HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Salvor
Date: September 28th, 2022

Document

Name	Smart Contract Code Review and Security Analysis Report for Salvor
Approved By	Noah Jelich | Lead Solidity SC Auditor at Hacken OU
Type	ERC721 token; Marketplace, Auction
Platform	AVM
Network	Avalanche
Language	Solidity
Methods	Manual Review, Automated Review, Architecture Review
Website	-
Timeline	06.09.2022 - 28.09.2022
Changelog	16.09.2022 - Initial Review 28.09.2022 - Second Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Salvor (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.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope

Repository:
    https://gitlab.com/salvor1/salvor-contracts
Commit:
    749ae16ce271d06698815f35ed0a78f47a2611a5
Documentation:
    Functional requirements

Integration and Unit Tests: Yes
Deployed Contracts Addresses: No
Contracts:

```
File: ./contracts/AuctionMarketplace/AuctionMarketplace.sol
SHA3: 32a416dbd1e4bb73b0853f2ce6f56562bae2ceeadf306ca71f84d3cfc157e349

File: ./contracts/BlindAuctionMarketplace/BlindAuctionMarketplace.sol
SHA3: b978fcff59ad5ef71d11aab241b4222f69a0f6ce9b86c04d5395a78d3663bdc7

File: ./contracts/BlindAuctionMarketplace/LibBlindAuction.sol
SHA3: 2def74d74b741a90342fc7be5728183d6fa43f381b421f4a2e27ccd67060a00b

File: ./contracts/DutchAuctionMarketplace/DutchAuctionMarketplace.sol
SHA3: f3aeeb918f678990fe0332c5f56df26517b685ecaf09c2a64ceefa59d04ab324

File: ./contracts/libs/LibShareholder.sol
SHA3: 690a17078ade227a15fb082b73dac3ad17cc5f576f6020c18539f864ffad1b56

File: ./contracts/Marketplace/lib/LibOrder.sol
SHA3: 0035b76acfeacd3429f99ba13bcf8d1e0676b60644f686bfe85cf5c5e69de929

File: ./contracts/Marketplace/Marketplace.sol
SHA3: 69b36632cc93bfca5f471791d9cac2d88b3dcadb7122f9b758a383bea2a831d1

File: ./contracts/Migrations.sol
SHA3: edbabc8a9c57405a33cfc4be8caa4b85ae5499bc6dfb7bfa08a82e655395545c

File: ./contracts/NFTCollectible/INFTCollectible.sol
SHA3: 07bc7a9f6de693843050bda27df25fb478216c9fa382bee4be14afedc63e117c

File: ./contracts/NFTCollectible/NFTCollectible.sol
SHA3: 24a170a3ac78746a043e96b89a321305b6d42f8754b8c4bf37c0e3cc86355e7d

File: ./contracts/PaymentManager/IPaymentManager.sol
SHA3: e53d8c53a32ed922fb870430671d1be1418a20c917f475b375b7255c6929e629

File: ./contracts/PaymentManager/PaymentManager.sol
SHA3: 9d009e092d670566658477b1584f2dbee0bf0466eb73c1e398ad638a4fcaf3e1
```

```
        File: ./contracts/Royalty/IRoyalty.sol
        SHA3: a6d556ecc9f8d7a2ebe6ec4bd271a32ab786d707f9599cc749dae3df4afd099c

        File: ./contracts/Royalty/LibRoyalty.sol
        SHA3: c3097f59d4709cf6b1e3dfe7077d1405b303c0b56a9e0bb5b484ac2c0203af20

        File: ./contracts/Royalty/Royalty.sol
        SHA3: 3724e4f628b2a6d474fc49475e67cbb9d15715f22dbb83bc5d155ef023eb1706
```

Second review scope

Repository:
https://gitlab.com/salvor1/salvor-contracts

Commit:
d397df3587c27acca8e4ef08f31e471eb5caaf15

Documentation:
Functional requirements

Integration and Unit Tests: Yes
Deployed Contracts Addresses: No
Contracts:

```
        File: ./contracts/AuctionMarketplace/AuctionMarketplace.sol
        SHA3: 5de831a23853734876d0e2e50f535f1bc449acb16d11cf16d617a9fd0b3089b7

        File: ./contracts/DutchAuctionMarketplace/DutchAuctionMarketplace.sol
        SHA3: 3f893af33073aff8fa9cc9226a30273da4842501cb887bd65fa9e2c5abd3632a

        File: ./contracts/ERC721Dummy.sol
        SHA3: a94fd0f7bac07afd6e1cffe5d4fb8c7d80b5fe9ffdd339f84b582b299adc64b8

        File: ./contracts/libs/LibShareholder.sol
        SHA3: 916f58a195fff0da9fc40910c5341bb95142e88b7eb5b00acb373194736f3268

        File: ./contracts/Marketplace/lib/LibOrder.sol
        SHA3: 4d706f8cdb1339ce82f2a47c9dd7bdca89c4ecc9a897fa533fc212e902fe4e44

        File: ./contracts/Marketplace/Marketplace.sol
        SHA3: e7711d1cc64c03539cc11de3046555220d6ca4c9137ac85ebd202cfd174d763b

        File: ./contracts/NFTCollectible/INFTCollectible.sol
        SHA3: 0bb4ba60a4a6e66e9883fd343dc5195b9359964ebd94000e471d627110230d95

        File: ./contracts/NFTCollectible/NFTCollectible.sol
        SHA3: 751ca7fd729763a1e69b9464d7da7c2c554a06c0cb0cfd32a6c35b6fee351d4d

        File: ./contracts/PaymentManager/IPaymentManager.sol
        SHA3: ab71f97100c1b5afd4d993111e37ce9e361947d5fef6f452f8c606ecdef3b96a

        File: ./contracts/PaymentManager/PaymentManager.sol
        SHA3: bc4cfb85491cc6d01c7aa1d33f3ca708eccd73d052d8110dd92d487c32c30931

        File: ./contracts/Royalty/IRoyalty.sol
        SHA3: 42ce65eafef427bf88156a07f87985a1a2f9e871a203992f2574e22b37b9aec9

        File: ./contracts/Royalty/LibRoyalty.sol
        SHA3: a0616a35d6c1b570dcb98dbd6d23915ed0c166f0dc8c9c022ad32bb120b4ec98

        File: ./contracts/Royalty/Royalty.sol
        SHA3: 663df5161d2105ca6e67af678d5418ba1a9f713addad90f6493bb2581e5bc4d4
```

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.

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 and a technical description for each contract were provided. Code explanations are well written, and the documentation is public.

Code quality

The total Code Quality score is **9** out of **10**. Style guide violation was found. Marketplace platforms were split into contracts, and the general architecture was well-designed.

Test coverage

Basic user interactions and setting environments were covered. Negative test cases were missing.
Test coverage of the project is 94.97%.

Security score

As a result of the audit, the code contains **1** low severity issue. The security score is **10** out of **10**.

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

Summary

According to the assessment, the Customer's smart contract has the following score: **9.45**.

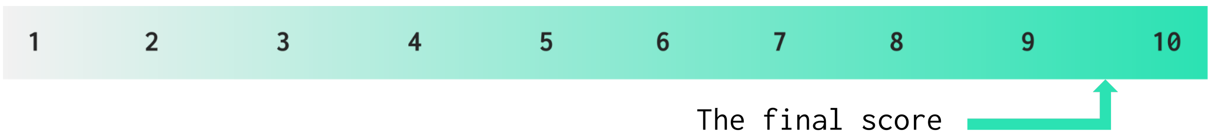

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
16 September 2022	12	6	4	3
28 September 2022	1	0	0	0

Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Passed
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless it is required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization	SWC-115	tx.origin should not be used for	Passed

through tx.origin		authorization.	
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122 EIP-155	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifier should always be used. All parameters from the signature should be used in signer recovery	Passed
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP standards violation	EIP	EIP standards should not be violated.	Passed
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Passed
Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of	Passed

		data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	
Style guide violation	**Custom**	Style guides and best practices should be followed.	Failed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Secure Oracles Usage	**Custom**	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
Tests Coverage	**Custom**	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Failed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

Salvor is an auction platform for ERC721Upgradeable tokens (NFTs), and it provides various auction patterns with the contracts below.

- AuctionMarketplace - an upgradable contract that allows NFT owners to start an English auction with "buy now" price option. Current bids are open to everyone. If a user wants to bid, the amount should be higher than the previous highest one, and the rise should match the required min bid increase amount.
- PaymentManager - an upgradable contract that manages the transfer commissions, royalties, and revenue shares for every marketplace contract. Each time the NFT owner changes, it can be expected 2 different fees:
    - Commission fee
        - A fee type that is taken from every NFT sale, if a commission percentage is set by the owner
    - Royalty fee
        - The second fee type is only taken if the NFT contract supports single or multi royalty standard(EIP-2981). The royalty fee will be distributed to the original NFT creator/creators.
- IPaymentManager - an interface of the PaymentManager contract.
- DutchAuctionMarketplace - an upgradable contract that provides a Dutch auction platform. Asset owner can start an auction at a starting price, which slowly decreases over time until the reserve price is reached. Auction immediately ends when a user catches the current Dutch price.
- LibShareHolder - a library that helps to store shareholder addresses and their percentages.
- LibOrder - a library that helps to store the order info and hash it.
- Marketplace - an upgradable contract that allows users to list and buy NFTs and make/accept offers to NFTs. Users can bid for any NFT asset without requiring a whitelisting process.
- INFTCollectible - an interface of the NFTCollectible contract.
- NFTCollectible - an ERC721 token contract that supports Royalty feature.
- IRoyalty - an interface of the Royalty contract.
- LibRoyalty - a library used by Royalty contract to store royalty info.
- Royalty - a helper contract that sets royalty addresses and percentages to signal a royalty amount to be paid to the NFT creator or rights holder every time the NFT is sold or re-sold.

Privileged roles

- NFT owners can:
    - start an auction by choosing a type

- settle the auction
- cancel the auction if it has not any offer

- Owner of the AuctionMarketplace contract can:
  - set payment manager address
  - set default bid increase percentage
  - set minimum price limit
  - set maximum duration period
  - set default auction bid period
  - pause/unpause the contract
- Owner of the DutchAuctionMarketplace can:
  - set the payment manager address
  - pause/unpause the contract
  - set minimum price limit
  - set maximum duration
  - set minimum drop interval
- Owner of the Marketplace contract can:
  - set the payment manager address
  - set minimum price limit
  - pause/unpause the contract
- Owner of the NFTCollectible contract can:
  - set the base token URI
  - set the base extension
  - set the default royalties
  - update the royalties
  - mint tokens infinitely
- Owner of the PaymentManager contract can:
  - add a marketplace address to the whitelist
  - remove a marketplace from the whitelist
  - set company wallet address
  - set commission percentage
  - pause/unpause the contract

Findings

■■■■ Critical

1. **Compilation issues**

   Some functions in the contract are in IRoyalty interface. They need to be overridden; otherwise, compilation will fail.

   Path: ./contracts/Royalty/Royalty.sol: getDefaultRoyalties(), getTokenRoyalties(), royaltyInfo(), multiRoyaltyInfo()

   Recommendation: Add "override" specifier to the functions.

   Status: Fixed (Revised commit: d397df3587c27acca8e4ef08f31e471eb5caaf15)

■■■ High

1. **Highly permissive owner access**

   defaultAuctionBidPeriod and *defaultBidIncreasePercentage* variables can be changed by the owner, even if there are already ongoing auctions.

   Path: ./contracts/AuctionMarketplace/AuctionMarketplace.sol

   Recommendation: Do not allow to change these variables for ongoing auctions. Assign their current values to the *Auction* struct in *auctions* mapping and for validations and checks, use the variables from *auctions[_nftContractAddress][_tokenId]*.

   Status: Fixed (Revised commit: d397df3587c27acca8e4ef08f31e471eb5caaf15)

2. **Highly permissive owner access**

   The owner can change the commission percentage anytime, and there is no limit to its value.

   This may lead users to face unpromised commission fees for their NFT sales.

   Path: ./contracts/PaymentManager/PaymentManager.sol

   Recommendation: Do not allow to change this variable for an auction that is already set.

   Status: Fixed (Revised commit: d397df3587c27acca8e4ef08f31e471eb5caaf15)

3. **Funds lock**

   If a royalty address or a shareholder address is a smart contract instead of an externally owned account and the buyout function fails on sending assets to these addresses, failed transfer amount will be saved here to be withdrawn later; *failedTransferBalance[_recipient] += _amount* (Line#242-PaymentManager). If there is no implementation in the smart contract to call the *withdrawFailedCredits* function, the entire balance will be locked inside the payment manager contract.

Same false protection design is implemented in AuctionMarketplace, BlindAuctionMarketplace, DutchAuctionMarketplace contracts.

Paths:

./contracts/PaymentManager/PaymentManager.sol:
withdrawFailedCredits()

./contracts/AuctionMarketplace/AuctionMarketplace.sol:
withdrawFailedTransferBalance()

./contracts/DutchAuctionMarketplace/DutchAuctionMarketplace.sol:
withdrawFailedCredits()

Recommendation: Only allow externally owned accounts to deposit/bid funds or fix the issue in a different way.

Status: Fixed (Revised commit: d397df3587c27acca8e4ef08f31e471eb5caaf15)

Medium

1. Requirement compliance violation / Wrong

Although require statement message says: “Drop Interval must be lower than minimum drop interval limit”, drop interval needs to be higher than the minimum limit.

Path:
./contracts/DutchAuctionMarketplace/DutchAuctionMarketplace.sol:
createDutchAuction()

Recommendation: Fix the requirement statement message.

Status: Fixed (Revised commit: d397df3587c27acca8e4ef08f31e471eb5caaf15)

Low

1. Floating pragma

Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Paths: all

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Status: Fixed (Revised commit: d397df3587c27acca8e4ef08f31e471eb5caaf15)

2. Missing events

Setting the default bid increase percentage, maximum duration period, and default auction bid period should emit an event for the internal state changes.

Path: ./contracts/AuctionMarketplace/AuctionMarketplace.sol: setDefaultBidIncreasePercentage(), setMaximumDurationPeriod(), setDefaultAuctionBidPeriod()

Recommendation: Emit events for the updates.

Status: Fixed (Revised commit: d397df3587c27acca8e4ef08f31e471eb5caaf15)

3. **Missing zero address validation**

address _paymentManager parameter in *initialize* function and *setPaymentManager* function is not validated against a default zero address possibility.

Path: ./contracts/AuctionMarketplace/AuctionMarketplace.sol: initialize(), setPaymentManager()

Recommendation: Implement zero address checks.

Status: Fixed (Revised commit: d397df3587c27acca8e4ef08f31e471eb5caaf15)

4. **State variables can be declared constant**

commissionPercentage and *maximumRoyaltyReceiversLimit* variables are initialized with a hard-coded value and never updated anywhere.

This causes extra Gas consumption and increases the code complexity.

Path: ./contracts/PaymentManager/PaymentManager.sol

Recommendation: Declare variables as a constant or make the variables dynamic.

Status: Fixed (Revised commit: d397df3587c27acca8e4ef08f31e471eb5caaf15)

5. **State variables' default visibility**

failedTransferBalance variable's visibility is not specified. Specifying state variables' visibility helps to catch incorrect assumptions about who can access the variable.

This makes the contract`s code quality and readability higher.

Path: ./contracts/DutchAuctionMarketplace/DutchAuctionMarketplace.sol

Recommendation: Specify variables as public, internal, or private. Explicitly define visibility for all state variables.

Status: Fixed (Revised commit: d397df3587c27acca8e4ef08f31e471eb5caaf15)

6. **Functions that can be declared as external**

setDefaultRoyalties, *saveRoyalties*, *getDefaultRoyalties*, *getTokenRoyalties*, *royaltyInfo*, *multiRoyaltyInfo* functions are declared public. In order to save Gas, public functions that are never called in the contract should be declared as external.

Path: ./contracts/Royalty/Royalty.sol

Recommendation: Use the external attribute for functions never called from the contract.

Status: Fixed (Revised commit: d397df3587c27acca8e4ef08f31e471eb5caaf15)

7. **Missing zero address validation**

address _paymentManager parameter in *initialize* function is not validated against a default zero address possibility.

Path: ./contracts/DutchAuctionMarketplace/DutchAuctionMarketplace.sol: initialize()

Recommendation: Implement a zero address check.

Status: Fixed (Revised commit: d397df3587c27acca8e4ef08f31e471eb5caaf15)

8. **Boolean equality**

isNotCancelled modifier compares to a boolean constant. Boolean constants can be used directly and do not need to be compared to true or false.

Path: ./contracts/Marketplace/Marketplace.sol

Recommendation: Remove the boolean equality.

Status: Fixed (Revised commit: d397df3587c27acca8e4ef08f31e471eb5caaf15)

9. **Style guide violation**

The provided projects should follow the official guidelines.

Paths: all

Recommendation: Follow the official Solidity guide: https://docs.soliditylang.org/en/v0.8.13/style-guide.html

Status: Reported (Revised commit: d397df3587c27acca8e4ef08f31e471eb5caaf15)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date 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 to 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, Consultant cannot guarantee the explicit security of the audited smart contracts.