Smart Contract Audit Report

GOYA Smart Contract

27 Sep 2022

Numen Cyber Labs - Security Services

Numen Cyber Technology Pte. Ltd.
11 North Buona Vista Drive, #04-09,
The Metropolis, Singapore 138589

Tel: 65-63555555
Fax: 65-63666666
Email: sales@numencyber.com
Web: https://numencyber.com

Table of Content

1 Executive Summary

Numen Cyber Technology was engaged by GOYA 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.

A High severity finding identified in no-transaction-fee mining and inability to create new transactions when large transactions were involved.

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.

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 and impact are categorized into three ratings: High, Medium and Low. Severity is determined by likelihood and impact and can be classified into four categories accordingly, Critical, High, Medium, Low shown in table 1.1.

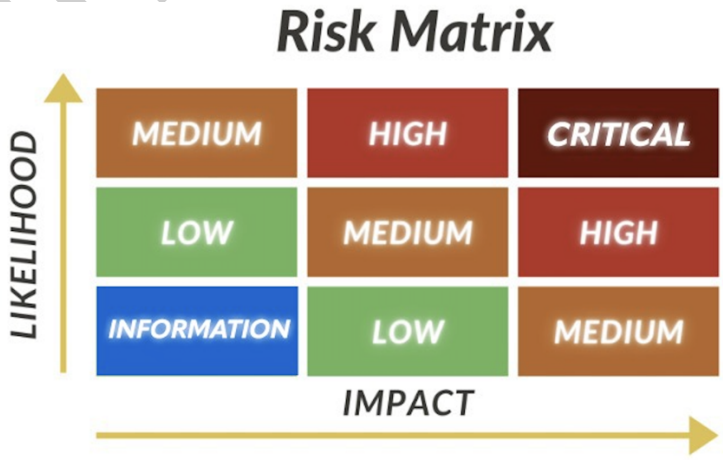

Table 1.1: Overall Risk Severity

To evaluate the risk, we will be going through a list of items, and each would be labelled with a severity category. The audit was performed with a systematic approach guided by a comprehensive assessment list carefully designed to identify known and impactful security issues. If our tool or analysis does not identify any issue, the contract can be considered safe regarding the assessed item. For any discovered issue, we might further deploy contracts on our private test environment and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.2.

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.
- Code and business security testing: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

Category	Assessment Item
Basic Coding Assessment	Apply Verification Control
	Authorization Access Control
	Forged Transfer Vulnerability
	Forged Transfer Notification
	Numeric Overflow
	Transaction Rollback Attack
	Transaction Block Stuffing Attack
	Soft_fail Attack
	Hard_fail Attack
	Abnormal Memo
	Abnormal Resource Consumption
	Secure Random Number
Advanced Source Code Scrutiny	Asset Security
	Cryptography Security
	Business Logic Review
	Source Code Functional Verification
	Account Authorization Control
	Sensitive Information Disclosure
	Circuit Breaker
	Blacklist Control
	System API Call Analysis

	Contract Deployment Consistency Check
Additional Recommendations	Semantic Consistency Checks
	Following Other Best Practices

Table 1.2: The Full List of Assessment Items

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [14], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development.

2 Findings Overview

2.1 Project info and Contract address

Project Name: GOYA
Project URL: NULL
Audit Time: 2022/9.23 - 2022/9.26
Language: solidity

Contract Name	Smart Contract Address
GoyaContract1.sol	
PriceConverter.sol	

2.2 Summary

Severity	Found	
Critical	0	
High	3	
Medium	0	
Low	0	
Informational	1	

2.3 Key Findings

Three high severities findings are related to owner authority, overflow caused by business logic and price acquisition. In addition, there is also 1 Informational finding.

ID	Severity	Findings Title	Status
NVE-001	High	Owner has higher authority	Ignore
NVE-002	High	Code logic flaws cause overflow	Fixed
NVE-003	High	There is a security risk in the calculation after the oracle machine obtains the price	Ignore
NVE-004	Informational	Notes on resellToken function	Ignore

Table 2.1: Key Audit Findings

3 Detailed Description of Findings

3.1 Owner has higher authority

ID: NVE-001
Severity: High
Likelihood: High
Impact: High

Location: CustomerContract1.sol
Category: Authority Issues

Description:

As shown in figure 1 below, the owner can call the approveAddress function to authorize the specified _tokenId to the _to address, after the owner calls the function, the _to address can transfer the NFT corresponding to the authorized tokenId at any time. There is a risk of excessive permissions, which may affect the security of user assets.

```
function approveAddress(uint256 _tokenId) external {
    require(owner == msg.sender, "Only mktplace owner can appoint approvers");
    _approve(contractOwner, _tokenId);
}
```

Figure 1 approveAddress function

Recommendations:

It is recommended that the project side delete this authorization function or modify the owner permission.

Result:

Pass

Fix Result:

Ignore (After communicating with the project party, this permission is required for the project design and is only used in special circumstances.)

3.2 Code logic flaws cause overflow

ID: NVE-002
Severity: High
Likelihood: High
Impact: High

Location:CustomerContract1.sol
Category: Business Issues

Description:

As shown in Figure 2 below,when the user owns the NFT or has the authorization of the specified tokenId, users can call transferNFTTo function to transfer NFT. The value of _itemsSold is incremented by 1 each time it is called，In fact, the NFT under the contract is not sold. As shown in Figure 3 below，When calling the fetchMarketItems function to query the unsold NFTs under the contract, it may cause an overflow, causing the function call to fail.

```
function transferNFTTo(address _from, address _to, uint256 tokenId) external nonReentrant {
    require(_isApprovedOrOwner(msg.sender, tokenId), "Transfer caller is not owner nor approved");
    idToMarketItem[tokenId].sold = true;
    idToMarketItem[tokenId].initialList = false;
    idToMarketItem[tokenId].seller = payable(address(0));
    idToMarketItem[tokenId].owner = payable(_to);

    idToMarketItem[tokenId].reservePriceUSD = 0;
    _itemsSold.increment();
    _transfer(_from, _to, tokenId);
}
```

Figure 2 transferNFTTo function

```
function fetchMarketItems() public view returns (MarketItem[] memory) {
  uint itemCount = _tokenIds.current();
  uint unsoldItemCount = _tokenIds.current() - _itemsSold.current();
  uint currentIndex = 0;

  MarketItem[] memory items = new MarketItem[](unsoldItemCount);
  for (uint i = 0; i < itemCount; i++) {
    if (idToMarketItem[i + 1].owner == address(this)) {
      uint currentId = i + 1;
      MarketItem storage currentItem = idToMarketItem[currentId];
      items[currentIndex] = currentItem;
      currentIndex += 1;
    }
  }
  return items;
}
```

Figure 3 fetchMarketItems function

Recommendations:

It is recommended that when calling the transferNFTTo function, determine whether the from address is this contract address.

Result:

Pass

Fix Result:

Fixed

The fixed code is as follows:

```
function transferNFTTo(address _from, address _to, uint256 tokenId) external nonReentrant {
    require(_isApprovedOrOwner(contractOwner, tokenId), "Transfer caller is not owner nor approved");
    idToMarketItem[tokenId].sold = true;
    idToMarketItem[tokenId].initialList = false;
    idToMarketItem[tokenId].seller = payable(address(0));
    idToMarketItem[tokenId].owner = payable(_to);

    idToMarketItem[tokenId].reservePriceUSD = 0;
    if (_from == address(this)){
        _itemsSold.increment();
    }
    _transfer(_from, _to, tokenId);
}
```

Figure 4 fixed function

3.3 There is a security risk in the calculation after the oracle machine obtains the price

ID: NVE-003
Severity: High
Likelihood: High
Impact: High

Location:CustomerContract1.sol
Category: price Issues

Description:

As shown in Figure 5 below, the ETH price in the contract is obtained by calling the latestRoundData function of chainlink，the data returned by the interface may be abnormal or inaccurate，if the returned value is 0, other function calls may fail in the morning. When calculating the price, pay special attention to the precision calculation.

```
library PriceConverter {

    function getPrice() internal view returns (uint256) {

        AggregatorV3Interface priceFeed = AggregatorV3Interface(
            0xd0D5e3DB44DE05E9F294BB0a3bEEaF030DE24Ada // Mumbai
            //0xAB594600376Ec9fD91F8e885dADF0CE036862dE0 // live mainnet
        );
        (, int256 answer, , , ) = priceFeed.latestRoundData();

        return uint256(answer * 10000000000);
    }

    // 1000000000
    function getConversionRate(uint256 ethAmount)
        internal
        view
        returns (uint256)
    {
        uint256 ethPrice = getPrice();
        uint256 ethAmountInUsd = (ethPrice * ethAmount) / 1000000000000000000;
        return ethAmountInUsd;
    }

    function getOrgUsdMatic(uint256 _amount) internal view returns (uint256) {
        uint256 ethPrice = getPrice() ;
        uint256  adjust_price = uint256 (ethPrice) * 1e18;
        uint256  usd = _amount * 1e18;
        uint256  rate = (usd * 1e18) / adjust_price;
        return rate;
    }
```

Figure 5 PriceConverter library

Recommendations:

It is recommended to make a non-zero judgment on the interface return value of chainlink,at the same time, in terms of price acquisition, a multi-data source method can be used to reduce errors.

Result:

Pass

Fix Result:

Ignore

3.4 Notes on resellToken function

ID: NVE-004
Severity: Informational
Likelihood: Informational
Impact: Informational

Location: CustomerContract1.sol
Category: Business Issues

Description:

As shown in Figure 5 below, when the user calls the original transfer function of ERC721 to transfer the NFT, the user who receives the NFT cannot call this function to secondary sales., because idToMarketItem[tokenId].owner may store the address of the last NFT that was transferred to you.

```
function resellToken(uint256 tokenId, uint256 _sellpriceUSD, uint256 price, bool _unlist) external payable nonReentrant {
  if (_unlist){
    require(msg.sender == idToMarketItem[tokenId].seller && idToMarketItem[tokenId].initialList == false, "Only s may unlist");
    idToMarketItem[tokenId].sold = true;
    idToMarketItem[tokenId].seller = payable(address(0));
    idToMarketItem[tokenId].owner = payable(msg.sender);
    _itemsSold.increment();
    _transfer(address(this), msg.sender, tokenId);
  }
  else {
    require(idToMarketItem[tokenId].owner == msg.sender, "Only item o");
    require(msg.value == listingPriceSecondary, "Price must be = Sec list price");

    idToMarketItem[tokenId].sold = false;
    idToMarketItem[tokenId].sellpriceUSD = _sellpriceUSD;
    idToMarketItem[tokenId].price = price;
    idToMarketItem[tokenId].seller = payable(msg.sender);
    idToMarketItem[tokenId].owner = payable(address(this));
    _itemsSold.decrement();
    _transfer(msg.sender, address(this), tokenId);
  }
}
```

Figure 6 PriceConverter library

Result:

Pass

Fix Result:

Ignore (After communicating with the project party, it conforms to the project design, here is only a reminder)

4 Conclusion

In this audit, we thoroughly analyzed **GOYA**'s smart contract implementation. The problems found are described and explained in detail in Section 3. The problems found in the audit have been brought up to the project party, ignored issues are in line with the project design, and permissions are only used for the project to properly function. We therefore deem the audit result to be a **PASS.** 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 Behaviour
- Description: Examine whether sensitive behaviour 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 Numen'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 Numen 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. Numen's position is that each company and individual are responsible for their own due diligence and continuous security. Numen'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.

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.