ExVul

SMART CONTRACT AUDIT REPORT

CysicNFT Smart Contract

AUGUST 2025

Contents

1. EXECUTIVE SUMMARY

ExVul Web3 Security was engaged by **CysicNFT** 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: 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 \ IMPACT	Informational	Low	Medium	High
High	INFO	MEDIUM	HIGH	CRITICAL
Medium	INFO	LOW	MEDIUM	HIGH
Low	INFO	LOW	LOW	MEDIUM

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 • Abnormal Resource Consumption
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	Audit Time	Language
CysicNFT	04/08/2025 - 06/08/2025	Solidity

Repository	Commit Hash
nft.zip	4ee8b81916f6145c01b6f04f2f040077848a30e1

2.2 Summary

Severity	Found
CRITICAL	0
HIGH	0
MEDIUM	1
LOW	1
INFO	4

2.3 Key Findings

Severity	Findings Title	Status
MEDIUM	setMaxId lacks a check for the new value, potentially invalidating already minted NFTs	Fixed
LOW	initialize() function lacks critical parameter validation	Fixed
INFO	State Variable Layout Can Be Optimized to Save Gas	Fixed
INFO	Lack Event Emission	Fixed
INFO	maxId may cause misunderstanding of the token ID range	Fixed
INFO	Redundant State Updates Cause Unnecessary Gas Consumption	Fixed

Table 2.3: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 setMaxId lacks a check for the new value, potentially invalidating already minted NFTs

SEVERITY: MEDIUM **STATUS:** Fixed

PATH:

`CysicNFT.sol`

DESCRIPTION:

The setMaxId() function allows the contract owner to arbitrarily set the value of maxId, but it does not check if the new _maxId is greater than or equal to the current maxId. Therefore, the owner could accidentally set maxId to a value smaller than its previous value, leading to unintended consequences.

```
function setMaxId(uint256 _maxId) public onlyOwner {
    maxId = _maxId;
}
```

IMPACT:

If maxId is decreased, all legitimately minted NFTs with IDs between the new maxId and the old maxId will become unusable.

RECOMMENDATIONS:

Add a required check within the function to ensure that the new _maxId must be greater than or equal to the current maxId. This prevents already issued tokens from being inadvertently “deactivated”.

```
function setMaxId(uint256 _maxId) public onlyOwner {
    require(_maxId > maxId, ”CysicNFT: Cannot decrease maxId”);
    maxId = _maxId;
}
```

3.2 initialize() function lacks critical parameter validation

SEVERITY: LOW **STATUS:** Fixed

PATH:

`CysicNFT.sol`

DESCRIPTION:

The initialize() function, which is responsible for setting the contract's critical initial state, does not perform any validity checks on the incoming _usdc parameter.

```
function initialize(address _usdc,uint256 _maxId, string memory uri_)
    public
    initializer
{
    USDC = IERC20(_usdc); // _usdc can be address(0)
    _uri = uri_;
    maxId = _maxId;
    transferable = false; // default is false
    __Pausable_init();
    __Ownable_init(msg.sender);
    __ReentrancyGuard_init();
    __ERC1155_init(_uri);
    __UUPSUpgradeable_init();
}
```

IMPACT:

A mistake during deployment could permanently cripple the contract's core functions. Setting an invalid _usdc address could lock all funds.

RECOMMENDATIONS:

Add require statements at the beginning of the function body for critical parameters to ensure robust contract initialization and prevent issues arising from deployment errors.

```
function initialize(address _usdc, uint256 _maxId, string memory uri_)
    public
    initializer
{
    require(_usdc != address(0), "CysicNFT: USDC address cannot be the
        zero address");

    USDC = IERC20(_usdc);
    _uri = uri_;
    // ... rest of the code
}
```

3.3 State Variable Layout Can Be Optimized to Save Gas

SEVERITY: INFO **STATUS:** Fixed

PATH:

`CysicNFT.sol`

DESCRIPTION:

The declaration order of state variables in the contract has not been optimized to take advantage of storage slot packing. The Solidity compiler packs multiple, consecutively declared variables that total less than 32 bytes into a single storage slot. By reordering the variables, USDC (address, 20 bytes) and transferable (bool, 1 byte) could be packed together, thus reducing the gas required for contract deployment and subsequent state writes.

Current Order (Unoptimized):

```
contract CysicNFT is ... {
    IERC20 public USDC;       // Slot 0 (address, uses 20 bytes)
    uint256 public maxId;      // Slot 1 (uses full 32 bytes)
    string private _uri;       // Slot 2 (dynamic type)
    bool public transferable;   // Slot 3 (uses a new slot as it's not
        adjacent to a packable type)
    ...
}
```

IMPACT:

The current state variable layout results in extra storage overhead, which slightly increases the deployment cost of the contract. Throughout the contract's lifecycle, write operations to these unpacked variables will also consume more gas than they would if optimized.

RECOMMENDATIONS:

Rearrange the state variable declarations in the contract to place variables that can be packed together (such as address and bool) adjacently.

```
contract CysicNFT is ... {
    uint256 public maxId;
    string private _uri;
    IERC20 public USDC;
    bool public transferable;

    // ... rest of the code
}
```

3.4 Lack Event Emission

SEVERITY: INFO **STATUS:** Fixed

PATH:

`CysicNFT.sol`

DESCRIPTION:

Several critical administrative functions in the contract directly modify core state variables but do not emit corresponding events upon successful execution. This applies to the setMaxId, setBaseURI, and setTransferable functions. The absence of event logs makes it difficult for off-chain services, front-end applications, and users to track these important state changes.

IMPACT:

This increases the opacity of centralized actions and creates unnecessary difficulties for off-chain indexing and data analysis.

RECOMMENDATIONS:

Add and emit events for all administrative functions that execute critical state changes.

3.5 maxId may cause misunderstanding of the token ID range

SEVERITY: INFO **STATUS:** Fixed

PATH:

`CysicNFT.sol`

DESCRIPTION:

The variable maxId is used in the contract as an exclusive upper bound to validate token IDs. For example, if the public value returned by maxId is 100, the actual maximum valid ID is 99. This naming convention could be misinterpreted by front-end or off-chain service developers as an inclusive upper bound (i.e., thinking the max ID is 100), which can lead to an off-by-one error in their logic.

IMPACT:

This naming convention is likely to cause misunderstanding for developers integrating with the contract.

RECOMMENDATIONS:

Determine whether maxId is index id or number of nfts to clarify the naming convention and prevent misunderstanding.

3.6 Redundant State Updates Cause Unnecessary Gas Consumption

SEVERITY: INFO **STATUS:** Fixed

PATH:

`CysicNFT.sol`

DESCRIPTION:

Several set functions in the contract (setMaxId, setBaseURI, setTransferable) do not check if the new input value is the same as the currently stored value before updating the state variable. Even if the new and old values are identical, the contract still performs an expensive storage write operation (SSTORE), which leads to unnecessary gas wastage.

IMPACT:

When the owner calls these functions and passes a value identical to the current one, the transaction will consume more gas than necessary.

RECOMMENDATIONS:

Before performing a state write, compare the new and old values first. If the values are the same, return early to skip the expensive storage write operation.

4. CONCLUSION

In this audit, we thoroughly analyzed **CysicNFT** 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
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	MEDIUM

5.2.5 System API

Description	Examine whether system API application issue present in the code
Result	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

Contact

Website
www.exvul.com

Email
contact@exvul.com

Twitter
@EXVULSEC

Github
github.com/EXVUL-Sec

ExVul