# Coterie Finance

**NFT Audit**

# Smart Contract Audit Report


**November 27, 2021**

# Introduction

## 1. About Coterie Finance

Coterie Finance is a collection of products that leverage blockchain to provide financial services to end users, with a focus on the NFT sector at the moment.

Coterie NFT Marketplace is a platform with unique features that allows artists and content creators to mint NFTs for their original works of art without having to write or comprehend any programing or blockchain code. Furthermore, the minted NFTs, as well as other externally procured NFTs, can be auctioned on the platform.

Some of the unique features of the NFT marketplace include:

- Carefully crafted auction system that maximises the revenue of content creators, and gives the best experience to NFT collectors
- An affiliate element that gives anyone the opportunity to earn on the marketplace, even without being an art creator or NFT collector.
- A content creators onboarding process to expose users to only legitimate contents.
- A social element that keeps users up-to-date on their favourite art and content creators, and collectors.

## 2. About ImmuneBytes

ImmuneBytes is a security start-up to provide professional services in the blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, dydx.

The team has been able to secure 105+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-up with a detailed analysis of the system ensuring security and managing the overall project.

Visit http://immunebytes.com/ to know more about the services.

# Documentation Details

The Coterie Finance team has provided the following doc for the purpose of audit:

--

## Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

## Audit Details

- Project Name: Coterie Financel
- Contracts Name: NftMkt.sol(Coterie)
- Languages: Solidity(Smart contract)
- Github commit/Smart Contract Address for audit: https://drive.google.com/file/d/1BQlN6bhR-lzuibIdN3QwIFQl7URK-4vr/view
- Deployed Address: 0x217321E3F1f8814F8c335b5FAf52Ee057e94522d
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

## Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:
    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

## Security Level References

Every issue in this report was assigned a severity level from the following:

**Admin/Owner Privileges** can be misused either intentionally or unintentionally.

**High severity issues** will bring problems and should be fixed.

**Medium severity issues** could potentially bring problems and should eventually be fixed.

**Low severity issues** are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

| Issues | High | Medium | Low |
|---|---|---|---|
| Open | 2 | 2 | 2 |
| Closed | - | - | - |

## Admin/Owner Privileges

The **admin/owner** of the **ERC721Marketplace** smart contract has various privileges over the smart contract. These privileges can be misused either intentionally or unintentionally (in case the admin's private key gets hacked). We assume that these extra rights will always be used appropriately. Some of these admin rights are listed below.

- **In the ERC721Marketplace contract, the *Owner* address controls various configuration parameters of the contract.**
  The **ERC721Marketplace** contract contains various crucial configuration parameters like **increaseBidFactor**, **refBonus**, **bidWindow**, **givingRefBonus**, **paused** flag, and many more variables. These parameters and the right to change them impact the working of the **ERC721Marketplace** contract. The owner has the right to change these parameters anytime.

*Recommendation*:
Consider hardcoding predefined ranges or validations for input variables in privileged access functions. Also, consider adding some governance for admin rights for smart contracts or use a multi-sig wallet as an admin/owner address.

## High Severity Issues

1. **Anyone can call the updatePayTo() function.**
   The **updatePayTo()** function in **ERC721Marketplace** smart contract is used to update the **payTo** mapping. However the function does not check the validity of the caller, hence anyone can call this function and update the **payTo** mapping for any **tokenId**.

```
function updatePayTo(uint256 auctionId, PaymentsTo[]memory paymentsTo) external {
     validatePayTo(paymentsTo);
     _updatePayTo(auctionId, paymentsTo);
 }
```

   *Recommendation*:
   Consider performing validation checks for the caller as well as input values before updating any contract state.

2. **The makeBid() function can be fooled in case of ETH as payment token.**
   The **makeBid**() function in **ERC721Marketplace** contract is intended to be used for bidding on NFTs. This function accepts **bidValue** as an input variable but in the case of ETH as the payment token, the contract considers **msg.value** as the bid's value.
   This function can be exploited by passing the minimum required ETH as **msg.value** for the new bid but passing a large integer value as the **bidValue** input variable. In this case, the large value passed as **bidValue** will be considered as the bidder's bid. If the exploiter passes a sufficiently large number as the **bidValue** then he cannot be outbidden.

```
if (_auction.paymentMethod == address(0)) {
    require(
        msg.value >= _auction.basePrice,
        CoterieMarket: Bid_value_must_be_>current_bid_value"
     );
 }

// ...
// at last

 Bid memory bid = Bid({
    createdAt: _blocktime(),
    currentBid : bidValue,
    bidder: _msgSender()
 });
 getBids[_id].push(bid);
```

*Recommendation*:
Consider storing **msg.value** as the **Bid.currentBid** in the case of ETH as the payment token.

## Medium Severity Issues

1. **The contract does not follow the check-effect-interaction pattern.**
   In Solidity smart contracts a check-effect-interaction (CEI) pattern is used for security purposes. This pattern suggests that all preconditions must be checked first followed by all internal state updates for the contract, and at last, any external call to another smart contract should be made. Sending ETH to any ethereum address is also considered an untrusted external call.
   The **ERC721Marketplace** contract breaks this pattern at most places, it updates all its internal states after making external calls which are not recommended.

   For instance, in the **cancelAuction**() function the state is updated after making an external call to an untrusted IERC721 token.

```
    IERC721(_auction.token).safeTransferFrom(
        address(this),
        _auction.owner,
        _auction.tokenId
    );
    _auction.status = Status.CANCELLED;
```

   *Recommendation*:
   Consider following the CEI pattern and updating all internal states before making any external call. More details can be found at https://docs.soliditylang.org/en/v0.6.12/security-considerations.html#use-the-checks-effects-interactions-pattern

2. **The use of for loops increases gas usage and may cause DoS.**
   The **ERC721Marketplace** smart contract makes extensive use of **for** loops inside itself. Implementing **for** loops over unbounded arrays increases the gas cost of transactions, if the arrays grow up to a substantially large size then the transactions may start getting reverted due to Ethereum's block gas limit.

```
function paymentSpliter(
    uint256 id,
    address paymentMtd,
    uint256 value
) internal {
```

```
        uint256 totalVal = value;
        if (value > 0)
            for (uint256 i = 0; i < payTo[id].length; i++) {
                uint256 val = 0;
                // ...
                // ...
            }
    }
```

*Recommendation*:
Consider avoiding the use of **for** loops over unbounded arrays as it may result in Denial of Service.

## Low Severity Issues

1. **Incorrect and Unused interface present in the smart contract.**
   The **NftMkt.sol** file contains an **IERC2981** interface which is currently unused. The interface also doesn't define the **royaltyInfo**() function correctly. The correct standard for the **IERC2981** interface can be found at https://eips.ethereum.org/EIPS/eip-2981.

   *Recommendation*:
   Consider correcting the interface or removing it if it is not used.

2. **Contract bytecode exceeds the 24KB bytecode size limit.**
   The **ERC721Marketplace** smart contract's size exceeds 24576 bytes (a limit introduced in Spurious Dragon) due to which the contract may not be deployable on the Ethereum mainnet without enabling the optimizer.

   *Recommendation*:
   Consider reducing the contract's size by dividing up the contract into sub-contracts or removing unnecessary functions and state variables.

## Unit Test

No unit tests were provided by the Coterie Finance team.

*Recommendation*:
Our team suggests that the developers should write extensive test cases for the contracts.

## Coverage Report

Coverage reports cannot be generated without unit test cases.

*Recommendation*:
We recommend 100% line and branch coverage for unit test cases.

## Automated Auditing

### Solhint Linting Violations

Solhint is an open-source project for linting solidity code, providing both security and style guide validations. It integrates seamlessly into most mainstream IDEs. We used Solhint as a plugin within our VScode for this analysis. Multiple Linting violations were detected by Solhint, it is recommended to use Solhint's npm package to lint the contracts.

### Contract Library

Contract-library contains the most complete, high-level decompiled representation of all Ethereum smart contracts, with security analysis applied to them in real-time. We performed analysis using contract Library on the Kovan address of the SporeToken, SporeStake and LiquidityFarming contracts used during manual testing:

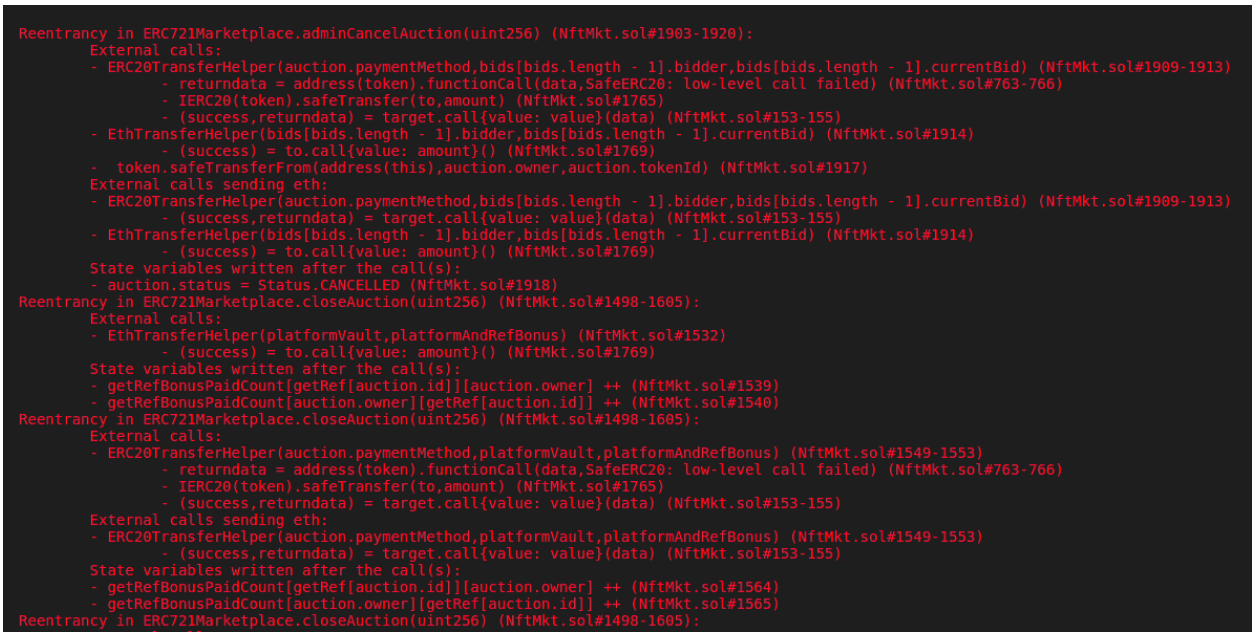
- ERC721Marketplace: 0x217321E3F1f8814F8c335b5FAf52Ee057e94522d

It raises no major concern for the contracts.

## Slither

Slither, an open-source static analysis framework. This tool provides rich information about Ethereum smart contracts and has critical properties. While Slither is built as a security-oriented
static analysis framework, it is also used to enhance the user's understanding of smart contracts, assist in code reviews, and detect missing optimizations.

The concerns slither raises have already been covered in the manual audit section.



```
Reentrancy in ERC721Marketplace.adminCancelAuction(uint256) (NftMkt.sol#1903-1920):
        External calls:
        - ERC20TransferHelper(auction.paymentMethod,bids[bids.length - 1].bidder,bids[bids.length - 1].currentBid) (NftMkt.sol#1909-1913)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (NftMkt.sol#763-766)
                - IERC20(token).safeTransfer(to,amount) (NftMkt.sol#1765)
                - (success,returndata) = target.call{value: value}(data) (NftMkt.sol#153-155)
        - EthTransferHelper(bids[bids.length - 1].bidder,bids[bids.length - 1].currentBid) (NftMkt.sol#1914)
                - (success) = to.call{value: amount}() (NftMkt.sol#1769)
        -  token.safeTransferFrom(address(this),auction.owner,auction.tokenId) (NftMkt.sol#1917)
        External calls sending eth:
        - ERC20TransferHelper(auction.paymentMethod,bids[bids.length - 1].bidder,bids[bids.length - 1].currentBid) (NftMkt.sol#1909-1913)
                - (success,returndata) = target.call{value: value}(data) (NftMkt.sol#153-155)
        - EthTransferHelper(bids[bids.length - 1].bidder,bids[bids.length - 1].currentBid) (NftMkt.sol#1914)
                - (success) = to.call{value: amount}() (NftMkt.sol#1769)
        State variables written after the call(s):
        - auction.status = Status.CANCELLED (NftMkt.sol#1918)
Reentrancy in ERC721Marketplace.closeAuction(uint256) (NftMkt.sol#1498-1605):
        External calls:
        - EthTransferHelper(platformVault,platformAndRefBonus) (NftMkt.sol#1532)
                - (success) = to.call{value: amount}() (NftMkt.sol#1769)
        State variables written after the call(s):
        - getRefBonusPaidCount[getRef[auction.id]][auction.owner] ++ (NftMkt.sol#1539)
        - getRefBonusPaidCount[auction.owner][getRef[auction.id]] ++ (NftMkt.sol#1540)
Reentrancy in ERC721Marketplace.closeAuction(uint256) (NftMkt.sol#1498-1605):
        External calls:
        - ERC20TransferHelper(auction.paymentMethod,platformVault,platformAndRefBonus) (NftMkt.sol#1549-1553)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (NftMkt.sol#763-766)
                - IERC20(token).safeTransfer(to,amount) (NftMkt.sol#1765)
                - (success,returndata) = target.call{value: value}(data) (NftMkt.sol#153-155)
        External calls sending eth:
        - ERC20TransferHelper(auction.paymentMethod,platformVault,platformAndRefBonus) (NftMkt.sol#1549-1553)
                - (success,returndata) = target.call{value: value}(data) (NftMkt.sol#153-155)
        State variables written after the call(s):
        - getRefBonusPaidCount[getRef[auction.id]][auction.owner] ++ (NftMkt.sol#1564)
        - getRefBonusPaidCount[auction.owner][getRef[auction.id]] ++ (NftMkt.sol#1565)
Reentrancy in ERC721Marketplace.closeAuction(uint256) (NftMkt.sol#1498-1605):
```

## Concluding Remarks

While conducting the audits of the Coterie Finance smart contract, it was observed that the contracts contain High, Medium and Low severity issues.

Our auditors suggest that High, Medium and Low severity issues should be resolved by the developers. The recommendations given will improve the operations of the smart contract.

## Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the Coterie Finance platform or its product nor this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for the code refactor by the team on critical issues.

***ImmuneBytes***