Neoki Multi Metaverse

Security Assessment - Final

24th March 2023

TRUSCOVA
TRUSTED CODE VALIDATION

Prepared for: Neoki Multi Metaverse

Prepared by: Muhammad Hassan

WE SECURE YOUR SMART CONTRACTS

About Truscova

Founded in last quarter of 2022 and headquartered in Bremen, Germany, Truscova aims to secure Web 3.0 by providing security assessment, audit, advisory and training services using formal verification and other leading technologies. Truscova founders include eminent scientists with more than 35 books, several patents, and more than 20,000 google scholar citations in the areas of verification, testing, security, and machine learning.

We focus on smart contracts testing and code review projects in Ethereum ecosystem, supporting client organizations in technology, defense, and finance industries, as well as government entities. We specialize in applying formal verification, dynamic analysis, fuzz testing, metamorphic testing, advanced coverage metrics, and static analysis to secure Web 3.0 projects.

We maintain an exhaustive list of publications at https://github.com/Truscova, with links to papers, presentations, public audit reports, and blog articles.

To explore our latest news and developments, please follow @truscova on Twitter or LinkedIn. You can also explore our repository at https://github.com/Truscova or blog articles at https://truscova.com/blog.php. To engage us directly, visit our "Contact" page at https://www.truscova.com/.

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Truscova in this project were performed in accordance with terms of the project agreement.

Truscova uses a combination of automated testing techniques and manual inspection to conduct security assessments and identify security flaws of the target system. Each method carries its own limitations.

Security assessments are time bound (hence not exhaustive) and are reliant on information provided by the client. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Contents

Executive Summary

Engagement Overview and Scope

Neoki Multi Metaverse engaged Truscova GmbH from 22 February 2023 till 16th March 2023 to review and audit the code base provided via online repository available at following link:

https://github.com/ojoaoguilherme-neoki

The scope of audit is limited to the following:

- https://github.com/ojoaoguilherme-neoki/token-sell (commit 9e2a6e6)
- https://github.com/ojoaoguilherme-neoki/neoki-nfts (commit f3bb2e5)
- https://github.com/ojoaoguilherme-neoki/contract-lands (commit 7943e38)
- https://github.com/ojoaoguilherme-neoki/contract-marketplace (commit 64d1205)

One security expert at Truscova audited the above code base using static analysis, symbolic execution, fuzzing, formal verification, and manual inspection. A summary of findings is discussed in the next subsection.

Summary of Findings

The uncovered vulnerabilities in codebase during the course of the audit are summarized in the following table:

Vulnerability Classification

Classification	*Count*
High Impact	8
Medium Impact	6
Low Impact	13
Informational	11
Undetermined	N/A

Vulnerability Categorization

Category	*Count*
Access Control	4
Configuration	2
Data Validation	13
Arithmetic	9
Denial of Service	4
Reentrancy	3
Notation	1

Notable Findings

Significant flaws that impact system are listed below.

- TCV-NK-1

  An attacker may get free NKO tokens without paying fee.
- TCV-NK-2

  An attacker can get free NKO tokens with BNB without paying fee.
- TCV-NK-3

  An attacker can get free NKO tokens with DAI without paying fee.
- TCV-NK-4

  An attacker can get free NKO tokens with BUSD without paying fee.
- TCV-NK-9

  Anyone can mint infinite number of NFTs.
- TCV-NK-10

  Anyone can mint infinite number of NFTs with Royalties.
- TCV-NK-12

  ADMIN_ROLE can mint infinite number of NKO tokens.
- TCV-NK-16

  URI_UPDATER role is never assigned.

Project Summary

Contact Information

Following officials from Truscova participated in this project.

Dr. Muhammad Naiman Jalil, naiman@truscova.com	Project Coordination
Dr. Muhammad Hassan, hassan@truscova.com	Security Assessment Expert

Project Timeline

The significant project events and milestones are as follows:

Date	**Event**
24th February, 2023	Pre-project Kickoff Call
2nd March, 2023	Preliminary Assessment Start
6th March, 2023	Contract Agreement
6th March, 2023	Assessment Start
15th March, 2023	Assessment Complete
16th March, 2023	Report Draft Complete and Shared with Neoki
20th March 2023	Report Readout Meeting
24th March 2023	Final Audit Report

Project Goals

The engagement was scoped to assess the security of Neoki Multi Metaverse's Lands, Marketplace, Token, and NFT contracts. Specifically, we focused to answer the following non-exhaustive list of questions:

- Are there appropriate access controls in place?
- Are user-provided parameters sufficiently validated?
- Are the arithmetic calculations and state changes performed during NFT purchases and token purchases, correct?
- How is oracle data obtained and handled?
- Could an attacker steal funds from the system?
- Is there a way for users to circumvent fees?
- Could an attacker take over the contract's ADMIN_ROLE?
- How does the minting of NFTs work?

Project Targets

The engagement involved a review and testing of the following targets:

Neoki Multi Metaverse

1. Repository https://github.com/ojoaoguilherme-neoki/token-sell
   (commit 9e2a6e6)
2. Repository https://github.com/ojoaoguilherme-neoki/neoki-nfts
   (commit f3bb2e5)
3. Repository https://github.com/ojoaoguilherme-neoki/contract-lands
   (commit 7943e38)
4. Repository https://github.com/ojoaoguilherme-neoki/contract-marketplace
   (commit 64d1205)

Codebase Type: Solidity

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches and their results include the following:

NeokiLands

This is a Solidity smart contract for a Non-fungible Token (NFT). It inherits from the ERC4907 and AccessControl contracts from the OpenZeppelin library, which implement the ERC721 standard and access control functionality, respectively. The contract has a maximum limit of 423,801 NFTs that can be minted, which is enforced in the onlyMint modifier. The contract owner is granted the DEFAULT_ADMIN_ROLE and MINTER_ROLE roles upon contract creation, which allows them to mint new NFTs. The mint function can be called by a user with the MINTER_ROLE to mint a new NFT, which is assigned a unique token ID and token URI. The updateURI function can be called by a user with the URI_UPDATER role to update the base URI for the token URI. In review of the contract, we reviewed the correctness of the following:

- The roles for access control are correctly placed.
- The arithmetic calculation for minting lands.
- We checked for reentrancy attack.

LandSell

This is a smart contract for selling LAND tokens on the Ethereum blockchain. The contract implements an ERC721Holder, which means that it can receive and hold ERC721 tokens, and an AccessControl, which allows specific roles to access certain functions. The contract defines a Map struct that holds information about each token being sold, including its owner, token ID, price, and whether it is sellable or not. It also imports several contracts from the OpenZeppelin library, including AccessControl, SafeERC20, and ERC721Holder. We conducted a review of this contract and investigated the following:

- We reviewed the bounds on all owner-controlled system parameters.
- We checked for appropriate access control for defining land prices, setting land for selling, buying land, and listing land to be sold.
- We reviewed if an attacker could bypass the access control on minting and burning functions.
- We checked for reentrancy attack.
- We checked for price manipulation of the lands when buying.
- We checked if it is possible to change the ownership of someone's land.
- We reviewed the arithmetic operations.

NikoToken

This is a smart contract for the NikoToken, which is an ERC20 token that is used to represent a digital asset. The contract is based on the OpenZeppelin ERC20 and AccessControl libraries. The contract has two roles defined using the bytes32 type and the keccak256 hash function: MINTER_ROLE and BURNER_ROLE. The contract owner is initially granted all three roles: DEFAULT_ADMIN_ROLE, MINTER_ROLE, and BURNER_ROLE. The constructor function creates the token by calling the ERC20 constructor and passing in the token name and symbol. It also mints and sends the initial supply of tokens to the specified receiver address. The mint function can be called by anyone who has been granted the MINTER_ROLE and allows them to mint new tokens and send them to a specified address. The burn function can be called by anyone who has been granted the BURNER_ROLE and allows them to burn (destroy) existing tokens held by a specified address. Note that there is no way to revoke the MINTER_ROLE or BURNER_ROLE from the original contract owner. We conducted a review of the contract and investigated the following:

- We reviewed if an attacker could bypass the access control on minting and burning functions.
- We checked the arithmetic operations.
- We checked if roles could be updated or not.

ERC4907

This smart contract implements the ERC721URIStorage standard and extends the IERC4907 interface. The contract allows the ownership of an NFT (non-fungible token) to be transferred to a user for a certain period of time. The contract defines a struct UserInfo which stores the address of the user and the timestamp until which the user can use the NFT. The contract maintains a mapping from token IDs to UserInfo structs to keep track of the ownership information of NFTs. We conducted a review of the contract and investigated the following:

- We checked the access control for setting the user and expiry information.

NeokiMarketplaceWithRoyalty

This smart contract implements a marketplace for non-fungible tokens (NFTs) and ERC-1155 tokens, with a royalty fee structure. The contract provides functionality for listing and buying tokens on the marketplace, as well as updating the fee and royalty structure. The contract imports various other contracts from the OpenZeppelin library, including ERC1155Holder, IERC2981, IERC1155, SafeERC20, AccessControl, ReentrancyGuard, and Counters. The contract has a struct called MarketItem that stores details about a token that is listed for sale, such as the item ID, token ID, amount, price, owner, and contract address. The contract constructor takes several input parameters, including addresses for the foundation, staking pool, NKO token contract, and an admin address. The contract also defines various events such as NewListing, NewBoughtItem, DeleteItem, UpdateItemPrice, UpdateItemAddAmount, UpdateItemRemoveAmount, and UpdateMarketplaceFee. The contract provides two external functions for buying and listing items on the marketplace. The listItem function lists ERC721/ERC1155 tokens on the marketplace, and the buyItem function allows users to purchase listed items. The contract also has functionality for changing the fee and royalty structure, with the ability to update the marketplace fee and add or remove royalties for specific items. We conducted a review and checked the following things:

- We checked the usage of counters in different functions for increment and decrements.
- We checked if the items can be bought without paying any fee.
- We checked if items (NFTs) sold and their counters are synchronized and consistent.
- We checked if an attacker could take ownership of someone else's NFTs.
- We checked if items listed and items removed are consistent.
- We reviewed the arithmetic operations to check for any rounding errors and results going to zero.

- We checked if the events are emitted at the end of functions for logging.

NeokiNftRoyalty

This smart contract implements a non-fungible token (NFT) standard based on ERC-1155 and ERC-2981. The contract is designed to be used for creating and managing NFTs with royalty payments. The contract imports the ERC1155 and ERC2981 contracts from the OpenZeppelin library. It also uses the Counters library to keep track of token IDs and their URIs. We checked the following things:

- We checked if proper access control is in place.
- We checked if the URI can be updated later on.

NikoSell

The purpose of this contract is to allow users to buy Niko token (NKO) using different currencies such as MATIC, BNB, DAI, and BUSD. The contract is also able to update the price of NKO, receive NKO tokens from the contract owner, withdraw tokens and ETH, and obtain the latest price feeds of MATIC and BNB. The contract uses the Chainlink AggregatorV3Interface to obtain the latest price feeds of MATIC and BNB. The contract also uses the OpenZeppelin AccessControl and ReentrancyGuard libraries to manage access control and prevent reentrancy attacks. The contract owner is assigned the ADMIN_ROLE and DEFAULT_ADMIN_ROLE upon contract deployment, giving them access to functions such as updating the NKO price, funding the contract with NKO tokens, and withdrawing tokens and ETH. We reviewed the following things:

- We checked access control placement for all functions.
- We checked for reentrancy attack in withdraw functions in particular and all functions in general.
- We checked arithmetic operations for buying NKO tokens.
- We checked if an attacker could transfer tokens to themselves free of charge.
- We checked if the contract keeps track of tokens sold.
- We checked if the address of BNB, BUSD, DAI, and MATIC could be updated later on.
- We reviewed the price oracles.

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity	Remarks (After Report Readout Meeting)
1	Attacker may get free NKO tokens	Data validation	High	Updated the vulnerability finding report.
2	Transfer zero number of NKO tokens with BNB but spend gas or get free NKO tokens	Data validation and Arithmetic	High	--
3	Transfer zero number of NKO tokens with DAI or get free tokens	Data validation and Arithmetic	High	--
4	Transfer zero number of NKO tokens with BUSD	Data validation and Arithmetic	High	--
5	Counters never decrement	Arithmetic	Low	Business Decision – To be kept as it is
6	Buying NFTs without any fee	Arithmetic	Medium	--
7	Buying NFTs without any fee	Arithmetic	Medium	--
8	Transaction reverts when removing items from list	Arithmetic	Medium	--
9	Anyone can mint infinite number of NFTs	Access control	High	Business Decision – To be kept as it is
10	Anyone can mint infinite number of NFTs with royalties	Access control	High	Business Decision – To be kept as it is
11	URI not updatable	Access control	Low	Business Decision – To be checked again by Neoki
12	Minting of infinite number of NKO tokens	Data validation	High	Business Decision – To be kept as it is
13	Not using scientific notation	Notation	Informational	--
14	Minting more lands than the maximum limit defined	Arithmetic	Low	--
15	Reentrancy	Reentrancy	Medium	--
16	URI_UPDATER role is never assigned	Access control	High	Business Decision – To be checked again by Neoki
17	Reentrancy	Reentrancy	Medium	--

18	Uninitialized local variables	Initialization	Informational	--
19	Local variable shadowing	Data validation	Informational	--
20	Address validation for zero address	Data validation	Low	--
21	Possible denial of service attack (DOS) – external calls inside loop	Denial of Service	Low	Business Decision – To be checked again by Neoki
22	Item amount underflow	Arithmetic	Medium	--
23	Unused state variables	Data validation	Informational	--
24	Reentrancy	Reentrancy	Low	--
25	Low level calls	Configuration	Informational	--
26	Low level calls	Configuration	Informational	--
27	Possible denial of service attack (DOS) – external calls inside loop	Denial of Service	Low	Business Decision – To be checked again by Neoki
28	Possible denial of service attack (DOS) – external calls inside loop	Denial of Service	Low	Business Decision – To be checked again by Neoki
29	Possible denial of service attack (DOS) – external calls inside loop	Denial of Service	Low	Business Decision – To be checked again by Neoki
30	Address validation for zero address	Data validation	Low	--
31	Address validation for zero address	Data validation	Low	--
32	Address validation for zero address	Data validation	Low	--
33	Address validation for zero address	Data validation	Low	--
34	Local variable shadowing	Data validation	Low	--
35	Uninitialized local variables	Initialization	Informational	--
36	Uninitialized local variables	Initialization	Informational	--
37	Uninitialized local variables	Initialization	Informational	--
38	Uninitialized local variables	Initialization	Informational	--

39	Uninitialized local variables	Initialization	Informational	--

Detailed Findings

1. Attacker may get free NKO tokens

Severity: High	Difficulty: Medium
Type: Data validation	Finding ID: TCV-NK-1
Target: NikoSell	

Description

This function “buyNikoWithMatic” (Figure 1) is designed to allow a user to buy a specified amount of NKO tokens using the MATIC cryptocurrency. The function takes in one parameter, buyAmount, which represents the amount of NKO that the user wants to buy. The function first retrieves the latest price of MATIC using the getMaticLatestPrice() function. It then calculates the cost of buyin`g the specified amount of NKO in MATIC, using the current price of NKO and the latest price of MATIC. The function then checks whether the user has sent enough MATIC to cover the cost of buying the NKO tokens. If the user has not sent enough MATIC, the function will throw an error and stop. If the user has sent enough MATIC, the function will transfer the MATIC to the contract using address(this).call{value: cost}(""); and transfer the specified amount of NKO to the user using NKO.safeTransfer(msg.sender, buyAmount);. If the user has sent more MATIC than the cost of buying the NKO tokens, the function will send the excess MATIC back to the user using (bool sentBack,) = msg.sender.call{value: msg.value - cost}("");. If this fails, the function will throw an error and stop. Finally, the function returns a boolean value indicating whether the MATIC transfer was successful. Solidity's integer division truncates. Thus, performing division before multiplication can lead to precision loss.

```
function buyNikoWithMatic(
        uint256 buyAmount
    ) external payable nonReentrant returns (bool) {
        uint256 matic = getMaticLatestPrice();
        uint256 cost = (buyAmount * ((nkoPrice * 1e18) / matic)) / 1e18;
        require(msg.value >= cost, "Sent insufficient MATIC");
        (bool sent, ) = address(this).call{value: cost}("");
        require(sent, "Failed to send ETH");
        NKO.safeTransfer(msg.sender, buyAmount);
        if (msg.value - cost > 0) {
            (bool sentBack, ) = msg.sender.call{value: msg.value - cost}("");
            require(sentBack, "Failed to send back remaining MATIC");
        }
        return sent;
    }
```

Figure 1.1: buyNikoWithMatic function from NikoSell contract NikoSell.sol#L60-74

Exploit Scenario

It is possible that the cost becomes zero because the contract uses division operation before multiplication. The price “nkoPrice” can be updated by the admin using the function “updateNikoPrice(uint256 amount) external onlyRole(ADMIN_ROLE) {...}”. It may happen that the price of NKO (nkoPrice) is mistakenly set to one (1) or a smaller value in general. As a result, the expression “(buyAmount * ((nkoPrice * 1e18) / bnbusd))” may result in value less than 1e18.

As an example, consider the following contract:

```
contract A {
            function f(uint n) public {
       coins = (oldSupply / n) * interest;
   }
}
```

If n is greater than oldSupply, coins will be zero. For example, with oldSupply = 5; n = 10, interest = 2, coins will be zero. If (oldSupply * interest / n) was used, coins would have been 1. In general, it's usually a good idea to re-arrange arithmetic to perform multiplication before division, unless the limit of a smaller type makes this dangerous.

Please check the following links for reference:

- Beware rounding with integer division
- Integer Division
- Precision Loss in Arithmetic Operations
- How to Bypass the Integer Division Error in Smart Contracts

Recommendation

It should be checked if cost is zero (0). In such a case, the transaction should revert. Additionally, add a state variable that keeps track of the contract's balance. Furthermore, rearrange arithmetic so that multiplication is done before division.

2. Transfer zero number of NKO tokens with BNB but spend gas or get free NKO tokens

Severity: High	Difficulty: Medium
Type: Data validation and Arithmetic	Finding ID: TCV-NK-2
Target: NikoSell	

Description

The function "buyNikoWithBNB" is designed to allow a user to buy a specified amount of NKO using the Binance Smart Chain (BSC) cryptocurrency BNB. The function takes in one parameter, buyAmount, which represents the amount of NKO that the user wants to buy. The function first retrieves the latest price of BNB in USD using the getBNBLatestPrice() function. It then calculates the cost of buying the specified amount of NKO in BNB, using the current price of NKO and the latest price of BNB.

The function then uses the safeTransferFrom function to transfer the required amount of BNB from the user to the contract. The contract address for BNB on the Binance Smart Chain is 0x3BA4c387f786bFEE076A58914F5Bd38d668B42c3. The safeTransferFrom function ensures that the transfer is successful and that the contract does not receive more BNB than it needs to complete the NKO purchase. Finally, the function transfers the specified amount of NKO tokens to the user. It's worth noting that this function does not return any value. Solidity's integer division truncates. Thus, performing division before multiplication can lead to precision loss.

```
function buyNikoWithBNB(uint256 buyAmount) external payable nonReentrant
{
        uint256 bnbusd = getBNBLatestPrice();
        uint256 cost = (buyAmount * ((nkoPrice * 1e18) / bnbusd)) / 1e18;

IERC20(0x3BA4c387f786bFEE076A58914F5Bd38d668B42c3).safeTransferFrom(
            msg.sender,
            address(this),
            cost
        );
        NKO.safeTransfer(msg.sender, buyAmount);
    }
```

Figure 2: buyNikoWithBNB function from NikoSell contract NikoSell.sol#L76-85

Exploit Scenario

The arithmetic calculation of cost can result in zero (0) even when buyAmount is not zero (0). The price "nkoPrice" can be updated by the admin using the function "updateNikoPrice(uint256 amount) external onlyRole(ADMIN_ROLE) {…}". It may happen that the price of NKO (nkoPrice) is mistakenly set to one (1). As a result, the expression "(buyAmount * ((nkoPrice * 1e18) / bnbusd))" may result in less than 1e18.

The division in Solidity always rounds down. There are no decimals. As a result, if "(buyAmount * ((nkoPrice * 1e18) / bnbusd))" is less than "1e18", the cost will be 0. However, the attacker will still be able to get NKO tokens.

Please check the following links for reference:

- Beware rounding with integer division
- Integer Division
- Precision Loss in Arithmetic Operations
- How to Bypass the Integer Division Error in Smart Contracts

Another scenario is that Alice calls the "buyNikoWithBNB" function with zero (0) "buyAmount" and N BNB tokens. The function never checks the buyAmount, as a result, the cost becomes zero (0). NikoSell contract transfers zero (0) BNB tokens from Alice to NikoSell contract as the cost is zero (0). This operation is also successful as it is allowed to transfer zero (0) tokens. NikoSell also transfers zero (0) NKO tokens to Alice. This results in gas usage but no transfer of tokens.

As an example, consider the following contract

```
contract A {
            function f(uint n) public {
       coins = (oldSupply / n) * interest;
    }
}
```

If n is greater than oldSupply, coins will be zero. For example, with oldSupply = 5; n = 10, interest = 2, coins will be zero. If (oldSupply * interest / n) was used, coins would have been 1. In general, it's usually a good idea to re-arrange arithmetic to perform multiplication before division, unless the limit of a smaller type makes this dangerous.

Recommendation

Proper data validation of buyAmount parameter should be done. Additionally, add a state variable that keeps track of the contract's balance. Furthermore, check that the cost is not zero (0). Consider ordering multiplication before division.

3. Transfer zero number of NKO tokens with DAI or get free tokens

Severity: High	Difficulty: Low
Type: Data validation and Arithmetic	Finding ID: TCV-NK-3
Target: NikoSell	

Description

The function "buyNikoWithDAI" allows users to buy NKO token with DAI. It takes in one argument, buyAmount, which is the amount of Niko tokens the user wants to buy. IERC20(0x8f3Cf7ad23Cd3CaDbD9735AFf958023239c6A063).safeTransferFrom() transfers DAI tokens from the user's address to the contract's address. NKO.safeTransfer(msg.sender, buyAmount) transfers NKO tokens from the contract's address to the user's address. Overall, this function enables users to exchange DAI tokens for Niko tokens at a fixed price, as specified by the nkoPrice variable.

```
function buyNikoWithDAI(uint256 buyAmount) external payable nonReentrant
{
      uint256 cost = (buyAmount * nkoPrice) / 1e18;

 IERC20(0x8f3Cf7ad23Cd3CaDbD9735AFf958023239c6A063).safeTransferFrom(
          msg.sender,
          address(this),
          cost
      );
      NKO.safeTransfer(msg.sender, buyAmount);
   }
```

Figure 3: buyNikoWithDAI function from NikoSell contract NikoSell.sol#L87-96

Exploit Scenario

The arithmetic calculation cost can result in zero (0) even when buyAmount is not zero (0), e.g., buyAmount * nkoPrice is less than 1e18. In this case the result will be less than one (1). Since, solidity does not support decimals, the result will not be 0.xx rather just zero (0). The price can be updated by the admin using the function "updateNikoPrice(uint256 amount) external onlyRole(ADMIN_ROLE) {…}". It may happen that the expression "buyAmount * nkoPrice" is less than 1e18. This is because Solidity does not support floating point arithmetic.

The division in Solidity always rounds down. There are no decimals. As a result, if "buyAmount * nkoPrice" is less than "1e18", the cost will be 0. However, the attacker will still be able to get NKO tokens.

Please check the following links for reference:

- Beware rounding with integer division
- Integer Division
- Precision Loss in Arithmetic Operations
- How to Bypass the Integer Division Error in Smart Contracts

Another scenario is that Alice calls the "buyNikoWithDAI" function with zero (0) "buyAmount" and N DAI tokens. The function never checks the buyAmount, as a result, the cost becomes zero (0). NikoSell contract transfers zero (0) DAI tokens from Alice to NikoSell contract as the cost is zero (0). This operation is also successful as it is allowed to transfer zero (0) tokens. NikoSell also transfers zero (0) NKO tokens to Alice. This results in gas usage but no transfer of tokens.

Recommendation

Proper data validation of buyAmount parameter should be done. Additionally, add a state variable that keeps track of the contract's balance. Furthermore, check that the cost does not result in zero (0). Require statements should be used.

4. Transfer zero number of NKO tokens with BUSD or get free tokens

Severity: High	Difficulty: Low
Type: Data validation and Arithmetic	Finding ID: TCV-NK-4
Target: NikoSell	

Description

The function “buyNikoWithBUSD” allows users to buy a certain amount of NKO tokens with BUSD (Binance USD) tokens. The division by 1e18 is to convert the result from wei (the smallest unit of ether) to BUSD.

IERC20(0xdAb529f40E671A1D4bF91361c21bf9f0C9712ab7).safeTransferFrom(msg.sender, address(this), cost) transfers the required amount of BUSD tokens from the user to the contract. The address 0xdAb529f40E671A1D4bF91361c21bf9f0C9712ab7 represents the address of the BUSD token contract, which is a standard ERC-20 token. The safeTransferFrom function ensures that the transfer is executed safely and the contract is notified when the transfer is successful. NKO.safeTransfer(msg.sender, buyAmount) transfers the specified amount of NKO tokens from the contract to the user's address. The safeTransfer function ensures that the

```
function buyNikoWithBUSD(uint256 buyAmount) external payable
nonReentrant {
        uint256 cost = (buyAmount * nkoPrice) / 1e18;


IERC20(0xdAb529f40E671A1D4bF91361c21bf9f0C9712ab7).safeTransferFrom(
            msg.sender,
            address(this),
            cost
        );
        NKO.safeTransfer(msg.sender, buyAmount);
    }
```

Figure 4: buyNikoWithBUSD function from NikoSell contract [NikoSell.sol#L98-107]

transfer is executed safely and the contract is notified when the transfer is successful.

Exploit Scenario

One scenario is that Alice calls the “buyNikoWithBUSD” function with zero (0) “buyAmount” and N BUSD tokens. The function never checks the buyAmount, as a result, the cost becomes zero (0). NikoSell contract transfers zero (0) BUSD tokens from Alice to NikoSell contract as the cost is zero (0). This operation is also successful as it is allowed to transfer zero (0) tokens. NikoSell also transfers zero (0) NKO tokens to Alice. This results in gas usage but no transfer of tokens.

Another scenario is if the buyAmount is not zero (0), but the updated value of “nkoPrice” is less than 1e18. The price can be updated by the admin using the function “updateNikoPrice(uint256 amount) external onlyRole(ADMIN_ROLE) {...}”. It may happen that the expression “buyAmount * nkoPrice” is less than 1e18. This is because Solidity does not support floating point arithmetic.

The division in Solidity always rounds down. There are no decimals. As a result, if “buyAmount * nkoPrice” is less than “1e18”, the cost will be 0. In this scenario, the attacker can get free NKO tokens.

Please check the following links for reference:

- Beware rounding with integer division
- Integer Division
- Precision Loss in Arithmetic Operations
- How to Bypass the Integer Division Error in Smart Contracts

Recommendation

Proper data validation of buyAmount parameter should be done. Additionally, add a state variable that keeps track of the contract's balance. Furthermore, check that the cost does not result in zero (0).

5. Counters never decrement

Severity: Low	Difficulty: Low
Type: Arithmetic	Finding ID: TCV-NK-5
Target: NeokiMarketplaceWithRoyalty	

Description

Several counters are defined in the NeokiMarketplaceWithRoyalty contract. The counters track the amount of NFTs sold and other items, etc. These counters are incremented but they never decrement.

```
…
 using SafeERC20 for IERC20;
   Counters.Counter public nftSold; // show on the UI
   Counters.Counter public totalSoledItems;
   Counters.Counter private _totalItems;
   Counters.Counter private _unavailableItems;
…
```

Figure 5: Snippet from NeokiMarketplaceWithRoyalty contract MarketplaceRoyalties.sol#L26-30

After report readout discussion with Neoki Multi Metaverse, they explained the concept and informed Truscova that it is done on purpose. It helps monitor the total number of NFTs sold by the marketplace.

Recommendation

Ensure that the counters for nftSold, totalSoledItems, _totalItems, and _unavailableItems are correctly decremented when the items are deleted.

6. Buying NFTs without any fee

Severity: Medium	Difficulty: Medium
Type: Arithmetic	Finding ID: TCV-NK-6
Target: NeokiMarketplaceWithRoyalty	

Description

The feeAmount will be zero (0) if paying * listingFee is less than 10000. When feeAmount becomes zero (0), the user can still buy items without paying any fee. The foundation and the stakingPool also do not get any tokens.

```
function buyItem(uint256 _itemId, uint256 _amount) external nonReentrant {
      require(_amount > 0, "Cannot buy amount of 0.");
      MarketItem storage item = marketItem[_itemId];

      // Fees
      uint256 paying = item.price * _amount;
      uint256 feeAmount = (paying * listingFee) / 10000;
…
```

Figure 6: Snippet from function buyItem in NeokiMarketplaceWithRoyalty contract
MarketplaceRoyalties.sol#L129-187

The division in Solidity always rounds down. There are no decimals. As a result, if "paying * listingFee" is less than "10000", feeAmount will be 0.

Please check the following links for reference:

- Beware rounding with integer division
- Integer Division
- Precision Loss in Arithmetic Operations
- How to Bypass the Integer Division Error in Smart Contracts

Recommendation

Ensure that the feeAmount is checked for zero (0) value. If feeAmount becomes zero (0), the transaction should revert.

7. Buying NFTs without any fee

Severity: Medium	Difficulty: Medium
Type: Arithmetic	Finding ID: TCV-NK-7
Target: NeokiMarketplaceWithRoyalty	

Description

The feeAmount can result in zero (0) and “paying” variable can result in a very small value. If the value is less than or equal to the royaltyAmount, an attacker can get the asset without paying any fee or any price for the item. Because “paying - (feeAmount + royaltyAmount)” will result in zero (0).

```
…
uint256 paying = item.price * _amount;
uint256 feeAmount = (paying * listingFee) / 10000;
nko.safeTransfer(item.owner, paying - (feeAmount + royaltyAmount));
      IERC1155 asset = IERC1155(item.nftContract);
      asset.safeTransferFrom(
         address(this),
         msg.sender,
         item.tokenId,
         _amount,
         ""
      );
```

Figure 7: Snippet from function buyItem in NeokiMarketplaceWithRoyalty contract
[MarketplaceRoyalties.sol#L155-163]

The division in Solidity always rounds down. There are no decimals. As a result, if “paying * listingFee” is less than “10000”, feeAmount will be 0.

Please check the following links for reference:

- Beware rounding with integer division
- Integer Division
- Precision Loss in Arithmetic Operations
- How to Bypass the Integer Division Error in Smart Contracts

Recommendation

Ensure that the “paying - (feeAmount + royaltyAmount)” is never zero (0).

8. Transaction reverts when removing items from list

Severity: Medium	Difficulty: Low
Type: Arithmetic	Finding ID: TCV-NK-8
Target: NeokiMarketplaceWithRoyalty	

Description

The function “removeMyListingItemAmount” allows the owner of a listed NFT item to remove a specified amount of the item from the market, and transfer it back to their own account. It retrieves the MarketItem struct with the given _itemId from the marketItem mapping and assigns it to a local item variable. It checks if the amount of the item is greater than 0 using the require statement. If the amount is 0, it means that there is no NFT to withdraw, and the function reverts. The function checks if the owner of the item is the same as the msg.sender who is calling the function using the require statement. If the owner is not the same as msg.sender, it means that the caller is not the owner of the listed item, and the function reverts. It gets a reference to the IERC1155 contract instance for the nftContract of the item using the asset variable. It subtracts the _removeAmount from the amount of the item. It transfers _removeAmount tokens of the item to the caller (msg.sender) using the safeTransferFrom function of the asset contract. If the amount of the item is now 0, it means that the item is no

```
function removeMyListingItemAmount(
        uint256 _itemId,
        uint256 _removeAmount
    ) public nonReentrant {
        MarketItem storage item = marketItem[_itemId];
        require(item.amount > 0, "There is no NFT to withdraw");
        require(item.owner == msg.sender, "Not the owner of the listed item");
        IERC1155 asset = IERC1155(item.nftContract);
        item.amount -= _removeAmount;

        asset.safeTransferFrom(
            address(this),
            msg.sender,
            item.tokenId,
            _removeAmount,
            ""
        );
...
```

Figure 8: Snippet from function removeMyListingItemAmount in NeokiMarketplaceWithRoyalty contract [MarketplaceRoyalties.sol#L275-297]

longer available for sale, and the function increments the _unavailableItems counter and deletes the item from the marketItem mapping.

The function never checks if the amount of items is greater than or equal to _removeAmount. As a consequence, the transaction will revert if the amount of items is smaller than _removeAmount.

Recommendation

Add a require statement to check if _removeAmount is less than or equal to the item.amount.

9. Anyone can mint infinite number of NFTs

Severity: High	Difficulty: Low
Type: Access control	Finding ID: TCV-NK-9
Target: NeokiNftRoyalty	

Description

The “mint” function allows to mint a new batch of tokens and assigns them to a specified account. The function also sets the token URI for the new tokens and returns the tokenId of the newly minted tokens. It increments the _tokenIdCounter using the _tokenIdCounter.increment() function to get a new token ID for the newly minted tokens. It assigns the new token ID to a tokenId variable. It calls the internal _mint function to mint the specified amount of tokens with the given tokenId and data to the account. It calls the setTokenUri function to set the token URI for the newly minted tokens. It returns the tokenId of the newly minted tokens.

```
function mint(
        address account,
        uint256 amount,
        string memory tokenURI,
        bytes memory data
    ) public returns (uint256) {
        _tokenIdCounter.increment();
        uint tokenId = _tokenIdCounter.current();

        _mint(account, tokenId, amount, data);
        setTokenUri(tokenId, tokenURI);
        return tokenId;
    }
```

Figure 9: mint function in NeokiNFTRoyalty contract NeokiNFTRoyalties.sol#L23-35

After report readout discussion with Neoki Multi Metaverse, they explained the concept and informed Truscova that it is a business decision.

Recommendation

Add proper access control that either the owner or someone with proper ROLE can access the mint function.

10. Anyone can mint infinite number of NFTs with royalties

Severity: High	Difficulty: Low
Type: Access control	Finding ID: TCV-NK-10
Target: NeokiNftRoyalty	

Description

The “mintWithRoyalties” function mints new tokens with royalties and sets their metadata URI. It takes in five arguments, it increments a _tokenIdCounter variable using the increment() function from the OpenZeppelin library, and assigns the current value to a variable called tokenId. The function calls the _mint function, passing in the account, tokenId, amount, and data arguments. This function mints new tokens and assigns them to the account address. Afterwards, it calls the setTokenUri function, passing in the tokenId and tokenURI arguments. This function sets the metadata URI of the token. At the end, it calls the _setTokenRoyalty function, passing in the tokenId, account, and royalty arguments. This function sets the royalty percentage for the token creator.

```
function mintWithRoyalties(
        address account,
        uint256 amount,
        string memory tokenURI,
        uint96 royalty,
        bytes memory data
    ) public returns (uint256) {
        _tokenIdCounter.increment();
        uint256 tokenId = _tokenIdCounter.current();

        _mint(account, tokenId, amount, data);
        setTokenUri(tokenId, tokenURI);
        _setTokenRoyalty(tokenId, account, royalty);
        return _tokenIdCounter.current();
    }
```

Figure 10: mintWithRoyalties function in NeokiNFTRoyalty contract NeokiNFTRoyalties.sol#L46-60

After report readout discussion with Neoki Multi Metaverse, they explained the concept and informed Truscova that it is a business decision.

Recommendation

Add proper access control that either the owner or someone with proper ROLE can access the mint function.

11. URI not updatable

Severity: Low	Difficulty: Low
Type: Access control	Finding ID: TCV-NK-11
Target: NeokiNftRoyalty	

Description

The contract NeokiNftRoyalty allows to set the URI of token only once. It does not allow to update the URI later on.

```
function mint(
    address account,
    uint256 amount,
    string memory tokenURI,
    bytes memory data
) public returns (uint256) {
    _tokenIdCounter.increment();
    uint tokenId = _tokenIdCounter.current();

    _mint(account, tokenId, amount, data);
    setTokenUri(tokenId, tokenURI);
    return tokenId;
}
```

Figure 11: mint function from NeokiNftRoyalty contract NeokiNFTRoyalties.sol#L23-35

After report readout discussion with Neoki Multi Metaverse, they informed Truscova that this is a business decision from their side. Neoki doesn't want to interfere with customer URIs.

Recommendation

Add a function to allow updating URI later on.

12. Minting of infinite number of NKO tokens

Severity: High	Difficulty: Low
Type: Data validation	Finding ID: TCV-NK-12
Target: NikoToken	

Description

The “mint” function allows a caller with the MINTER_ROLE role to mint new tokens and assign them to a specified address. It ensures that only authorized parties are able to mint new tokens. However, there is no upper limit on how many tokens can be minted. Hence, NKO token can have theoretically infinite supply.

```
function mint(address to, uint256 amount) public {
        require(
           hasRole(MINTER_ROLE, msg.sender),
           "Caller does not have a MINTER_ROLE"
        );
        _mint(to, amount);
    }
```

Figure 11: mint function from NikoToken contract NKO.sol#L18-24

After report readout discussion with Neoki Multi Metaverse, they explained the concept and informed Truscova that it is a business decision.

Recommendation

Add information on maximum supply of the NKO token.

13. Not using scientific notation

Severity: Informational	Difficulty: -
Type: Notation	Finding ID: TCV-NK-13
Target: NikoToken	

Description

The constructor is assigning ROLES and also minting the initial supply of tokens. However, the initial supply is written in a format which is prone to errors. Literals with many digits are difficult to read and review.

```
constructor(address receiver) ERC20("Niko Token", "NKO") {
      _mint(receiver, 3000000000 * 10 ** decimals());
      _grantRole(DEFAULT_ADMIN_ROLE, receiver);
      _grantRole(MINTER_ROLE, receiver);
      _grantRole(BURNER_ROLE, receiver);
   }
```

Figure 13: constructor from NikoToken contract NKO.sol#L11-16

Recommendation

Always use Ether suffix, Time suffix, or the scientific notation.

14. Minting more lands than the maximum limit defined

Severity: Low	Difficulty: Medium
Type: Arithmetic	Finding ID: TCV-NK-14
Target: LandSell	

Description

The function “mintLands” allows an account with the ADMIN_ROLE role to mint a specified number of tokens from the land contract and assign them to the current contract. It ensures that only authorized parties are able to mint new tokens.

```
function mintLands(uint256 amount) external
onlyRole(ADMIN_ROLE) {
        for (uint256 index; index < amount; index++) {
           land.mint(address(this));
        }
    }
```

Figure 14: mintLands from LandSell contract [LandSell.sol#L44-48]

Exploit Scenario

ADMIN_ROLE can give large amount as input, where amount is greater than maximum number of lands defined. In the mint function of Lands, the modifier first checks if the current number of lands is less than or equal to maximum lands, and then increments the counter. This means if the maxLands is already reached, the condition will be satisfied. As next step, the currentAmountOfLands will increment by one (1). Hence, an additional land is minted.

Recommendation

In Lands.sol, the modifier “onlyMint” should be modified to check “currentAmountOfLands < maxLands” instead of “currentAmountOfLands <= maxLands”.

15. Reentrancy

Severity: Medium	Difficulty: Low
Type: Reentrancy	Finding ID: TCV-NK-15
Target: LandSell	

Description

The function "buyLand" allows a buyer to purchase LAND tokens from the sell contract. After a successful purchase, the ownership of the LAND tokens is transferred to the buyer and the corresponding Map structs are deleted from the landMap mapping. At the time of NKO safeTransferFrom function call, an attacker can re-enter buyLand. However, they won't be able

```
function buyLand(uint256[] memory tokenIds) external {
        for (uint256 index; index < tokenIds.length; index++) {
            Map memory map = landMap[tokenIds[index]];

            require(
                address(this) == land.ownerOf(map.tokenId),
                "Sell Contract does not own the LAND requested"
            );
            require(map.sellable == true, "LAND not for sell");
            nko.safeTransferFrom(msg.sender, map.owner, map.price);
            land.safeTransferFrom(address(this), msg.sender,
map.tokenId);
            delete landMap[tokenIds[index]];
        }
    }
```

Figure 15: buyLand function from LandSell contract *LandSell.sol#122-135*

to do any damage.

Recommendation

Use reentrancy guard from openzeppelin or apply check-effects-interactions pattern

16. URI_UPDATER role is never assigned

Severity: High	Difficulty: Low
Type: Access control	Finding ID: TCV-NK-16
Target: NeokiLands	

Description

The function "updateURI" allows a URI_UPDATER to update the baseURI. But the role URI_UPDATER is never assigned to anyone. Hence, updateURI function can never be executed.

```
function updateURI(string memory uri) public returns (string
memory) {
        require(
            hasRole(URI_UPDATER, msg.sender),
            "Caller does not have URI_UPDATER role"
        );
        baseURI = uri;
        return baseURI;
    }
```

Figure 16: updateURI function from NeokiLands contract Lands.sol#L79-86

This is a business decision from Neoki Multi Metaverse that they don't want to interfere with the URIs of any Lands sold to buyers. However, the contract never assigns the role of URI_UPDATER to any buyer.

Recommendation

Ensure that the role URI_UPDATER is properly assigned and correct access control is in place. In this regard, please implement a function to assign the URI_UPDATER role to buyers.

17. Reentrancy

Severity: Medium	Difficulty: Low
Type: Reentrancy	Finding ID: TCV-NK-17
Target: LandSell	

Description

The function “listLand” allows an owner of a LAND token to list the LAND token for sale at a specified price. It ensures that the LAND token is not already listed, that the caller is the owner of the LAND token, and that the price is greater than 0. After a successful listing, the LAND token ownership is transferred to the sell contract and the corresponding Map struct is added to the landMap mapping. However, they won’t be able to do any damage.

```
function listLand(uint256 tokenId, uint256 price) external {
        require(
           landMap[tokenId].tokenId == 0,
           "Cannot override existing LAND listed"
        );

        require(
           land.ownerOf(tokenId) == msg.sender,
           "Caller does not own LAND"
        );
        require(price > 0, "LAND price must be greater than 0");
        land.safeTransferFrom(msg.sender, address(this), tokenId);
        Map memory map = Map({
           tokenId: tokenId,
           owner: msg.sender,
           price: price,
           sellable: true
        });
        landMap[tokenId] = map;
    }
```

Figure 17: listLand function from LandSell contract LandSell.sol#157-176

Recommendation

Use reentrancy guard from openzeppelin or apply check-effects-interactions pattern.

18. Uninitialized local variables

Severity: Informational	Difficulty: -
Type: Initialization	Finding ID: TCV-NK-18
Target: LandSell	

Description

- LandSell.adminSetLandForSell(bool[],uint256[]).index (contracts/LandSell.sol#82) is a local variable never initialized

Recommendation

Initialize all the variables. If a variable is meant to be initialized to zero, explicitly set it to zero to improve code readability.

19. Local variable shadowing

Severity: Low	Difficulty: Low
Type: Data validation	Finding ID: TCV-NK-19
Target: NeokiLands	

Description

- NeokiLands.constructor(string,string,string)._symbol (contracts/Lands.sol#21) shadows:
    - ERC721._symbol (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#27) (state variable)

```
contract Bug {
    uint owner;

    function sensitive_function(address owner) public {
        // ...
        require(owner == msg.sender);
    }

    function alternate_sensitive_function() public {
        address owner = msg.sender;
        // ...
        require(owner == msg.sender);
    }
}
```

Figure 19: Local variable shadowing example

In Figure 19, “sensitive_function.owner” shadows “Bug.owner”. As a result, the use of owner in “sensitive_function” might be incorrect.

Recommendation

Rename the local variables that shadow another component.

20. Address validation for zero address

Severity: Low	Difficulty: -
Type: Data validation	Finding ID: TCV-NK-20
Target: LandSell	

Description

- LandSell.constructor(address,address,address)._treasury (contracts/LandSell.sol#36) lacks a zero-check on :
    - treasury = _treasury (contracts/LandSell.sol#39)

Recommendation

Check that the address is not zero.

21. Possible denial of service attack (DOS) – external calls inside loop

Severity: Low	Difficulty: Low
Type: Denial of Service	Finding ID: TCV-NK-21
Target: LandSell	

Description

Calls inside a loop might lead to a denial-of-service attack. However, this is a business decision of Neoki Multi Metaverse to keep the implementation the way it is.

- LandSell.mintLands(uint256) (contracts/LandSell.sol#44-48) has external calls inside a loop:
  - land.mint(address(this))
  - (contracts/LandSell.sol#46)

Recommendation

Use pull over push pattern for external calls. You may mint smaller batches of lands.

22. Item amount underflow

Severity: Medium	Difficulty: Medium
Type: Arithmetic	Finding ID: TCV-NK-22
Target: NeokiMarketplaceWithRoyalty	

Description

In buyItem function, it is never checked if marketplace has greater than or equal to the requested _amount. As a consequence, the item.amount can underflow. The transaction will revert.

```
…
asset.safeTransferFrom(
        address(this),
        msg.sender,
        item.tokenId,
        _amount,
        ""
    );
    item.amount -= _amount;

    for (uint i = 0; i < _amount; i++) {
        nftSold.increment();
    }
…
```

Figure 22: Snippet of buyItem function from NeokiMarketplaceWithRoyalty contract MarketplaceRoyalties.sol#164

Recommendation

Add a require statement to ensure that _amount is always less than or equal to item.amount.

23. Unused state variables

Severity: Informational	Difficulty: -
Type: Data validation	Finding ID: TCV-NK-23
Target: NeokiMarketplaceWithRoyalty	

Description

Unused state variables.

- NeokiMarketplaceWithRoyalty._INTERFACE_ID_ERC1155Receiver (contracts/MarketplaceRoyalties.sol#20) is never used in NeokiMarketplaceWithRoyalty (contracts/MarketplaceRoyalties.sol#14-370)

Recommendation

Remove unused state variables.

24. Reentrancy

Severity: Low	Difficulty: Medium
Type: Reentrancy	Finding ID: TCV-NK-24
Target: NeokiNftRoyalty	

Description

In “mint” and “mintWithRoyalties” function, an external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could

```
function mint(
        address account,
        uint256 amount,
        string memory tokenURI,
        bytes memory data
    ) public returns (uint256) {
        _tokenIdCounter.increment();
        uint tokenId = _tokenIdCounter.current();

        _mint(account, tokenId, amount, data);
        setTokenUri(tokenId, tokenURI);
        return tokenId;
    }

function mintWithRoyalties(
        address account,
        uint256 amount,
        string memory tokenURI,
        uint96 royalty,
        bytes memory data
    ) public returns (uint256) {
        _tokenIdCounter.increment();
        uint256 tokenId = _tokenIdCounter.current();

        _mint(account, tokenId, amount, data);
        setTokenUri(tokenId, tokenURI);
        _setTokenRoyalty(tokenId, account, royalty);
        return _tokenIdCounter.current();
```

Figure 24: mint and mintWithRoyalties functions from NeokiNftRoyalty contract
[NeokiNFTRoyalties.sol#L23-60]()

re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour.

Recommendation

Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

25. Low level calls

Severity: Informational	Difficulty: Low
Type: Configuration	Finding ID: TCV-NK-25
Target: NikoSell	

Description

The use of low-level calls is error-prone. Low-level calls do not check for code existence or call success.

- Low level call in NikoSell.buyNikoWithMatic(uint256) (contracts/NikoSell.sol#60-74):
    - (sent) = address(this).call{value: cost}() (contracts/NikoSell.sol#66)
    - (sentBack) = msg.sender.call{value: msg.value - cost}() (contracts/NikoSell.sol#70)

```
function buyNikoWithMatic(
        uint256 buyAmount
    ) external payable nonReentrant returns (bool) {
        uint256 matic = getMaticLatestPrice();
        uint256 cost = (buyAmount * ((nkoPrice * 1e18) / matic)) / 1e18;
        require(msg.value >= cost, "Sent insufficient MATIC");
        (bool sent, ) = address(this).call{value: cost}("");
        require(sent, "Failed to send ETH");
        NKO.safeTransfer(msg.sender, buyAmount);
        if (msg.value - cost > 0) {
            (bool sentBack, ) = msg.sender.call{value: msg.value - cost}("");
            require(sentBack, "Failed to send back remaining MATIC");
        }
        return sent;
    }
```

Figure 25: buyNikoWithMatic function in NikoSell contract NikoSell.sol#L60-74

Recommendation

Avoid low-level calls. Check the call success. If the call is meant for a contract, check for code existence.

26. Low level calls

Severity: Informational	Difficulty: Low
Type: Configuration	Finding ID: TCV-NK-26
Target: NikoSell	

Description

The use of low-level calls is error-prone. Low-level calls do not check for code existence or call success.

- Low level call in NikoSell.withdrawBalance(uint256) (contracts/NikoSell.sol#38-46):
    - (sent) = msg.sender.call{value: amount}() (contracts/NikoSell.sol#44)

```
function withdrawBalance(uint256 amount) external onlyRole(ADMIN_ROLE) {
        require(
            address(this).balance >= amount,
            "Insuffient balance for amount requested"
        );

        (bool sent, ) = msg.sender.call{value: amount}("");
        require(sent, "Failed to send ETH");
    }
```

Figure 26: withdrawBalance function in NikoSell contract [NikoSell.sol#L38-46]

Recommendation

Avoid low-level calls. Check the call success. If the call is meant for a contract, check for code existence.

27. Possible denial of service attack (DOS) – external calls inside loop

Severity: Low	Difficulty: Low
Type: Denial of Service	Finding ID: TCV-NK-27
Target: LandSell	

Description

Calls inside a loop might lead to a denial-of-service attack. However, this is a business decision of Neoki Multi Metaverse to keep the implementation the way it is.

- LandSell.definePricePerRange(uint256,uint256,uint256) (contracts/LandSell.sol#54-71) has external calls inside a loop:
    - require(bool,string)(land.ownerOf(index) == address(this),Sell Contract not owner of LAND)
    - (contracts/LandSell.sol#60-63)

Recommendation

Use pull over push pattern for external calls. You may use smaller batches of operations.

28. Possible denial of service attack (DOS) – external calls inside loop

Severity: Low	Difficulty: Low
Type: Denial of Service	Finding ID: TCV-NK-28
Target: LandSell	

Description

Calls inside a loop might lead to a denial-of-service attack. However, this is a business decision of Neoki Multi Metaverse to keep the implementation the way it is.

- LandSell.buyLand(uint256[]) (contracts/LandSell.sol#122-135) has external calls inside a loop:
    - require(bool,string)(address(this) == land.ownerOf(map.tokenId),Sell Contract does not own the LAND requested)
    - (contracts/LandSell.sol#126-129)

Recommendation

Use pull over push pattern for external calls. You may use smaller batches of operations.

29. Possible denial of service attack (DOS) – external calls inside loop

Severity: Low	Difficulty: Low
Type: Denial of Service	Finding ID: TCV-NK-29
Target: LandSell	

Description

Calls inside a loop might lead to a denial-of-service attack. However, this is a business decision of Neoki Multi Metaverse to keep the implementation the way it is.

- LandSell.buyLand(uint256[]) (contracts/LandSell.sol#122-135) has external calls inside a loop:
  - land.safeTransferFrom(address(this),msg.sender,map.tokenId)
  - (contracts/LandSell.sol#132)

Recommendation

Use pull over push pattern for external calls. You may use smaller batches of operations.

30. Address validation for zero address

Severity: Low	Difficulty: -
Type: Data validation	Finding ID: TCV-NK-30
Target: NeokiMarketplaceWithRoyalty	

Description

- NeokiMarketplaceWithRoyalty.constructor(address,address,address,address)._foundation (contracts/MarketplaceRoyalties.sol#49) lacks a zero-check on :
    - foundation = _foundation (contracts/MarketplaceRoyalties.sol#54)

Recommendation

Check that the address is not zero.

31. Address validation for zero address

Severity: Low	Difficulty: -
Type: Data validation	Finding ID: TCV-NK-31
Target: NeokiMarketplaceWithRoyalty	

Description

- NeokiMarketplaceWithRoyalty.constructor(address,address,address,address)._staking Pool (contracts/MarketplaceRoyalties.sol#50) lacks a zero-check on :
    - stakingPool = _stakingPool (contracts/MarketplaceRoyalties.sol#55)

Recommendation

Check that the address is not zero.

32. Address validation for zero address

Severity: Low	Difficulty: -
Type: Data validation	Finding ID: TCV-NK-32
Target: NeokiMarketplaceWithRoyalty	

Description

- NeokiMarketplaceWithRoyalty.updateStakingPool(address).newAddress (contracts/MarketplaceRoyalties.sol#324) lacks a zero-check on :
  - stakingPool = newAddress (contracts/MarketplaceRoyalties.sol#329)

Recommendation

Check that the address is not zero.

33. Address validation for zero address

Severity: Low	Difficulty: -
Type: Data validation	Finding ID: TCV-NK-33
Target: NeokiMarketplaceWithRoyalty	

Description

- NeokiMarketplaceWithRoyalty.updateFoundation(address).newAddress (contracts/MarketplaceRoyalties.sol#335) lacks a zero-check on :
    - foundation = newAddress (contracts/MarketplaceRoyalties.sol#340)

Recommendation

Check that the address is not zero.

34. Local variable shadowing

Severity: Low	Difficulty: Low
Type: Data validation	Finding ID: TCV-NK-34
Target: NeokiLands	

Description

- NeokiLands.constructor(string,string,string)._symbol (contracts/Lands.sol#21) shadows:
    - ERC721._symbol (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#27) (state variable)

Recommendation

Rename the local variables that shadow another component.

35. Uninitialized local variables

Severity: Informational	Difficulty: -
Type: Initialization	Finding ID: TCV-NK-35
Target: LandSell	

Description

LandSell.buyLand(uint256[]).index (contracts/LandSell.sol#123) is a local variable never initialized

Recommendation

Initialize all the variables. If a variable is meant to be initialized to zero, explicitly set it to zero to improve code readability.

36. Uninitialized local variables

Severity: Informational	Difficulty: -
Type: Initialization	Finding ID: TCV-NK-36
Target: LandSell	

Description

LandSell.mintLands(uint256).index (contracts/LandSell.sol#45) is a local variable never initialized

Recommendation

Initialize all the variables. If a variable is meant to be initialized to zero, explicitly set it to zero to improve code readability.

37. Uninitialized local variables

Severity: Informational	Difficulty: -
Type: Initialization	Finding ID: TCV-NK-37
Target: NeokiMarketplaceWithRoyalty	

Description

NeokiMarketplaceWithRoyalty.getAllItems().itemIndex (contracts/MarketplaceRoyalties.sol#197) is a local variable never initialized

Recommendation

Initialize all the variables. If a variable is meant to be initialized to zero, explicitly set it to zero to improve code readability.

38. Uninitialized local variables

Severity: Informational	Difficulty: -
Type: Initialization	Finding ID: TCV-NK-38
Target: NeokiMarketplaceWithRoyalty	

Description

NeokiMarketplaceWithRoyalty.buyItem(uint256,uint256).royaltyAmount (contracts/MarketplaceRoyalties.sol#139) is a local variable never initialized

Recommendation

Initialize all the variables. If a variable is meant to be initialized to zero, explicitly set it to zero to improve code readability.

39. Uninitialized local variables

Severity: Informational	Difficulty: -
Type: Initialization	Finding ID: TCV-NK-39
Target: NeokiMarketplaceWithRoyalty	

Description

NeokiMarketplaceWithRoyalty.buyItem(uint256,uint256).receiver (contracts/MarketplaceRoyalties.sol#138) is a local variable never initialized

Recommendation

Initialize all the variables. If a variable is meant to be initialized to zero, explicitly set it to zero to improve code readability.

Summary of Recommendations

The Neoki Multi Metaverse is a work in progress with multiple planned iterations. Truscova recommends that Neoki Multi Metaverse address the findings detailed in this report and take the following additional steps prior to deployment:

- Identify and analyze all system properties that are expected to hold.
- Use Fuzz testing with Foundry or Echidna to test and validate those system properties.
- Develop a detailed incident response plan to ensure that any issues that arise can be addressed promptly and without confusion.
- Ensure that all arithmetic is performed correctly.
- Ensure proper access control.
- Create good documentation.