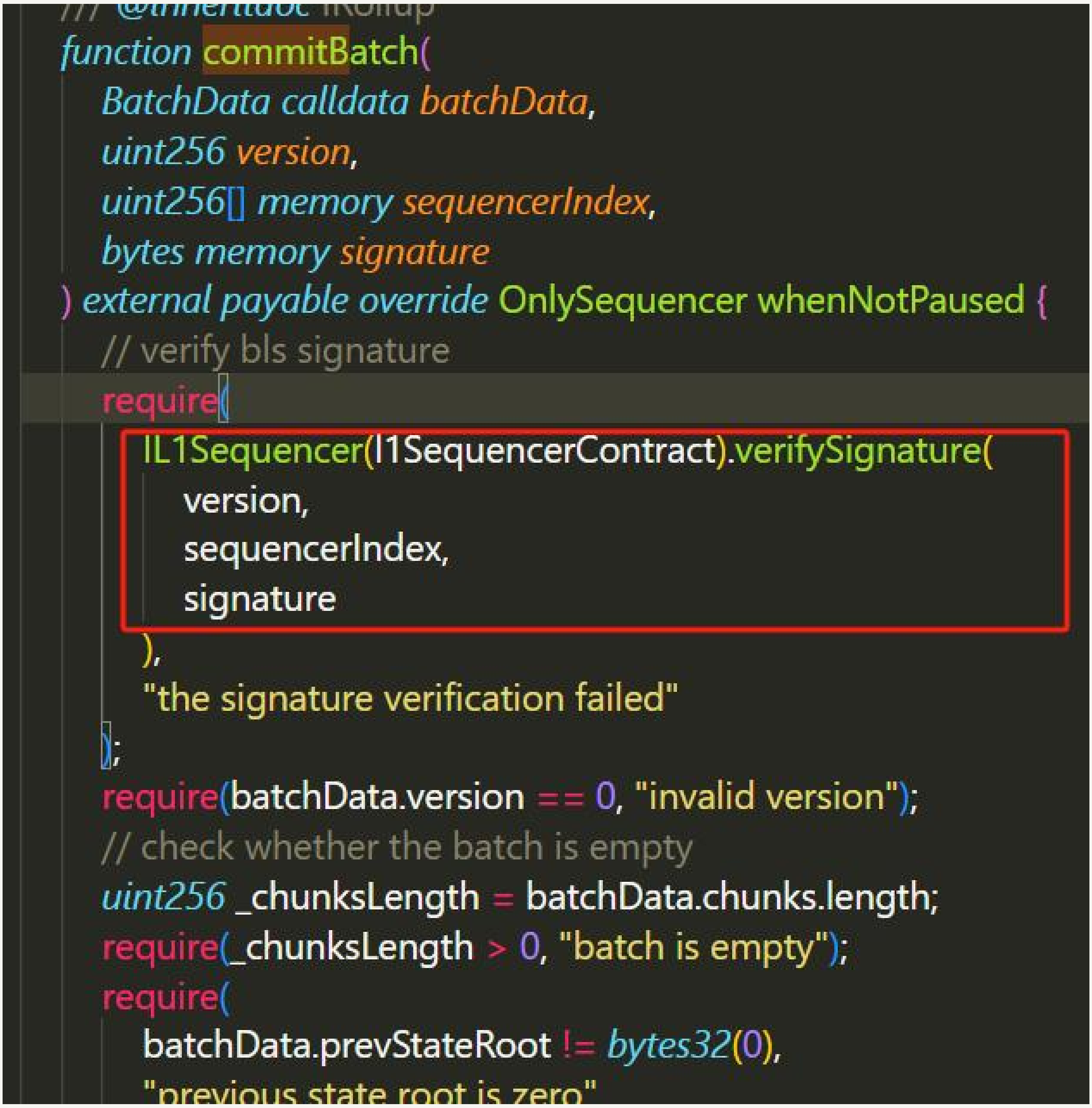
As shown in the figure below, Sequencers could submit batches that are not consensus-determined

The process of consensus verification depends on

- The aggregated BLS signatures
- The batch of transactions
- The consensus-determined state

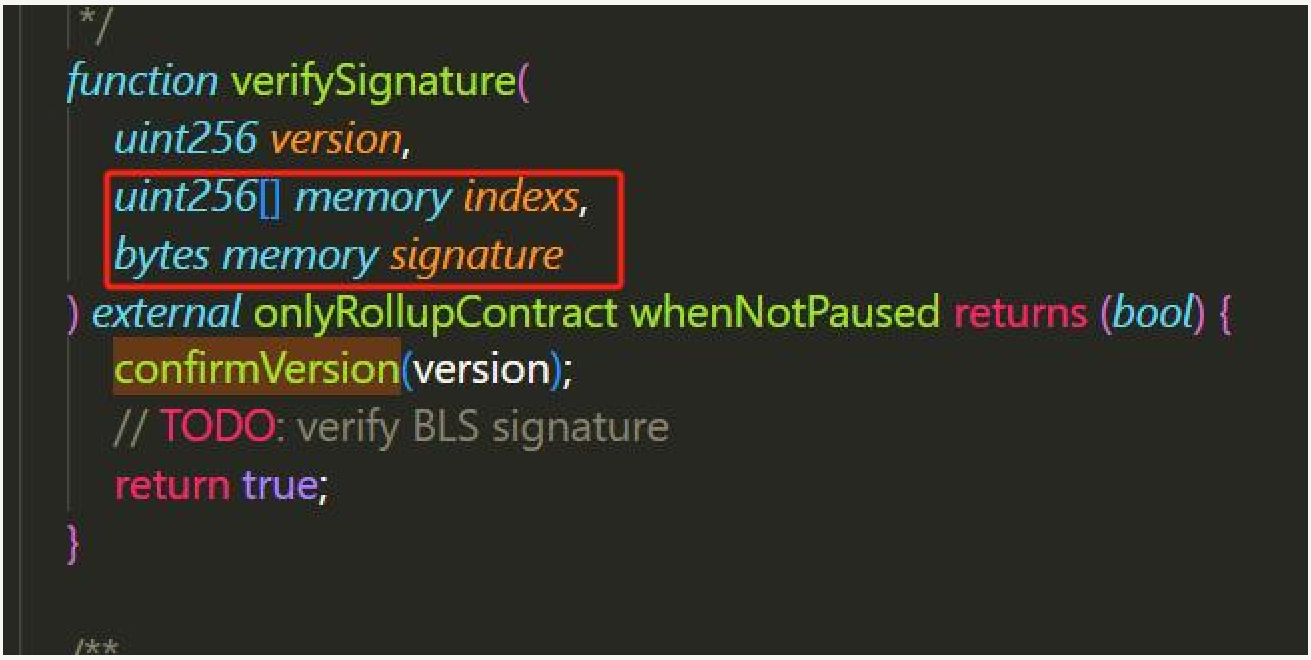
Which is submitted by sequencers. However in the current implementation, the sequencers could submit batches that are not consensus-determined because the `Rollup` contract could not verify the submitted BLS signatures along with the batches to confirm the consensus.

The root cause of this issue is from the function `L1Sequencer.verifySignature()`, which does not verify the BLS signature.



```
/// @inheritdoc IRollup
function commitBatch(
    BatchData calldata batchData,
    uint256 version,
    uint256[] memory sequencerIndex,
    bytes memory signature
) external payable override OnlySequencer whenNotPaused {
    // verify bls signature
    require(
        IL1Sequencer(l1SequencerContract).verifySignature(
            version,
            sequencerIndex,
            signature
        ),
        "the signature verification failed"
    );
    require(batchData.version == 0, "invalid version");
    // check whether the batch is empty
    uint256 _chunksLength = batchData.chunks.length;
    require(_chunksLength > 0, "batch is empty");
    require(
        batchData.prevStateRoot != bytes32(0),
        "previous state root is zero"
```

Figure 3.1.1 Rollup.sol



```
*/
function verifySignature(
    uint256 version,
    uint256[] memory indexs,
    bytes memory signature
) external onlyRollupContract whenNotPaused returns (bool) {
    confirmVersion(version);
    // TODO: verify BLS signature
    return true;
}

/**
```

Figure 3.1.2 L1Sequencer.sol

Recommendations:

ExVul Web3 Labs recommends checking code logic.

Result: Confirmed

Fix Result: Ignore

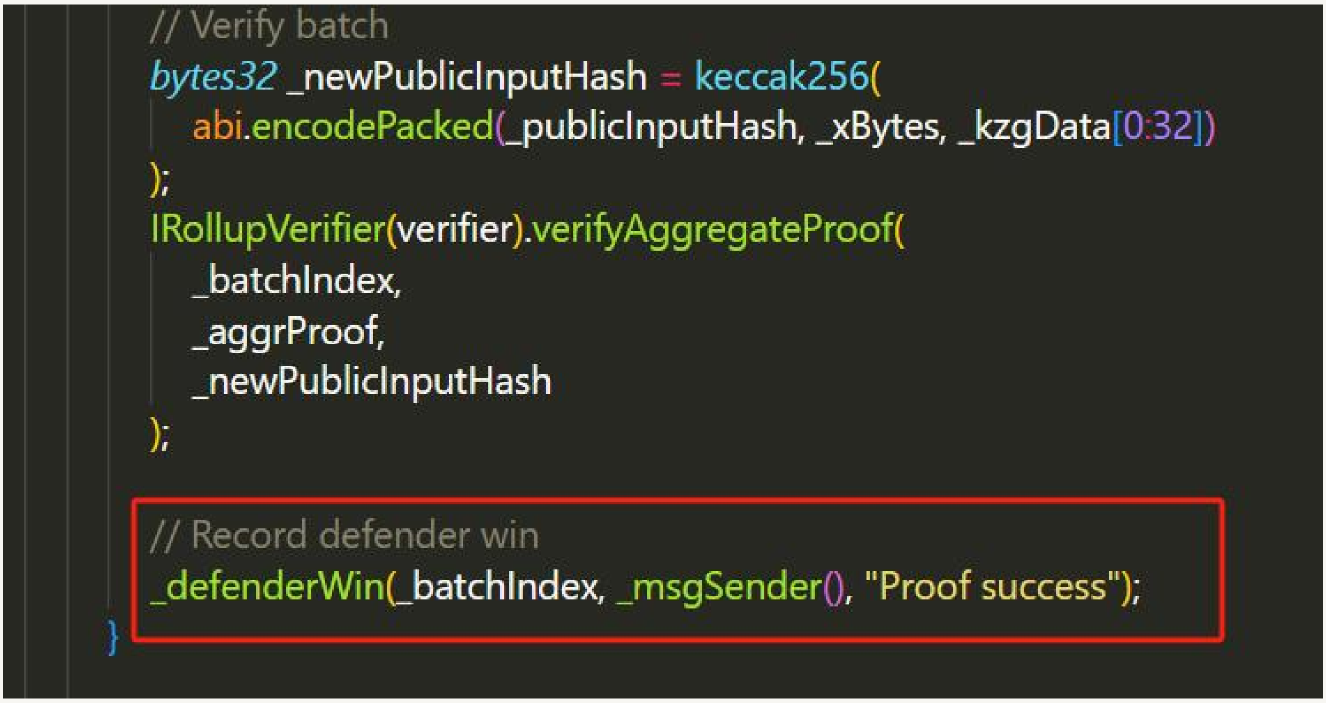
3.2 Missing permission control

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

Description:

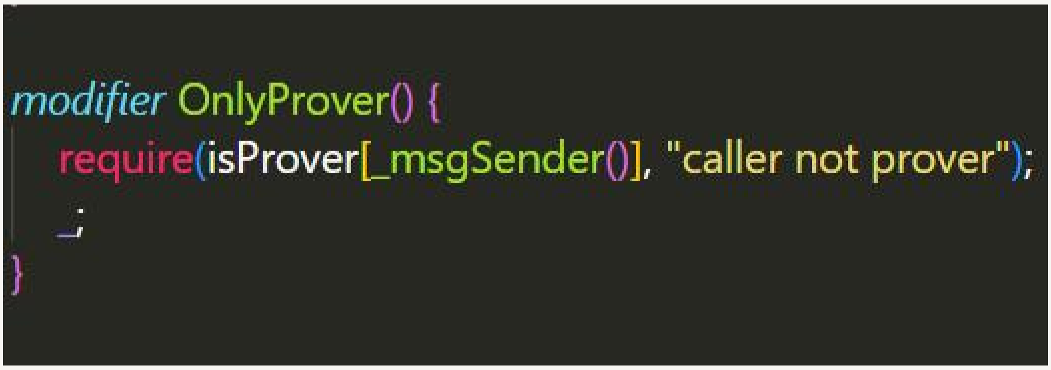
As shown in the figure below, the proveState function does not use permission control. The `OnlyProver` is defined in the context but is not used.

When the challenge fails, the reward is sent to msg.sender, `_defenderWin(_batchIndex, _msgSender(), "Proof success");`
Front-running results in funds being sent to the front-runner.



```
    // Verify batch
    bytes32 _newPublicInputHash = keccak256(
        abi.encodePacked(_publicInputHash, _xBytes, _kzgData[0:32])
    );
    IRollupVerifier(verifier).verifyAggregateProof(
        _batchIndex,
        _aggrProof,
        _newPublicInputHash
    );

    // Record defender win
    _defenderWin(_batchIndex, _msgSender(), "Proof success");
}
```

Figure 3.2.1 Rollup.sol



```
modifier OnlyProver() {
    require(isProver[_msgSender()], "caller not prover");
    _;
}
```

Figure 3.2.2 Rollup.sol

Recommendations:

ExVul Web3 Labs recommends adding permission checks and caller verification.

Result: Confirmed

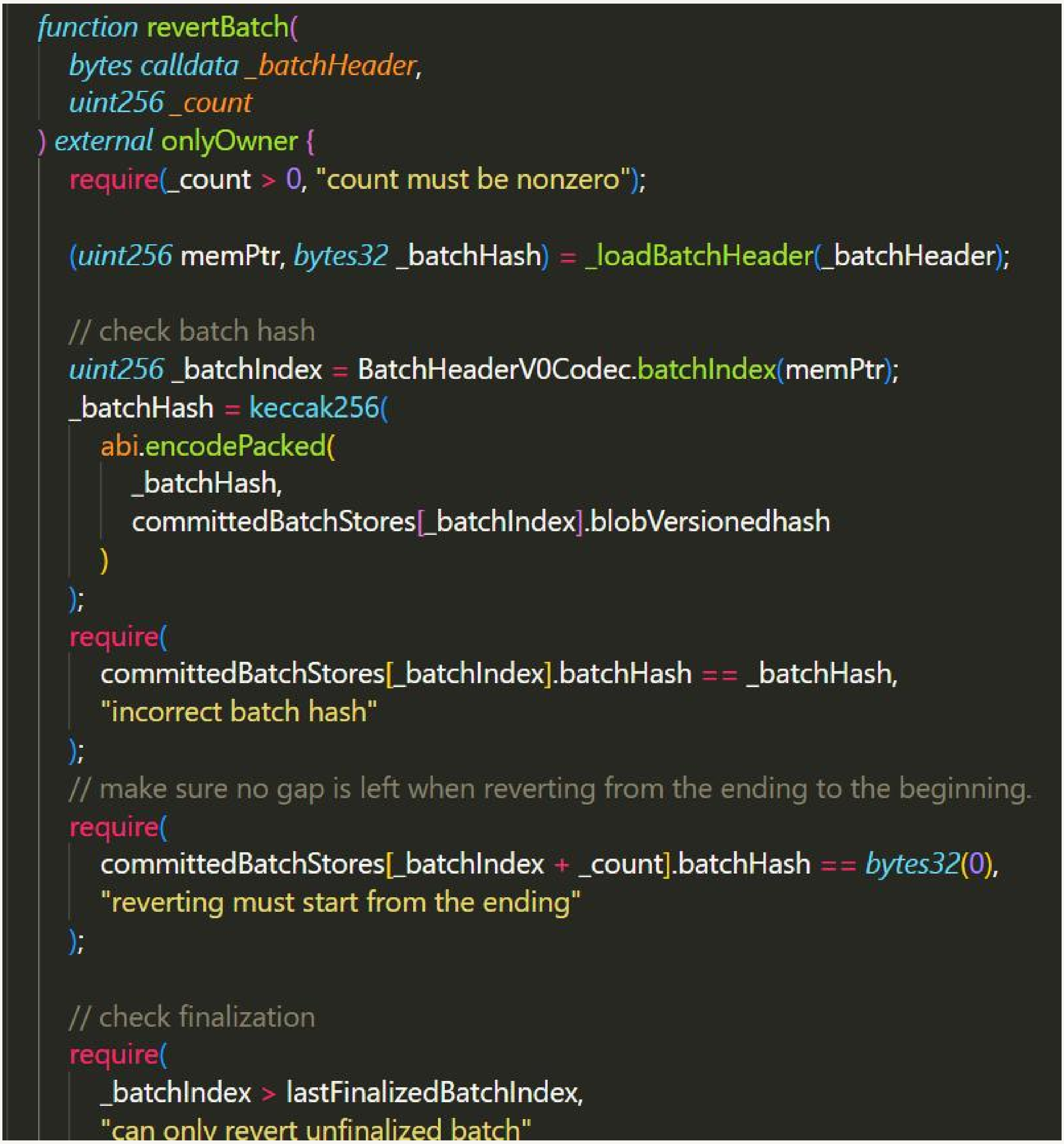
Fix Result: Ignored

3.3 revertBatch function locks funds

ID:	NVE-003	**Location:**	Rollup.sol
Severity:	Medium	**Category:**	Business Issues
Likelihood:	Medium	**Impact:**	Medium

Description:

As shown in the figure below, in the revertBatch function, the contract owner can delete the commit of challengeState, and the funds of challengeState will be locked in the contract. And if a challenge has been initiated, if the batch that initiated the challenge is deleted at this time, then the next batch is actually already in the challenge. If the verification time exceeds, the next challenge will become invalid, resulting in verification failure.



```
function revertBatch(
    bytes calldata _batchHeader,
    uint256 _count
) external onlyOwner {
    require(_count > 0, "count must be nonzero");

    (uint256 memPtr, bytes32 _batchHash) = _loadBatchHeader(_batchHeader);

    // check batch hash
    uint256 _batchIndex = BatchHeaderV0Codec.batchIndex(memPtr);
    _batchHash = keccak256(
        abi.encodePacked(
            _batchHash,
            committedBatchStores[_batchIndex].blobVersionedhash
        )
    );
    require(
        committedBatchStores[_batchIndex].batchHash == _batchHash,
        "incorrect batch hash"
    );
    // make sure no gap is left when reverting from the ending to the beginning.
    require(
        committedBatchStores[_batchIndex + _count].batchHash == bytes32(0),
        "reverting must start from the ending"
    );

    // check finalization
    require(
        _batchIndex > lastFinalizedBatchIndex,
        "can only revert unfinalized batch"
```

Recommendations:

ExVul Web3 Labs recommends setting loop limit.

Result: Confirmed

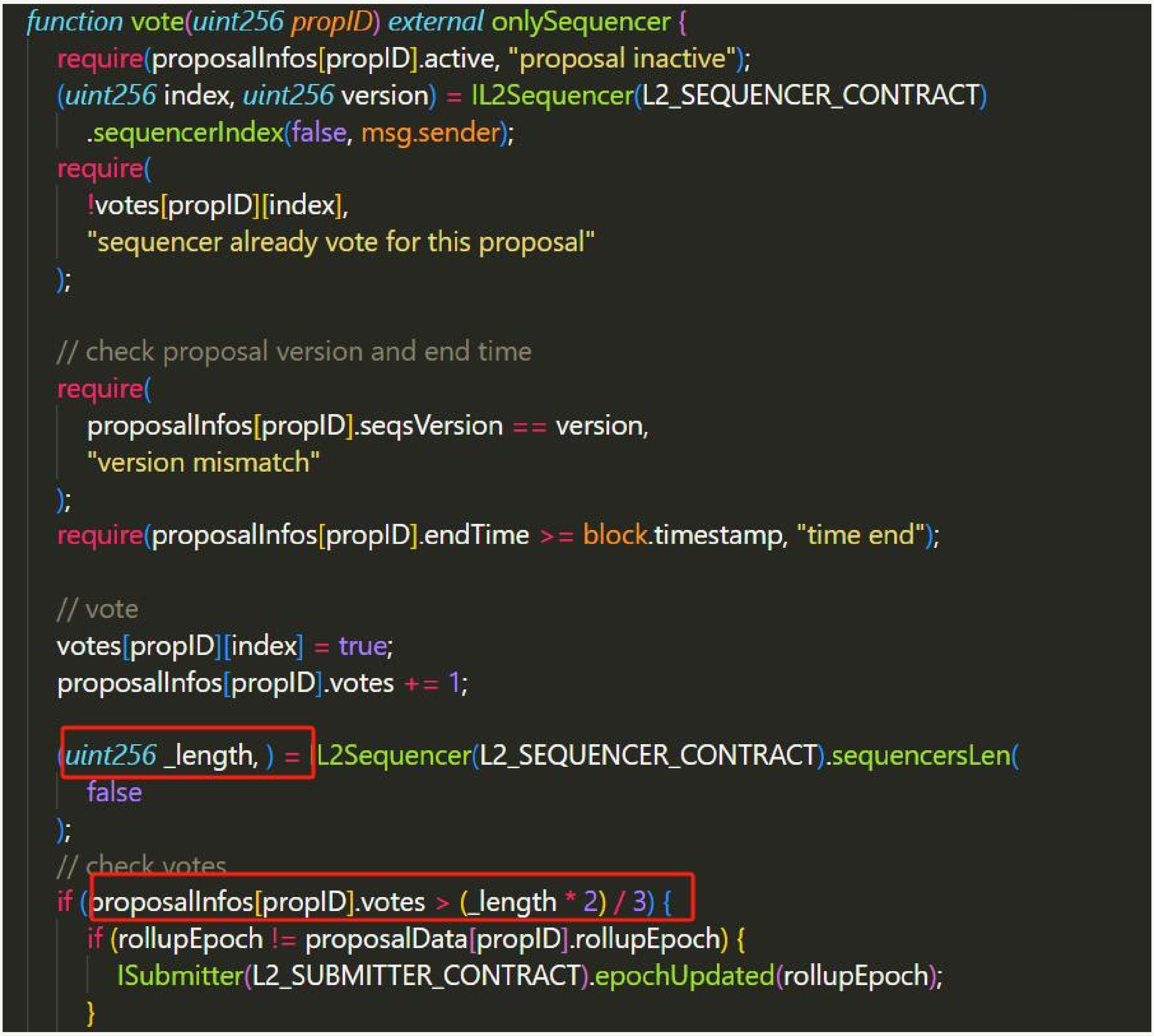
Fix Result: Ignored

3.4 Check the number of verifiers

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

Description:

As shown in the figure below, in the vote function, the _lenght of the number of signers obtained should be greater than or equal to 3. Otherwise, when judging the number of votes below, only one person needs to vote to complete the judgment.



```
function vote(uint256 propID) external onlySequencer {
    require(proposalInfos[propID].active, "proposal inactive");
    (uint256 index, uint256 version) = IL2Sequencer(L2_SEQUENCER_CONTRACT)
        .sequencerIndex(false, msg.sender);
    require(
        !votes[propID][index],
        "sequencer already vote for this proposal"
    );

    // check proposal version and end time
    require(
        proposalInfos[propID].seqsVersion == version,
        "version mismatch"
    );
    require(proposalInfos[propID].endTime >= block.timestamp, "time end");

    // vote
    votes[propID][index] = true;
    proposalInfos[propID].votes += 1;

    (uint256 _length, ) = IL2Sequencer(L2_SEQUENCER_CONTRACT).sequencersLen(
        false
    );
    // check votes
    if (proposalInfos[propID].votes > (_length * 2) / 3) {
        if (rollupEpoch != proposalData[propID].rollupEpoch) {
            ISubmitter(L2_SUBMITTER_CONTRACT).epochUpdated(rollupEpoch);
        }
```

Figure 3.4.1 Gov.sol

Recommendations:

ExVul Web3 Labs recommends adding judgment of the number of verifiers.

Result: Confirmed

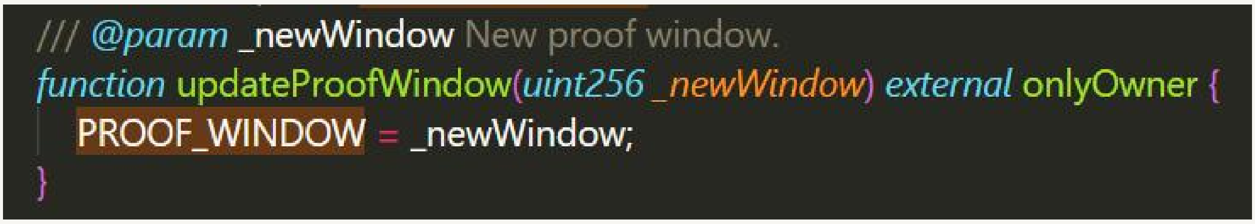
Fix Result: Ignored

3.5 The updateProofWindow function affects the results of prove

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

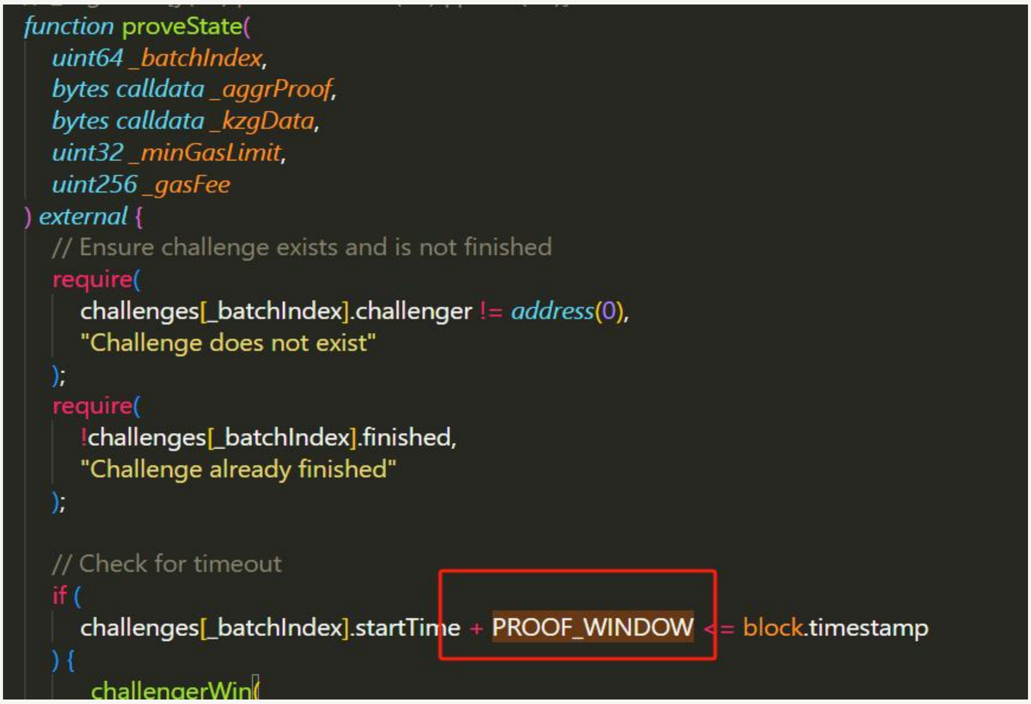
Description:

As shown in the figure below, The contract owner can call the updateProofWindow function at any time to update the proof_window variable. The modification will affect the challenge verification. If PROOF_WINDOW is zero, there will be no challenge period and no one will be able to challenge. In addition, if PROOF_WINDOW is set to a small value or zero, it may cause legal challenges or proofs to be judged to have timed out at inappropriate times.
When updating, there is no check whether the updated value is equal to the original value.



```
/// @param _newWindow New proof window.
function updateProofWindow(uint256 _newWindow) external onlyOwner {
    PROOF_WINDOW = _newWindow;
}
```

Figure 3.5.1 Staking.sol



```
function proveState(
    uint64 _batchIndex,
    bytes calldata _aggrProof,
    bytes calldata _kzgData,
    uint32 _minGasLimit,
    uint256 _gasFee
) external {
    // Ensure challenge exists and is not finished
    require(
        challenges[_batchIndex].challenger != address(0),
        "Challenge does not exist"
    );
    require(
        !challenges[_batchIndex].finished,
        "Challenge already finished"
    );

    // Check for timeout
    if (
        challenges[_batchIndex].startTime + PROOF_WINDOW <= block.timestamp
    ) {
        challengerWin(
```

Figure 3.5.2 Staking.sol

Recommendations:

ExVul Web3 Labs recommends the contract owner using multi-signature wallet management and PROOF_WINDOW is not zero.

Result: Confirmed

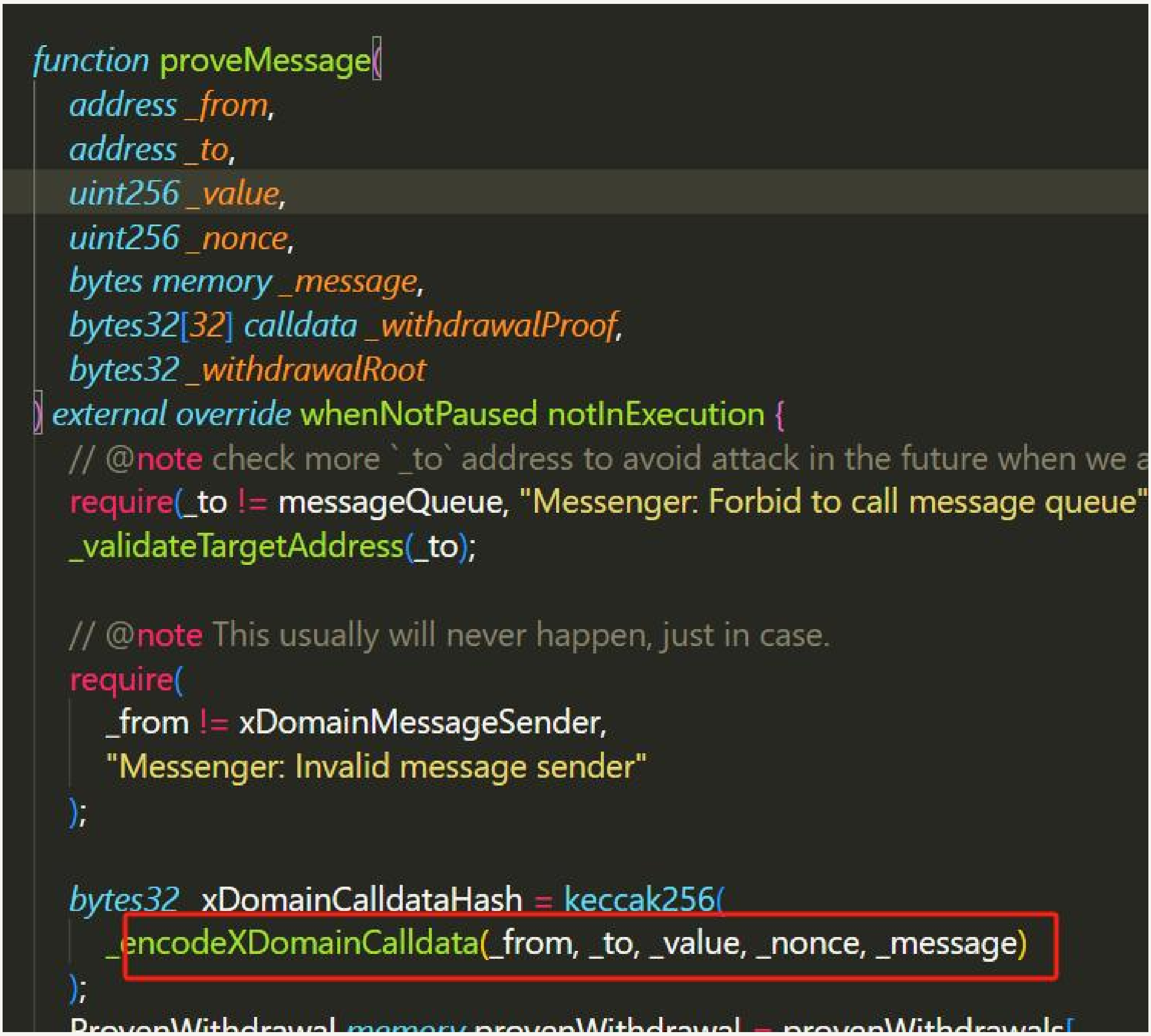
Fix Result: Ignored

3.6 Cross-chain message replay

ID:	NVE-006	**Location:**	L1CrossDomainMessenger.sol
Severity:	Medium	**Category:**	Business Issues
Likelihood:	Low	**Impact:**	Medium

Description:

As shown in the figure below, in the proveMessage function, _xDomainCalldataHash does not have chainid parameter verification, which may lead to cross-chain message replay.



```
function proveMessage(
    address _from,
    address _to,
    uint256 _value,
    uint256 _nonce,
    bytes memory _message,
    bytes32[32] calldata _withdrawalProof,
    bytes32 _withdrawalRoot
) external override whenNotPaused notInExecution {
    // @note check more `_to` address to avoid attack in the future when we a
    require(_to != messageQueue, "Messenger: Forbid to call message queue"
    _validateTargetAddress(_to);

    // @note This usually will never happen, just in case.
    require(
        _from != xDomainMessageSender,
        "Messenger: Invalid message sender"
    );

    bytes32 _xDomainCalldataHash = keccak256(
        _encodeXDomainCalldata(_from, _to, _value, _nonce, _message)
    );
    ProvenWithdrawal memory provenWithdrawal = provenWithdrawals[
```

Figure 3.6.1 L1CrossDomainMessenger.sol

Recommendations:

ExVul Web3 Labs recommends adding chainid check.

Result: Confirmed

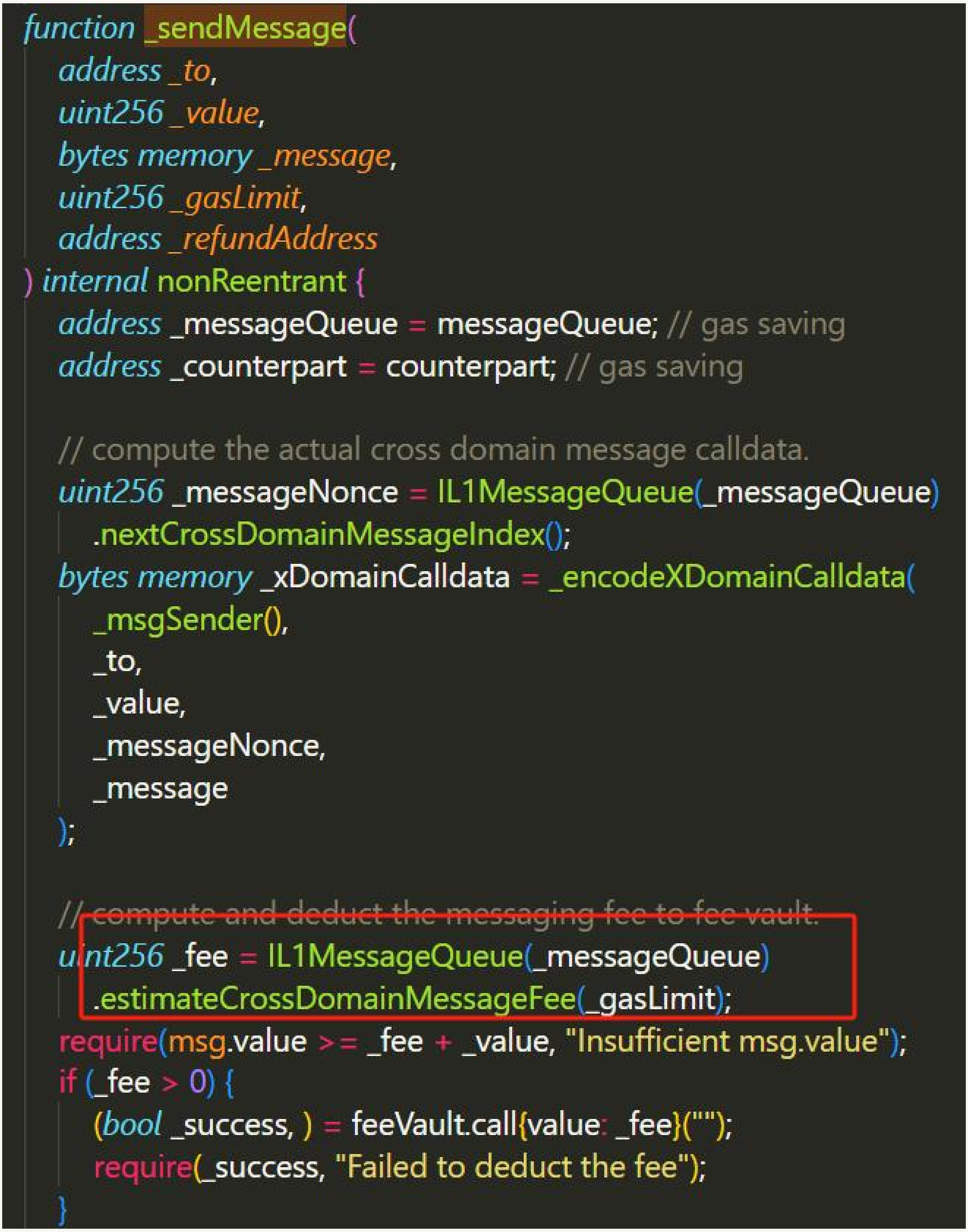
Fix Result: Ignored

3.7 The _fee condition may be bypassed

ID:	NVE-007	**Location:**	L1CrossDomainMessenger.sol
Severity:	Medium	**Category:**	Business Issues
Likelihood:	Low	**Impact:**	Medium

Description:

As shown in the figure below, The _sendMessage function is used to send messages. If the l2BaseFee set in L1MessageQueue is zero, theoretically the caller gasLimit enters any value and l2BaseFee is zero. Calculate _gasLimit * l2BaseFee through estimateCrossDomainMessageFee and the return value _fee is zero. When _fee is zero, you can bypass the fee check msg.value >= _fee + _value, and enter a reasonable value for _gasLimit to bypass the verification in the appendCrossDomainMessage function.



```
function _sendMessage(
    address _to,
    uint256 _value,
    bytes memory _message,
    uint256 _gasLimit,
    address _refundAddress
) internal nonReentrant {
    address _messageQueue = messageQueue; // gas saving
    address _counterpart = counterpart; // gas saving

    // compute the actual cross domain message calldata.
    uint256 _messageNonce = IL1MessageQueue(_messageQueue)
        .nextCrossDomainMessageIndex();
    bytes memory _xDomainCalldata = _encodeXDomainCalldata(
        _msgSender(),
        _to,
        _value,
        _messageNonce,
        _message
    );

    // compute and deduct the messaging fee to fee vault.
    uint256 _fee = IL1MessageQueue(_messageQueue)
        .estimateCrossDomainMessageFee(_gasLimit);
    require(msg.value >= _fee + _value, "Insufficient msg.value");
    if (_fee > 0) {
        (bool _success, ) = feeVault.call{value: _fee}("");
        require(_success, "Failed to deduct the fee");
    }
```

Figure 3.7.1 L1CrossDomainMessenger.sol

Recommendations:

ExVul Web3 Labs recommends the contract owner setting l2BaseFee to a reasonable value and l2BaseFee cannot be zero.

Result: Confirmed

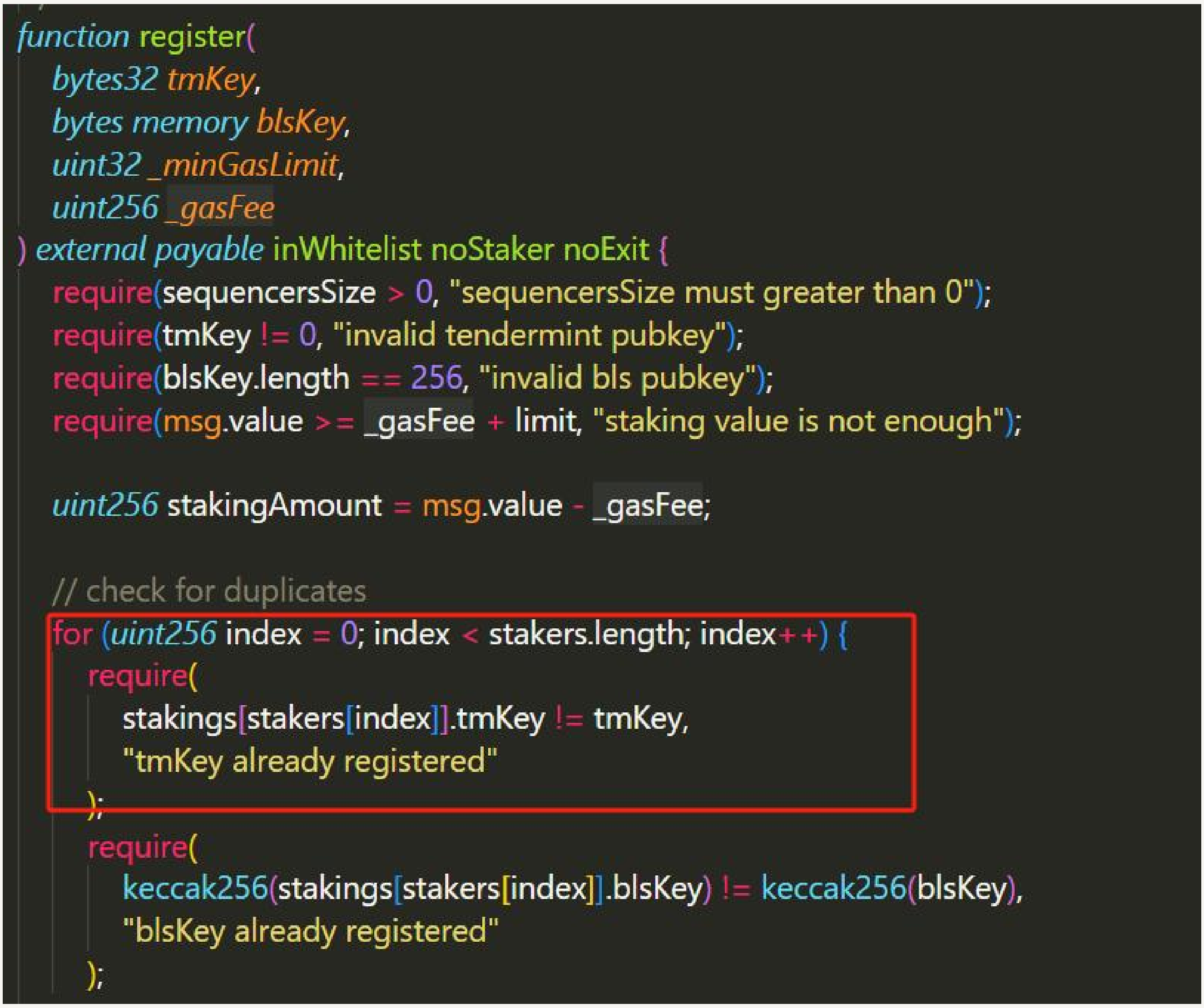
Fix Result: Ignored

3.8 Gas may exceed limits

ID:	NVE-008	**Location:**	Staking.sol
Severity:	Medium	**Category:**	Business Issues
Likelihood:	Low	**Impact:**	Medium

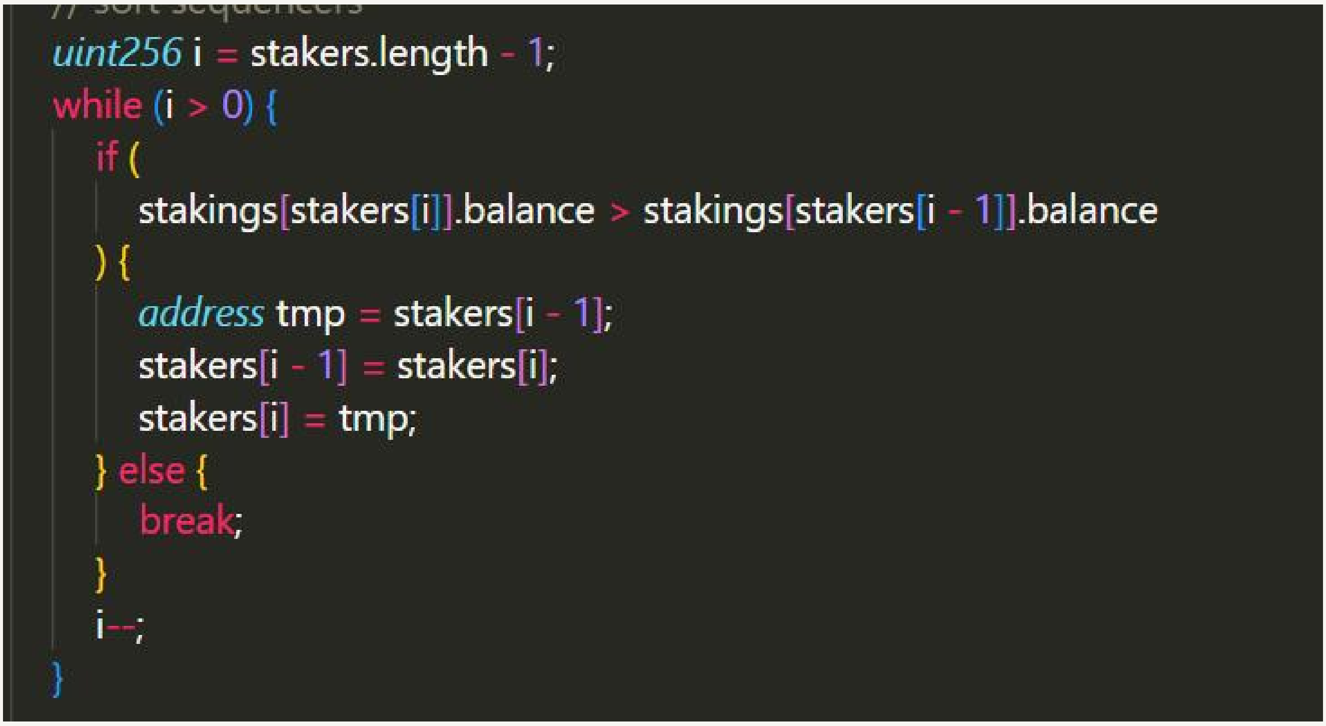
Description:

As shown in the figure below, there are three loop operations in the register function. When the length of the stakers array is large, the gas limit may be exceeded.



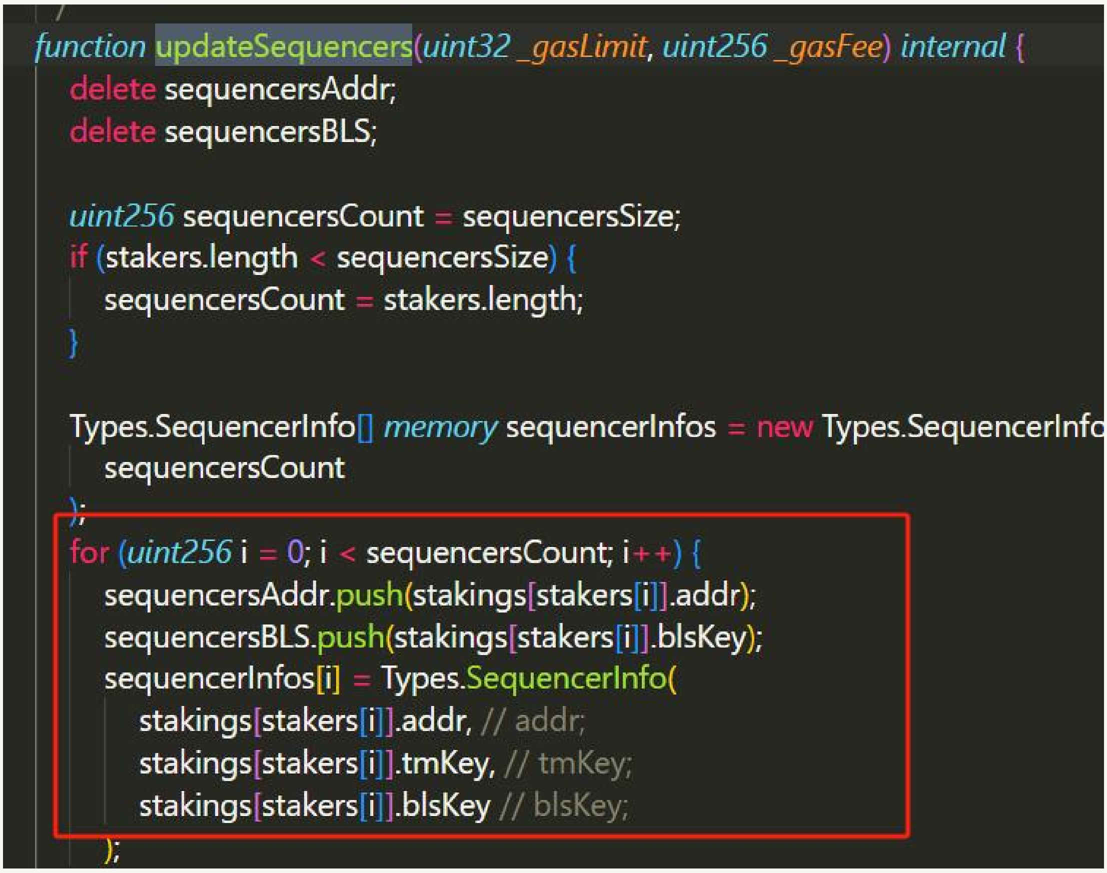
```solidity
function register(
    bytes32 tmKey,
    bytes memory blsKey,
    uint32 _minGasLimit,
    uint256 _gasFee
) external payable inWhitelist noStaker noExit {
    require(sequencersSize > 0, "sequencersSize must greater than 0");
    require(tmKey != 0, "invalid tendermint pubkey");
    require(blsKey.length == 256, "invalid bls pubkey");
    require(msg.value >= _gasFee + limit, "staking value is not enough");

    uint256 stakingAmount = msg.value - _gasFee;

    // check for duplicates
    for (uint256 index = 0; index < stakers.length; index++) {
        require(
            stakings[stakers[index]].tmKey != tmKey,
            "tmKey already registered"
        );
        require(
            keccak256(stakings[stakers[index]].blsKey) != keccak256(blsKey),
            "blsKey already registered"
        );
```

Figure 3.8.1 Staking.sol



```
// sort sequencers
uint256 i = stakers.length - 1;
while (i > 0) {
    if (
        stakings[stakers[i]].balance > stakings[stakers[i - 1]].balance
    ) {
        address tmp = stakers[i - 1];
        stakers[i - 1] = stakers[i];
        stakers[i] = tmp;
    } else {
        break;
    }
    i--;
}
```

Figure 3.8.2 Staking.sol



```
function updateSequencers(uint32 _gasLimit, uint256 _gasFee) internal {
    delete sequencersAddr;
    delete sequencersBLS;

    uint256 sequencersCount = sequencersSize;
    if (stakers.length < sequencersSize) {
        sequencersCount = stakers.length;
    }

    Types.SequencerInfo[] memory sequencerInfos = new Types.SequencerInfo
        sequencersCount
    );
    for (uint256 i = 0; i < sequencersCount; i++) {
        sequencersAddr.push(stakings[stakers[i]].addr);
        sequencersBLS.push(stakings[stakers[i]].blsKey);
        sequencerInfos[i] = Types.SequencerInfo(
            stakings[stakers[i]].addr, // addr;
            stakings[stakers[i]].tmKey, // tmKey;
            stakings[stakers[i]].blsKey // blsKey;
        );
```

Figure 3.8.3 Staking.sol

Recommendations:

ExVul Web3 Labs recommends setting loop limit.

Result: Confirmed

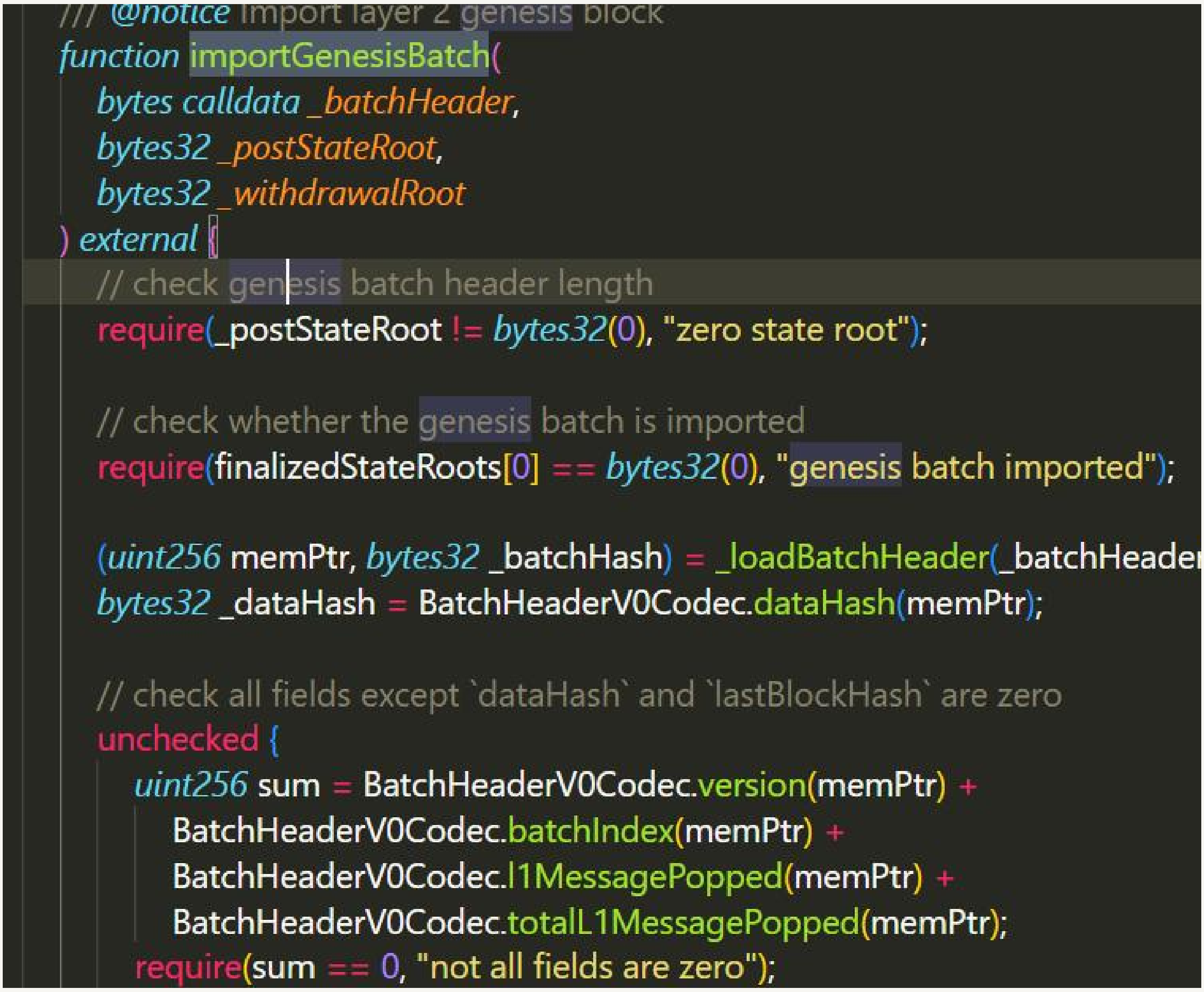
Fix Result: Ignored

3.9 importGenesisBatch function init

ID:	NVE-009	**Location:**	Rollup.sol
Severity:	Low	**Category:**	Business Issues
Likelihood:	Low	**Impact:**	Low

Description:

As shown in the figure below, The importGenesisBatch function can only be called once, but the function is public. If it is not called correctly in time after the contract is deployed, the contract data will be destroyed.



```
/// @notice Import layer 2 genesis block
function importGenesisBatch(
    bytes calldata _batchHeader,
    bytes32 _postStateRoot,
    bytes32 _withdrawalRoot
) external {
    // check genesis batch header length
    require(_postStateRoot != bytes32(0), "zero state root");

    // check whether the genesis batch is imported
    require(finalizedStateRoots[0] == bytes32(0), "genesis batch imported");

    (uint256 memPtr, bytes32 _batchHash) = _loadBatchHeader(_batchHeader
    bytes32 _dataHash = BatchHeaderV0Codec.dataHash(memPtr);

    // check all fields except `dataHash` and `lastBlockHash` are zero
    unchecked {
        uint256 sum = BatchHeaderV0Codec.version(memPtr) +
            BatchHeaderV0Codec.batchIndex(memPtr) +
            BatchHeaderV0Codec.l1MessagePopped(memPtr) +
            BatchHeaderV0Codec.totalL1MessagePopped(memPtr);
        require(sum == 0, "not all fields are zero");
```

Figure 3.9.1Rollup.sol

Recommendations:

ExVul Web3 Labs recommends initializing the function in time or setting the calling permission.

Result: Confirmed

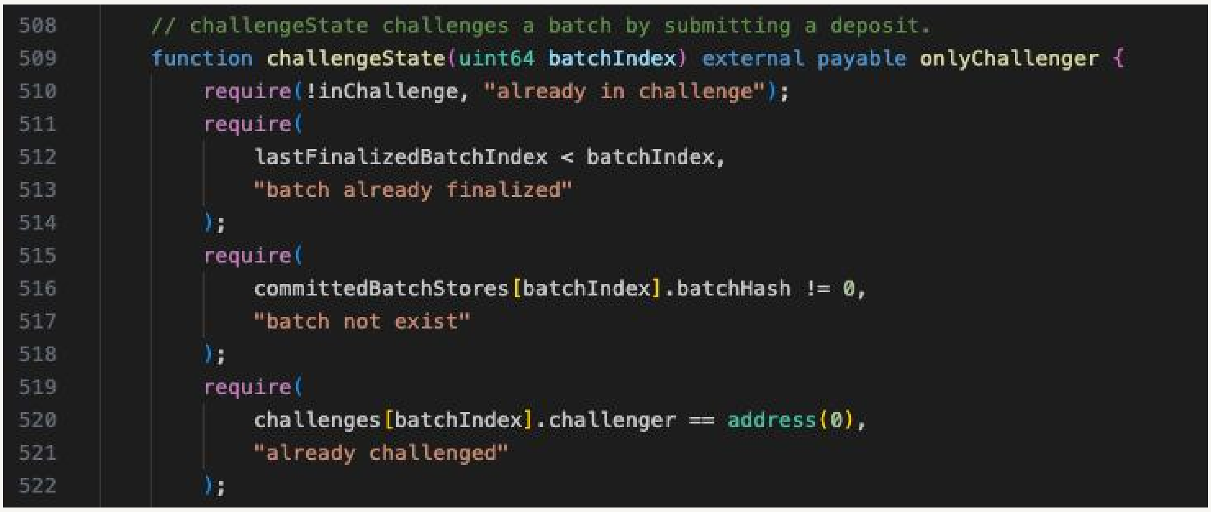
Fix Result: Ignored

3.10 inChallenge is always false

ID:	NVE-010	**Location:**	Rollup.sol
Severity:	Low	**Category:**	Business Issues
Likelihood:	Low	**Impact:**	Low

Description:

As shown in the figure below, The challengeState function is used to initiate a challenge. When initiating a challenge, !inChallenge will be judged, but the inChallenge variable is always false. There is no code to modify the state of the inChallenge variable, so multiple challenges can be initiated.
If there are too many challenges, the challenge period of the entire project may be extended indefinitely. This will result in a longer time for final confirmation of the batch



```
    // challengeState challenges a batch by submitting a deposit.
    function challengeState(uint64 batchIndex) external payable onlyChallenger {
        require(!inChallenge, "already in challenge");
        require(
            lastFinalizedBatchIndex < batchIndex,
            "batch already finalized"
        );
        require(
            committedBatchStores[batchIndex].batchHash != 0,
            "batch not exist"
        );
        require(
            challenges[batchIndex].challenger == address(0),
            "already challenged"
        );
```

Figure 3.10.1 Rollup.sol

Recommendations:

ExVul Web3 Labs recommends modifying the code logic.

Result: Confirmed

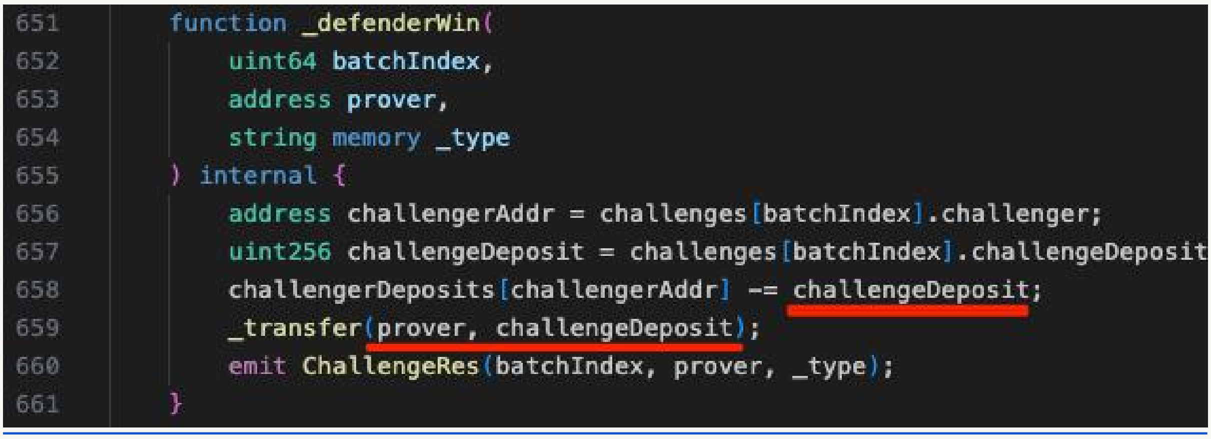
Fix Result: Ignored

3.11 Fund allocation does not match documentation

ID:	NVE-011	**Location:**	Rollup.sol
Severity:	Low	**Category:**	Business Issues
Likelihood:	Low	**Impact:**	Low

Description:

As shown in the figure below, The transfer of all challenger funds to the prover in the _defenderWin function is inconsistent with the official documentation. The official description: If the challenge fails, it means that there is no problem with the challenged batch, and the verifier's pledged funds will be completely seized. 80% of the seized funds will be given to the DAO vault and 20% will be paid to the questioning Sequencer.
doc：https://docs.morphl2.io/docs/how-morph-works/responsive-validity-proof/how-rvp-applied



```
function _defenderWin(
    uint64 batchIndex,
    address prover,
    string memory _type
) internal {
    address challengerAddr = challenges[batchIndex].challenger;
    uint256 challengeDeposit = challenges[batchIndex].challengeDeposit
    challengerDeposits[challengerAddr] -= challengeDeposit;
    _transfer(prover, challengeDeposit);
    emit ChallengeRes(batchIndex, prover, _type);
}
```

Figure 3.11.1 Rollup.sol

Recommendations:

ExVul Web3 Labs recommends checking the code logic.

Result: Confirmed

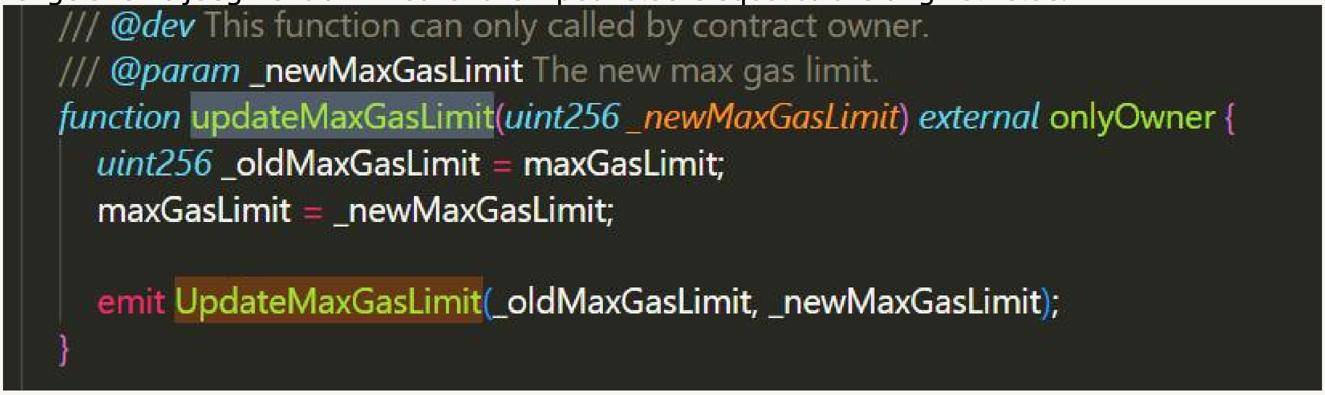
Fix Result: Ignored

3.12 updateMaxGasLimit function

ID:	NVE-012	**Location:**	L1MessageQueueWithGasPriceOracle.sol
Severity:	Medium	**Category:**	Business Issues
Likelihood:	Low	**Impact:**	Medium

Description:

As shown in the figure below, When updating the _newMaxGasLimit variable, there is no limit range and no judgment on whether the input value is equal to the original value.



```
/// @dev This function can only called by contract owner.
/// @param _newMaxGasLimit The new max gas limit.
function updateMaxGasLimit(uint256 _newMaxGasLimit) external onlyOwner {
    uint256 _oldMaxGasLimit = maxGasLimit;
    maxGasLimit = _newMaxGasLimit;

    emit UpdateMaxGasLimit(_oldMaxGasLimit, _newMaxGasLimit);
}
```

Figure 3.12.1 L1MessageQueueWithGasPriceOracle.sol

Recommendations:

ExVul Web3 Labs recommends adding corresponding judgment.

Result: Confirmed

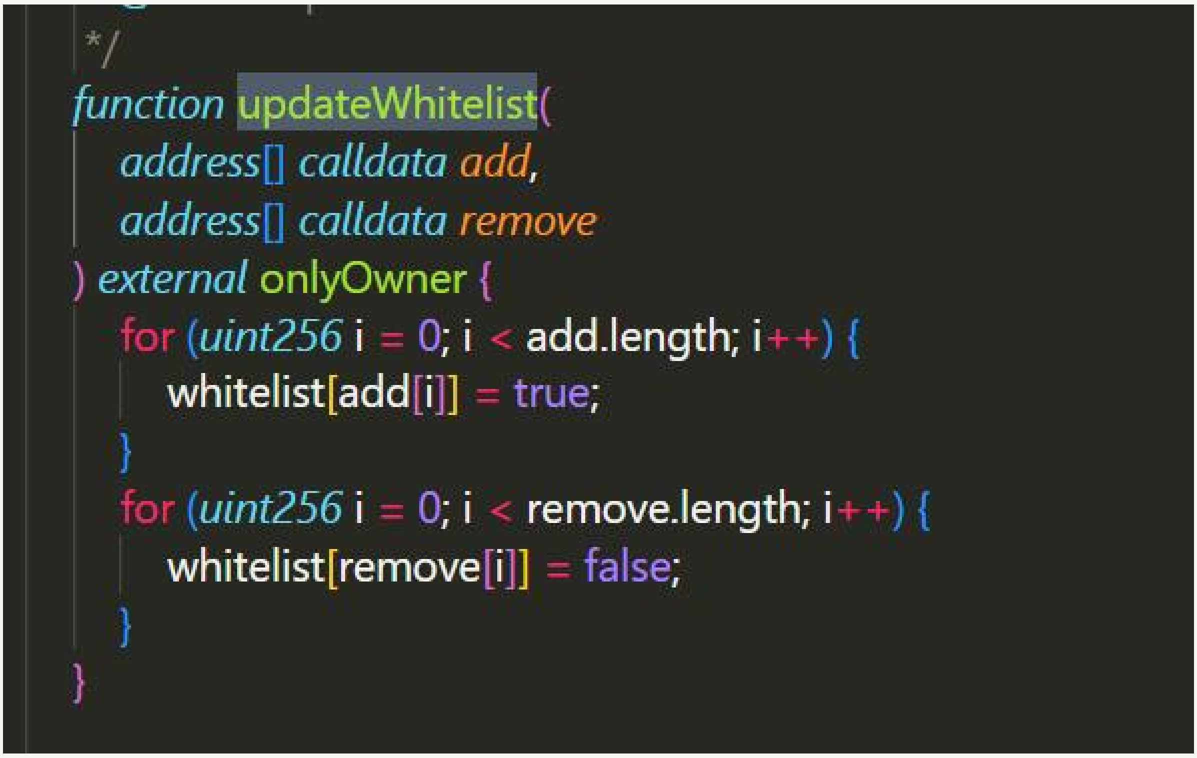
Fix Result: Ignored

3.13 add and remove list checks

ID:	NVE-013	**Location:**	Staking.sol
Severity:	Medium	**Category:**	Business Issues
Likelihood:	Low	**Impact:**	Medium

Description:

As shown in the figure below, when the contract owner calls the updateWhitelist function, it needs to detect whether there are duplicate addresses in the add and remove lists, and whether these addresses have been added or removed.



```
*/
function updateWhitelist(
    address[] calldata add,
    address[] calldata remove
) external onlyOwner {
    for (uint256 i = 0; i < add.length; i++) {
        whitelist[add[i]] = true;
    }
    for (uint256 i = 0; i < remove.length; i++) {
        whitelist[remove[i]] = false;
    }
}
```

Figure 3.13.1 Staking.sol

Recommendations:

ExVul Web3 Labs recommends adding corresponding judgment.

Result: Confirmed

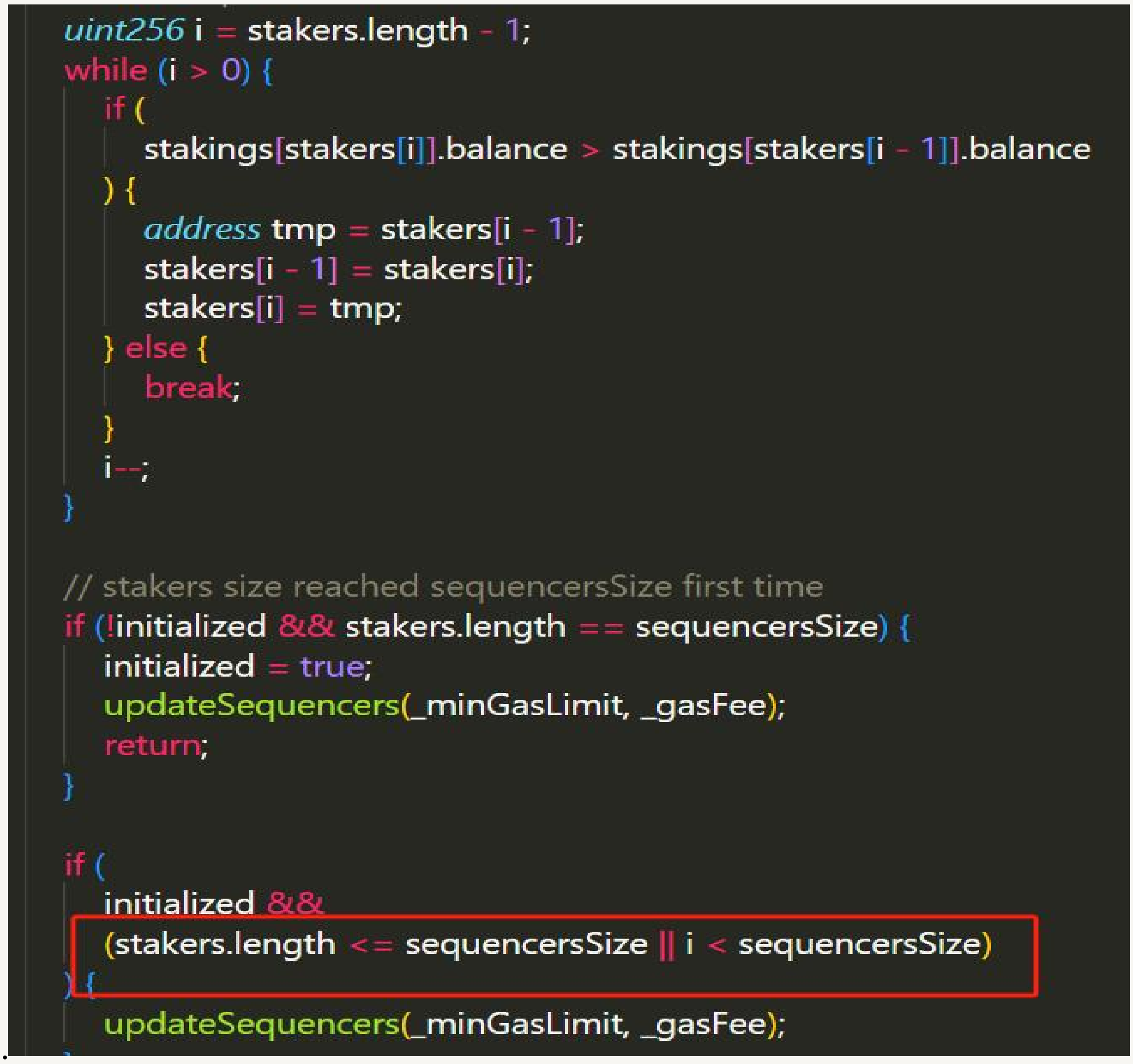
Fix Result: Ignored

3.14 Code logic redundancy

ID:	NVE-014	**Location:**	Staking.sol
Severity:	Low	**Category:**	Business Issues
Likelihood:	Low	**Impact:**	Low

Description:

As shown in the figure below, in the register function, the variable i is always smaller than stakers.length, when stakers.length<=sequencersSize, the i<sequencersSize always holds, so you can delete the stakers.length<=sequencersSize.



```
uint256 i = stakers.length - 1;
while (i > 0) {
    if (
        stakings[stakers[i]].balance > stakings[stakers[i - 1]].balance
    ) {
        address tmp = stakers[i - 1];
        stakers[i - 1] = stakers[i];
        stakers[i] = tmp;
    } else {
        break;
    }
    i--;
}

// stakers size reached sequencersSize first time
if (!initialized && stakers.length == sequencersSize) {
    initialized = true;
    updateSequencers(_minGasLimit, _gasFee);
    return;
}

if (
    initialized &&
    (stakers.length <= sequencersSize || i < sequencersSize)
) {
    updateSequencers(_minGasLimit, _gasFee);
```

Figure 3.14.1 Staking.sol

Recommendations:

ExVul Web3 Labs recommends removing redundant code.

Result: Confirmed

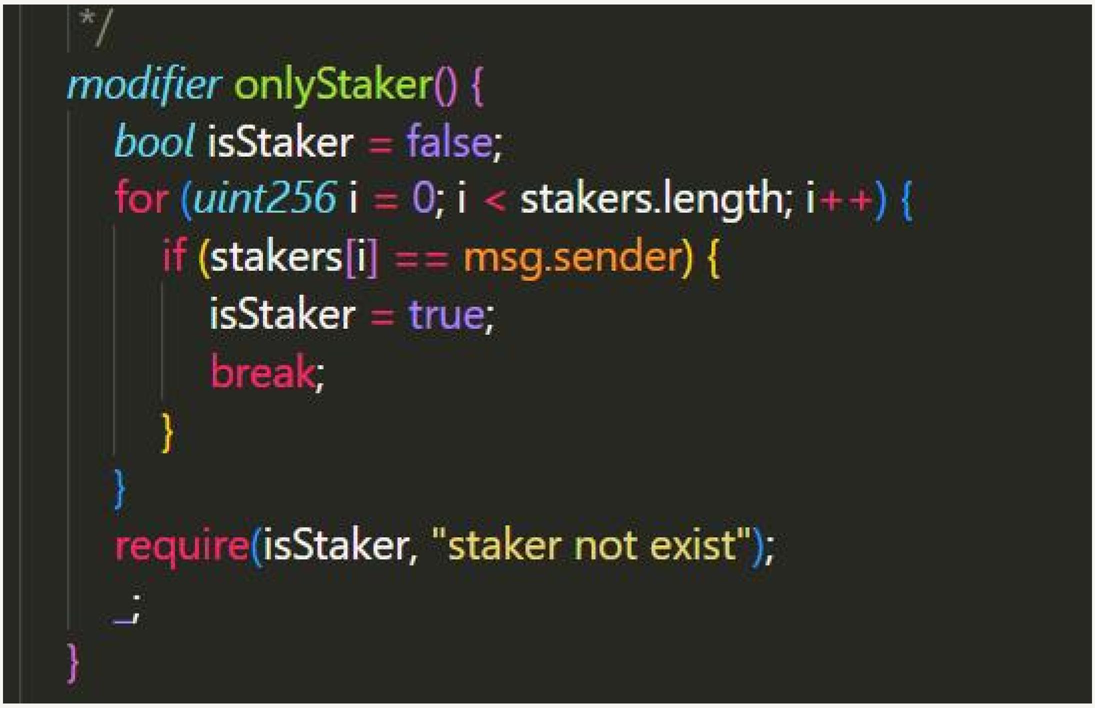
Fix Result: Ignored

3.15 error log message

ID:	NVE-015	**Location:**	Staking.sol
Severity:	Informational	**Category:**	Business Issues
Likelihood:	Informational	**Impact:**	Informational

Description:

As shown in the figure below, The noStaker modifier error message does not match the definition. It should be something like no staker.



```
*/
modifier onlyStaker() {
    bool isStaker = false;
    for (uint256 i = 0; i < stakers.length; i++) {
        if (stakers[i] == msg.sender) {
            isStaker = true;
            break;
        }
    }
    require(isStaker, "staker not exist");
    _;
}
```

Figure 3.15.1 Staking.sol

Recommendations:

ExVul Web3 Labs recommends adding corresponding judgment.

Result: Confirmed

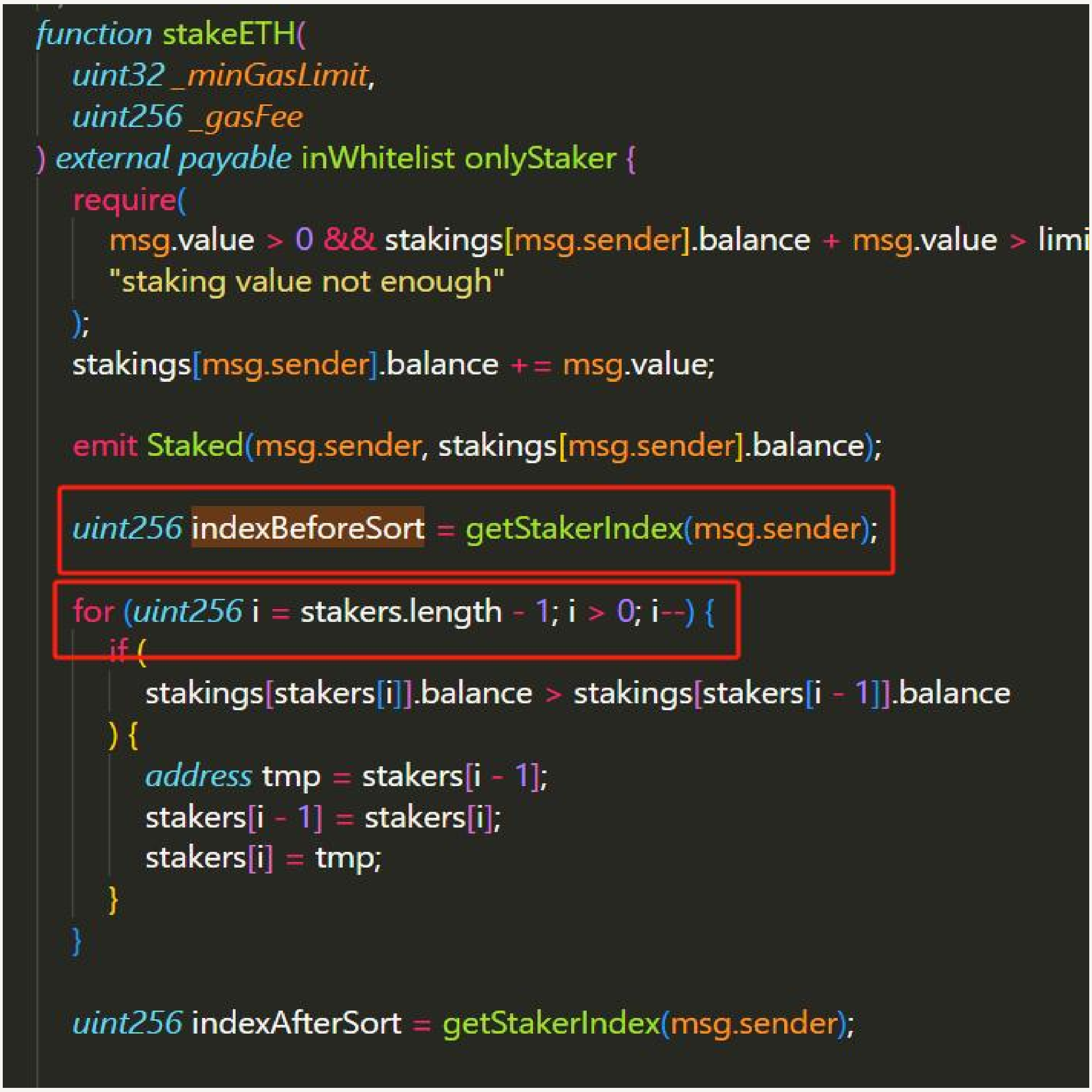
Fix Result: Ignored

3.16 Gas optimization

ID:	NVE-016	Location:	Staking.sol
Severity:	Informational	Category:	Business Issues
Likelihood:	Informational	Impact:	Informational

Description:

As shown in the figure below, in the stakeETH function, when looping, you should start looping directly from "i" equal to "indexBeforSort" instead of looping from the end of the array.



```
function stakeETH(
    uint32 _minGasLimit,
    uint256 _gasFee
) external payable inWhitelist onlyStaker {
    require(
        msg.value > 0 && stakings[msg.sender].balance + msg.value > limi
        "staking value not enough"
    );
    stakings[msg.sender].balance += msg.value;

    emit Staked(msg.sender, stakings[msg.sender].balance);

    uint256 indexBeforeSort = getStakerIndex(msg.sender);

    for (uint256 i = stakers.length - 1; i > 0; i--) {
        if (
            stakings[stakers[i]].balance > stakings[stakers[i - 1]].balance
        ) {
            address tmp = stakers[i - 1];
            stakers[i - 1] = stakers[i];
            stakers[i] = tmp;
        }
    }

    uint256 indexAfterSort = getStakerIndex(msg.sender);
```

Figure 3.16.1 Staking.sol

Recommendations:

ExVul Web3 Labs recommends modifying code logic.

Result: Confirmed

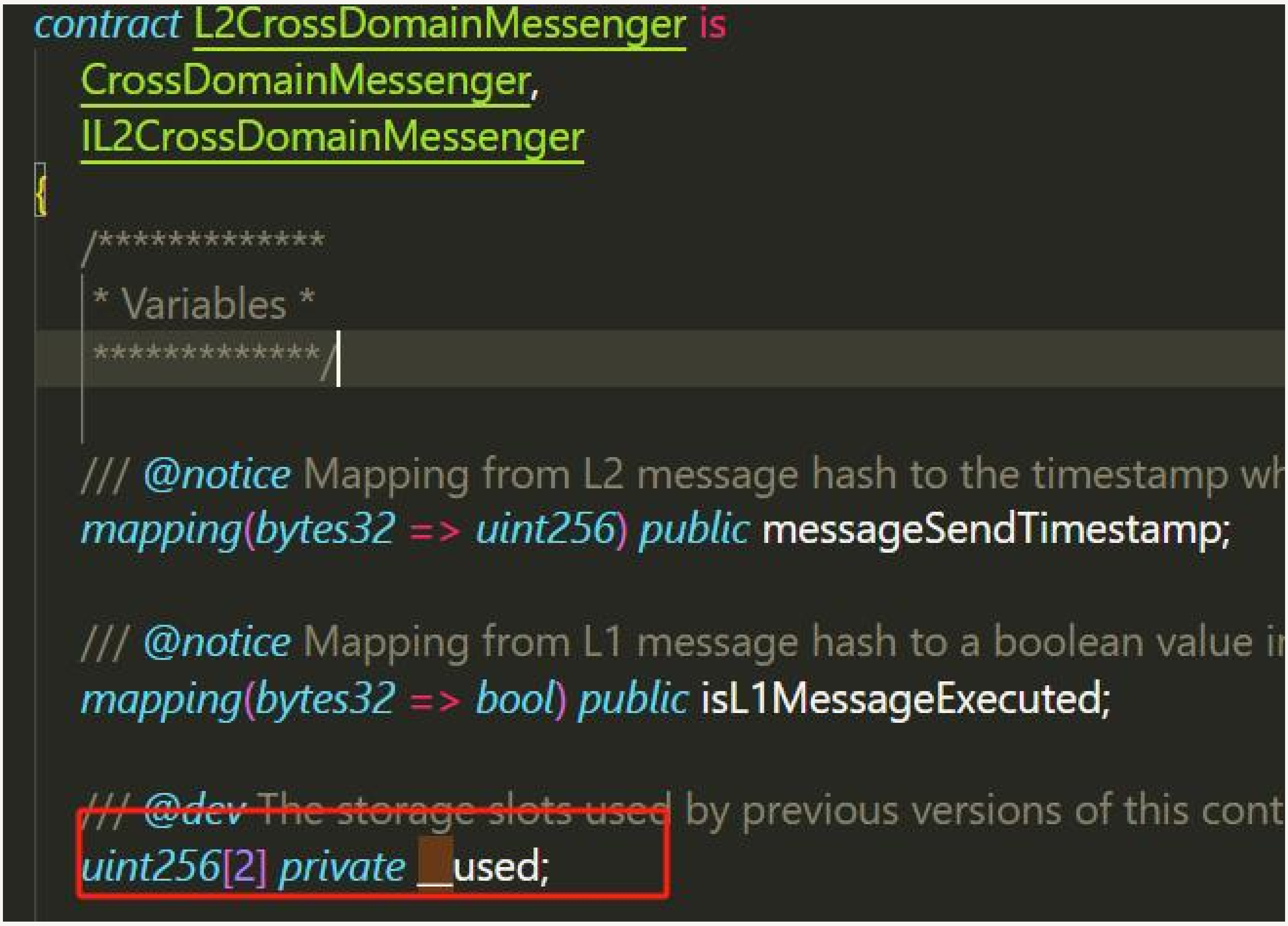
Fix Result: Ignored

3.17 Unused variables

ID:	NVE-017	**Location:**	L2CrossDomainMessenger.sol, CrossDomainMessenger.sol
Severity:	Informational	**Category:**	Business Issues
Likelihood:	Informational	**Impact:**	Informational

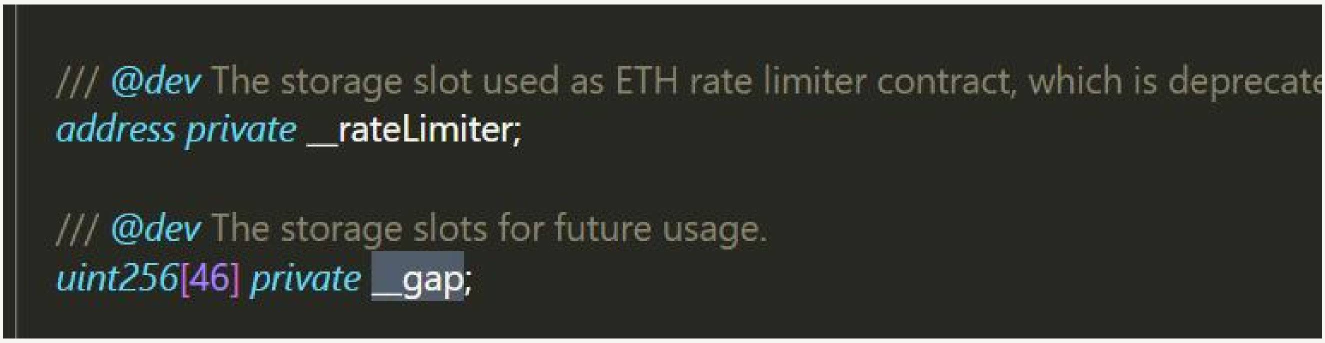
Description:

As shown in the figure below, the __used,__rateLimiter and __gap variables are private attributes and are not used in the contract.



```
contract L2CrossDomainMessenger is
    CrossDomainMessenger,
    IL2CrossDomainMessenger
{
    /*************
     * Variables *
     *************/

    /// @notice Mapping from L2 message hash to the timestamp wh
    mapping(bytes32 => uint256) public messageSendTimestamp;

    /// @notice Mapping from L1 message hash to a boolean value i
    mapping(bytes32 => bool) public isL1MessageExecuted;

    /// @dev The storage slots used by previous versions of this cont
    uint256[2] private __used;
```

Figure 3.17.1 L2CrossDomainMessenger.sol



```
/// @dev The storage slot used as ETH rate limiter contract, which is deprecate
address private __rateLimiter;

/// @dev The storage slots for future usage.
uint256[46] private __gap;
```

Figure 3.17.2 CrossDomainMessenger.sol

Recommendations:

ExVul Web3 Labs recommends removing redundant code.

Result: Confirmed

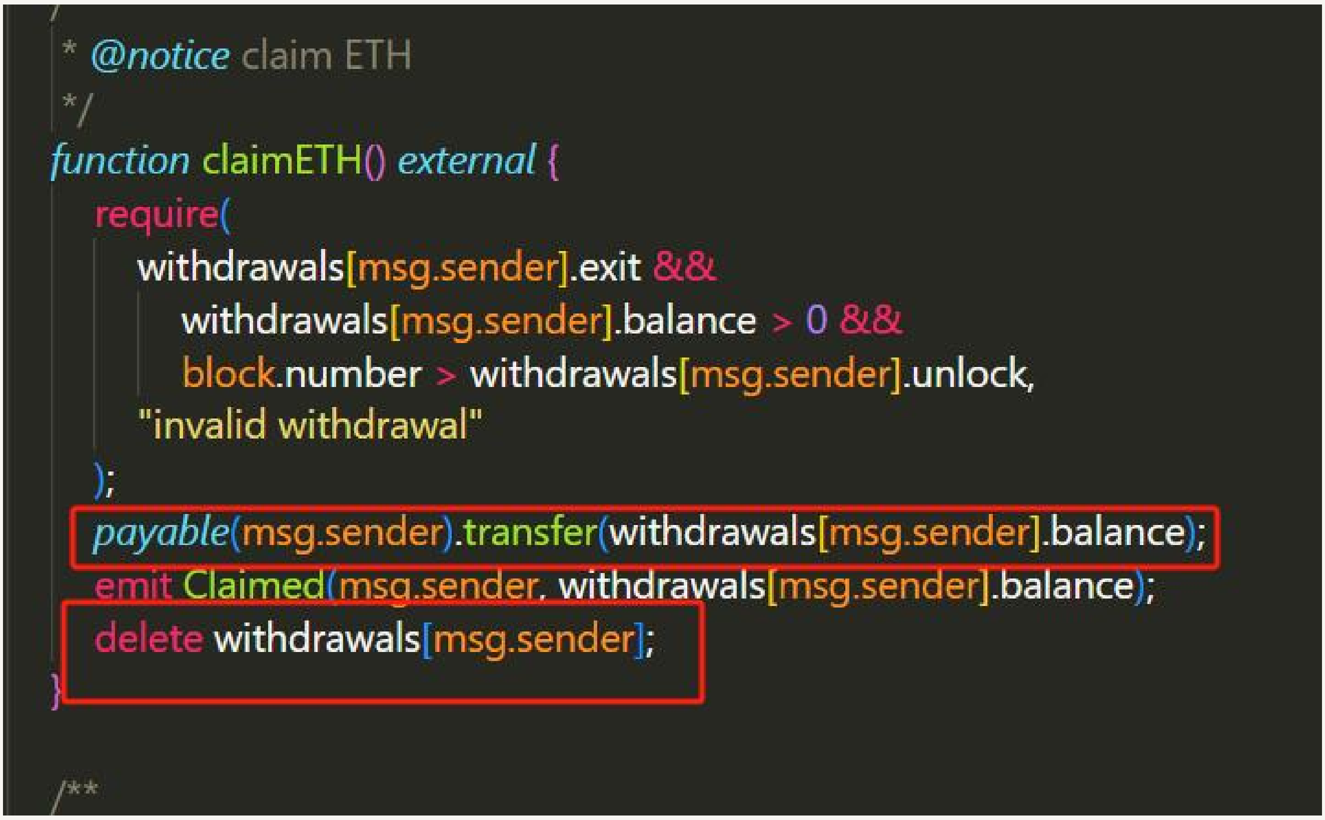
Fix Result: Ignored

3.18 claimEth function does not follow the CEI rule

ID:	NVE-018	**Location:**	Stake.sol
Severity:	Informational	**Category:**	Business Issues
Likelihood:	Informational	**Impact:**	Informational

Description:

As shown in the figure below, The claimEth function does not follow the CEI rules and transfers the tokens first and then updates the information.



```
 * @notice claim ETH
 */
function claimETH() external {
    require(
        withdrawals[msg.sender].exit &&
            withdrawals[msg.sender].balance > 0 &&
            block.number > withdrawals[msg.sender].unlock,
        "invalid withdrawal"
    );
    payable(msg.sender).transfer(withdrawals[msg.sender].balance);
    emit Claimed(msg.sender, withdrawals[msg.sender].balance);
    delete withdrawals[msg.sender];
}

/**
```

Figure 3.18.1 Stake.sol

Recommendations:

ExVul Web3 Labs recommends updating the information before transferring tokens.

Result: Confirmed

Fix Result: Ignored

4. CONCLUSION

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