HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: DualMint Limited
Date: October 21, 2022

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for DualMint Limited
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	NFT Marketplace
Platform	EVM
Network	Ethereum
Language	Solidity
Methods	Manual Review, Automated Review, Architecture Review
Website	https://www.dualmint.com/
Timeline	22.09.2022 - 21.10.2022
Changelog	27.09.2022 - Initial Review 12.10.2022 - Second Review 21.10.2022 - Third Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by DualMint Limited (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://github.com/DualMint/Marketplace
Commit:
353aac8a82ec48937af3920f073dc89cea2e0429
Documentation: NatSpec

Integration and Unit Tests: Yes
Deployed Contracts Addresses:
Contracts:
File: ./contracts/FactoryNFT1155.sol
SHA3: a587b5d451e45945d806709a7c6123045b88dee83f9209ff4bc0c74f404689ad

File: ./contracts/Market.soll
SHA3: 4ba472bf09cb527d8cd03290cb161394da565ae0c8902ce02be13076af6ef0ec

File: ./contracts/NFT1155.sol
SHA3: ff4c0a2b68658195df6c0194ee28a9bb0ef86b99272ec7ed8bd0f9f314467663

Second review scope

Repository:
https://github.com/DualMint/Marketplace
Commit:
ed2ecc8112b66457d4d1e6173c404f3ddd538a2d
Documentation: NatSpec

Integration and Unit Tests: Yes
Deployed Contracts Addresses:
Contracts:
File: ./contracts/FactoryNFT1155.sol
SHA3: 34b843e269e8a2a50de06ffa90982ac5e26b9f3f5b344d41a9c87f437a6880e2

File: ./contracts/Market.sol
SHA3: f41d9ea87359e29ec287141589ba54a5a84ea25a31ebc4429ea38a033bcdd52c

File: ./contracts/NFT1155.sol
SHA3: 0e4ff91c9ff332bb2b75d765b2e886ffd29448fa0a91d14204299fc2a66e42f2

Third review scope

Repository:
https://github.com/DualMint/Marketplace
Commit:
664be9e7409862161e69c7a29cae39192a09b877

Documentation: NatSpec

Integration and Unit Tests: Yes
Deployed Contracts Addresses:
Contracts:
File: ./contracts/FactoryNFT1155.sol
SHA3: 5eebf4c08cb1829b63b38157766701f8c989b71a242448100ec0a5b60cb3b2f4

File: ./contracts/Market.sol
SHA3: 159f2db2f12a8fa243be9f9aca15196d2b0f6e25db3f0f41ea38d922fb615879

File: ./contracts/NFT1155.sol
SHA3: 11d8187ba9a11ce36284c173156e34b70357de0b264fcfec8f29d2410b79ad41

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.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

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.
- Technical descriptions are provided.

Code quality

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

- The development environment is configured.
- NatSpec annotation covers all code.
- The code follows the Solidity style guide.
- The code contains multiple duplications of calculations.

Test coverage

Test coverage of the project is **100.00%**.

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is partially missed.
- Not all branches are covered.

Security score

As a result of the audit, the code contains **1** medium severity issue. The security score is **9** 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: **9.1**.

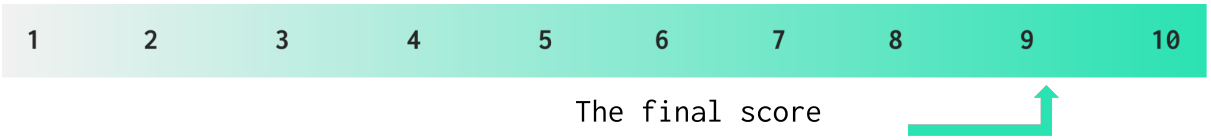

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
26 September 2022	9	6	8	2
12 October 2022	2	1	0	0
21 October 2022	0	1	0	0

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Checked Items

We have audited the Customers' smart contracts for commonly known and more specific vulnerabilities. Here are some items 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.	Not Relevant
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.	Not Relevant
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed

Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Not Relevant
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 identifiers should always be used. All parameters from the signature should be used in signer recovery	Not Relevant
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.	Not Relevant

Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
Style guide violation	**Custom**	Style guides and best practices should be followed.	Passed
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.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

DualMint Limited is a platform that aims to provide real Luxury Goods & Collectibles through an NFT market. The system supports ERC1155 NFTs. The system allows creation, direct purchases, auctions, and offers to listed items from the users. The system supports ERC20 tokens as a paid currency. There is a fee and royalties structure in the system. These fees and royalties are deducted from each sale, auction, or offer.
The system consists of the following contracts:

- *FactoryNFT1155.sol* - *ECR1155* handler
- *NFT1155.sol* - *ERC1155* contract for the Dualmint marketplace that facilitates minting NFT assets and updating their URIs
- *Market.sol* - *NFTMarket* contract for the Dualmint marketplace that facilitates creation, buying, selling and auctions of tokenized versions of luxury items

Privileged roles

- The *owner* of the *Market* contract can arbitrarily set royalties, commission percentage, main currency for market, create auction/direct sale on behalf of NFT owner, complete royalties distribution. It is therefore entitled to impersonate or change the logic of critical components of the system at will.
- The *admin* of the *NFT1155* contract can mint new assets on behalf of the NFT contract owner and modify token URIs.
- *The owner* of the *NFT1155* can mint new assets.

Risks

- Only the owner of the Market contract can complete any incomplete royalty loops.

Findings

■■■■ Critical

1. Denial of Service Vulnerability

If the address with the winning bid were a contract without *ERC1155Receiver* implementation, it would be impossible to complete the asset's sale.

It will lead to a lock of the seller’s funds and royalties.

Path: ./contracts/Market.sol

Functions: endAuction, distributionOfFundsAndAssets

Recommendation: use a pull pattern for asset withdrawal. Create a separate method from distribution of royalties and fees for asset withdrawal by the address of the winning bid.

Status: Fixed(ed2ecc8112b66457d4d1e6173c404f3ddd538a2d)

2. Funds Lock

The owner of the market contract can change the main payment currency at any time. In case the token is changed before users withdraw their amount from the old sale/auction, there may not be enough new tokens in the contract.

Tokens can get stuck on the contract.

Path: ./contracts/Market.sol

Function: setTokenAddress

Recommendation: remove function or implement a mechanism to prevent described cases.

Status: Fixed(ed2ecc8112b66457d4d1e6173c404f3ddd538a2d)

■■■ High

1. Non-Finalized Code

The production code should not contain any functions or variables that are being used solely in the test environment.

Path: ./contracts/NFT1155.sol

Recommendation: remove *.hardhat/console.sol* import.

Status: Fixed(ed2ecc8112b66457d4d1e6173c404f3ddd538a2d)

2. Requirements Violation

According to the requirement, the minimum price for sale items should be greater than 1 USDC, but comparing values does not take into account USDC token decimals.

Path: ./contracts/Market.sol

Functions: createMarketItem, assistedCreateMarketItem

Recommendation: consider actual token decimals in requirement.

Status: Fixed(ed2ecc8112b66457d4d1e6173c404f3ddd538a2d)

3. **Undocumented Behavior**

The NatSpec of the function resellItem() states that it should be used for sale of an asset previously purchased on the marketplace.

It is possible to resell an item that was already on auction as a new one. In this case, previous owners will not receive royalties.

Path: ./contracts/Market.sol

Functions: create, asistcreateItem

Recommendation: inline requirements with the documentation.

Status: Fixed(ed2ecc8112b66457d4d1e6173c404f3ddd538a2d)

4. **Denial of Service Vulnerability**

Hardcoded Gas amounts are used in the contract. It should not be estimated how much Gas will be needed for some remaining operations. Gas costs may change dramatically along with different blockchain forks. There may be cases when the remaining Gas after performing the operations in the loop will be less than that required for the remaining operations.

It can lead to a block of distribution of the royalties for certain item Ids.

Path: ./contracts/Market.sol

Functions: distributionOfFundsAndAssets, completeRoyaltyLoop

Recommendation: leave the distribution for up to 3 sales and commission in function distributionOfFundsAndAssets. Implement a pagination pattern to function completeRoyaltyLoop and distribute the rest of the royalties with this method.

Status: Fixed(ed2ecc8112b66457d4d1e6173c404f3ddd538a2d)

5. **Undocumented Behavior**

According to the documentation, FactoryNFT1155 contract is for the Dualmint marketplace. Function deployNFT1155 allows users to create NFT for the marketplace specified by the user and Dualmint wallet as an admin.

DualMint admin will have access to minting the user NFT’s and allowing them for sale on different marketplaces.

Path: ./contracts/FactoryNFT1155.sol

Function: deployNFT1155

Recommendation: remove the possibility to set marketplace and instead set DualMint market contract as FactoryNFT1155 constructor parameter.

Status: Fixed(ed2ecc8112b66457d4d1e6173c404f3ddd538a2d)

6. **Finalized Code**

   payable addresses are used for the address of the deployer and the owner of a market item. The current implementation of NFT market contract cannot perform operations with the chain's native currency.

   Path: ./contracts/Market.sol

   Recommendation: remove payable from mentioned addresses or add a description to the documentation why it is needed.

   Status: Fixed(ed2ecc8112b66457d4d1e6173c404f3ddd538a2d)

7. **Invalid Calculations; Denial of Service Vulnerability**

   The contract allows to exceed 100% for the sum of DM commission and royalty's percentage.

   It can lead to reaching a failing assert statement for underflow.

   Path: ./contracts/Market.sol

   Functions: setRoyalties, setCommissionPercent

   Recommendation: add a check to the methods setRoyalties and setCommissionPercent to check if the total percentage does not exceed 100 %.

   Status: Fixed(ed2ecc8112b66457d4d1e6173c404f3ddd538a2d)

Medium

1. **Missing Events Emit on Changing Important Values**

   The contract does not emit any events after changing important values. It is recommended to emit events after changing important values. This will allow everyone to easily noticed such changes.

   Path: ./contracts/Market.sol

   Functions: setRoyalties, setCommissionPercent, setTokenAddress, directMarketSale, resellItem

   Recommendation: implement event emits after changing contract values.

   Status: Fixed(ed2ecc8112b66457d4d1e6173c404f3ddd538a2d)

2. **Unchecked Token Transfer**

   ERC20 transfer functions return bool after transfers, and it is important to implement a return value check for this return value. This issue leads to unintended behavior of the contract regarding token transfer results.

   Path: ./contracts/Market.sol

Functions: CreateBid, directMarketSale, withdrawFunds

Recommendation: implement a return value check for token transfers.

Status: Fixed(ed2ecc8112b66457d4d1e6173c404f3ddd538a2d)

3. **Missing Allowance Check**

   The contract calls token transfer functions without checking the allowance.

   Path: ./contracts/Market.sol

   Functions: CreateBid, directMarketSale

   Recommendation: implement allowance check in mentioned functions.

   Status: Fixed(ed2ecc8112b66457d4d1e6173c404f3ddd538a2d)

4. **Code Duplication**

   In the source code, there is a duplication of function implementation:

   https://github.com/DualMint/Marketplace/blob/353aac8a82ec48937af3920f073dc89cea2e0429/contracts/NFT1155.sol#L58

   and
   https://github.com/DualMint/Marketplace/blob/353aac8a82ec48937af3920f073dc89cea2e0429/contracts/NFT1155.sol#L43

   Avoid the use of duplicate code, as it will lead to an increase in Gas usage during the deployment.

   Path: ./contracts/NFT1155.sol

   Functions: createToken, assistedCreateToken

   Recommendation: create an internal method for mentioned functions and remove code duplication.

   Status: Fixed(ed2ecc8112b66457d4d1e6173c404f3ddd538a2d)

5. **Code Duplication**

   The code contains multiple duplications of calculations.

   Gas consumption is not optimal. Readability is decreased.

   Path: ./contracts/Market.sol

   Recommendation: use local variables to store calculated values.

   Status: Reported

6. **Usage of Hardcoded Values**

   Hardcoded values are used in calculations.

   Readability is decreased.

Path: ./contracts/Market.sol

Recommendation: move hardcoded values to constants.

Status: Fixed(ed2ecc8112b66457d4d1e6173c404f3ddd538a2d)

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.

Path: ./contracts/FactoryNFT1155.sol

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

Status: Fixed(ed2ecc8112b66457d4d1e6173c404f3ddd538a2d)

2. Style Guide Violation

The provided projects should follow the official guidelines.

Path: ./contracts/FactoryNFT1155.sol,./contracts/Market.sol

Recommendation: follow the official Solidity guidelines.

Status: Fixed(664be9e7409862161e69c7a29cae39192a09b877)

3. State Variables' Default Visibility

The explicit visibility makes it easier to catch incorrect assumptions about who can access the variable.

Path: ./contracts/FactoryNFT1155.sol

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

Status: Fixed(ed2ecc8112b66457d4d1e6173c404f3ddd538a2d)

4. Unindexed Events

Having indexed parameters in the events makes it easier to search for these events using indexed parameters as filters.

Path: ./contracts/FactoryNFT1155.sol

Recommendation: use the "*indexed*" keyword to the event parameters

Status: Fixed(ed2ecc8112b66457d4d1e6173c404f3ddd538a2d)

5. Missing Zero Address Validation

Address parameters are being used without checking against the possibility of 0x0.

Path: ./contracts/FactoryNFT1155.sol

Function: deployNFT1155

Recommendation: implement zero address checks.

Status: Fixed(ed2ecc8112b66457d4d1e6173c404f3ddd538a2d)

6. **Functions that Can Be Declared External**

"*public*" functions that are never called by the contract should be declared "external" to save Gas.

Paths: ./contracts/FactoryNFT1155.sol, ./contracts/Market.sol, ./contracts/NFT1155.sol

Functions: deployNFT1155, initialize, createToken, assistedCreateToken, updateUri

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

Status: Fixed(664be9e7409862161e69c7a29cae39192a09b877)

7. **Typos in Documentation**

Any typos encountered in the provided documentation should be addressed.

Paths: ./contracts/FactoryNFT1155.sol, ./contracts/Market.sol, ./contracts/NFT1155.sol

Recommendation: fix typos.

Status: Fixed(ed2ecc8112b66457d4d1e6173c404f3ddd538a2d)

8. **Redundant Import**

The use of unnecessary imports will increase the Gas consumption of the code. Thus they should be removed from the code.

Path: ./contracts/Market.sol

Imports: ContextUpgradeable, IERC1155ReceiverUpgradeable

Recommendation: remove the redundant imports.

Status: Fixed(ed2ecc8112b66457d4d1e6173c404f3ddd538a2d)

9. **Variable Shadowing**

Function parameter owner shadows existing function from OwnableUpgradeable inherited contract.

Path: ./contracts/Market.sol

Function: assistedCreateMarketItem

Recommendation: Rename related argument.

Status: Fixed(ed2ecc8112b66457d4d1e6173c404f3ddd538a2d)

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.