



# **BLOCKCHAIN AUDIT REPORT**

**QuarkChain**

**JUNE 2025**

## Contents

<b>1. EXECUTIVE SUMMARY</b>	<b>4</b>
1.1 Methodology	4
<b>2. FINDINGS OVERVIEW</b>	<b>7</b>
2.1 Project Info And Contract Address	7
2.2 Summary	7
2.3 Key Findings	8
<b>3. DETAILED DESCRIPTION OF FINDINGS</b>	<b>9</b>
3.1 Missing Zero Check in Operator Fee Refund Process	16
3.2 Nil Pointer Dereference in distributeGas Function	9
3.3 Missing Zero Address Validation in batchMint	10
3.4 Empty Array Handling Vulnerability in SoulGasToken	11
3.5 Lack of Events for Critical Operations	13
3.6 Missing Zero Address Validation in addMinters Function	14
3.7 Fixed Timeout for Blob Upload	15
<b>4. CONCLUSION</b>	<b>17</b>
<b>5. APPENDIX</b>	<b>18</b>
5.1 Basic Coding Assessment	18
5.1.1 Apply Verification Control	18
5.1.2 Authorization Access Control	18
5.1.3 Forged Transfer Vulnerability	18
5.1.4 Transaction Rollback Attack	18
5.1.5 Transaction Block Stuffing Attack	19
5.1.6 Soft Fail Attack Assessment	19
5.1.7 Hard Fail Attack Assessment	19
5.1.8 Abnormal Memo Assessment	19
5.1.9 Abnormal Resource Consumption	19
5.1.10 Random Number Security	20
5.2 Advanced Code Scrutiny	20
5.2.1 Cryptography Security	20
5.2.2 Account Permission Control	20
5.2.3 Malicious Code Behavior	20
5.2.4 Sensitive Information Disclosure	21

5.2.5	System API	21
6.	<b>DISCLAIMER</b>	<b>22</b>
7.	<b>REFERENCES</b>	<b>23</b>

# 1. EXECUTIVE SUMMARY

Exvul Web3 Security was engaged by QUARKCHAIN to review Blockchain 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.

<b>Likelihood</b>	<b>High</b>	INFO	MEDIUM	HIGH	CRITICAL
	<b>Medium</b>	INFO	LOW	MEDIUM	HIGH
	<b>Low</b>	INFO	LOW	LOW	MEDIUM
		<b>Informational</b>	<b>Low</b>	<b>Medium</b>	<b>High</b>
		<b>IMPACT</b>			

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
P2P Communication Security	Connection Number Occupation Audit
	Eclipse Attack
	Packet Size Limit
	Node Communication Protocol Security
RPC Interface Security	RPC Sensitive Interface Permissions
	Traditional Web Security
	RPC Interface Security
Consensus Mechanism Security	Design Of Consensus Mechanism
	Implementation Of Consensus Verification
	Incentive Mechanism Audit
Transaction processing Security	Transaction Signature Logic
	Transaction Verification Logic
	Transaction Processing Logic
	Transaction Fee Setting
	Transaction Replay
Cryptography Security	Random Number Range And Probability Distribution
	Cryptographic Algorithm Implementation/Use
Wallet Module& Account Security Audit	Private Key / Mnemonic Word Storage Security
	Private Key / Mnemonic Word Usage Security
	Private key/mnemonic generation algorithm
Others Security Audit	Database Security

	Thread Security
	File Permission Security
	Historical Vulnerability Security

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

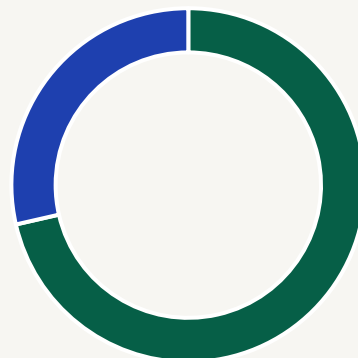
### 2.1 Project Info

Project Name	Audit Time	Language
QuarkChain	June 16 2025 – June 27 2025	Go;Solidity

Scope	Commit
<a href="https://op-geth.quarkchain.io/">https://op-geth.quarkchain.io/</a>	self: 6160d64630056b665ba3ce8623a82b0bdb084c47 upstream: 2b9abb39077cb88f6e8a513f09a5ea2c2569dfed
<a href="https://op-node.quarkchain.io/">https://op-node.quarkchain.io/</a>	self: 98b2647899a4aa856b1b194b409db1519a742752 upstream: c8b9f62736a7dad7e569719a84c406605f4472e6

### 2.2 Summary

Severity	Found
<b>CRITICAL</b>	0
<b>HIGH</b>	0
<b>MEDIUM</b>	0
<b>LOW</b>	5
<b>INFO</b>	2



## 2.3 Key Findings

Severity	Findings Title	Status
LOW	Nil Pointer Dereference in distributeGas Function	Fixed
LOW	Missing Zero Address Validation in batchMint	Acknowledge
LOW	Empty Array Handling Vulnerability in SoulGasToken	Acknowledge
LOW	Lack of Events for Critical Operations	Fixed
LOW	Missing Zero Address Validation in addMinters Function	Acknowledge
INFO	Fixed Timeout for Blob Upload	Acknowledge
INFO	Missing Zero Check in Operator Fee Refund Process	Acknowledge

Table 2.3: Key Audit Findings



### 3. DETAILED DESCRIPTION OF FINDINGS

#### 3.1 Nil Pointer Dereference in distributeGas Function

Location	Severity	Category
op-geth	LOW	Business Logic

##### Description:

In the distributeGas function, if pool1 is nil, the code branches to:

```

1  if pool1 == nil {
2      // pool1 empty, all to pool2
3      quota1 = new(uint256.Int)
4      quota2 = amount.Clone()
5
6      pool2.Sub(pool2, quota2) // <-- Potential nil pointer dereference
7      return
8  }

```

The function assumes that when pool1 is nil, pool2 must be non-nil. However, there's no explicit check that prevents both pools from being nil simultaneously.

##### Recommendations:

Add a defensive check at the beginning of the function:

```

1  func (st *stateTransition) distributeGas(amount, pool1, pool2 *uint256.Int) (quota1, quota2 *uint256.Int) {
2      if amount == nil {
3          panic("amount should not be nil")
4      }
5      if st.usedSGTBalace == nil {
6          panic("should not happen when usedSGTBalace is nil")
7      }
8      if pool1 == nil && pool2 == nil {
9          // Both pools empty - return amount as is with no distribution
10         return amount, nil
11     }
12     // ... existing code ...
13 }

```

Result	FixResult
Confirmed	Fixed

### 3.2 Missing Zero Address Validation in batchMint

Location	Severity	Category
op-node	LOW	Input Validation

#### Description:

The batchMint function in the SoulGasToken contract lacks validation to prevent minting tokens to the zero address (address(0)). When authorized minters call this function with an array of recipient addresses, the code fails to verify that none of these addresses are the zero address before proceeding with token minting.

```

1 function batchMint(address[] calldata _accounts, uint256[] calldata _values) external {
2     // we don't explicitly check !IS_BACKED_BY_NATIVE here, because if IS_BACKED_BY_NATIVE,
3     // there's no way to add a minter.
4     require(_accounts.length == _values.length, "SGT: invalid arguments");
5
6     SoulGasTokenStorage storage $ = _getSoulGasTokenStorage();
7     require($.minters[msgSender()], "SGT: not a minter");
8
9     for (uint256 i = 0; i < _accounts.length; i++) {
10         _mint(_accounts[i], _values[i]); // No zero address check!
11     }
12 }

```

#### Recommendations:

Add a validation loop before the minting process to check all recipient addresses in the array.

Result	FixResult
Confirmed	Acknowledge

### 3.3 Empty Array Handling Vulnerability in SoulGasToken

Location	Severity	Category
op-node	LOW	Input Validation

#### Description:

The SoulGasToken contract has multiple batch functions that don't explicitly check for or handle empty arrays. When empty arrays are passed to these functions, transactions still execute and consume gas without performing any meaningful actions.

```

1 function batchDepositForAll(address[] calldata _accounts, uint256 _value) external payable {
2     require(IS_BACKED_BY_NATIVE, "SGT: batchDepositForAll should only be called when IS_BACKED_BY_NATIVE");
3
4     for (uint256 i = 0; i < _accounts.length; i++) {
5         _mint(_accounts[i], _value);
6     }
7     require(msg.value == _value * _accounts.length, "SGT: unexpected msg.value");
8 }

```

Similar issues exist in:

- batchDepositFor
- batchWithdrawFrom
- batchMint
- addMinters/delMinters
- addBurners/delBurners
- allowSgtValue/disallowSgtValue
- batchBurnFrom

#### Recommendations:

Add array length validation at the beginning of each affected function to ensure arrays contain at least one element before processing.

Result	FixResult
Confirmed	Acknowledge

### 3.4 Lack of Events for Critical Operations

Location	Severity	Category
op-node	LOW	Code Quality

#### Description:

The SoulGasToken contract fails to emit events for critical operations that affect token balances and native currency transfers. This absence represents a significant transparency issue in the contract.

```

1  function withdrawFrom(address _account, uint256 _value) external {
2      require(IS_BACKED_BY_NATIVE, "SGT: withdrawFrom should only be called when IS_BACKED_BY_NATIVE");
3      SoulGasTokenStorage storage $ = _getSoulGasTokenStorage();
4      require($.burners[msgSender()], "SGT: not the burner");
5      _burn(_account, _value);
6      payable(msgSender()).transfer(_value);
7      // No event emission
8  }
9
10 function burnFrom(address _account, uint256 _value) external {
11     require(!IS_BACKED_BY_NATIVE, "SGT: burnFrom should only be called when !IS_BACKED_BY_NATIVE");
12     SoulGasTokenStorage storage $ = _getSoulGasTokenStorage();
13     require($.burners[msgSender()], "SGT: not the burner");
14     _burn(_account, _value);
15     // No event emission
16 }

```

#### Recommendations:

Add comprehensive event logging for all critical operations to ensure proper auditability and off-chain integration support.

Result	FixResult
Confirmed	Fixed

### 3.5 Missing Zero Address Validation in addMinters Function

Location	Severity	Category
op-node	LOW	Input Validation

#### Description:

The addMinters function in the SoulGasToken contract lacks validation to prevent the zero address (address(0)) from being added to the minters whitelist:

```

1 function addMinters(address[] calldata _minters) external onlyOwner {
2     require(!IS_BACKED_BY_NATIVE, "SGT: addMinters should only be called when !IS_BACKED_BY_NATIVE");
3     SoulGasTokenStorage storage $ = _getSoulGasTokenStorage();
4     uint256 i;
5     for (i = 0; i < _minters.length; i++) {
6         $.minters[_minters[i]] = true; // No check for address(0)
7     }
8 }

```

#### Recommendations:

Add zero address validation within the loop to prevent adding invalid addresses to the minters mapping.

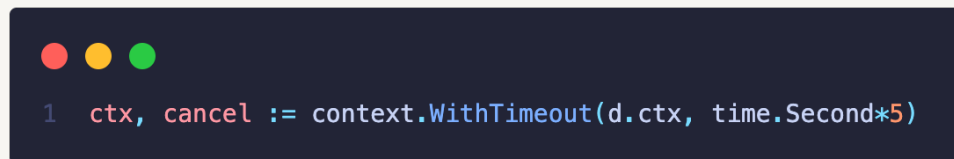
Result	FixResult
Confirmed	Acknowledge

### 3.6 Fixed Timeout for Blob Upload

Location	Severity	Category
op-node	INFO	Code Quality

#### Description:

A hard-coded 5-second timeout for blob uploads could be problematic. If blobs are large or network conditions are poor, this timeout may be too short, causing uploads to fail unnecessarily. This could lead to a denial of service if valid blocks are repeatedly rejected.



```
1 ctx, cancel := context.WithTimeout(d.ctx, time.Second*5)
```

#### Recommendations:

Implement configurable timeout with dynamic adjustment based on blob size and network conditions, or provide fallback mechanisms for failed uploads.

Result	FixResult
Confirmed	Acknowledge

### 3.7 Missing Zero Check in Operator Fee Refund Process

Location	Severity	Category
op-geth	INFO	Input Validation

#### Description:

The function `refundIsthmusOperatorCost()` in `state_transition.go` lacks a check for zero-value refunds before calling `refundGas()`. When the calculated refund amount `refundedOperatorCost` is zero, the code still unnecessarily processes a gas refund operation.

```

1 func (st *stateTransition) refundIsthmusOperatorCost() {
2     operatorCostGasLimit := st.evm.Context.OperatorCostFunc(st.msg.GasLimit, st.evm.Context.Time)
3     operatorCostGasUsed := st.evm.Context.OperatorCostFunc(st.gasUsed(), st.evm.Context.Time)
4
5     if operatorCostGasUsed.Cmp(operatorCostGasLimit) > 0 { // Sanity check.
6         panic(fmt.Sprintf("operator cost gas used (%d) > operator cost gas limit (%d)", operatorCostGasUsed, operatorCostGasLimit))
7     }
8
9     refundedOperatorCost := new(uint256.Int).Sub(operatorCostGasLimit, operatorCostGasUsed)
10    st.refundedOperatorFee = refundedOperatorCost.Clone()
11    st.refundGas(refundedOperatorCost) // No check if refundedOperatorCost is zero!
12 }

```

#### Recommendations:

Add a simple zero-value check before processing the refund:

```

1 func (st *stateTransition) refundIsthmusOperatorCost() {
2     operatorCostGasLimit := st.evm.Context.OperatorCostFunc(st.msg.GasLimit, st.evm.Context.Time)
3     operatorCostGasUsed := st.evm.Context.OperatorCostFunc(st.gasUsed(), st.evm.Context.Time)
4
5     if operatorCostGasUsed.Cmp(operatorCostGasLimit) > 0 {
6         panic(fmt.Sprintf("operator cost gas used (%d) > operator cost gas limit (%d)", operatorCostGasUsed, operatorCostGasLimit))
7     }
8
9     refundedOperatorCost := new(uint256.Int).Sub(operatorCostGasLimit, operatorCostGasUsed)
10    st.refundedOperatorFee = refundedOperatorCost.Clone()
11
12    // Only refund if there's an actual amount to refund
13    if refundedOperatorCost.Sign() > 0 {
14        st.refundGas(refundedOperatorCost)
15    }
16 }

```

Result	FixResult
Confirmed	Acknowledge



## 4. CONCLUSION

---

In this audit, we thoroughly analyzed **QuarkChain** Blockchain 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	<b>CRITICAL</b>

#### 5.1.2 Authorization Access Control

Description	Permission checks for external integral functions
Result	Not found
Severity	<b>CRITICAL</b>

#### 5.1.3 Forged Transfer Vulnerability

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

#### 5.1.4 Transaction Rollback Attack

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

### 5.1.5 Transaction Block Stuffing Attack

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

### 5.1.6 Soft Fail Attack Assessment

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

### 5.1.7 Hard Fail Attack Assessment

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

### 5.1.8 Abnormal Memo Assessment

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

### 5.1.9 Abnormal Resource Consumption

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

### 5.1.10 Random Number Security

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

## 5.2 Advanced Code Scrutiny

### 5.2.1 Cryptography Security

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

### 5.2.2 Account Permission Control

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

### 5.2.3 Malicious Code Behavior

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

#### 5.2.4 Sensitive Information Disclosure

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

#### 5.2.5 System API

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

## 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](https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology)

## Contact



Website

[www.exvul.com](http://www.exvul.com)



Email

[contact@exvul.com](mailto:contact@exvul.com)



Twitter

[@EXVULSEC](https://twitter.com/EXVULSEC)



Github

[github.com/EXVUL-Sec](https://github.com/EXVUL-Sec)

**EV ExVul**