

Cysic Smart Contract

# SMART CONTRACT AUDIT REPORT

March 2025



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

# Table of 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 .....	7
<b>3. DETAILED DESCRIPTION OF FINDINGS .....</b>	<b>8</b>
3.1 depositNative Fails to Receive Correct Token .....	8
3.2 Potential Centralization Risks in SideBridge's rescueNative/rescueERC20 .....	9
3.3 Unexpected Multisig Execution Behavior .....	10
3.4 Potential DoS Attack in MainBridge.withdrawNative .....	11
3.5 Arbitrary targetToken Selection in MainBridge.mapToken .....	12
3.6 Incorrect Amount Calculation Due to Decimals Mismatch .....	13
<b>4. CONCLUSION .....</b>	<b>14</b>
<b>5. APPENDIX .....</b>	<b>15</b>
5.1 Basic Coding Assessment .....	15
5.1.1 Apply Verification Control .....	15
5.1.2 Authorization Access Control .....	15
5.1.3 Forged Transfer Vulnerability .....	15
5.1.4 Transaction Rollback Attack .....	15
5.1.5 Transaction Block Stuffing Attack .....	15
5.1.6 Soft Fail Attack Assessment .....	15
5.1.7 Hard Fail Attack Assessment .....	15
5.1.8 Abnormal Memo Assessment .....	16
5.1.9 Abnormal Resource Consumption .....	16
5.1.10 Random Number Security .....	16

5.2 Advanced Code Scrutiny .....	16
5.2.1 Cryptography Security .....	16
5.2.2 Account Permission Control .....	16
5.2.3 Malicious Code Behavior .....	16
5.2.4 Sensitive Information Disclosure .....	16
5.2.5 System API .....	16
<b>6. DISCLAIMER .....</b>	<b>18</b>
<b>7. REFERENCES .....</b>	<b>19</b>

# 1. EXECUTIVE SUMMARY

Exvul Web3 Security was engaged by Cysic 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
	Semantic Consistency Checks
Additional Recommendations	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: Cysic

Audit Time: March 20, 2025 - March 28, 2025

Language: solidity

Soure code	Link
Cysic	private
Commit Hash	5825b2592a0fd74d11eef5f2d7e63946

### 2.2 Summary

Severity	Found	
Critical	0	
High	1	<div></div>
Medium	2	<div></div>
Low	3	<div></div>
Informational	0	

### 2.3 Key Findings

ID	Severity	Findings Title	Status	Confirm
NVE- 001	High	depositNative Fails to Receive Correct Token	Fixed	Confirmed
NVE- 002	Medium	Potential Centralization Risks in SideBridge's rescueNative/rescueERC20	Fixed	Confirmed
NVE- 003	Medium	Unexpected Multisig Execution Behavior	Fixed	Confirmed
NVE- 004	Low	Potential DoS Attack in MainBridge.withdrawNative	Acknowledge	Confirmed
NVE- 005	Low	Arbitrary targetToken Selection in MainBridge.mapToken	Fixed	Confirmed
NVE- 006	Low	Incorrect Amount Calculation Due to Decimals Mismatch	Acknowledge	Confirmed

Table 2.3: Key Audit Findings

## 3. DETAILED DESCRIPTION OF FINDINGS

### 3.1 depositNative Fails to Receive Correct Token

<b>ID:</b>	NVE-001	<b>Location:</b>	MainBridge.sol
<b>Severity:</b>	High	<b>Category:</b>	Business Issues
<b>Likelihood:</b>	High	<b>Impact:</b>	High

#### Description:

The issue arises because different chains have different native tokens. For instance, the cysic chain's native token is cys, not eth ([cysic-Network-Whitepaper](#)). When users bridge native tokens (e.g., ETH, AVAX) from a sidechain to the cysic chain using the SideBridge.depositNative method, they receive the cysic chain's native token \$cys instead of the expected wrapped token (e.g., WETH). This occurs because the MainBridge.depositNative method incorrectly assumes that native tokens from other chains should be directly converted to \$cys rather than their corresponding wrapped tokens. For example, bridging 1 ETH results in receiving 1 \$cys instead of 1 WETH, causing incorrect asset valuation and potential financial loss.

#### Dual Token Model

The goal of the Cysic Network is to establish a decentralized and reliable proving and verification service that fosters community growth and self-sustainability. The token will be used to incentivize provers, verifiers, and validators within the protocol, establishing an effective governance and reward distribution mechanism. Cysic Network uses a dual-token model, consisting of the network token and governance token. Each token plays a specific role in the network, working together to build the Cysic Network ecosystem:

- **\$CYS:** The \$CYS token is the native token of the Cysic Network and is used to pay transaction fees, block rewards, and other network-related activities. \$CYS ensures the liveness and vitality of the network through its transaction fee mechanism and serves as one of the incentives for users to participate in network activities.
- **\$CGT:** \$CGT is the governance token and is non-transferable. It can be obtained by staking \$CYS and completing computation tasks. The un-staking process takes longer than the staking process, as implemented in the Cosmos SDK.

#### Recommendations:

The MainBridge.depositNative method should be modified to return the corresponding wrapped native token instead of the cysic chain's native token.

Result: **Fixed**



### 3.2 Potential Centralization Risks in SideBridge's rescueNative/rescueERC20

<b>ID:</b>	NVE-002	<b>Location:</b>	SideBridge.sol
<b>Severity:</b>	Medium	<b>Category:</b>	Business Issues
<b>Likelihood:</b>	High	<b>Impact:</b>	Medium

#### Description:

Signers in the SideBridge contract can exploit the rescueNative and rescueERC20 methods to transfer users' frozen cross-chain assets to a multi-signature wallet, even under normal conditions. This unauthorized access allows signers to move assets without user consent, breaking the peg between wrapped tokens on the cysic chain and native assets on the sidechain. Users are left unable to transfer assets back to the sidechain due to insufficient funds in SideBridge, leading to locked assets and potential financial loss.

```

273     function rescueNative() external onlyMultiSig {
274         uint256 balance = address(this).balance;
275         require(balance > 0, "No native tokens to rescue");
276         @>> (bool success, ) = address(multiSigWallet).call{value: balance}("");
277         require(success, "Native token rescue failed");
278         emit NativeTokenRescued(balance, address(multiSigWallet));
279     }

```

#### Recommendations:

To prevent unauthorized fund movements, the rescueNative and rescueERC20 methods should only allow transfers of assets explicitly authorized by users, or impose certain restrictions on admin withdrawals.

**Result:** Fixed

### 3.3 Unexpected Multisig Execution Behavior

<b>ID:</b>	NVE-003	<b>Location:</b>	MultiSigWallet.sol
<b>Severity:</b>	Medium	<b>Category:</b>	Business Issues
<b>Likelihood:</b>	Medium	<b>Impact:</b>	Medium

#### Description:

The static confirmation count in transactions can lead to unexpected execution when signers and required confirmations change via `replaceSigners`. Specifically, the `confirmTransaction` method in `MultiSigWallet.sol` uses static values for confirmations and required confirmations, which do not dynamically reflect current signers or thresholds. This allows old confirmations to be counted even after signers are updated, enabling transactions to execute with insufficient or outdated confirmations from new signers.

```

90     function confirmTransaction(
91         uint256 _txId
92     ) public onlySigner txExists(_txId) notConfirmed(_txId) notExecuted(_txId) {
93         Transaction storage transaction = transactions[_txId];
94         transaction.confirmations++;
95         hasConfirmed[_txId][msg.sender] = true;
96         emit TransactionConfirmed(_txId, msg.sender);
97         @>> if (transaction.confirmations >= requiredConfirmations) {
98             executeTransaction(_txId);
99         }
100     }

```

#### Recommendations:

Modify the transaction confirmation logic to dynamically reference the current list of signers and required confirmations instead of relying on static values.

**Result:** Fixed

### 3.4 Potential DoS Attack in MainBridge.withdrawNative

<b>ID:</b>	NVE-004	<b>Location:</b>	MainBridge.sol
<b>Severity:</b>	Low	<b>Category:</b>	Business Issues
<b>Likelihood:</b>	Medium	<b>Impact:</b>	Low

#### Description:

Malicious users can exploit the MainBridge.withdrawNative function on the cysic chain to launch denial-of-service (DoS) attacks by initiating numerous small withdrawals. This function lacks access controls, allowing attackers to repeatedly submit tiny withdrawal requests (e.g., 1 wei) to sidechain addresses, overwhelming off-chain validators and consuming withdrawal resources. Legitimate users face significant delays or failures when attempting to withdraw substantial amounts (e.g., 100 ether) due to the backlog of trivial transactions.

```

407     function withdrawNative(
408         uint256 _targetChainId,
409         address _recipient
410     ) external payable whenNotPaused {
411         require(
412             _targetChainId != cysicChainId,
413             "Cannot withdraw to Cysic chain"
414         );
415         @>> require(msg.value > 0, "Amount must be greater than 0");
416         require(_recipient != address(0), "Invalid recipient address");
417         require(
418             tokenMapping[_targetChainId][NATIVE_TOKEN] != address(0),
419             "Native token not supported for target chain"
420         );
421     }

```

#### Recommendations:

To mitigate DoS risks, implement a minimum withdrawal threshold (MIN\_WITHDRAW) that requires users to withdraw amounts above a specified value. This increases the cost of launching attacks while preserving normal withdrawal functionality for legitimate users.

**Result:** Acknowledge

### 3.5 Arbitrary targetToken Selection in MainBridge.mapToken

<b>ID:</b>	NVE-005	<b>Location:</b>	MainBridge.sol
<b>Severity:</b>	Low	<b>Category:</b>	Business Issues
<b>Likelihood:</b>	Medium	<b>Impact:</b>	Low

#### Description:

When using MainBridge.mapToken, selecting an existing token as targetToken that isn't a cysicToken created by MainBridge causes deposit and withdrawal failures. This happens because MainBridge lacks the necessary canMint() and canBurn() permissions for the targetToken, preventing it from minting or burning tokens during user transactions.

```

385      // Mark nonce as used
386      usedNonces[messageHash] = true;
387
388      // Increment nonce after successful verification
389      depositChainNonce[_sourceChainId]++;
390
391      // Get the mapped token address
392      @>> address mappedToken = tokenMapping[_sourceChainId][_token];
393
394      // Get the Cysic token contract using the mapped token address
395      @>> ICysicToken cysicToken = ICysicToken(mappedToken);
396
397      // Check if this contract has mint permission
398      require(cysicToken.canMint(address(this)), "Bridge cannot mint token");
399
400      // Mint tokens to recipient
401      cysicToken.mint(_recipient, _amount);
402
403      // Emit completion events
404      emit FinishDeposit(_token, _recipient, _amount, _sourceChainId);
405  }

```

#### Recommendations:

Add a verification step in MainBridge.mapToken to ensure MainBridge has canMint() and canBurn() permissions for the targetToken. This can be done by checking if targetToken is a valid cysicToken created by MainBridge and confirming that MainBridge holds the required permissions before allowing the mapping. This ensures seamless deposit and withdrawal operations.

**Result:** Fixed

### 3.6 Incorrect Amount Calculation Due to Decimals Mismatch

<b>ID:</b>	NVE-005	<b>Location:</b>	MainBridge.sol
<b>Severity:</b>	Low	<b>Category:</b>	Business Issues
<b>Likelihood:</b>	Medium	<b>Impact:</b>	Low

#### Description:

When using MainBridge.createToken, if the corresponding tokens on the sidechain and cysic chain have different decimals, the transfer amount is incorrectly calculated on a 1:1 basis. This occurs because MainBridge.createToken allows setting decimals without ensuring consistency between chains, leading to errors during user deposit/withdrawal operations. For example, a token with 18 decimals on the sidechain and 6 decimals on the cysic chain would result in incorrect amount conversions.

```

246
247     it("Should create new token when target token is zero address", async function () {
248         const TestToken = await ethers.getContractFactory("TestToken");
249         const newToken = await TestToken.deploy("Test Token", "TT", 18);
250         await newToken.waitForDeployment();
251
252         // Get all addresses upfront
253         const [mainBridgeAddress, newTokenAddress] = await Promise.all([
254             mainBridge.getAddress(),
255             newToken.getAddress()
256         ]);
257
258         const tokenName = "New Cysic Token";
259         const tokenSymbol = "NCT";
260         const tokenDecimals = 6; // Test with 6 decimals
261
262         const mapTokenTx = await multiSig.connect(signers[0]).submitTransaction(
263             mainBridgeAddress,
264             0,

```

#### Recommendations:

Ensure that corresponding tokens on both chains have matching decimals. If decimals differ, implement proper amount conversions during calculations to reflect the correct value based on the decimal differences.

**Result:** Acknowledge

## 4. CONCLUSION

In this audit, we thoroughly analyzed Cysic 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](https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology)



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



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



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



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

**EV** ExVul