SHELLBOXES

Unicrypt's Marketplace Smart Contracts

Smart Contract Security Audit

Prepared by ShellBoxes

Dec 24th, 2022 - Jan 9th, 2023

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	Unicrypt
Version	1.0
Classification	Public

Scope

Repository	Commit Hash
https://github.com/chainsulting/nft-marketplace-audit	abf82208fd4c8bd9f4817244ac3a67a595aaf53c

Re-Audit

Repository	Commit Hash
https://github.com/chainsulting/nft-marketplace	842d44a247e9e676a7ad7d3743f65d1f1b12b38e

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction

Unicrypt engaged ShellBoxes to conduct a security assessment on the Unicrypt's Marketplace Smart Contracts beginning on Dec 24th, 2022 and ending Jan 9th, 2023. 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 Unicrypt

Started in June 2020, Unicrypt provides an ever-growing suite of decentralized services. The objective is to bring value to the DeFi space as a whole by delivering disruptive, flexible and audited technology. Strengthen your project and reward your communities using our services.

Issuer	Unicrypt
Website	`https://unicrypt.network/`
Type	Solidity Smart Contract
Documentation	`https://docs.unicrypt.network`
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 Disclaimer

This audit report highlights security issues that were identified within the scope of the audit, which includes all smart contracts in the NFT Marketplace repository. It should be noted that during the Re-Audit phase, a new contract named MarketplaceOffers was added to the NFT Marketplace repository in the fixes commit. While this new contract is inherited by the main marketplace contract, it was not included in our scope of the audit. We are not responsible for any issues that may be present within this new contract, and it is the client's responsibility to ensure the security and integrity of this contract.

Despite the client's developers having performed unit tests with 100% coverage of the audited contracts, the client has not taken any action to address or mitigate the risks associated with most of the identified issues in this report. Therefore, we advise the client to take the necessary action to fix as many issues as possible in their next version of the project to ensure the security and integrity of their smart contracts.

2.2 Summary

The following is a synopsis of our conclusions from our analysis of the Unicrypt's Marketplace Smart Contracts 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.

2.3 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 , 2 high-severity, 5 medium-severity, 2 low-severity, 1 undetermined-severity vulnerabilities.

Vulnerabilities	Severity	Status
SHB.1. Bypassing Ownership Requirements in OwnableMultiple Contract	HIGH	Fixed
SHB.2. Unchecked NFT contract in _createNewListing function	HIGH	Acknowledged
SHB.3. EOA Verification Missing In _bidOnListing Function	MEDIUM	Acknowledged
SHB.4. Cancellation Fees Can Be Bypassed Using The endAuction Function	MEDIUM	Fixed
SHB.5. Possible Re-Entrancy Attacks	MEDIUM	Acknowledged
SHB.6. Unlimited Listing and Cancellation Fees Can Lead to a Denial of Service	MEDIUM	Acknowledged
SHB.7. Centralization of Power in Owner Role	MEDIUM	Acknowledged
SHB.8. Front-Run That Can Lead To Dos	LOW	Acknowledged
SHB.9. Loss of Precision in The Calculation of feeAmount	LOW	Acknowledged
SHB.10. Missmatch Between The Code And Natspec	UNDETERMINED	Mitigated

3 Finding Details

SHB.1 Bypassing Ownership Requirements in OwnableMultiple Contract

- Severity : HIGH
- Status : Fixed
- Likelihood : 2
- Impact : 3

Description:

The OwnableMultiple contract includes a function called removeOwners which allows an owner to remove other owners from the contract. However, this function can be exploited by an owner to bypass the requirement that the contract must always have at least one owner.

Exploit Scenario:

1. A malicious owner calls the addOwner function and adds the address 0x0 as an owner.
2. The attacker then calls the removeOwners function and removes all other owners from the contract, except for the address 0x0.
3. As a result, the contract is left with only one owner, the address 0x0. This contradicts the intended behavior of the contract, which requires that it always has at least one owner.

Files Affected:

SHB.1.1: OwnableMultiple.sol

```
function _addOwner(address _newOwner) internal {
       bool exists = false;
       for (uint256 j = 0; j < owners.length; j++) {
           if (owners[j] == _newOwner) {
              exists = true;
              break;
           }
       }
       if (!exists) {
           owners.push(_newOwner);
           emit OwnerAdded(_newOwner);
       }
   }
```

SHB.1.2: OwnableMultiple.sol

```
function removeOwners(address[] memory _oldOwners) external onlyOwner {
        require(
            _oldOwners.length < ownersLength(),
            "Ownable: Can not remove all owners"
        );
        _removeOwners(_oldOwners);
    }
```

Recommendation:

It is recommended that the contract implements a check to prohibit the 0x0 address from being added as an owner. This will prevent an owner from bypassing the requirement of having at least one owner in the contract and potentially leaving the contract with no owners. This can be achieved by adding a check in the addOwner function to ensure that the address being added is not equal to 0x0. This check should be placed before any other checks or updates to the contract state, to ensure that the 0x0 address is not added as an owner.

Updates

The Unicrypt team has resolved the issue by implementing a two-step ownable process. This process includes the addition of a new feature which allows an owner to propose the addition of a new address and add it to the pendingOwners mapping using the proposeOwner function. The proposed owner must then call the acceptOwnership function to officially become an owner of the contract.This added layer of oversight ensures that the contract always has at least one owner, effectively preventing the exploit identified during the security audit.

SHB.1.3: OwnableMultiple.sol

```
    function _proposeOwner(address _newOwner) internal {
        require(_newOwner != address(0), "Ownable: Zero address");
        require(!owners[_newOwner], "Ownable: Owner already exists");
        require(!pendingOwners[_newOwner], "Ownable: Pending already
            ↪ exists");
        pendingOwners[_newOwner] = true;
        emit PendingOwnerAdded(_newOwner);
    }
```

SHB.1.4: OwnableMultiple.sol

```
    function acceptOwnership() external {
        require(pendingOwners[msg.sender], "Ownable: no pending owner");
        _addOwner(msg.sender);
        delete pendingOwners[msg.sender];
        emit PendingOwnerRemoved(msg.sender);
    }
```

SHB.2 Unchecked NFT contract in _createNewListing function

- Severity : HIGH
- Status : Acknowledged
- Likelihood : 3
- Impact : 2

Description:

The _createNewListing function accepts an address representing an NFT contract as an input parameter. The function then calls the safeTransferNFTFrom function on this contract to transfer the NFT item to the marketplace contract. However, the contract calling _createNewListing does not verify that the contract at the given address is actually an NFT contract that supports the safeTransferNFTFrom function.

Exploit Scenario:

1. The attacker creates a malicious contract with the same function signature as the safeTransferNFTFrom function.
2. The attacker then passes this contract's address as the _collection parameter to _createNewListing
3. When a user buys the malicious NFT, the malicious NFT contract then executes its malicious action, transferring all the user's funds to the attacker's account.

This exploit allows the attacker to transfer funds from the user to the attacker's account using the marketplace contract as a means of facilitating the transfer.

Files Affected:

SHB.2.1: MarketPlace.sol

```
function _createNewListing(
        address _seller,
        address _collection,
        uint256 _itemId,
        uint256 _itemAmount,
        uint256 _startPrice,
        uint256 _buyPrice,
        uint256 _enddate
    ) private returns (uint256) {
        // check input parameters
        require(_collection.code.length > 0, "Marketplace: Invalid
           ↪ collection");
        require(_startPrice <= _buyPrice, "Marketplace: Invalid prices");
        require(_buyPrice > 10_000, "Marketplace: Buy price too low");
        require(_enddate > block.timestamp, "Marketplace: Invalid end date");

        // transfer item to market place
        (bool success, bool isERC721) = INFT(_collection).safeTransferNFTFrom(
            _seller,
            address(this),
            _itemId,
            _itemAmount
        );
        require(success, "Marketplace: Sending NFT failed")
```

Recommendation:

To mitigate this issue, the contract calling _createNewListing should verify that the contract at the given address is a valid NFT contract by checking its implementation of the IERC721 or IERC1155 interface. This can be done using the IERC721.supportsInterface or IERC1155.supportsInterface functions. If the contract does not implement the necessary interface, the _createNewListing function should reject the input.

Updates

The Unicrypt team acknowledged the risk, stating that it is desired to support as many NFTs as possible and, relying on EIP165, excludes NFT contract which might not have supportsInterface function. Additionally, this function alone does not guarantee the absence of malicious code. Even some well known NFTs do not implement these standards and will be excluded (like CryptoPunks).

SHB.3 EOA Verification Missing In _bidOnListing Function

- Severity : MEDIUM
- Status : Acknowledged

- Likelihood : 3
- Impact : 1

Description:

The function _bidOnListing does not verify that msg.sender is an external-owned account (EOA). This allows a contract to bid on a listing and potentially execute malicious code. It also contradicts the comments cited in the code.

Files Affected:

SHB.3.1: Marketplace.sol

```
     /* @custom:requirement Requirements:
     *
     * - listing with `_listingId` must exist.
     * - `_amount` must be greater than current bid amount and start price
     * - `_amount` must be lower than or equal to buy price
     * - The listing's end date must be greater than current timestamp
     * - `msg.sender` must be an external owned account (not a contract)
     *
     * Emits {BidOnListing} or {NFTSold} event.
     */
    function _bidOnListing(uint256 _listingId, uint256 _amount) private {
        // check listing parameter
        ListingLib.Listing storage listing = listings[_listingId];
        require(
            block.timestamp < listing.endDate,
            "Marketplace: Auction has ended"
        );
        require(
            (listing.currentPrice == 0 && listing.startPrice <= _amount)
                (listing.currentPrice != 0 && listing.currentPrice <
                   ↪ _amount),
            "Marketplace: Bid too low"
        );
        require(_amount <= listing.buyPrice, "Marketplace: Bid too high")
            ↪ ;

        // refund old top bidder
        if (listing.topBidder != address(0)) {
            _safeTransferETHWithFallback(
                listing.topBidder,
                listing.currentPrice
            );
        }

        // update listing
        listing.topBidder = msg.sender;
        listing.currentPrice = _amount;
```

Recommendation:

Consider adding a check to the _bidOnListing function to verify that msg.sender is an externally owned account (EOA) rather than a contract. This can be done by using the isExternalAccount function. This function takes an address as an input and returns a boolean value indicating whether the account at that address is an EOA or a contract.

SHB.3.2: Marketplace.sol

```
    function isExternalAccount(address account) internal view returns (bool)
        ↪  {
          return account.code.length == 0;
        }
```

Here's an example of how you could use the isExternalAccount function in the _bidOnListing function :

SHB.3.3: Marketplace.sol

```
    // Check if the sender is an EOA
    if (!isExternalAccount(msg.sender)) {
        // msg.sender is a contract, throw an error
        revert("Contracts are not allowed to bid on listings");
    }
```

By adding this check to the _bidOnListing function, you can help ensure that only EOAs are able to bid on listings and prevent contracts from attempting to do so.

Updates

The client acknowledged this issue, stating it is unlikely for a bidder being a malicious smart contract to cause an issue. Additionally, they will fix that mismatch between the code and the natspec comment by adjusting the documentation.

SHB.4 Cancellation Fees Can Be Bypassed Using The endAuction Function

- Severity : MEDIUM
- Status : Fixed
- Likelihood : 2
- Impact : 2

Description:

The endAuction function allows users to cancel a listing without paying a cancellation fee, while the cancelListing function requires a cancellation fee. This indicates that there is a misalignment between the two functions, and some users could potentially bypass the fee by using the endAuction function instead of the cancelListing function.

Exploit Scenario:

1. A seller creates a listing on the marketplace.
2. The end date of the listing arrives.
3. The seller then calls the endAuction function to cancel the listing, without paying the cancellation fee.

Files Affected:

SHB.4.1: Marketplace.sol

```
    function cancelListing(uint256 _listingId) external payable override {
        // check for sufficient cancellation fee
        require(
            protocolFees.cancellationFee(listings[_listingId].collection) ==
                msg.value,
            "Marketplace: Invalid cancellation fee"
        );

        _cancelListing(_listingId);

        // transfer cancellation fee to fee receiver
        _transferFee(msg.value);
    }
```

SHB.4.2: Marketplace.sol

```
    function endAuction(uint256 _listingId) external override {
        _endAuction(_listingId);
    }
```

Recommendation:

Consider implementing one of the following recommendations:

1. Charge the lister in the endAuction function when the endDate has passed and there are no bidders on the auction. This ensures that the sellers who choose to end their listing using the endAuction function are still required to pay a fee.
2. Do not charge the seller when calling the cancelListing function after the endDate and no bids were paid in the auction. This eliminates the discrepancy between the two functions and ensures that all users are subject to the same fee structure.

Additionally, consider updating the documentation and the comments within the code to explain the purpose and behavior of the endAuction function, and its relationship with the cancellation fee.

Updates

The Unicrypt team has fixed this issue by implementing the second recommendation we have stated, which is not charging the seller when calling the cancelListing function after the endDate by adding a require statement which reverts when calling the cancelListing if the endDate has reached.

SHB.4.3: Marketplace.sol

```
    function _cancelListing(uint256 _listingId, uint256 _cancellationFeeAmount)
        private
     {
        // check listing parameter
        ListingLib.Listing memory listing = listings[_listingId];
        require(listing.seller == msg.sender, "Marketplace: Sender not seller");
        require(
            listing.topBidder == address(0),
            "Marketplace: Listing has bidder"
        );
        require(
            listing.endDate > block.timestamp,
            "Marketplace: Listing ended"
        );

        // delete listing from state
        delete listings[_listingId];
```

SHB.5 Possible Re-Entrancy Attacks

- Severity : MEDIUM
- Status : Acknowledged
- Likelihood : 1
- Impact : 3

Description:

The function createNewListing is vulnerable to a re-entrancy attack because it calls an external contract function _safeTransferETHWithFallback before updating the state of the contract.

A re-entrancy attack occurs when an attacker is able to repeatedly call an external contract function that can modify the state of the contract, before the state of the contract is updated. In this case, the attacker could call _safeTransferETHWithFallback multiple times before the state of the contract is updated, allowing them to potentially exploit the contract's logic.

The same issue has been found in the _finalizeAuction function.

Exploit Scenario:

1. The owner will change the address of protocolFees.feeTo() to an address that he controls and can modify the fallback function in order to call other functions once again.

Files Affected:

SHB.5.1: Marketplace.sol

```
    function createNewListing(
        address _collection,
        uint256 _itemId,
        uint256 _itemAmount,
        uint256 _startPrice,
        uint256 _buyPrice,
        uint256 _enddate
    ) external payable override {
        // check for sufficient listing fee
        require(
            protocolFees.listingFee(_collection) == msg.value,
            "Marketplace: Invalid listing fee"
        );

        // create new listing
        _createNewListing(
            msg.sender,
            _collection,
            _itemId,
            _itemAmount,
            _startPrice,
            _buyPrice,
            _enddate
        );

        // transfer listing fee to fee receiver
        _transferFee(msg.value);
```

SHB.5.2: Marketplace.sol

```
function _finalizeAuction(uint256 _listingId) private {
        ListingLib.Listing memory listing = listings[_listingId];

        // delete listing from state
        delete listings[_listingId];

        // transfer NFT to buyer
        (bool sentNFT, ) = INFT(listing.collection).safeTransferNFTFrom(
            address(this),
            listing.topBidder,
            listing.itemId,
            listing.itemAmount
        );

        if (sentNFT) {
            // emit sold event
            emit NFTSold(
               listing.collection,
               listing.seller,
               listing.topBidder,
               _listingId,
               listing.itemId,
               listing.itemAmount,
               listing.currentPrice
            );

            // calculate buying fee and transfer to fee receiver
            uint256 feeAmount = (protocolFees.buyingFee(listing.collection) *
               listing.currentPrice) / protocolFees.FEE_DENOMINATOR();
            _safeTransferETHWithFallback(protocolFees.feeTo(), feeAmount);

            // transfer buying amount to seller
            _safeTransferETHWithFallback(
               listing.seller,
               listing.currentPrice - feeAmount
            );
        } else {
            // refund seller and buyer if buyer rejects receiving NFT
            emit CancelListing(
               listing.seller,
               listing.collection,
               _listingId,
               listing.itemId,
               listing.itemAmount
            );
            _safeTransferETHWithFallback(
               listing.topBidder,
               listing.currentPrice
            );
            INFT(listing.collection).safeTransferNFTFrom(
               address(this),
               listing.seller,
               listing.itemId,
               listing.itemAmount
            );
        }
    }
```

SHB.5.3: Marketplace.sol

```
function _transferFee(uint256 _amount) private {
        require(msg.value >= _amount, "Marketplace: Insufficient fee
            ↪ amount");
        _safeTransferETHWithFallback(protocolFees.feeTo(), _amount);
    }
```

Recommendation:

To prevent reentrancy attacks in the functions, you can use the ReentrancyGuard library from OpenZeppelin.

Updates

The Unicrypt team acknowledged this issue due to its unlikeliness to happen as per their statement.

SHB.6 Unlimited Listing and Cancellation Fees Can Lead to a Denial of Service

- Severity : MEDIUM
- Status : Acknowledged

- Likelihood : 1
- Impact : 3

Description:

The function _setFees in the contract allows an owner to set the listing, cancellation, and buying fees for the marketplace without any limitation. This can potentially lead to a denial of service attack, where an attacker can set the fees to extremely high values, making it impossible for users to list or cancel their items on the marketplace.

Files Affected:

SHB.6.1: ProtocolFees.sol

```
    function _setFees(
        uint256 _listingFee,
        uint256 _cancellationFee,
        uint256 _buyingFee
    ) private {
        require(_buyingFee < MAX_RELATIVE_FEE, "ProtocolFees: Fee too
            ↪ high");
        defaultListingFee = _listingFee;
        defaultCancellationFee = _cancellationFee;
        defaultBuyingFee = _buyingFee;
    }
```

Recommendation:

To mitigate this issue, it is recommended to add limitations on the fees that can be set through the _setFees function. For example, the fees could be capped at a maximum value or set as a percentage of the item's price. This would prevent attackers from setting excessively high fees and ensure the usability of the marketplace for all users.

Updates

The Unicrypt team acknowledged this issue, stating that it is unlikely to happen.

SHB.7 Centralization of Power in Owner Role

- Severity : MEDIUM
- Status : Acknowledged
- Likelihood : 1
- Impact : 3

Description:

The removeOwners function allows any contract owner to remove other owners from the contract. This creates a centralization of power within the owner role, as a malicious owner could potentially remove all other owners and have complete control over the contract.

Files Affected:

SHB.7.1: OwnableMultiple.sol

```
function removeOwners(address[] memory _oldOwners) external onlyOwner {
    require(
        _oldOwners.length < ownersLength(),
        "Ownable: Can not remove all owners"
    );
    _removeOwners(_oldOwners);
}
```

Recommendation:

Consider implementing additional checks and controls to prevent a single owner from removing all other owners. This could include requiring a majority vote from the remaining owners before allowing an owner to be removed, or implementing a limit on the number of owners that can be removed at one time.

Updates

The Unicrypt team acknowledged this issue, stating that it is unlikely to happen.

SHB.8 Front-Run That Can Lead To Dos

- Severity : LOW
- Status : Acknowledged
- Likelihood : 1
- Impact : 2

Description:

There is a potential front-running attack in the cancelListing function. If the owner of the contract is able to change the cancellationFee for a specific collection at a critical moment before the listing is cancelled, the transaction will revert.

Exploit Scenario:

For example, consider the following scenario:

1. A user creates a listing for an NFT from collection A with a listing fee of 1 ETH and a cancellation fee of 0.5 ETH.
2. The owner of the contract changes the cancellation fee for collection A to 10 ETH.
3. The user decides to cancel their listing.
4. The cancelListing function checks that the cancellation fee for collection A is 10 ETH and requires the user to pay this fee in order to cancel the listing.
5. The user pays the 0.5 ETH fee and the transaction is reverted.

In this scenario, the owner of the contract was able to front-run the user by changing the cancellation fee at a critical moment. This could be considered unfair to the user and could also potentially discourage them from using the marketplace in the future.

Files Affected:

SHB.8.1: MarketPlace.sol

```
function cancelListing(uint256 _listingId) external payable override {
        // check for sufficient cancellation fee
        require(
            protocolFees.cancellationFee(listings[_listingId].collection)
                ↪  ==
                msg.value,
            "Marketplace: Invalid cancellation fee"
        );
        _cancelListing(_listingId);

        // transfer cancellation fee to fee receiver
        _transferFee(msg.value);
}
```

Recommendation:

To prevent this kind of attack, it would be necessary to ensure that the fees for a specific collection cannot be changed while a listing is active. This could be achieved by adding a state variable to the contract that tracks whether the fees for a specific collection have been changed since the last listing was created, and preventing changes to the fees if this variable is set to true.

Updates

The Unicrypt team acknowledged this issue for the reason being it is unlikely to happen.

SHB.9 Loss of Precision in The Calculation of feeAmount

- Severity : LOW
- Status : Acknowledged
- Likelihood : 1
- Impact : 2

Description:

The _finalizeAuction function calculates the buying fee by multiplying the protocolFees.buyingFee(listing.collection) value by listing.currentPrice and dividing the result by protocolFees.FEE_DENOMINATOR(). However, this calculation can lead to a loss of precision as it is performed using integer division, which discards any remainder. This can result in a fee being transferred lower than intended, potentially leading to a loss of funds for the contract.

Exploit Scenario:

An attacker could potentially take advantage of the fact that Solidity does not support floating point values to avoid paying fees when buying an NFT. If the buyingFee is set to 2500, which is the maximum value for this fee, and the currentPrice of the NFT is less than 4, the result of buyingFee * currentPrice will be less than 10000 (the denominator). This means that no fees will be acquired, even though the buyingFee is set to a high value.

Files Affected:

SHB.9.1: Marketplace.sol

```
// calculate buying fee and transfer to fee receiver
uint256 feeAmount =(protocolFees.buyingFee(listing.collection) * listing.currentPrice)/protocolFees.FEE\_DENOMINATOR();

_safeTransferETHWithFallback(protocolFees.feeTo(), feeAmount);
```

Recommendation:

To prevent this issue, it is recommended to perform the calculation using a fixed point math library or by storing the fee amounts in a smaller denomination and multiplying by the correct factor when needed. This will ensure that the full fee amount is transferred as intended without any loss of precision.

Updates

The Unicrypt team acknowledged this issue, stating that it is a tolerable amount for their business logic as far as it is less than the DENOMINATOR.

SHB.10 Missmatch Between The Code And Natspec

- Severity : UNDETERMINED
- Status : Mitigated
- Likelihood : 3
- Impact : -

Description:

The _createNewListing function has a requirement in its natspec documentation that states _startPrice must be lower than _buyPrice. However, the implementation in the code checks if _startPrice is less than or equal to _buyPrice. This could potentially allow listings to be created with _startPrice equal to _buyPrice, which may not be the intended behavior.

Updates

The Unicrypt team fixed this issue, by updating the natspec for the _createNewListing but the issue still remains in createNewListing.

SHB.10.1: Marketplace.sol

```
* - `_listing.startPrice` must be lower than `_buyPrice`
```

SHB.10.2: Marketplace.sol

```
* - `_listing.startPrice` must be lower than or equal to `_buyPrice`
```

4 Best Practices

BP.1 Remove initialization

Description:

In the _addOwner function, the exists variable is declared, and the default value is assigned to it. However, in solidity, there is no need to initialize a variable with its type's default value, this is done automatically after the variable declaration.

Files Affected:

BP.1.1: OwnableMultiple.sol

```
function _addOwner(address _newOwner) internal {
        bool exists = false;
```

Status - Acknowledged

The Unicrypt team acknowledged this best practice for the reason being the importance of the code readability.

BP.2 Use Mapping For Efficient Lookups And Updates In OwnableMultiple Contract

Description:

It is recommended to use a mapping to store the owners in the OwnableMultiple contract instead of an array, as it allows for more efficient lookups and updates. The contract should also have a constructor function that adds the deployer as the initial owner, and include functions for adding and removing individual owners that are restricted to be called only by owners using the onlyOwner modifier. To add or remove multiple owners, these functions can be implemented by iterating through an array of addresses and calling the addOwner and removeOwner functions for each address. Additionally, it is important to include checks to prevent the null address (0x0) from being added as an owner and to ensure that an owner is not removed if they are not a current owner. These checks can help prevent potential issues with the contract.

Status - Fixed

The Unicrypt team used mapping for lookups inside the OwnableMultiple contract, making the contract more efficient and less costly in terms of gas.

BP.3 Optimizing Storage and Gas Usage with the delete Statement

Description:

The delete statement is more optimized than individualFees[_collection].isActivated = false, because it will completely remove the element from the mapping and free up storage space, while setting a boolean value to false will just update the value without freeing up any storage.

Using the delete statement will also reduce the number of entries in the mapping, which can be beneficial if the mapping has many entries, and you want to reduce the gas cost of iterating over the mapping. However, if you need to keep track of the inactive collections and their fees, it might be more appropriate to set a boolean value to false instead of deleting the element from the mapping.

In general, it's a good practice to use the delete statement only when you no longer need the element and want to free up storage, and to use updates to boolean values or other variables when you want to change the state of an element without deleting it.

Files Affected:

BP.3.1: ProtocolFees.sol

```
    function setIndividualFees(
        address _collection,
        bool _activate,
        uint256 _listingFee,
        uint256 _cancellationFee,
        uint256 _buyingFee
    ) external onlyOwner {
        require(_buyingFee < MAX_RELATIVE_FEE, "ProtocolFees: Fee too
            ↪ high");
        if (_activate) {
            individualFees[_collection] = IndividualFee(
                _activate,
                _listingFee,
                _cancellationFee,
                _buyingFee
            );
        } else {
            individualFees[_collection].isActivated = false;
        }
    }
```

Status - Acknowledged

The Unicrypt team acknowledged this best practice, stating that they need hasIndividualFees mapping to each address in their business logic, so they prefer to set it to false instead of deleting it.

BP.4 Separating activate/deactivate individualFees logic for clarity and maintainability

Description:

The setIndividualFees function is intended to set the fees for a particular collection. The function takes in several parameters including a boolean _activate which determines whether the fees are being activated or deactivated.

Its recommended to separate the activate/deactivate logic into a separate function and keep the setIndividualFees function focused on just setting the collection fee parameters. This would also make the code easier to understand and maintain, as the purpose of each function would be clearer.

For example, you could have a separate activateIndividualFee function that takes in the address of the collection and a boolean value indicating whether to activate or deactivate the fee. This function could then handle the logic for setting the isActivated field of the IndividualFee struct. Here is how the revised functions could look:

Files Affected:

BP.4.1: ProtocolFees.sol

```
    function activateIndividualFee(address _collection, bool _activate) external onlyOwner {
        individualFees[_collection].isActivated = _activate;
}

function setIndividualFees(
    address _collection,
    uint256 _listingFee,
    uint256 _cancellationFee,
    uint256 _buyingFee
) external onlyOwner {
    require(_buyingFee < MAX_RELATIVE_FEE, "ProtocolFees: Fee too high");
    individualFees[_collection] = IndividualFee(
          true,
        _listingFee,
        _cancellationFee,
        _buyingFee
    );
}
```

Status - Acknowledged

The Unicrypt team acknowledged this best practice, according to them this would require more function calls making the contract unoptimized, however they will still take this best practice into consideration.

BP.5 Public Function Can Be Declared as External

Description:

In Solidity, functions with a public scope are automatically generated with a function signature in the contract's application binary interface (ABI). This function signature is used by external contracts and applications to call the function.

If a public function is not called internally within the contract, it can be declared as external to save on gas costs. This is because the EVM (Ethereum Virtual Machine) will not generate code to execute the function body, which reduces the amount of gas needed to deploy and run the contract.

Files Affected:

BP.5.1: OwnableMultiple.sol

```
    function addOwner(address _newOwner) public onlyOwner {
        _addOwner(_newOwner);
    }
```

BP.5.2: OwnableMultiple.sol

```
    function addOwners(address[] memory _newOwners) public onlyOwner {
        _addOwners(_newOwners);
    }
```

Status - Fixed

The Unicrypt team implemented this best practice, which saves gas.

BP.6 Using the isOwner function to check for the presence of an owner

Description:

Consider using the isOwner function to check if a given address is an owner instead of manually iterating through the owners array.

This would make the code more concise and easier to read, as it avoids the need to manually iterate through the owners array and checks for the presence of the address.

Here is how the _addOwner function could be modified to use the isOwner function :

BP.6.1: OwnableMultiple.sol

```
function _addOwner(address _newOwner) internal {
      if (!isOwner(_newOwner)) {
          owners.push(_newOwner);
          emit OwnerAdded(_newOwner);
      }
   }
```

Status - Fixed

The Unicrypt team updated the ownableMultiple logic and used the isOwner function as a modifier for many newly implemented functions making the code optimized.

5 Tests

Results:

→ Gasless marketplace

✓ Should settle gasless listing with bid

✓ Should settle gasless listing with buy now

✓ Should transfer amounts correctly

✓ Should revert settling gasless listing with invalid signature

✓ Should revert settling gasless listing with invalid listing

✓ Should revert settling gasless listing multiple times

→ Marketplace batching

→ Create new listings batch

✓ Should create multiple listings with different ERC721

✓ Should create multiple listings with one ERC721 contract

✓ Should create multiple listings with different ERC1155

✓ Should create multiple listings with ERC721 and ERC1155

✓ Should revert creating multiple listings with length missmatch

✓ Should revert creating multiple listing with one invalid listing

✓ Should revert creating multiple listings with insufficient listing fee

✓ Should transfer proper listing fee

✓ Should transfer proper listing fee with individual fee set

✓ Should refund overpaid listing fee

- → Cancel listings batch
  - ✓ Should cancel multiple listings
  - ✓ Should transfer cancellation fee on cancel multiple listings
  - ✓ Should refund overpaid cancellation fee
  - ✓ Should revert cancelling multiple listings with one invalid
- → Bid on listings batch
  - ✓ Should bid on multiple listings
  - ✓ Should buy multiple listings
  - ✓ Should revert bidding on multiple listings with one invalid bid
  - ✓ Should transfer bidded amount to marketplace
  - ✓ Should refund overpaid bidded amount
  - ✓ Should revert with length missmatch
- → End listings batch
  - ✓ Should end multiple listings
  - ✓ Should revert ending multiple listings with one invalid
- → Settle listings with signature batch
  - ✓ Should settle multiple listings
  - ✓ Should revert with one invalid
  - ✓ Should revert with length missmatch
  - ✓ Should transfer amounts correctly
  - ✓ Should refund overpaid amount

→ Marketplace with ERC1155

- → Create new listings

✓ Should create a new listing with ERC1155

✓ Should transfer ERC1155 on listing creation

✓ Should transfer listing fee to fee receiver

✓ Should revert with insufficient listing fee sent

✓ Should revert with start price higher than buy price

✓ Should revert with buy price too low

✓ Should revert with invalid end date

✓ Should revert listing not owned NFT

✓ Should revert listing more ERC1155 than owned

→ Cancel listings

✓ Should cancel created listing by seller

✓ Should transfer NFT back to seller

✓ Should transfer cancellation fee to fee receiver

✓ Should revert cancelling not owned listing

✓ Should revert cancel listing with insufficient cancellation fee

✓ Should revert cancel non existing listing

✓ Should revert cancel listing twice

✓ Should revert cancel listing with bidder

→ Bid on listings

✓ Should bid on created listing

✓ Should transfer bidded amount to marketplace

✓ Should revert bidding when auction ended

✓ Should revert bidding lower than start price

✓ Should revert bidding lower current bid

✓ Should revert bidding more than buy price

✓ Should refund previous bidder on new bid

✓ Should refund reverting contract with WETH

→ End listing

✓ Should end listing on bidding buy price

✓ Should end listing when auction is over

✓ Should transfer NFT to buyer on buy now

✓ Should transfer NFT to buyer on end auction

✓ Should transfer buying fee to fee receiver

✓ Should transfer buying amount to seller

✓ Should transfer NFT back to seller if there is no buyer

✓ Should revert ending before auction is over

✓ Should refund buyer and seller on buyer reverting receiving NFT

→ Marketplace with ERC721

✓ Should initialize contract correctly

✓ Should revert deploying contract with invalid address

→ Create new listings

✓ Should create a new listing with ERC721

✓ Should transfer ERC721 to marketplace on listing creation

✓ Should transfer listing fee to fee receiver

✓ Should revert with non ERC721 or ERC1155 token

✓ Should revert with insufficient listing fee sent

✓ Should revert with start price higher than buy price

```
✓ Should revert with buy price too low
✓ Should revert with invalid end date
✓ Should revert listing not owned NFT
✓ Should revert with invalid collection address
→ Cancel listings
✓ Should cancel created listing by seller
✓ Should transfer ERC721 back to seller
✓ Should transfer cancellation fee to fee receiver
✓ Should revert cancelling not owned listing
✓ Should revert cancel listing with insufficient cancellation fee
✓ Should revert cancel non existing listing
✓ Should revert cancel listing twice
✓ Should revert cancel listing with bidder
→ Bid on listings
✓ Should bid on created listing
✓ Should transfer bidded amount to marketplace
✓ Should revert bidding when auction ended
✓ Should revert bidding lower than start price
✓ Should revert bidding lower current bid
✓ Should revert bidding more than buy price
✓ Should refund previous bidder on new bid
✓ Should refund reverting contract with WETH
→ End listing
✓ Should end listing on bidding buy price
```

- ✓ Should end listing when auction is over
- ✓ Should transfer NFT to buyer on buy now
- ✓ Should transfer NFT to buyer on end auction
- ✓ Should transfer buying fee to fee receiver
- ✓ Should transfer buying amount to seller
- ✓ Should transfer NFT back to seller if there is no buyer
- ✓ Should revert ending before auction is over

→ OwnableMultiple

✓ Should initialize contract correctly

✓ Should add new owner by owner

✓ Should add multiple new owners by owner

✓ Should revert adding new owner by non-owner

✓ Should revert adding new owners by non-owner

✓ Should not add owner twice

✓ Should not revert adding owner twice on adding multiple new owners

✓ Should remove owner by owner

✓ Should remove multiple owners by owner

✓ Should revert removing owner by non-owner

✓ Should revert removing multiple owners by non-owner

✓ Should not remove all owners

→ ProtocolFees

```
✓ Should initialize contract correctly
✓ Should set fee receiver by owner
✓ Should revert setting fee receiver by non-owner
✓ Should revert setting fee receiver to zero address
✓ Should set fees by owner
✓ Should revert setting fee higher than 25
✓ Should set individual fees for collections
✓ Should revert setting individual fees for collections
✓ Should deacctivate individual fees for collections

124 passing (43s)
```

Coverage:

The code coverage results were obtained by running npx hardhat coverage in the nft-marketplace-audit-main project. We found the following results :

- Statements Coverage : 100%
- Branches Coverage : 96.67%
- Functions Coverage : 100%
- Lines Coverage : 100%

6 Conclusion

In this audit, we examined the design and implementation of Unicrypt's Marketplace Smart Contracts and discovered several issues of varying severity. Unicrypt team addressed 2 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 Unicrypt Team to maintain a high level of vigilance and to keep those findings in mind in order to avoid any future complications.

7 Scope Files

7.1 Audit

Files	MD5 Hash
nft-marketplace-audit-main/contracts/OwnableMultiple.sol	cbefdc3162633bccbb3b76623a30b757
nft-marketplace-audit-main/contracts/ProtocolFees.sol	542a62ea6f6a4aace10cd5a8c4b1e3af
nft-marketplace-audit-main/contracts/MarketPlace.sol	6659dae8ec0bf2d81fb5f67d5372dfd9
nft-marketplace-audit-main/contracts/interfaces/INFT.sol	5d5abba636c677d7720fa498f525dc32
nft-marketplace-audit-main/contracts/interfaces/IWETH.sol	35a7e47ef06cf75598c43ecdf9759684
nft-marketplace-audit-main/contracts/interfaces/IMarketplace.sol	7c9db70dff2ea6dd596e0c7462e3cc87
nft-marketplace-audit-main/contracts/interfaces/IProtocolFees.sol	e6a6de93beb3e892fff4d137f824c574
nft-marketplace-audit-main/contracts/interfaces/IOwnableMultiple.sol	dab1fcf1b9e7fdd29fdd2826a9515045
nft-marketplace-audit-main/contracts/libraries/NFTTransfer.sol	f48485960888b964f4e5392a323abb8d
nft-marketplace-audit-main/contracts/libraries/ListingLib.sol	91225e9727c0e393040d514a8be12438

7.2 Re-Audit

Files	MD5 Hash
nft-marketplace-audit/contracts/OwnableMultiple.sol	0f2eef54945c06c1b59f7b38aba74215
nft-marketplace-audit/contracts/Marketplace.sol	b3dae17da8c258431c404e2463d47ec8
nft-marketplace-audit/contracts/ProtocolFees.sol	13e0beacd3915b0cd9e7fe4d3bae86d5
nft-marketplace-audit/contracts/interfaces/IWETH.sol	35a7e47ef06cf75598c43ecdf9759684
nft-marketplace-audit/contracts/interfaces/IProtocolFees.sol	dddc022353b79d88e6b440f29f9e3951
nft-marketplace-audit/contracts/interfaces/IOwnableMultiple.sol	7116e5d2df4dd4d2e84b3ddd56f3b39c
nft-marketplace-audit/contracts/interfaces/INFT.sol	5d5abba636c677d7720fa498f525dc32
nft-marketplace-audit/contracts/interfaces/IMarketplace.sol	599a8725c43fe0f4b6a8501987e9318e
nft-marketplace-audit/contracts/libraries/ListingLib.sol	91225e9727c0e393040d514a8be12438
nft-marketplace-audit/contracts/libraries/NFTTransfer.sol	f48485960888b964f4e5392a323abb8d

8 Disclaimer

Shellboxes reports should not be construed as ”endorsements” or ”disapprovals” of particular teams or projects. These reports do not reflect the economics or value of any ”product” or ”asset” produced by any team or project that engages Shellboxes to do a security evaluation, nor should they be regarded as such. Shellboxes Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the examined technology, nor do they provide any indication of the technology's proprietors, business model, business or legal compliance. Shellboxes Reports should not be used in any way to decide whether to invest in or take part in a certain project. These reports don't offer any kind of investing advice and shouldn't be used that way. Shellboxes Reports are the result of a thorough auditing process designed to assist our clients in improving the quality of their code while lowering the significant risk posed by blockchain technology. According to Shellboxes, each business and person is in charge of their own due diligence and ongoing security. Shellboxes does not guarantee the security or functionality of the technology we agree to research; instead, our purpose is to assist in limiting the attack vectors and the high degree of variation associated with using new and evolving technologies.

SHELLBOXES

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