SHELLBOXES

Niftopia

Smart Contract Security Audit

Prepared by ShellBoxes

July 22nd, 2022 - September 1st, 2022

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	Metaverse Trading
Version	1.0
Classification	Public

Scope

The Niftopia Contract in the Niftopia Repository

Repo	Commit Hash
https://github.com/wale-cyberpunk/niftopia-swaping	e21c7f2dfad461623adfe88160514575d4c8dcce
https://github.com/wale-cyberpunk/niftopia-swaping/tree/updated-contract	147a91e663a8a94859925815db690dd22cb1b98e

Files	MD5 Hash
Updated Swap contract & project/contract/Swap Connect.sol	55a48b6616fba794b31acab402268343
Marketing/contract/Marketing.sol	4b2c5b5ad76bbedff47e6699e324420a

Re-Audit Files

Files	MD5 Hash
Marketing.sol	f3fb2772e5a423837312de8ee836b960
swapContract.sol	f0555bcb5abc872b53706278eddee1ff

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction

Metaverse Trading engaged ShellBoxes to conduct a security assessment on the Niftopia beginning on July 22nd, 2022 and ending September 1st, 2022. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Metaverse Trading

Niftopia is a Secondary nft market that gives nft's owner a chance to generate income from their nft collection and to create a non-crypto currency trading market.
The Niftopia market weill be devided into two specific markets:

- Trading market: Here, users will be able to trade their nft's for other nft's. They can list their nft's to receive offers or send offers to other users. The offers may include a pack of nft's in exchange for one or several.
- Marketing market: Here users will be able to list their nft's so companies, artists or any person can buy the rights to use an nft in marketing campaigns or other promotional or artistic events.

Issuer	Metaverse Trading
Website	https://niftopia.io/
Type	Solidity Smart Contract
Audit Method	Whitebox

1.2 Approach & Methodology

ShellBoxes used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.2.1 Risk Methodology

Vulnerabilities or bugs identified by ShellBoxes are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

Impact				
	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low
			Likelihood	

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the Niftopia implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include 5 critical-severity, 2 high-severity, 5 medium-severity, 11 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
A.1. Token Holders Can Drain The Contract	CRITICAL	Fixed
A.2. Reentrancy Leads To The Draining Of The Contract	CRITICAL	Fixed
B.1. Reentrancy Leads To The Draining Of The Contract	CRITICAL	Fixed
B.2. Missing Verification Over valueOne, valueTwo And swapFee	CRITICAL	Fixed
B.3. Anyone Is Authorized To Close Any SwapIntent	CRITICAL	Fixed
A.3. The Collateral Can Be Withdrawn Anytime	HIGH	Fixed
A.4. Wrong endTime Calculation When Purchasing Marketing	HIGH	Fixed
A.5. Centralization Risk	MEDIUM	Fixed
A.6. User Might Purchase A Marketing For Free	MEDIUM	Fixed
A.7. Missmatch Between The Code And The Error Message	MEDIUM	Fixed
B.4. swapFee Is Only Utilized When The Swap Is Cancelled	MEDIUM	Fixed

B.5. Missing Transfer Verification	MEDIUM	Fixed
A.8. The marketingFee Variable Is Initialized But Not Implemented	LOW	Fixed
A.9. totalBalance Is Incompatible With The Contract Balance	LOW	Fixed
A.10. Avoid using .send() to transfer Ether	LOW	Fixed
A.11. Marketing Types And Options Can Be Duplicated	LOW	Fixed
A.12. For Loop Over Dynamic Array	LOW	Acknowledged
A.13. Usage of block.timestamp	LOW	Acknowledged
A.14. Floating Pragma	LOW	Fixed
B.6. Avoid using .transfer() to transfer Ether	LOW	Fixed
B.7. Missing Address Verification	LOW	Fixed
B.8. Owner Can Renounce Ownership	LOW	Fixed
B.9. Floating Pragma	LOW	Fixed

3 Finding Details

A Marketing.sol

A.1 Token Holders Can Drain The Contract [CRITICAL]

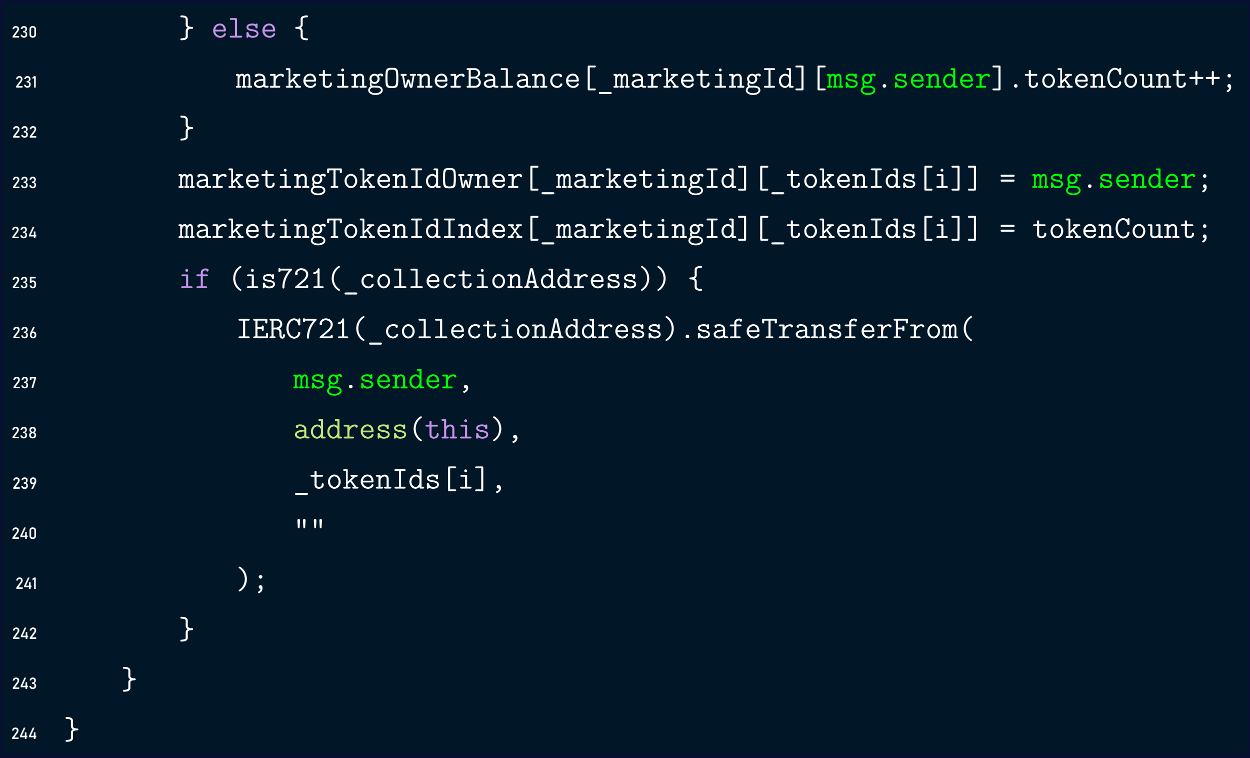
Description:

When a marketing is created, anyone can add items to it. When the marketing is purchased, the holders of the items have the option of withdrawing a portion of the marketing balance based on the number of items that were added. The token holder can drain the contract, by burning all of his marketing items, and then add them back, which would reset the withdrewAmount to zero. That will provide him infinite withdrawals and drain the contract's funds.

Code:

Listing 1: Marketing.sol

```
function _addItemInMarketing(
    uint256 _marketingId,
    address _collectionAddress,
    uint256[] memory _tokenIds
) private {
    uint256 tokenCount = marketings[_marketingId].tokenIds.length;
    for (uint256 i = 0; i < _tokenIds.length; i++) {
        marketings[_marketingId].tokenIds.push(_tokenIds[i]);
        tokenCount++;
        if ( marketingOwnerBalance[_marketingId][msg.sender].tokenCount == 0 ) {
            marketingOwnerBalance[_marketingId][msg.sender] = TokenOwner({
                tokenCount: 1,
                withdrewAmount: 0
            });
```



```
        } else {
            marketingOwnerBalance[_marketingId][msg.sender].tokenCount++;
        }
        marketingTokenIdOwner[_marketingId][_tokenIds[i]] = msg.sender;
        marketingTokenIdIndex[_marketingId][_tokenIds[i]] = tokenCount;
        if (is721(_collectionAddress)) {
            IERC721(_collectionAddress).safeTransferFrom(
                msg.sender,
                address(this),
                _tokenIds[i],
                ""
            );
        }
    }
}
```

Risk Level:

Likelihood – 5
Impact - 5

Recommendation:

When adding new items whenever the tokenCount is zero, it is advised to not reset the withdrewAmount.

Status - Fixed

The Niftopia team has fixed the issue by only incrementing tokenCount and not resetting the withdrewAmount.

A.2 Reentrancy Leads To The Draining Of The Contract [CRITICAL]

Description:

The withdrawCollateral and the withdrawMarketingAmount functions are exposed to a reentrancy attacks, a user can call the withdrawCollateral using a contract, if this contract contains in its fallback function a call to the same function the user can drain the contract since the collateral is set to zero after the transfer call, and the same scenario goes for the withdrawMarketingAmount function since the withdrewAmount is updated after the transfer call.

Code:

Listing 2: Marketing.sol

```
function withdrawCollateral(uint256 _marketingId, uint256 _purchaseId) external onlyPurchaseCreator(_marketingId, _purchaseId){
    require(_marketingId > 0 && _marketingId < marketingId, "markeing ID is not existing");
    Marketing storage marketing = marketings[_marketingId];
    if (!marketing.isExclusive) {
        marketings[_marketingId].currentPurchaseId = 0;
    }
    uint256 collateralAMount = marketingPurchases[_marketingId][_purchaseId].collateral;
     bool isSent = payable(msg.sender).send(collateralAMount);
    require(isSent, "Failed to send Ether");
    marketingPurchases[_marketingId][_purchaseId].collateral = 0;
    emit WithdrawCollateral(_marketingId, _purchaseId, collateralAMount, uint32(block.timestamp));
}
```

Listing 3: Marketing.sol

```
function withdrawMarketingAmount(uint256 _marketingId) external {
    require(_marketingId > 0 && _marketingId < marketingId, "range out
        ↪ of marketings");

    Marketing memory marketing = marketings[_marketingId];
    TokenOwner storage tokenOwner = marketingOwnerBalance[_marketingId][
        msg.sender
    ];
    uint256 withdrawableFees = _withdrawableMarketingFees( marketing,
        ↪ tokenOwner );
    bool isSent = payable(msg.sender).send(withdrawableFees);
    require(isSent, "Failed to send Ether");
    tokenOwner.withdrewAmount += withdrawableFees;
    emit WithdrawMarketingAmount( _marketingId, msg.sender,
        ↪ withdrawableFees, uint32(block.timestamp));
}
```

Risk Level:

Likelihood – 5
Impact - 5

Recommendation:

Consider setting the collateral to zero and updating the withdrewAmount before the transfer calls, an additional security layer can be added by using the nonReentrant modifier from the ReentrancyGuard contract.

Status - Fixed

The Niftopia team has fixed the issue by using the nonReentrant modifier from the ReentrancyGuard contract.

A.3 The Collateral Can Be Withdrawn Anytime [HIGH]

Description:

In order for a user to purchase a marketing, he has to pay the marketing.depositValue as a collateral in addition to the daily price depending on the duration of the purchase. However, the collateral can be withdrawn instantly using the withdrawCollateral function.

Code:

Listing 4: Marketing.sol

```
function withdrawCollateral(uint256 _marketingId, uint256 _purchaseId) external onlyPurchaseCreator(_marketingId, _purchaseId){
    require(_marketingId > 0 && _marketingId < marketingId, "markeing ID is not existing");
    Marketing storage marketing = marketings[_marketingId];
    if (!marketing.isExclusive) {
        marketings[_marketingId].currentPurchaseId = 0;
    }
    uint256 collateralAMount = marketingPurchases[_marketingId][_purchaseId].collateral;
     bool isSent = payable(msg.sender).send(collateralAMount);
    require(isSent, "Failed to send Ether");
    marketingPurchases[_marketingId][_purchaseId].collateral = 0;
    emit WithdrawCollateral(_marketingId, _purchaseId, collateralAMount, uint32(block.timestamp));
}
```

Risk Level:

Likelihood – 5
Impact - 4

Recommendation:

Consider locking the collateral until the end of the purchase duration to assure its value.

Status - Fixed

The Niftopia team has fixed the issue by mandating that the marketing expire before allowing withdrawal of the collateral.

A.4 Wrong endTime Calculation When Purchasing Marketing [HIGH]

Description:

The endTime is calculated the wrong way when purchasing a marketing, this attribute should be calculated using the following formula: marketing.startDate + 86400 * _duration or endTime + 86400 * _duration.

Code:

Listing 5: Marketing.sol

```
//check end date and duration compaire
uint32 endTime = uint32(block.timestamp);
if (endTime < marketing.startDate) {
    endTime = (marketing.startDate + 86400) * _duration;
} else {
    endTime = (endTime + 86400) * _duration;
}
```

Risk Level:

Likelihood – 4
Impact - 4

Recommendation:

Consider using the correct formula to calculate the endTime attribute when purchasing a marketing.

Status - Fixed

The Niftopia team has fixed the issue by using the correct formula to calculate the endTime attribute.

A.5 Centralization Risk [MEDIUM]

Description:

The burnMarketing function allows the marketing creator to burn the marketing without checking if someone has already a valid purchase. This represents a significant centralization risk where the marketing creator can cancel the user's purchases.

Code:

Listing 6: Marketing.sol

```
function burnMarketing(uint256 _marketingId) external onlyMarketingCreator(_marketingId) {
    require( _marketingId > 0 && _marketingId < marketingId, "range out of marketings" );

    Marketing memory marketing = marketings[_marketingId];

    marketings[_marketingId].isActive = false;

    if (!marketing.isCollection) {

        for (uint256 i = 0; i < marketing.tokenIds.length; i++) {

            IERC721(marketing.collection).safeTransferFrom(
                address(this),
                msg.sender,
                marketing.tokenIds[i],
                ""
            );

            if (marketingOwnerBalance[_marketingId][msg.sender].tokenCount > 0 ) {
                marketingOwnerBalance[_marketingId][msg.sender].tokenCount --;
            }
            marketingTokenIdOwner[_marketingId][marketing.tokenIds[i]] = address(0x0);
            marketingTokenIdIndex[_marketingId][marketing.tokenIds[i]] = 0;
            delete marketings[_marketingId].tokenIds[i];
        }

    }

    emit ChangeMarketingStatus(_marketingId, false, uint32(block.timestamp));
}
```

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

Consider preventing the marketing burn operation whenever there is a valid purchase in the marketing.

Status - Fixed

The issue has been resolved by the Niftopia team by making sure the marketing is not purchased before burning it.

A.6 User Might Purchase A Marketing For Free [MEDIUM]

Description:

Every time a marketing purchase is made under the contract, the purchaseMarketing function is used, which increases the marketing balance by ((marketing.dailyPrice * _duration) * 95) / 100. Due to a type conversion issue, if the value of marketing.dailyPrice is less than 2, the user will be permitted to purchase a marketing for free.

Code:

Listing 7: Marketing.sol

```
require(endTime <= marketing.endDate, "date range out");
require( msg.value == marketing.dailyPrice * _duration + marketing.depositValue, "you have to deposit enough money" );

uint256 _collateral = marketing.depositValue;
uint32 purchasedTime = uint32(block.timestamp);

Purchase memory purchase = Purchase({
    creator: payable(msg.sender),
    duration: _duration,
    collateral: _collateral,
    purchasedTime: purchasedTime,
    endTime: endTime
});
marketing.currentPurchaseId++;

marketingPurchases[_marketingId][marketing.currentPurchaseId] = purchase;
```

```
marketing.balance = marketing.balance + ((marketing.dailyPrice *
    ↪ _duration) * 95) / 100;

totalBalance += msg.value;
```

Risk Level:

Likelihood – 3
Impact - 4

Recommendation:

Consider requiring the value of the daily pricing to be higher or equal than two.

Status - Fixed

The Niftopia team has resolved the issue by requiring the daily price to be greater than 0.00001 ether which is equivalent to 10000000000000 Wei.

A.7 Missmatch Between The Code And The Error Message [MEDIUM]

Description:

The purchaseMarketing function contains a condition that makes sure if the marketing is not exclusive, the purchased cannot get completed if the marketing was already bought. However, the error message is: this marketing is exclusive and already was bought. which does not match with the condition that assures the marketing is not exclusive.

Code:

Listing 8: Marketing.sol

```
require(marketing.isActive, "this marketing is disabled");
if (!marketing.isExclusive) {
    require( marketing.currentPurchaseId == 0, "this marketing is
        ↪ exclusive and already was bought." );
}
```

Risk Level:

Likelihood – 4
Impact - 2

Recommendation:

Consider matching the code with what is mentioned in the error message.

Status - Fixed

The Niftopia team has resolved the issue by changing the condition in the if statement to match the error message.

A.8 The marketingFee Variable Is Initialized But Not Implemented [LOW]

Description:

The variable marketingFee is initialized, but it is not used to implement a fee structure in the contract.

Code:

Listing 9: Marketing.sol

```
uint256 public marketingFee = 0;
```

Risk Level:

Likelihood – 4
Impact - 2

Recommendation:

Consider utilizing the marketingFee variable to build a fee structure, or eliminating it if the fee structure is not part of the business logic.

Status - Fixed

The Niftopia team has resolved the issue by removing the marketingFee variable.

A.9 totalBalance Is Incompatible With The Contract Balance [LOW]

Description:

The totalBalance variable represents the contract balance. In the withdrawCollateral and withdrawMarketingAmount functions, this variable is not updated. Therefore, the getTotalBalanceOfContract function it will return an inaccurate value.

Code:

Listing 10: Marketing.sol

```
function getTotalBalanceOfContract() public view returns (uint256) {
    return totalBalance;
}
```

Risk Level:

Likelihood – 3
Impact - 1

Recommendation:

Consider updating the totalBalance variable in the withdrawCollateral and withdrawMarketingAmount functions, or using the address(this).balance to get the balance of the contract.

Status - Fixed

The Niftopia team has fixed the issue by removing the totalBalance variable and using the address(this).balance to get the balance of the contract.

A.10 Avoid using .send() to transfer Ether [LOW]

Description:

Although transfer() and send() are recommended as a security best-practice to prevent reentrancy attacks because they only forward 2300 gas, the gas repricing of opcodes may break deployed contracts.

Code:

Listing 11: Marketing.sol

```
function withdrawCollateral(uint256 _marketingId, uint256 _purchaseId) external onlyPurchaseCreator(_marketingId, _purchaseId){
    require(_marketingId > 0 && _marketingId < marketingId, "markeing ID is not existing");
    Marketing storage marketing = marketings[_marketingId];
    if (!marketing.isExclusive) {
        marketings[_marketingId].currentPurchaseId = 0;
    }
    uint256 collateralAMount = marketingPurchases[_marketingId][_purchaseId].collateral;
     bool isSent = payable(msg.sender).send(collateralAMount);
    require(isSent, "Failed to send Ether");
    marketingPurchases[_marketingId][_purchaseId].collateral = 0;
    emit WithdrawCollateral(_marketingId, _purchaseId, collateralAMount, uint32(block.timestamp));
}
```

Listing 12: Marketing.sol

```
function withdrawMarketingAmount(uint256 _marketingId) external {
    require(_marketingId > 0 && _marketingId < marketingId, "range out of marketings");

    Marketing memory marketing = marketings[_marketingId];
    TokenOwner storage tokenOwner = marketingOwnerBalance[_marketingId][
        msg.sender
    ];
    uint256 withdrawableFees = _withdrawableMarketingFees( marketing, tokenOwner );
    bool isSent = payable(msg.sender).send(withdrawableFees);
    require(isSent, "Failed to send Ether");
    tokenOwner.withdrewAmount += withdrawableFees;
    emit WithdrawMarketingAmount( _marketingId, msg.sender, withdrawableFees, uint32(block.timestamp));
}
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

Consider using .call{value: ... }("") instead, without hardcoded gas limits along with checks-effects-interactions pattern or reentrancy guards for reentrancy protection.

Status - Fixed

The Niftopia team has resolved the issue by using .call{value: ... }("") instead of send().

A.11 Marketing Types And Options Can Be Duplicated [LOW]

Description:

Due to the lack of a duplication check in the setMarketingTypes and setMarketingOptions functions, the same type or option can be added to the contract more than once, which might impact the contract's logic.

Code:

Listing 13: Marketing.sol

```solidity
function setMarketingTypes(string memory marketingType) external onlyAdmin {
    require(
        keccak256(abi.encodePacked(marketingType)) !=
            keccak256(abi.encodePacked("")),
        ""
    );
    marketingTypes[marketingTypeNumber] = marketingType;
    marketingTypeNumber++;
}
```

Listing 14: Marketing.sol

```solidity
function setMarketingOptions(string memory marketingOption)
    external
    onlyAdmin
{
    require(
        keccak256(abi.encodePacked(marketingOption)) !=
            keccak256(abi.encodePacked("")),
        ""
    );
    marketingOptions[marketingOptionNumber] = marketingOption;
    marketingOptionNumber++;
}
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

To prevent the marketing types and options from being duplicated, consider including a duplication check.

Status - Fixed

The Niftopia team has fixed the issue by preventing the marketing types and options from being duplicated.

A.12 For Loop Over Dynamic Array [LOW]

Description:

When smart contracts are deployed or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size array that grows in size over time can result in a Denial of Service attack. Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.

Code:

Listing 15: Marketing.sol

```
for (uint32 i = 0; i < _marketing.optionIds.length; i++) {
    require(
        _marketing.optionIds[i] > 0 && _marketing.optionIds[i] <
           ↪ marketingOptionNumber,
        "NiftopiaMarketing::Marketing options are incurrect"
    );
}
```

Listing 16: Marketing.sol

```
for (uint256 i = 0; i < _tokenIds.length; i++) {
    marketings[_marketingId].tokenIds.push(_tokenIds[i]);
    tokenCount++;
    if ( marketingOwnerBalance[_marketingId][msg.sender].tokenCount == 0
        ↪ ) {
        marketingOwnerBalance[_marketingId][msg.sender] = TokenOwner({
            tokenCount: 1,
            withdrewAmount: 0
        });
    } else {
        marketingOwnerBalance[_marketingId][msg.sender].tokenCount++;
    }
    marketingTokenIdOwner[_marketingId][_tokenIds[i]] = msg.sender;
    marketingTokenIdIndex[_marketingId][_tokenIds[i]] = tokenCount;
    if (is721(_collectionAddress)) {
        IERC721(_collectionAddress).safeTransferFrom(
            msg.sender,
            address(this),
            _tokenIds[i],
            ""
        );
    }
}
```

Listing 17: Marketing.sol

```
for (uint256 i = 0; i < _tokenIds.length; i++) {
    require( marketingTokenIdOwner[_marketingId][_tokenIds[i]] == msg.
        ↪ sender, "you don't owner of the token" );
    tokenIndex = marketingTokenIdIndex[_marketingId][_tokenIds[i]];
    delete marketings[_marketingId].tokenIds[tokenIndex];
    if (marketingOwnerBalance[marketingId][msg.sender].tokenCount > 0) {
        marketingOwnerBalance[marketingId][msg.sender].tokenCount--;
    }
    marketingTokenIdOwner[marketingId][_tokenIds[i]] = address(0x0);
    marketingTokenIdIndex[marketingId][_tokenIds[i]] = 0;
    if (is721(_collectionAddress)) {
        IERC721(_collectionAddress).safeTransferFrom(
            address(this),
            msg.sender,
            _tokenIds[i],
            ""
        );}
}
```

Listing 18: Marketing.sol

```
    for (uint256 i = 0; i < marketing.tokenIds.length; i++) {

        IERC721(marketing.collection).safeTransferFrom(
            address(this),
            msg.sender,
            marketing.tokenIds[i],
            ""
        );

        if (marketingOwnerBalance[_marketingId][msg.sender].tokenCount > 0 ) {
            marketingOwnerBalance[_marketingId][msg.sender].tokenCount--;
        }
        marketingTokenIdOwner[_marketingId][marketing.tokenIds[i]] = address(0x0);
        marketingTokenIdIndex[_marketingId][marketing.tokenIds[i]] = 0;
        delete marketings[_marketingId].tokenIds[i];
    }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

Avoid actions that involve looping across the entire data structure. If you really must loop over an array of unknown size, arrange for it to consume many blocks and thus multiple transactions.

Status - Acknowledged

The Niftopia team has acknowledged the issue, stating that there is no alternative to implement their business logic.

A.13 Usage of block.timestamp [LOW]

Description:

block.timestamp is used in the contract. The variable block is a set of variables. The timestamp does not always reflect the current time and may be inaccurate. The value of a block can be influenced by miners. Maximal Extractable Value attacks require a timestamp of up to 900 seconds. There is no guarantee that the value is right, all what is guaranteed is that it is higher than the timestamp of the previous block.

Code:

Listing 19: Marketing.sol

```
function createMarketing(Marketing memory _marketing) external {
    ensureIsNotZeroAddr(_marketing.collection);
    require(_marketing.dailyPrice > 0, "");
    require(_marketing.typeId > 0 && _marketing.typeId <
        ↪ marketingTypeNumber, "");
    require(_marketing.startDate > uint32(block.timestamp), "start date
        ↪ error");
    require(_marketing.endDate > _marketing.startDate, "end date error")
        ↪ ;
```

Listing 20: Marketing.sol

```
function updatePeriodOfMarketing(
    uint256 _marketingId,
    uint32 _startDate,
    uint32 _endDate
) external onlyMarketingCreator(_marketingId) {
    require(_startDate > uint32(block.timestamp), "start date error");
    require(_endDate > _startDate, "end date error");
    Marketing storage marketing = marketings[_marketingId];
    marketing.startDate = _startDate;
    marketing.endDate = _endDate;

    emit UpdatePeriodOfMarketing( _marketingId, _startDate, _endDate,
        ↪ uint32(block.timestamp) );
}
```

Listing 21: Marketing.sol

```
uint32 endTime = uint32(block.timestamp);
if (endTime < marketing.startDate) {
    endTime = (marketing.startDate + 86400) * _duration;
} else {
    endTime = (endTime + 86400) * _duration;
}
require(endTime <= marketing.endDate, "date range out");
require( msg.value == marketing.dailyPrice * _duration + marketing.
    ↪ depositValue, "you have to deposit enough money" );

uint256 _collateral = marketing.depositValue;
uint32 purchasedTime = uint32(block.timestamp);
```

Listing 22: Marketing.sol

```
uint32 endTime = uint32(block.timestamp);
if (endTime < marketing.startDate) {
    endTime = marketing.startDate + 86400 * _duration;
} else {
    endTime = endTime + 86400 * _duration;
}
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Verify that a delay of 900 seconds will not harm the logic of the contract.

Status - Acknowledged

The Niftopia team has acknowledged the issue, stating that 900 seconds delay will not impact the business logic.

A.14 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma 0.8.6. Contracts should be deployed using the same compiler version. Locking the pragma helps ensure that contracts will not unintentionally be deployed using another pragma, which in some cases may be an obsolete version, that may introduce issues to the contract system.

Code:

Listing 23: Marketing.sol

```solidity
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.6;
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Consider locking the pragma version. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status - Fixed

The Niftopia team has fixed the issue by locking the pragma version to 0.8.4.

B SwapConnect.sol

B.1 Reentrancy Leads To The Draining Of The Contract [CRITICAL]

Description:

The cancelSwapIntent function is exposed to a reentrancy attack, a user can call the cancelSwapIntent using a contract, if this contract contains in its fallback function a call to the same function the user can drain the contract. The reentrancy attack occurs in the transfer of the swapFee and the valueOne.

Code:

Listing 24: SwapConnect.sol

```
function cancelSwapIntent(uint256 _swapId) public {
    require(swapList[msg.sender][swapMatch[_swapId]].addressOne == msg.
        ↪ sender, "You're not the interested counterpart");
    require(swapList[msg.sender][swapMatch[_swapId]].status ==
        ↪ swapStatus.Opened, "Swap Status is not opened");
    //Rollback
    if(swapList[msg.sender][swapMatch[_swapId]].swapFee>0)
        payable(msg.sender).transfer(swapList[msg.sender][swapMatch[
            ↪ _swapId]].swapFee);
    uint256 i;
    for(i=0; i<nftsOne[_swapId].length; i++) {
        if(nftsOne[_swapId][i].typeStd == ERC20) {
            ERC20Interface(nftsOne[_swapId][i].dapp).transfer(swapList[
                ↪ msg.sender][swapMatch[_swapId]].addressOne, nftsOne[
                ↪ _swapId][i].blc[0]);
        }
        else if(nftsOne[_swapId][i].typeStd == ERC721) {
            ERC721Interface(nftsOne[_swapId][i].dapp).safeTransferFrom(
                ↪ address(this), swapList[msg.sender][swapMatch[_swapId
                ↪ ]].addressOne, nftsOne[_swapId][i].tokenId[0], nftsOne[
                ↪ _swapId][i].data);
        }
        else if(nftsOne[_swapId][i].typeStd == ERC1155) {
            ERC1155Interface(nftsOne[_swapId][i].dapp).
                ↪ safeBatchTransferFrom(address(this), swapList[msg.
                ↪ sender][swapMatch[_swapId]].addressOne, nftsOne[_swapId
                ↪ ][i].tokenId, nftsOne[_swapId][i].blc, nftsOne[_swapId
                ↪ ][i].data);
        }
        else {
            customInterface(dappRelations[nftsOne[_swapId][i].dapp]).
                ↪ bridgeSafeTransferFrom(nftsOne[_swapId][i].dapp,
                ↪ dappRelations[nftsOne[_swapId][i].dapp], swapList[msg.
                ↪ sender][swapMatch[_swapId]].addressOne, nftsOne[_swapId
                ↪ ][i].tokenId, nftsOne[_swapId][i].blc, nftsOne[_swapId
                ↪ ][i].data);
        }
    }

    if(swapList[msg.sender][swapMatch[_swapId]].valueOne > 0)
        swapList[msg.sender][swapMatch[_swapId]].addressOne.transfer(
            ↪ swapList[msg.sender][swapMatch[_swapId]].valueOne);
```

Risk Level:

Likelihood – 5
Impact - 5

Recommendation:

Consider using the nonReentrant modifier from the ReentrancyGuard contract.

Status - Fixed

The Niftopia team has fixed the issue by using the nonReentrant modifier from the ReentrancyGuard contract.

B.2 Missing Verification Over valueOne, valueTwo And swapFee [CRITICAL]

Description:

Users are able to exchange numerous assets in addition to a certain number of native tokens using the createSwapIntent function.
The swap closer should pay valueTwo, and the swap creator should pay valueOne + swapFee

of native tokens. The createSwapIntent function does not verify if msg.value and valueOne + swapFee are equivalent. The closeSwapIntent method does not ensure that msg.value is equivalent to valueTwo in the same way.

Code:

Listing 25: SwapConnect.sol

```
if(swapList[_swapCreator][swapMatch[_swapId]].valueTwo>0)
    swapList[_swapCreator][swapMatch[_swapId]].addressOne.transfer(
        ↪ swapList[_swapCreator][swapMatch[_swapId]].valueTwo);
```

Listing 26: SwapConnect.sol

```
if(swapList[_swapCreator][swapMatch[_swapId]].valueOne > 0)
    swapList[_swapCreator][swapMatch[_swapId]].addressTwo.transfer(
        ↪ swapList[_swapCreator][swapMatch[_swapId]].valueOne);
```

Risk Level:

Likelihood – 5
Impact - 5

Recommendation:

Consider verifying the valueOne + swapFee to be the same as msg.value in the createSwapIntent function, and verifying the valueTwo to be the same as msg.value in the closeSwapIntent function.

Status - Fixed

The issue has been resolved by the Niftopia team by making sure that the values of the parameters valueOne + swapFee and valueTwo in the functions createSwapIntent and closeSwapIntent are identical to msg.value.

B.3 Anyone Is Authorized To Close Any SwapIntent [CRITICAL]

Description:

While creating a swapIntent, the creator can specify the addressTwo to only allow it to be able to close the swap, or he can leave it as address(0) to allow anyone to close it. The closeSwapIntent function does not implement this logic, and it overwrites the addressTwo value with the msg.sender without checking its previous value for authorization.

Code:

Listing 27: SwapConnect.sol

```
function closeSwapIntent(address _swapCreator, uint256 _swapId) payable public whenNotPaused {
    require(swapList[_swapCreator][swapMatch[_swapId]].status == swapStatus.Opened, "Swap Status is not opened");

    swapList[_swapCreator][swapMatch[_swapId]].addressTwo = payable(msg.sender);
    swapList[_swapCreator][swapMatch[_swapId]].swapEnd = block.timestamp;
    swapList[_swapCreator][swapMatch[_swapId]].status = swapStatus.Closed;
```

Risk Level:

Likelihood – 5
Impact - 5

Recommendation:

Consider adding a condition in the closeSwapIntent function that allows only the addressTwo to close the swap and allows anyone to do so if the addressTwo is equal to the address(0).

Status - Fixed

The Niftopia team has fixed the issue by verifying the addressTwo to be the same as the msg.sender when the addressTwo is not set.

B.4 swapFee Is Only Utilized When The Swap Is Cancelled [MEDIUM]

Description:

The business logic states that the user should pay fees when creating a swap or accepting a swap. However, the contact utilizes the swapFee only when the swap is cancelled.

Code:

Listing 28: SwapConnect.sol

```
if(swapList[msg.sender][swapMatch[_swapId]].swapFee>0)
    payable(msg.sender).transfer(swapList[msg.sender][swapMatch[_swapId
        ↪ ]].swapFee);
```

Risk Level:

Likelihood – 0
Impact - 0

Recommendation:

Consider implementing the swap fee in the createSwapIntent function.

Status - Fixed

The Niftopia team has resolved the issue by implementing the swap fee in the createSwapIntent function.

B.5 Missing Transfer Verification [MEDIUM]

Description:

The ERC20 standard token implementation functions return the transaction status as a Boolean. It is a good practice to check for the return status of the function call to ensure that the transaction was executed successfully. It is the developer's responsibility to enclose these function calls with require() to ensure that, when the intended ERC20 function call returns false, the caller transaction also fails.

Code:

Listing 29: SwapConnect.sol

```
if(nftsOne[_swapIntent.id][i].typeStd == ERC20) {
    ERC20Interface(nftsOne[_swapIntent.id][i].dapp).transferFrom(
        ↪ _swapIntent.addressOne, address(this), nftsOne[_swapIntent.id
        ↪ ][i].blc[0]);
}
```

Listing 30: SwapConnect.sol

```
if(nftsTwo[_swapId][i].typeStd == ERC20) {
    ERC20Interface(nftsTwo[_swapId][i].dapp).transferFrom(swapList[
        ↪ _swapCreator][swapMatch[_swapId]].addressTwo, swapList[
        ↪ _swapCreator][swapMatch[_swapId]].addressOne, nftsTwo[_swapId
        ↪ ][i].blc[0]);
}
```

Listing 31: SwapConnect.sol

```
if(nftsOne[_swapId][i].typeStd == ERC20) {
    ERC20Interface(nftsOne[_swapId][i].dapp).transfer(swapList[
        ↪ _swapCreator][swapMatch[_swapId]].addressTwo, nftsOne[_swapId
        ↪ ][i].blc[0]);
}
```

Listing 32: SwapConnect.sol

```
if(nftsOne[_swapId][i].typeStd == ERC20) {
    ERC20Interface(nftsOne[_swapId][i].dapp).transfer(swapList[msg.
        ↪ sender][swapMatch[_swapId]].addressOne, nftsOne[_swapId][i].
        ↪ blc[0]);
}
```

Risk Level:

Likelihood – 2
Impact - 4

Recommendation:

Use the safeTransfer function from the safeERC20 Implementation, or put the transfer call inside an assert or require to verify that it returned true.

Status - Fixed

The Niftopia team has resolved the issue by using the safeTransfer function from the safeERC20 Implementation.

B.6 Avoid using .transfer() to transfer Ether [LOW]

Description:

Although transfer() and send() are recommended as a security best-practice to prevent reentrancy attacks because they only forward 2300 gas, the gas repricing of opcodes may break deployed contracts.

Code:

Listing 33: SwapConnect.sol

```
if(swapList[_swapCreator][swapMatch[_swapId]].valueTwo>0)
```

```
    swapList[_swapCreator][swapMatch[_swapId]].addressOne.transfer(
        ↪ swapList[_swapCreator][swapMatch[_swapId]].valueTwo);
```

Listing 34: SwapConnect.sol

```
if(swapList[_swapCreator][swapMatch[_swapId]].valueOne > 0)
    swapList[_swapCreator][swapMatch[_swapId]].addressTwo.transfer(
        ↪ swapList[_swapCreator][swapMatch[_swapId]].valueOne);
```

Listing 35: SwapConnect.sol

```
if(swapList[msg.sender][swapMatch[_swapId]].swapFee>0)
    payable(msg.sender).transfer(swapList[msg.sender][swapMatch[_swapId
        ↪ ]].swapFee);
```

Listing 36: SwapConnect.sol

```
if(swapList[msg.sender][swapMatch[_swapId]].valueOne > 0)
    swapList[msg.sender][swapMatch[_swapId]].addressOne.transfer(
        ↪ swapList[msg.sender][swapMatch[_swapId]].valueOne);
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

Consider using .call{ value: ... }("") instead, without hard-coded gas limits along with checks-effects-interactions pattern or reentrancy guards for reentrancy protection.

Status - Fixed

The Niftopia team has solved the issue by using .call{ value: ... }("") instead of .transfer().

B.7 Missing Address Verification [LOW]

Description:

Certain functions lack a safety check in the address, the address-type arguments should include a zero-address test, otherwise, the contract's functionality may become inaccessible. The _dapp and the _customInterface arguments should be verified to be different from address(0).

Code:

Listing 37: SwapConnect.sol

```
// Handle dapp relations for the bridges
function setDappRelation(address _dapp, address _customInterface) public onlyOwner {
    dappRelations[_dapp] = _customInterface;
}
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

We recommend that you make sure the addresses provided in the arguments are different from the address(0).

Status - Fixed

The Niftopia team has fixed the issue by verifying the _customInterface argument to be different from the address(0).

B.8 Owner Can Renounce Ownership [LOW]

Description:

Typically, the account that deploys the contract is also its owner. Consequently, the owner is able to engage in certain privileged activities in his own name. In smart contracts, the renounceOwnership function is used to renounce ownership, which means that if the contract's ownership has never been transferred, it will never have an Owner, rendering some owner-exclusive functionality unavailable.

Code:

Listing 38: SwapConnect.sol

```
contract SwapConnect is Ownable, Pausable, IERC721Receiver,
    ↪ IERC1155Receiver {
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

We recommend that you prevent the owner from calling renounceOwnership without first transferring ownership to a different address. Additionally, if you decide to use a multi-signature wallet, then the execution of the renounceOwnership will require for at least two or more users to be confirmed. Alternatively, you can disable Renounce Ownership functionality by overriding it.

Status - Fixed

The Niftopia team has resolved the issue by overriding the renounceOwnership function in order to disable the functionality.

B.9 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma 0.8.12. Contracts should be deployed using the same compiler version. Locking the pragma helps ensure that contracts will not unintentionally be deployed using another pragma, which in some cases may be an obsolete version, that may introduce issues to the contract system.

Code:

Listing 39: SwapConnect.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.12;
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Consider locking the pragma version. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version.

Status - Fixed

The Niftopia team has fixed the issue by locking the pragma version to 0.8.4.

4 Best Practices

BP.1 State variables that could be declared immutable

Description:

The above constant state variable should be declared immutable to save gas. Add the immutable attribute to state variables that never change after contact

Code:

Listing 40: SwapConnect.sol

```
address constant ERC20 = 0x90b7cf88476cc99D295429d4C1Bb1ff52448abeE;
address constant ERC721 = 0x58874d2951524F7f851bbBE240f0C3cF0b992d79;
address constant ERC1155 = 0xEDfdd7266667D48f3C9aB10194C3d325813d8c39;
```

Listing 41: SwapConnect.sol

```
uint256 constant secs = 86400;
```

BP.2 Unnecessary Initializations

Description:

When a variable is declared in solidity, it gets initialized with its type's default value. Thus, there is no need to initialize a variable with the default value.

Code:

Listing 42: Marketing.sol

```
uint256 tokenIndex = 0;
```

Listing 43: Marketing.sol

```
uint256 private totalBalance = 0;
```

Listing 44: Marketing.sol

```
uint256 public marketingFee = 0;
```

Listing 45: Marketing.sol

```
_marketing.currentPurchaseId = 0;
_marketing.balance = 0;
```

BP.3 Public Function Can Be Called External

Description:

Functions with a public scope that are not called inside the contract should be declared external to reduce the gas fees.

Code:

Listing 46: SwapConnect.sol

```
function createSwapIntent(swapIntent memory _swapIntent, swapStruct[]
    ↪ memory _nftsOne, swapStruct[] memory _nftsTwo) payable public
    ↪ whenNotPaused {
```

Listing 47: Marketing.sol

```
function calculateMarketingFees(uint256 _marketingId, address _ownerAddr
    ↪ ) public view returns (uint256) {
```

Listing 48: Marketing.sol

```
function getTokenOwnerData(address _tokenOwnerAddress, uint256
    ↪ _marketingId) public view returns (TokenOwner memory){
```

Listing 49: Marketing.sol

```
function getLastMarketingId() public view returns (uint256) {
```

Listing 50: Marketing.sol

```
function getMarketingDetail(uint256 _marketingId) public view returns ( Marketing memory) {
```

Listing 51: Marketing.sol

```
function getMarketingPurchaseDetail( uint256 _marketingId, uint256 _purchaseId) public view returns (Purchase memory) {
```

5 Static Analysis (Slither)

Description:

ShellBoxes expanded the coverage of the specific contract areas using automated testing methodologies. Slither, a Solidity static analysis framework, was one of the tools used. Slither was run on all-scoped contracts in both text and binary formats. This tool can be used to test mathematical relationships between Solidity instances statically and variables that allow for the detection of errors or inconsistent usage of the contracts' APIs throughout the entire codebase.

Results:

```
NiftMarketing._addItemInMarketing(uint256,address,uint256[]) (Marketing.
   ↪ sol#255-283) uses a dangerous strict equality:
       - marketingOwnerBalance[_marketingId][msg.sender].tokenCount == 0
          ↪  (Marketing.sol#264)
NiftMarketing.calculateMarketingFees(uint256,address) (Marketing.sol
   ↪ #564-570) uses a dangerous strict equality:
       - require(bool,string)(marketing.creator == _ownerAddr,Not
          ↪ marketing creator) (Marketing.sol#567)
NiftMarketing.onlyMarketingCreator(uint256) (Marketing.sol#124-131) uses
   ↪  a dangerous strict equality:
       - require(bool,string)(msg.sender == marketings[_marketingId].
          ↪ creator,You must be creator of this marketing) (Marketing.
          ↪ sol#126-129)
NiftMarketing.onlyPurchaseCreator(uint256,uint256) (Marketing.sol
   ↪ #133-145) uses a dangerous strict equality:
       - require(bool,string)(msg.sender == marketingPurchases[
          ↪ _marketingId][_purchaseId].creator,You must be creator of
          ↪ this purchase) (Marketing.sol#140-143)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #dangerous-strict-equalities
```

```
Reentrancy in NiftMarketing.burnMarketing(uint256) (Marketing.sol
    ↪ #503-532):
        External calls:
        - IERC721(marketing.collection).safeTransferFrom(address(this),
            ↪ msg.sender,marketing.tokenIds[i],) (Marketing.sol#514-519)
        State variables written after the call(s):
        - delete marketings[_marketingId].tokenIds[i] (Marketing.sol#526)
Reentrancy in NiftMarketing.createMarketing(NiftMarketing.Marketing) (
    ↪ Marketing.sol#175-222):
        External calls:
        - _addItemInMarketing(marketingId,_marketing.collection,tokenIds)
            ↪  (Marketing.sol#217)
                - IERC721(_collectionAddress).safeTransferFrom(msg.sender,
                    ↪ address(this),_tokenIds[i],) (Marketing.sol
                    ↪ #275-280)
        State variables written after the call(s):
        - marketingId ++ (Marketing.sol#219)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-1

NiftMarketing._addItemInMarketing(uint256,address,uint256[]) (Marketing.
    ↪ sol#255-283) has external calls inside a loop: IERC721(
    ↪ _collectionAddress).safeTransferFrom(msg.sender,address(this),
    ↪ _tokenIds[i],) (Marketing.sol#275-280)
NiftMarketing._burnMarketingItem(uint256,address,uint256[]) (Marketing.
    ↪ sol#298-322) has external calls inside a loop: IERC721(
    ↪ _collectionAddress).safeTransferFrom(address(this),msg.sender,
    ↪ _tokenIds[i],) (Marketing.sol#314-319)
NiftMarketing.burnMarketing(uint256) (Marketing.sol#503-532) has
    ↪ external calls inside a loop: IERC721(marketing.collection).
    ↪ safeTransferFrom(address(this),msg.sender,marketing.tokenIds[i],)
    ↪  (Marketing.sol#514-519)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ /#calls-inside-a-loop
```

```
Reentrancy in NiftMarketing.burnMarketing(uint256) (Marketing.sol
   ↪ #503-532):
       External calls:
       - IERC721(marketing.collection).safeTransferFrom(address(this),
          ↪ msg.sender,marketing.tokenIds[i],) (Marketing.sol#514-519)
       State variables written after the call(s):
       - marketingOwnerBalance[_marketingId][msg.sender].tokenCount -- (
          ↪ Marketing.sol#522)
       - marketingTokenIdIndex[_marketingId][marketing.tokenIds[i]] = 0
          ↪ (Marketing.sol#525)
       - marketingTokenIdOwner[_marketingId][marketing.tokenIds[i]] =
          ↪ address(0x0) (Marketing.sol#524)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #reentrancy-vulnerabilities-2

Reentrancy in NiftMarketing.addItemInMarketingCollection(uint256,address
   ↪ ,uint256[]) (Marketing.sol#240-253):
       External calls:
       - _addItemInMarketing(marketingId,_collectionAddress,_tokenIds) (
          ↪ Marketing.sol#250)
              - IERC721(_collectionAddress).safeTransferFrom(msg.sender,
                 ↪ address(this),_tokenIds[i],) (Marketing.sol
                 ↪ #275-280)
       Event emitted after the call(s):
       - AddItemInMarketing(_marketingId,_collectionAddress,_tokenIds,
          ↪ uint32(block.timestamp)) (Marketing.sol#252)
Reentrancy in NiftMarketing.burnMarketingItem(uint256,address,uint256[])
   ↪  (Marketing.sol#286-296):
       External calls:
       - _burnMarketingItem(_marketingId,_collectionAddress,_tokenIds) (
          ↪ Marketing.sol#294)
              - IERC721(_collectionAddress).safeTransferFrom(address(
                 ↪ this),msg.sender,_tokenIds[i],) (Marketing.sol
```

```
                    ↪ #314-319)
        Event emitted after the call(s):
        - BurnMarketingItems(_marketingId,_collectionAddress,_tokenIds,
            ↪ uint32(block.timestamp)) (Marketing.sol#295)
Reentrancy in NiftMarketing.createMarketing(NiftMarketing.Marketing) (
    ↪ Marketing.sol#175-222):
        External calls:
        - _addItemInMarketing(marketingId,_marketing.collection,tokenIds)
            ↪  (Marketing.sol#217)
                - IERC721(_collectionAddress).safeTransferFrom(msg.sender,
                    ↪ address(this),_tokenIds[i],) (Marketing.sol
                    ↪ #275-280)
        Event emitted after the call(s):
        - CreatMarketing(_marketing.collection,tokenIds,_marketing.typeId
            ↪ ,_marketing.optionIds,_marketing.isExclusive,_marketing.
            ↪ dailyPrice,_marketing.penaltyValue,_marketing.depositValue
            ↪ ,_marketing.startDate,_marketing.endDate,_marketing.
            ↪ isCollection,msg.sender,marketingId,uint32(block.timestamp
            ↪ )) (Marketing.sol#220)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #reentrancy-vulnerabilities-3

NiftMarketing.createMarketing(NiftMarketing.Marketing) (Marketing.sol
    ↪ #175-222) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(_marketing.dailyPrice > 0,) (Marketing.sol
            ↪ #177)
        - require(bool,string)(_marketing.typeId > 0 && _marketing.typeId
            ↪  < marketingTypeNumber,) (Marketing.sol#178)
        - require(bool,string)(_marketing.startDate > uint32(block.
            ↪ timestamp),start date error) (Marketing.sol#179)
        - require(bool,string)(_marketing.endDate > _marketing.startDate,
            ↪ end date error) (Marketing.sol#180)
```

```
       - require(bool,string)(_marketing.optionIds[i] > 0 && _marketing.
          ↪ optionIds[i] < marketingOptionNumber,NiftopiaMarketing::
          ↪ Marketing options are incurrect) (Marketing.sol#183-186)
       - require(bool,string)(is721(_marketing.collection),
          ↪ NiftopiaMarketing::Collection must be ERC721 token.) (
          ↪ Marketing.sol#213)
       - require(bool,string)(tokenIds.length > 0,token Id is required)
          ↪ (Marketing.sol#216)
NiftMarketing.updatePeriodOfMarketing(uint256,uint32,uint32) (Marketing.
   ↪ sol#225-237) uses timestamp for comparisons
       Dangerous comparisons:
       - require(bool,string)(_startDate > uint32(block.timestamp),start
          ↪  date error) (Marketing.sol#230)
NiftMarketing._addItemInMarketing(uint256,address,uint256[]) (Marketing.
   ↪ sol#255-283) uses timestamp for comparisons
       Dangerous comparisons:
       - i < _tokenIds.length (Marketing.sol#261)
       - marketingOwnerBalance[_marketingId][msg.sender].tokenCount == 0
          ↪  (Marketing.sol#264)
NiftMarketing.burnMarketingItem(uint256,address,uint256[]) (Marketing.
   ↪ sol#286-296) uses timestamp for comparisons
       Dangerous comparisons:
       - require(bool,string)(marketings[_marketingId].currentPurchaseId
          ↪  == 0,can't withdraw items) (Marketing.sol#292)
NiftMarketing._burnMarketingItem(uint256,address,uint256[]) (Marketing.
   ↪ sol#298-322) uses timestamp for comparisons
       Dangerous comparisons:
       - marketingOwnerBalance[marketingId][msg.sender].tokenCount > 0 (
          ↪ Marketing.sol#308)
NiftMarketing.purchaseMarketing(uint256,uint8) (Marketing.sol#331-403)
   ↪ uses timestamp for comparisons
       Dangerous comparisons:
       - endTime < marketing.startDate (Marketing.sol#345)
       - require(bool,string)(endTime <= marketing.endDate,date range
          ↪ out) (Marketing.sol#350)
       - require(bool,string)(marketing.proposal.endtime < block.
          ↪ timestamp,voting result is not finalized) (Marketing.sol
          ↪ #358)
NiftMarketing.vote(uint256,bool) (Marketing.sol#405-433) uses timestamp
   ↪ for comparisons
       Dangerous comparisons:
       - require(bool,string)(marketing.proposal.endtime > block.
          ↪ timestamp,voting result is not finalized) (Marketing.sol
          ↪ #412)
NiftMarketing.placeProposal(uint256,uint256) (Marketing.sol#435-477)
   ↪ uses timestamp for comparisons
       Dangerous comparisons:
       - marketing.proposal.endtime < block.timestamp (Marketing.sol
          ↪ #445)
NiftMarketing.upgradeDurationOfPurchase(uint256,uint256,uint8) (
   ↪ Marketing.sol#479-501) uses timestamp for comparisons
       Dangerous comparisons:
       - require(bool,string)(_duration > purchase.duration,_duration >
          ↪ purchase.duration) (Marketing.sol#482)
       - require(bool,string)(msg.value == marketing.dailyPrice * (
          ↪ _duration - purchase.duration),you have to deposit enough
          ↪ money) (Marketing.sol#483)
       - endTime < marketing.startDate (Marketing.sol#486)
       - require(bool,string)(endTime <= marketing.endDate,date range
          ↪ out) (Marketing.sol#493)
NiftMarketing.burnMarketing(uint256) (Marketing.sol#503-532) uses
   ↪ timestamp for comparisons
       Dangerous comparisons:
       - i < marketing.tokenIds.length (Marketing.sol#512)
       - marketingOwnerBalance[_marketingId][msg.sender].tokenCount > 0
          ↪ (Marketing.sol#521)
```

```
NiftMarketing.withdrawCollateral(uint256,uint256) (Marketing.sol
    ↪ #535-546) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(isSent,Failed to send Ether) (Marketing.
            ↪ sol#543)
NiftMarketing.withdrawMarketingAmount(uint256) (Marketing.sol#549-561)
    ↪ uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(isSent,Failed to send Ether) (Marketing.
            ↪ sol#558)
NiftMarketing.calculateMarketingFees(uint256,address) (Marketing.sol
    ↪ #564-570) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(marketing.creator == _ownerAddr,Not
            ↪ marketing creator) (Marketing.sol#567)
NiftMarketing.getTokenOwnerData(address,uint256) (Marketing.sol#582-586)
    ↪  uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(marketingOwnerBalance[_marketingId][
            ↪ _tokenOwnerAddress].tokenCount > 0,address is bad) (
            ↪ Marketing.sol#584)
NiftMarketing.getMarketingDetail(uint256) (Marketing.sol#592-603) uses
    ↪ timestamp for comparisons
        Dangerous comparisons:
        - marketing.proposal.endtime < block.timestamp (Marketing.sol
            ↪ #597)
NiftMarketing.getTimeDiffence(uint256) (Marketing.sol#605-608) uses
    ↪ timestamp for comparisons
        Dangerous comparisons:
        - (marketing.proposal.endtime < block.timestamp) (Marketing.sol
            ↪ #607)
NiftMarketing.getMarketingPurchaseDetail(uint256,uint256) (Marketing.sol
    ↪ #610-615) uses timestamp for comparisons
        Dangerous comparisons:
```

```
       - require(bool,string)(_purchaseId > 0 && _purchaseId <=
          ↪ marketings[_marketingId].currentPurchaseId,purchase id
          ↪ error) (Marketing.sol#612)
NiftMarketing.ensureIsMarketingAssets(uint256,address,uint256[]) (
   ↪ Marketing.sol#617-627) uses timestamp for comparisons
       Dangerous comparisons:
       - require(bool,string)(marketings[_marketingId].isCollection,can'
          ↪ t add items in personal marketing) (Marketing.sol#623)
       - require(bool,string)(_collectionAddress == marketings[
          ↪ _marketingId].collection,collection address must be same
          ↪ to marketing collection address.) (Marketing.sol#625)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #block-timestamp

Different versions of Solidity is used:
       - Version used: ['^0.8.0', '^0.8.6']
       - ^0.8.0 (../../../openzeppelin-contracts/contracts/security/
          ↪ ReentrancyGuard.sol#3)
       - ^0.8.0 (../../../openzeppelin-contracts/contracts/token/ERC1155
          ↪ /IERC1155.sol#3)
       - ^0.8.0 (../../../openzeppelin-contracts/contracts/token/ERC1155
          ↪ /IERC1155Receiver.sol#3)
       - ^0.8.0 (../../../openzeppelin-contracts/contracts/token/ERC1155
          ↪ /utils/ERC1155Holder.sol#3)
       - ^0.8.0 (../../../openzeppelin-contracts/contracts/token/ERC1155
          ↪ /utils/ERC1155Receiver.sol#3)
       - ^0.8.0 (../../../openzeppelin-contracts/contracts/token/ERC20/
          ↪ ERC20.sol#3)
       - ^0.8.0 (../../../openzeppelin-contracts/contracts/token/ERC20/
          ↪ IERC20.sol#3)
       - ^0.8.0 (../../../openzeppelin-contracts/contracts/token/ERC20/
          ↪ extensions/IERC20Metadata.sol#3)
       - ^0.8.0 (../../../openzeppelin-contracts/contracts/token/ERC721/
          ↪ IERC721.sol#3)
```

```
        - ^0.8.0 (../../../openzeppelin-contracts/contracts/token/ERC721/
           ↪ IERC721Receiver.sol#3)
        - ^0.8.0 (../../../openzeppelin-contracts/contracts/token/ERC721/
           ↪ utils/ERC721Holder.sol#3)
        - ^0.8.0 (../../../openzeppelin-contracts/contracts/utils/Context
           ↪ .sol#3)
        - ^0.8.0 (../../../openzeppelin-contracts/contracts/utils/
           ↪ introspection/ERC165.sol#3)
        - ^0.8.0 (../../../openzeppelin-contracts/contracts/utils/
           ↪ introspection/IERC165.sol#3)
        - ^0.8.6 (Marketing.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #different-pragma-directives-are-used

Context._msgData() (../../../openzeppelin-contracts/contracts/utils/
   ↪ Context.sol#20-22) is never used and should be removed
ERC20._burn(address,uint256) (../../../openzeppelin-contracts/contracts/
   ↪ token/ERC20/ERC20.sol#274-289) is never used and should be
   ↪ removed
ERC20._mint(address,uint256) (../../../openzeppelin-contracts/contracts/
   ↪ token/ERC20/ERC20.sol#251-261) is never used and should be
   ↪ removed
NiftMarketing.ensureIsZeroAddr(address) (Marketing.sol#641-643) is never
   ↪  used and should be removed
NiftMarketing.is1155(address) (Marketing.sol#633-635) is never used and
   ↪ should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #dead-code

Pragma version^0.8.0 (../../../openzeppelin-contracts/contracts/security
   ↪ /ReentrancyGuard.sol#3) necessitates a version too recent to be
   ↪ trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../../../openzeppelin-contracts/contracts/token/
   ↪ ERC1155/IERC1155.sol#3) necessitates a version too recent to be
```

```
    ↪ trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../../../openzeppelin-contracts/contracts/token/
    ↪ ERC1155/IERC1155Receiver.sol#3) necessitates a version too recent
    ↪  to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../../../openzeppelin-contracts/contracts/token/
    ↪ ERC1155/utils/ERC1155Holder.sol#3) necessitates a version too
    ↪ recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../../../openzeppelin-contracts/contracts/token/
    ↪ ERC1155/utils/ERC1155Receiver.sol#3) necessitates a version too
    ↪ recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../../../openzeppelin-contracts/contracts/token/
    ↪ ERC20/ERC20.sol#3) necessitates a version too recent to be
    ↪ trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../../../openzeppelin-contracts/contracts/token/
    ↪ ERC20/IERC20.sol#3) necessitates a version too recent to be
    ↪ trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../../../openzeppelin-contracts/contracts/token/
    ↪ ERC20/extensions/IERC20Metadata.sol#3) necessitates a version too
    ↪  recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../../../openzeppelin-contracts/contracts/token/
    ↪ ERC721/IERC721.sol#3) necessitates a version too recent to be
    ↪ trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../../../openzeppelin-contracts/contracts/token/
    ↪ ERC721/IERC721Receiver.sol#3) necessitates a version too recent
    ↪ to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../../../openzeppelin-contracts/contracts/token/
    ↪ ERC721/utils/ERC721Holder.sol#3) necessitates a version too
    ↪ recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../../../openzeppelin-contracts/contracts/utils/
    ↪ Context.sol#3) necessitates a version too recent to be trusted.
    ↪ Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../../../openzeppelin-contracts/contracts/utils/
    ↪ introspection/ERC165.sol#3) necessitates a version too recent to
    ↪ be trusted. Consider deploying with 0.6.12/0.7.6
```

```
Pragma version^0.8.0 (../../../openzeppelin-contracts/contracts/utils/
    ↪ introspection/IERC165.sol#3) necessitates a version too recent to
    ↪  be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.6 (Marketing.sol#2) necessitates a version too recent
    ↪  to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.6 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #incorrect-versions-of-solidity

Parameter NiftMarketing.createMarketing(NiftMarketing.Marketing).
    ↪ _marketing (Marketing.sol#175) is not in mixedCase
Parameter NiftMarketing.updatePeriodOfMarketing(uint256,uint32,uint32).
    ↪ _marketingId (Marketing.sol#226) is not in mixedCase
Parameter NiftMarketing.updatePeriodOfMarketing(uint256,uint32,uint32).
    ↪ _startDate (Marketing.sol#227) is not in mixedCase
Parameter NiftMarketing.updatePeriodOfMarketing(uint256,uint32,uint32).
    ↪ _endDate (Marketing.sol#228) is not in mixedCase
Parameter NiftMarketing.addItemInMarketingCollection(uint256,address,
    ↪ uint256[])._marketingId (Marketing.sol#241) is not in mixedCase
Parameter NiftMarketing.addItemInMarketingCollection(uint256,address,
    ↪ uint256[])._collectionAddress (Marketing.sol#242) is not in
    ↪ mixedCase
Parameter NiftMarketing.addItemInMarketingCollection(uint256,address,
    ↪ uint256[])._tokenIds (Marketing.sol#243) is not in mixedCase
Parameter NiftMarketing.burnMarketingItem(uint256,address,uint256[]).
    ↪ _marketingId (Marketing.sol#287) is not in mixedCase
Parameter NiftMarketing.burnMarketingItem(uint256,address,uint256[]).
    ↪ _collectionAddress (Marketing.sol#288) is not in mixedCase
Parameter NiftMarketing.burnMarketingItem(uint256,address,uint256[]).
    ↪ _tokenIds (Marketing.sol#289) is not in mixedCase
Parameter NiftMarketing.changeMarketingStatus(uint256)._marketingId (
    ↪ Marketing.sol#325) is not in mixedCase
Parameter NiftMarketing.purchaseMarketing(uint256,uint8)._marketingId (
    ↪ Marketing.sol#331) is not in mixedCase
```

```
Parameter NiftMarketing.purchaseMarketing(uint256,uint8)._duration (
    ↪ Marketing.sol#331) is not in mixedCase
Parameter NiftMarketing.vote(uint256,bool)._marketingId (Marketing.sol
    ↪ #405) is not in mixedCase
Parameter NiftMarketing.vote(uint256,bool)._vote (Marketing.sol#405) is
    ↪ not in mixedCase
Parameter NiftMarketing.placeProposal(uint256,uint256)._offerPrice (
    ↪ Marketing.sol#435) is not in mixedCase
Parameter NiftMarketing.placeProposal(uint256,uint256)._marketingId (
    ↪ Marketing.sol#435) is not in mixedCase
Parameter NiftMarketing.upgradeDurationOfPurchase(uint256,uint256,uint8)
    ↪ ._marketingId (Marketing.sol#479) is not in mixedCase
Parameter NiftMarketing.upgradeDurationOfPurchase(uint256,uint256,uint8)
    ↪ ._purchaseId (Marketing.sol#479) is not in mixedCase
Parameter NiftMarketing.upgradeDurationOfPurchase(uint256,uint256,uint8)
    ↪ ._duration (Marketing.sol#479) is not in mixedCase
Parameter NiftMarketing.burnMarketing(uint256)._marketingId (Marketing.
    ↪ sol#503) is not in mixedCase
Parameter NiftMarketing.withdrawCollateral(uint256,uint256)._marketingId
    ↪  (Marketing.sol#535) is not in mixedCase
Parameter NiftMarketing.withdrawCollateral(uint256,uint256)._purchaseId
    ↪ (Marketing.sol#535) is not in mixedCase
Parameter NiftMarketing.withdrawMarketingAmount(uint256)._marketingId (
    ↪ Marketing.sol#549) is not in mixedCase
Parameter NiftMarketing.calculateMarketingFees(uint256,address).
    ↪ _marketingId (Marketing.sol#564) is not in mixedCase
Parameter NiftMarketing.calculateMarketingFees(uint256,address).
    ↪ _ownerAddr (Marketing.sol#564) is not in mixedCase
Parameter NiftMarketing.getTokenOwnerData(address,uint256).
    ↪ _tokenOwnerAddress (Marketing.sol#582) is not in mixedCase
Parameter NiftMarketing.getTokenOwnerData(address,uint256)._marketingId
    ↪ (Marketing.sol#582) is not in mixedCase
Parameter NiftMarketing.getMarketingDetail(uint256)._marketingId (
    ↪ Marketing.sol#592) is not in mixedCase
```

```
Parameter NiftMarketing.getTimeDiffence(uint256)._marketingId (Marketing
    ↪ .sol#605) is not in mixedCase
Parameter NiftMarketing.getMarketingPurchaseDetail(uint256,uint256).
    ↪ _marketingId (Marketing.sol#610) is not in mixedCase
Parameter NiftMarketing.getMarketingPurchaseDetail(uint256,uint256).
    ↪ _purchaseId (Marketing.sol#610) is not in mixedCase
Parameter NiftMarketing.ensureIsMarketingAssets(uint256,address,uint256
    ↪ []).._marketingId (Marketing.sol#618) is not in mixedCase
Parameter NiftMarketing.ensureIsMarketingAssets(uint256,address,uint256
    ↪ [])._collectionAddress (Marketing.sol#619) is not in mixedCase
Parameter NiftMarketing.ensureIsMarketingAssets(uint256,address,uint256
    ↪ [])._tokenIds (Marketing.sol#620) is not in mixedCase
Parameter NiftMarketing.is721(address)._nft (Marketing.sol#629) is not
    ↪ in mixedCase
Parameter NiftMarketing.is1155(address)._nft (Marketing.sol#633) is not
    ↪ in mixedCase
Parameter NiftMarketing.ensureIsNotZeroAddr(address)._addr (Marketing.
    ↪ sol#637) is not in mixedCase
Parameter NiftMarketing.ensureIsZeroAddr(address)._addr (Marketing.sol
    ↪ #641) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #conformance-to-solidity-naming-conventions

Reentrancy in NiftMarketing.withdrawCollateral(uint256,uint256) (
    ↪ Marketing.sol#535-546):
        External calls:
        - isSent = address(msg.sender).send(collateralAMount) (Marketing.
            ↪ sol#542)
        State variables written after the call(s):
        - marketingPurchases[_marketingId][_purchaseId].collateral = 0 (
            ↪ Marketing.sol#544)
        Event emitted after the call(s):
        - WithdrawCollateral(_marketingId,_purchaseId,collateralAMount,
            ↪ uint32(block.timestamp)) (Marketing.sol#545)
```

```
Reentrancy in NiftMarketing.withdrawMarketingAmount(uint256) (Marketing.
   ↪ sol#549-561):
      External calls:
      - isSent = address(msg.sender).send(withdrawableFees) (Marketing.
         ↪ sol#557)
      State variables written after the call(s):
      - tokenOwner.withdrewAmount += withdrawableFees (Marketing.sol
         ↪ #559)
      Event emitted after the call(s):
      - WithdrawMarketingAmount(_marketingId,msg.sender,
         ↪ withdrawableFees,uint32(block.timestamp)) (Marketing.sol
         ↪ #560)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #reentrancy-vulnerabilities-4

NiftMarketing.marketingFee (Marketing.sol#51) should be constant
NiftMarketing.votingDays (Marketing.sol#44) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
   ↪ #state-variables-that-could-be-declared-constant

onERC1155Received(address,address,uint256,uint256,bytes) should be
   ↪ declared external:
      - ERC1155Holder.onERC1155Received(address,address,uint256,uint256
         ↪ ,bytes) (../../../openzeppelin-contracts/contracts/token/
         ↪ ERC1155/utils/ERC1155Holder.sol#11-19)
onERC1155BatchReceived(address,address,uint256[],uint256[],bytes) should
   ↪  be declared external:
      - ERC1155Holder.onERC1155BatchReceived(address,address,uint256[],
         ↪ uint256[],bytes) (../../../openzeppelin-contracts/
         ↪ contracts/token/ERC1155/utils/ERC1155Holder.sol#21-29)
name() should be declared external:
      - ERC20.name() (../../../openzeppelin-contracts/contracts/token/
         ↪ ERC20/ERC20.sol#61-63)
symbol() should be declared external:
        - ERC20.symbol() (../../../openzeppelin-contracts/contracts/token
            ↪ /ERC20/ERC20.sol#69-71)
decimals() should be declared external:
        - ERC20.decimals() (../../../openzeppelin-contracts/contracts/
            ↪ token/ERC20/ERC20.sol#86-88)
totalSupply() should be declared external:
        - ERC20.totalSupply() (../../../openzeppelin-contracts/contracts/
            ↪ token/ERC20/ERC20.sol#93-95)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (../../../openzeppelin-contracts/
            ↪ contracts/token/ERC20/ERC20.sol#100-102)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (../../../openzeppelin-
            ↪ contracts/contracts/token/ERC20/ERC20.sol#112-115)
allowance(address,address) should be declared external:
        - ERC20.allowance(address,address) (../../../openzeppelin-
            ↪ contracts/contracts/token/ERC20/ERC20.sol#120-122)
approve(address,uint256) should be declared external:
        - ERC20.approve(address,uint256) (../../../openzeppelin-contracts
            ↪ /contracts/token/ERC20/ERC20.sol#131-134)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (../../../
            ↪ openzeppelin-contracts/contracts/token/ERC20/ERC20.sol
            ↪ #149-163)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (../../../openzeppelin
            ↪ -contracts/contracts/token/ERC20/ERC20.sol#177-180)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (../../../openzeppelin
            ↪ -contracts/contracts/token/ERC20/ERC20.sol#196-204)
onERC721Received(address,address,uint256,bytes) should be declared
    ↪ external:
        - ERC721Holder.onERC721Received(address,address,uint256,bytes)
            ↪ (../../../openzeppelin-contracts/contracts/token/ERC721/
        ↪ utils/ERC721Holder.sol#19-26)
vote(uint256,bool) should be declared external:
        - NiftMarketing.vote(uint256,bool) (Marketing.sol#405-433)
placeProposal(uint256,uint256) should be declared external:
        - NiftMarketing.placeProposal(uint256,uint256) (Marketing.sol
            ↪ #435-477)
calculateMarketingFees(uint256,address) should be declared external:
        - NiftMarketing.calculateMarketingFees(uint256,address) (
            ↪ Marketing.sol#564-570)
getTokenOwnerData(address,uint256) should be declared external:
        - NiftMarketing.getTokenOwnerData(address,uint256) (Marketing.sol
            ↪ #582-586)
getLastMarketingId() should be declared external:
        - NiftMarketing.getLastMarketingId() (Marketing.sol#588-590)
getMarketingDetail(uint256) should be declared external:
        - NiftMarketing.getMarketingDetail(uint256) (Marketing.sol
            ↪ #592-603)
getTimeDiffence(uint256) should be declared external:
        - NiftMarketing.getTimeDiffence(uint256) (Marketing.sol#605-608)
getMarketingPurchaseDetail(uint256,uint256) should be declared external:
        - NiftMarketing.getMarketingPurchaseDetail(uint256,uint256) (
            ↪ Marketing.sol#610-615)
getTotalBalanceOfContract() should be declared external:
        - NiftMarketing.getTotalBalanceOfContract() (Marketing.sol
            ↪ #645-647)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #public-function-that-could-be-declared-external
```

Conclusion:

Most of the vulnerabilities found by the analysis have already been addressed by the smart contract code review.

6 Conclusion

In this audit, we examined the design and implementation of Niftopia contract and discovered several issues of varying severity. Metaverse Trading team addressed 21 issues raised in the initial report and implemented the necessary fixes, while classifying the rest as a risk with low-probability of occurrence. Shellboxes' auditors advised Metaverse Trading Team to maintain a high level of vigilance and to keep those findings in mind in order to avoid any future complications.

SHELLBOXES

For a Contract Audit, contact us at contact@shellboxes.com