VIDMA

VRJAM

SMART CONTRACT AUDIT

Project: VRJAM
Date: October 19th, 2022

TABLE OF CONTENTS

SUMMARY

Vidma is pleased to present this audit report outlining our assessment of code, smart contracts, and other important audit insights and suggestions for management, developers, and users.

VRJAM is an immersive, live experience platform built on the Matic blockchain designed to magnify the connectedness of human beings in virtual space. VRJAM is a multiplayer social gaming and live events platform.

The audited scope included the Marketplace contract and VRJAMCollection. Marketplace allows users to create orders to swap the VRJAMCollection NFTs by proposing the price by the buyer which is approved by the seller. Swap of orders is possible only by signing the message by one of the participant seller or buyer. The platform will charge a fee for each closed order. VRJAMCollection is pausable. There is a possibility for the user with the pauser role to stop any transfer for all users on the platform.

During the audit process, the Vidma team found several issues. A detailed summary and the current state are displayed in the table below.

Severity of the issue	Total found	Resolved	Unresolved
Critical	0 issues	0 issues	0 issues
High	0 issues	0 issues	0 issues
Medium	0 issues	0 issues	0 issues
Low	6 issues	6 issues	0 issues
Informational	4 issues	2 issues	2 issues
Total	**10 issues**	**8 issues**	**2 issues**

After evaluating the findings in this report and the final state after fixes, the Vidma auditors can state that the contracts are fully operational and secure. Under the given circumstances, we set the following risk level:

Our auditors are evaluating the initial commit given for the scope of the audit and the last commit with the fixes. This approach helps us adequately and sequentially evaluate the quality of the code. Code style, optimization of the contracts, the number of issues, and risk level of the issues are all taken into consideration. The Vidma team has developed a transparent scoring system presented below.

Severity of the issue	Resolved	Unresolved
Critical	1	10
High	0.8	7
Medium	0.5	5
Low	0.2	0.5
Informational	0	0.1

Please note that the points are deducted out of 100 for each and every issue on the list of findings (according to the current status of the issue). Issues marked as "not valid" are not subject to point deduction.

Based on the **overall result of the audit**, the Vidma audit team grants the following score:

In addition to manual check and static analysis, the auditing team has conducted a number of integrated autotests to ensure the given codebase has an adequate performance and security level.

The test results and the coverage can be found in the accompanying section of this audit report.

Please be aware that this audit does not certify the definitive reliability and security level of the contract. This document describes all vulnerabilities, typos, performance issues, and security issues found by the Vidma audit team. If the code is still under development, we highly recommend running one more audit once the code is finalized.

SCOPE OF WORK

VRJAM

VRJAM is a metaverse platform that empowers rightsholders to create and share digital content and virtual events with global audiences.

Within the scope of this audit, two independent auditors thoroughly investigated the given codebase and analyzed the overall security and performance of the smart contracts.

The audit was conducted from October 11, 2022 to October 12, 2022. The outcome is disclosed in this document.

The review of the fixes was done on Wednesday, October 19. The outcome is disclosed in this document.

The scope of work for the given audit consists of the following contracts:

- LibOrder;
- Marketplace;
- VRJAMCollection;
- ERC1155.

The source code was taken from the following **sources**:

Repository 1:
https://github.com/vrjlive/VRJAM-ERC1155-Token

Initial commit submitted for the audit:
b27f5d509c26b846b7e7553605646deb676fd2ee

Last commit reviewed by the auditing team:
97406d0d47dd48bc05541a93d902b5e6367b2ed2

Repository 2:
https://github.com/vrjlive/VRJAM-Marketplace-LibOrder-Contract

Initial commit submitted for the audit:
481f6d1ba08b261b3789f782ae7aadea7ff6a4ba

Last commit reviewed by the auditing team:
7ef76e95980bad9f356fe8b26e92c247fd63bd33

As a reference to the contracts logic, business concept, and the expected behavior of the codebase, the VRJAM team has provided the following documentation:
https://vrjam.com/wp-content/uploads/2022/10/VRJAM-Whitepaper-v3.5.pdf.zip

WORKFLOW OF THE AUDITING PROCESS

Vidma audit team uses the most sophisticated and contemporary methods and well-developed techniques to ensure contracts are free of vulnerabilities and security risks. The overall workflow consists of the following phases:

Phase 1: The research phase

Research

After the Audit kick-off, our security team conducts research on the contract's logic and expected behavior of the audited contracts.

Documentation reading

Vidma auditors do a deep dive into your tech documentation with the aim of discovering all the behavior patterns of your codebase and analyzing the potential audit and testing scenarios.

The outcome

At this point, the Vidma auditors are ready to kick off the process. We set the auditing strategies and methods and are prepared to conduct the first audit part.

Phase 2: Manual part of the audit

Manual check

During the manual phase of the audit, the Vidma team manually looks through the code in order to find any security issues, typos, or discrepancies with the logic of the contract. The initial commit as stated in the agreement is taken into consideration.

Static analysis check

Static analysis tools are used to find any other vulnerabilities in smart contracts that were missed after a manual check.

The outcome

An interim report with the list of issues.

Phase 3: Testing part of the audit

Integration tests

Within the testing part, Vidma auditors run integration tests using the Truffle or Hardhat testing framework. The test coverage and the test results are inserted in the accompanying section of this audit report.

The outcome

Second interim report with the list of new issues found during the testing part of the audit process.

STRUCTURE AND ORGANIZATION OF THE FINDINGS

For simplicity in reviewing the findings in this report, Vidma auditors classify the findings in accordance with the severity level of the issues. (from most critical to least critical).

All issues are marked as “Resolved” or “Unresolved”, depending on if they have been fixed by VRJAM or not. The issues with “Not Relevant” status are left on the list of findings but are not eligible for the score points deduction.

The latest commit with the fixes reviewed by the auditors is indicated in the “Scope of Work” section of the report.

The Vidma team always provides a detailed description of the issues and recommendations on how to fix them.

Classification of found issues is graded according to 6 levels of severity described below:

Critical

The issue affects the contract in such a way that funds may be lost or allocated incorrectly, or the issue could result in a significant loss.

Example: Underflow/overflow, precisions, locked funds.

High

The issue significantly affects the ability of the contract to compile or operate. These are potential security or operational issues.

Example: Compilation errors, pausing/unpausing of some functionality, a random value, recursion, the logic that can use all gas from block (too many iterations in the loop), no limitations for locking period, cooldown, arithmetic errors which can cause underflow, etc.

Medium

The issue slightly impacts the contract's ability to operate by slightly hindering its intended behavior.

Example: Absence of emergency withdrawal of funds, using assert for parameter sanitization.

Low

The issue doesn't contain operational or security risks, but are more related to optimization of the codebase.

Example: Unused variables, inappropriate function visibility (public instead of external), useless importing of SCs, misuse or disuse of constant and immutable, absent indexing of parameters in events, absent events to track important state changes, absence of getters for important variables, usage of string as a key instead of a hash, etc.

Informational

Are classified as every point that increases onboarding time and code reading, as well as the issues which have no impact on the contract's ability to operate.

Example: Code style, NatSpec, typos, license, refactoring, naming convention (or unclear naming), layout order, functions order, lack of any type of documentation.

MANUAL REPORT

Lack of event emits while some important data is set

Low | ML – 01 | Resolved

In the Marketplace contract should be event emit for state variables overriding *_beneficiary* and *_fee* in the appropriate functions *setBeneficiary()* and *setFee()*.

Recommendation:

Consider the emitting events in these specific cases to be able to track critical data change.

State Variable Default Visibility

Low | ML – 02 | Resolved

In the LibOrder library constant ORDER_TYPEHASH doesn't have variable visibility and it is public by default. Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

Recommendation:

Consider defining visibility for this constant.

Useless imports

Low | TL – 01 | Resolved

There are unused imports in the codebase:

- ERC1155.sol and Strings.sol in the VRJAMColection contract;
- IERC165.sol in the Marketplace contract.

Recommendation:

Consider removing useless imports to decrease the contract's bytecode.

Event indexing is missed

Low | TL – 02 | Resolved

Params event indexing is missed for *MatchOrders()* event of the Merketplace contract. The indexed parameters for logged events give the possibility to search for these events using the indexed parameters as filters (solidity official docs)

Recommendation:

Consider adding event param indexing where it is missed.

Chain id getter optimization

Low | TL – 03 | Resolved

There is an assembly block that gets chain id in Marketplace contract but in the Solidity compiler version 0.8.* or higher *block.chainid* is available as a global variable.

Recommendation:

Consider using *block.chainid*.

Floating pragma

Low | TL – 04 | Resolved

The current version of solc in the audited contracts is ^0.8.4 and it is better to lock the pragma to a specific version.

Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. Locking the pragma helps to 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.

Recommendation:

Consider locking pragma to a specific version.

Functions visibility optimization

Informational | MI – 01 | Unresolved

In the contracts Marketplace, VRJAMCollection and ERC1155 there are public functions that can be marked as external as they are not called anywhere inside the contract.

Recommendation:

Consider changing visibility from public to external.

Re-Audit:

Consider changing visibility to external in next functions:

- Marketplace.sol: *version()*, *beneficiary()*, *collection()*, *fee()*, *setBeneficiary()*, *setFee()*, *pause()*, *unpause()*;
- VRJAMCollection.sol: *name()*, *symbol()*, *version()*, *baseURI()*, *props()*, *setBaseURI()*, *pause()*, *unpause()*, *mint()*, *mintBatch()*;
- Appropriate functions in ERC1155.sol.

Functions order at contracts

Informational | MI – 02 | Unresolved

The functions in contracts Marketplace and VRJAMCollection are not grouped according to their visibility and order.

Functions should be grouped according to their visibility and ordered in the following way:

- constructor;
- receive function (if exists);
- fallback function (if exists);
- external;
- public;
- internal;
- private.

Recommendation:

Consider changing functions order according to solidity documentation and style guide.

Unnecessary comments

Informational | MI – 03 | Resolved

In the ERC1155 contract there is an unnecessary comment in 252 line.

Recommendation:

Consider removing the unnecessary comments.

Unimplemented functionality

Informational | TI – 01 | Resolved

In the marketplace contract, there is pausable functionality. Based on the natspec it should not allow make orders when a contract is paused. But in fact *matchOrders()* function has no limitation and check for paused contract or not.

Recommendation:

Consider removing pausable functionality in case it is useless or add *whenNotPaused()* modifier for *matchOrders()* function.

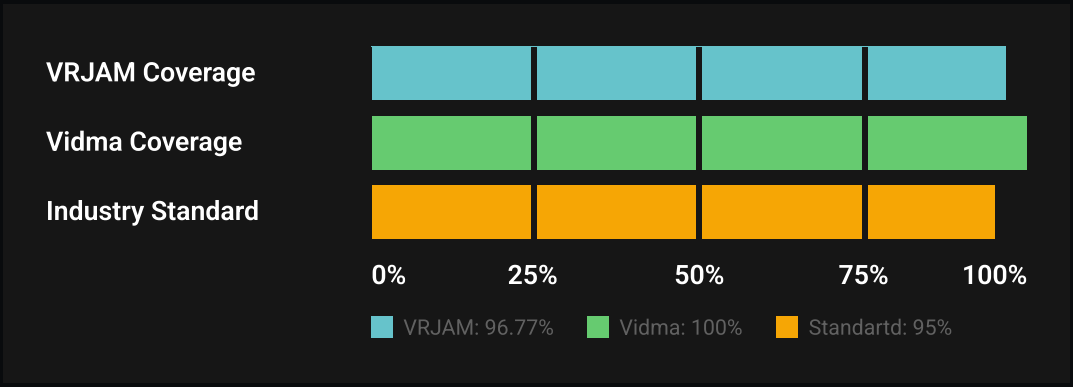
Re-Audit:

whenNotPaused() modifier for *matchOrders()* function was added.

TEST RESULTS

To verify the security of contracts and their performance, a number of integration tests were carried out using the Truffle testing framework.

In this section, we provide both tests written by VRJAM and tests written by Vidma auditors.



It is important to note that Vidma auditors do not modify, edit or add tests to the existing tests provided in the VRJAM repository. We write totally separate tests with code coverage of a minimum of 95% to meet industry standards.

Tests written by VRJAM

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
contracts\	92.94	94.64	81.25	93.33
LIbOrder.sol	100.00	100.00	100.00	100.00
Marketplace.sol	91.84	92.5	71.43	92.00
VRJAMCollection.sol	93.94	100.00	87.5	94.59
contracts\token\ERC1155\	100.00	90.91	100.00	100.00
ERC1155.sol	100.00	90.91	100.00	100.00
All Files	**96.77**	**93.00**	**88.89**	**96.92**

Test Results

```
Contract: Marketplace
  ✓ has collection
  ✓ has beneficiary
  ✓ has fee numerator
  ✓ has fee denominator
  ✓ has domain name
  ✓ has domain version
  ✓ deployer has the default admin role
  ✓ deployer has the admin role
  roles after deployment
    ✓ only default admin has the default admin role
    ✓ only operator has the admin role
```

```
  ✓ only default admin can set default admin
  ✓ only default admin can set admin (44ms)
  ✓ other cannot set default admin (175ms)
  ✓ admin cannot set other admin (189ms)
setBeneficiary
  ✓ sets new beneficiary
  - emits {SetBeneficiary} log
  ✓ reverts for non-admin (169ms)
setFee
  ✓ sets new fee
  - emits {setFee} log
  ✓ reverts when the fee numerator exceeds the fee denumerator
  ✓ reverts for non-admin (114ms)
matchOrders (alice - buyer, bob - seller)
  pre-mint
    execute ask
      ✓ swaps from alice (158ms)
      ✓ swaps from alice (138ms)
      ✓ swaps from alice with higher bid price (117ms)
      ✓ swaps from alice with zero price (173ms)
      ✓ swaps from alice with low price (117ms)
      ✓ swaps from alice with insignificant price (119ms)
      ✓ swaps from alice with zero ask start (156ms)
      ✓ swaps from alice with zero ask end (115ms)
      ✓ swaps from alice with zero ask start & end (115ms)
      ✓ swaps from alice with zero bid start (115ms)
      ✓ swaps from alice with zero bid end (117ms)
      ✓ swaps from alice with zero bid start & end (115ms)
      ✓ reverts from bob (105ms)
      ✓ reverts from alice when seller is not signer (110ms)
      ✓ reverts reused order (388ms)
      ✓ reverts when insufficient token allowance (170ms)
      ✓ reverts when insufficient token balance (173ms)
      ✓ reverts when nft is not approved (173ms)
      ✓ reverts when nft is not owned (262ms)
      ✓ reverts when ask start is not valid (105ms)
      ✓ reverts when ask end is not valid (69ms)
      ✓ reverts when bid start is not valid (71ms)
      ✓ reverts when bid end is not valid (116ms)
      ✓ reverts when no signed order (111ms)
      ✓ reverts when ask seller and bid buyer are the same (324ms)
      ✓ reverts when collection address does not match (129ms)
```

```
    ✓ reverts when asset ID does not match (72ms)
    ✓ reverts when payment token does not match (93ms)
    ✓ reverts when ask price higher than bid price (128ms)
  signed bid (execute bid)
    ✓ swaps from bob (113ms)
    ✓ swaps from bob (112ms)
    ✓ swaps from bob with higher bid price (117ms)
    ✓ swaps from bob with zero price (111ms)
    ✓ swaps from bob with low price (117ms)
    ✓ swaps from bob with insignificant price (115ms)
    ✓ swaps from bob with zero ask start (126ms)
    ✓ swaps from bob with zero ask end (110ms)
    ✓ swaps from bob with zero ask start & end (115ms)
    ✓ swaps bob with zero bid start (114ms)
    ✓ swaps from bob with zero bid end (111ms)
    ✓ swaps from bob with zero bid start & end (111ms)
    ✓ reverts from alice (172ms)
    ✓ reverts from bob when buyer is not signer (114ms)
    ✓ reverts reused order (385ms)
    ✓ reverts when insufficient token allowance (133ms)
    ✓ reverts when insufficient token balance (160ms)
    ✓ reverts when nft is not approved (258ms)
    ✓ reverts when nft is not owned (202ms)
    ✓ reverts when ask start is not valid (203ms)
    ✓ reverts when ask end is not valid (64ms)
    ✓ reverts when bid start is not valid (82ms)
    ✓ reverts when bid end is not valid (69ms)
    ✓ reverts when no signed order (74ms)
    ✓ reverts when ask seller and bid buyer are the same (172ms)
    ✓ reverts when collection address does not match (96ms)
    ✓ reverts when asset ID does not match (78ms)
    ✓ reverts when payment token does not match (108ms)
    ✓ reverts when ask price higher than bid price (69ms)
pre-mint & lazy-mint
  execute ask
    ✓ swaps ask data from alice (154ms)
    ✓ swaps bid data from alice (161ms)
    ✓ swaps bid data from alice with higher bid price (200ms)
    ✓ swaps bid data from alice with zero price (164ms)
    ✓ swaps bid data from alice with low price (156ms)
    ✓ swaps bid data from alice with insignificant price (153ms)
    ✓ swaps bid data from alice with zero ask start (159ms)
```

```
    ✓ swaps bid data from alice with zero ask end (154ms)
    ✓ swaps bid data from alice with zero ask start & end (160ms)
    ✓ swaps bid data from alice with zero bid start (157ms)
    ✓ swaps bid data from alice with zero bid end (155ms)
    ✓ swaps bid data from alice with zero bid start & end (153ms)
    ✓ reverts ask data from bob (132ms)
    ✓ reverts bid data from bob (90ms)
    ✓ reverts ask data from alice when seller is not
      signer (146ms)
    ✓ reverts ask data from bob when seller is not
      signer (130ms)
    ✓ reverts reused data (345ms)
    ✓ reverts when insufficient token allowance (212ms)
    ✓ reverts when insufficient token balance (222ms)
    ✓ reverts when nft is not approved (265ms)
    ✓ reverts when nft is not owned (325ms)
    ✓ reverts when ask start is not valid (66ms)
    ✓ reverts when ask end is not valid (135ms)
    ✓ reverts when bid start is not valid (91ms)
    ✓ reverts when bid end is not valid (67ms)
    ✓ reverts when no signed order (45ms)
    ✓ reverts when ask seller and bid buyer are the same (107ms)
    ✓ reverts when collection address does not match (134ms)
    ✓ reverts when asset ID does not match (121ms)
    ✓ reverts when payment token does not match (97ms)
    ✓ reverts when ask price higher than bid price (164ms)
  signed bid (execute bid)
    ✓ swaps ask data from bob (151ms)
    ✓ swaps bid data from bob (155ms)
    ✓ swaps bid data from bob with higher bid price (185ms)
    ✓ swaps bid data from bob with zero price (154ms)
    ✓ swaps bid data from bob with low price (157ms)
    ✓ swaps bid data from bob with insignificant price (155ms)
    ✓ swaps bid data from bob with zero ask start (157ms)
    ✓ swaps bid data from bob with zero ask end (159ms)
    ✓ swaps bid data from bob with zero ask start & end (152ms)
    ✓ swaps bid data from bob with zero bid start (154ms)
    ✓ swaps bid data from bob with zero bid end (158ms)
    ✓ swaps bid data from bob with zero bid start & end (158ms)
    ✓ reverts ask data from alice (122ms)
    ✓ reverts bid data from alice (172ms)
```

```
    ✓ reverts ask data from bob when buyer is not signer (119ms)
    ✓ reverts ask data from alice when buyer is not
      signer (148ms)
    ✓ reverts reused data (336ms)
    ✓ reverts when insufficient token allowance (220ms)
    ✓ reverts when insufficient token balance (248ms)
    ✓ reverts when nft is not approved (329ms)
    ✓ reverts when nft is not owned (261ms)
    ✓ reverts when ask start is not valid (122ms)
    ✓ reverts when ask end is not valid (101ms)
    ✓ reverts when bid start is not valid (104ms)
    ✓ reverts when bid end is not valid (97ms)
    ✓ reverts when no signed order (67ms)
    ✓ reverts when ask seller and bid buyer are the same (67ms)
    ✓ reverts when collection address does not match (127ms)
    ✓ reverts when asset ID does not match (120ms)
    ✓ reverts when payment token does not match (140ms)
    ✓ reverts when ask price higher than bid price (84ms)
  lazy-mint
    signed ask (execute ask)
      ✓ swaps ask data from alice (165ms)
      ✓ swaps bid data from alice (146ms)
      ✓ swaps ask data from alice with higher bid price (150ms)
      ✓ swaps ask data from alice with zero price (147ms)
      ✓ swaps ask data from alice with low price (149ms)
      ✓ swaps ask data from alice with insignificant price (145ms)
      ✓ swaps ask data from alice with zero ask start (193ms)
      ✓ swaps ask data from alice with zero ask end (145ms)
      ✓ swaps ask data from alice with zero ask start & end (146ms)
      ✓ swaps ask data from alice with zero bid start (147ms)
      ✓ swaps ask data from alice with zero bid end (146ms)
      ✓ swaps ask data from alice with zero bid start & end (145ms)
      ✓ reverts when nft is not owned (306ms)
      ✓ reverts when tokenize with fee (229ms)
    signed bid (execute bid)
      ✓ swaps ask data from bob (146ms)
      ✓ swaps bid data from bob (152ms)
      ✓ swaps ask data from bob with higher bid price (146ms)
      ✓ swaps ask data from bob with zero price (147ms)
      ✓ swaps ask data from bob with low price (173ms)
      ✓ swaps ask data from bob with insignificant price (148ms)
      ✓ swaps ask data from bob with zero ask start (145ms)
```

```
        ✓ swaps ask data from bob with zero ask end (143ms)
        ✓ swaps ask data from bob with zero ask start & end (146ms)
        ✓ swaps ask data from bob with zero bid start (153ms)
        ✓ swaps ask data from bob with zero bid end (146ms)
        ✓ swaps ask data from bob with zero bid start & end (146ms)
        ✓ reverts when nft is not owned (277ms)
        ✓ reverts when tokenize with fee (206ms)

Contract: VRJAMCollection
  ✓ has a name
  ✓ has a symbol
  ✓ has a base URI [ERC1155]
  ✓ has a base URI [ERC1155URIStorage]
  ✓ deployer has the default admin role
  ✓ deployer has the minter role
  ✓ deployer has the pauser role
  ✓ deployer has the URI setter role
  roles after deployment
    ✓ only default admin has the default admin role
    ✓ only admin has the minter role
    ✓ only admin has the pauser role
    ✓ only admin has the URI setter role
    ✓ only default admin can set default admin
    ✓ only default admin can set minter
    ✓ other cannot set default admin (151ms)
    ✓ minter cannot set other minter (231ms)
  like an ERC1155
    balanceOf
      ✓ reverts when queried about the zero address
      when accounts don't own tokens
        ✓ returns zero for given addresses
      when accounts own some tokens
        ✓ returns the amount of tokens owned by the given
          addresses
    balanceOfBatch
      ✓ reverts when input arrays don't match up
      ✓ reverts when one of the addresses is the zero address (67ms)
      when accounts don't own tokens
        ✓ returns zeros for each account
      when accounts own some tokens
        ✓ returns amounts owned by each account in order passed
```

```
      ✓ returns multiple times the balance of the same address
        when asked
    setApprovalForAll
      ✓ sets approval status which can be queried via
        isApprovedForAll
      ✓ emits an ApprovalForAll log
(node:11788) DeprecationWarning: expectEvent.inLogs() is deprecated.
Use expectEvent() instead.
(Use `node --trace-deprecation ...` to show where the warning was
created)
      ✓ can unset approval for an operator
      ✓ reverts if attempting to approve self as an operator (73ms)
    safeTransferFrom
      ✓ reverts when transferring more than balance (52ms)
      ✓ reverts when transferring to zero address
      when called by the multiTokenHolder
        ✓ debits transferred balance from sender
        ✓ credits transferred balance to receiver
        ✓ emits a TransferSingle log
        ✓ preserves existing balances which are not transferred
          by multiTokenHolder
      when called by an operator on behalf of the multiTokenHolder
        when operator is not approved by multiTokenHolder
          ✓ reverts
        when operator is approved by multiTokenHolder
          ✓ debits transferred balance from sender
          ✓ credits transferred balance to receiver
          ✓ emits a TransferSingle log
          ✓ preserves operator's balances not involved in the
            transfer
      when sending to a valid receiver
        without data
          ✓ debits transferred balance from sender
          ✓ credits transferred balance to receiver
          ✓ emits a TransferSingle log
          ✓ calls onERC1155Received
        with data
          ✓ debits transferred balance from sender
          ✓ credits transferred balance to receiver
          ✓ emits a TransferSingle log
          ✓ calls onERC1155Received
      to a receiver contract returning unexpected value
```

```
    ✓ reverts (98ms)
  to a receiver contract that reverts
    ✓ reverts (57ms)
  to a contract that does not implement the required function
    ✓ reverts (48ms)
safeBatchTransferFrom
  ✓ reverts when transferring amount more than any of
    balances (77ms)
  ✓ reverts when ids array length doesn't match amounts array
    length (122ms)
  ✓ reverts when transferring to zero address (64ms)
  when called by the multiTokenHolder
    ✓ debits transferred balances from sender
    ✓ credits transferred balances to receiver
    ✓ emits a TransferBatch log
  when called by an operator on behalf of the multiTokenHolder
    when operator is not approved by multiTokenHolder
      ✓ reverts
    when operator is approved by multiTokenHolder
      ✓ debits transferred balance from sender
      ✓ credits transferred balance to receiver
      ✓ emits a TransferBatch log
      ✓ preserves operator's balances not involved in the
        transfer
  when sending to a valid receiver
    without data
      ✓ debits transferred balance from sender
      ✓ credits transferred balance to receiver
      ✓ emits a TransferBatch log
      ✓ calls onERC1155BatchReceived
    with data
      ✓ debits transferred balance from sender
      ✓ credits transferred balance to receiver
      ✓ emits a TransferBatch log
      ✓ calls onERC1155Received
  to a receiver contract returning unexpected value
    ✓ reverts (110ms)
  to a receiver contract that reverts
    ✓ reverts (81ms)
  to a receiver contract that reverts only on single transfers
    ✓ debits transferred balances from sender
    ✓ credits transferred balances to receiver
```

```
      ✓ emits a TransferBatch log
      ✓ calls onERC1155BatchReceived
    to a contract that does not implement the required function
      ✓ reverts (91ms)
  ERC165
    ✓ supportsInterface uses less than 30k gas (43ms)
    ✓ all interfaces are reported as supported
    ✓ all interface functions are in ABI
internal functions
  _mint
    ✓ reverts with a zero destination address (67ms)
    with minted tokens
      ✓ emits a TransferSingle event
      ✓ emits an URI event
      ✓ credits the minted amount of tokens
      ✓ credits the tokens with correct URI
  _mintBatch
    ✓ reverts with a zero destination address (70ms)
    ✓ reverts if length of inputs do not match (343ms)
    with minted batch of tokens
      ✓ emits a TransferBatch event
      ✓ credits the minted batch of tokens
      ✓ credits the tokens with correct URI
  _burn
    ✓ reverts when burning the zero account's tokens
    ✓ reverts when burning a non-existent token id
    ✓ reverts when burning more than available tokens (113ms)
    with minted-then-burnt tokens
      ✓ emits a TransferSingle event
      ✓ accounts for both minting and burning
      ✓ has an empty token URI after total burn
  _burnBatch
    ✓ reverts when burning the zero account's tokens (41ms)
    ✓ reverts if length of inputs do not match (243ms)
    ✓ reverts when burning a non-existent token id
    with minted-then-burnt tokens
      ✓ reverts if length of inputs do not match (91ms)
      ✓ emits a TransferBatch event
      ✓ accounts for both minting and burning
      ✓ has an empty tokens URI after total burn (80ms)
  ERC1155MetadataURI
    ✓ emits no URI event in constructor
```

```
    ✓ sets the initial URI for both minted and unminted tokens
  sets a new base URI
    ✓ emits no URI event
    ✓ sets the new URI for both minted and unminted tokens
ERC1155Burnable
  burn
    ✓ holder can burn their tokens
    ✓ approved operators can burn the holder's tokens
    ✓ unapproved accounts cannot burn the holder's tokens
  burnBatch
    ✓ holder can burn their tokens (42ms)
    ✓ approved operators can burn the holder's tokens (55ms)
    ✓ unapproved accounts cannot burn the holder's tokens
ERC1155Supply
  before mint
    ✓ exist
    ✓ totalSupply
  after mint
    single
      ✓ exist
      ✓ totalSupply
    batch
      ✓ exist
      ✓ totalSupply
  after burn
    single
      ✓ exist
      ✓ totalSupply
    batch
      ✓ exist
      ✓ totalSupply
Properties
  before mint
    ✓ exist
    ✓ props
  after mint
    single
      ✓ exist
      ✓ props
    batch
      ✓ exist
      ✓ props
```

```
      after burn
        single
          ✓ exist
          ✓ props
        batch
          ✓ exist
          ✓ props (39ms)
    Pausable
      when token is paused
        ✓ reverts when trying to safeTransferFrom from holder (49ms)
        ✓ reverts when trying to safeTransferFrom from
          operator (41ms)
        ✓ reverts when trying to safeBatchTransferFrom from
          holder (62ms)
        ✓ reverts when trying to safeBatchTransferFrom from
          operator (60ms)
        ✓ reverts when trying to mint (49ms)
        ✓ reverts when trying to mintBatch (78ms)
        ✓ reverts when trying to burn
        ✓ reverts when trying to burnBatch (42ms)
        balanceOf
          ✓ returns the amount of tokens owned by the given
            address
        isApprovedForAll
          ✓ returns the approval of the operator

309 passing (3m)
2 pending
```

Tests written by Vidma auditors

Test Coverage

File	% Stmts	% Branch	% Funcs	% Lines
contracts\	100.00	100.00	100.00	100.00
LIbOrder.sol	100.00	100.00	100.00	100.00
Marketplace.sol	100.00	100.00	100.00	100.00
VRJAMCollection.sol	100.00	100.00	100.00	100.00
contracts\token\ERC1155\	100.00	95.45	100.00	100.00
ERC1155.sol	100.00	95.45	100.00	100.00
All Files	**100.00**	**98.00**	**100.00**	**100.00**

Test Results

```
Contract: VRJAMCollection
  initialization
    ✓ should set a name correct
    ✓ should set a symbol correct
    ✓ should set a base URI correct
    ✓ should set initial roles correct (58ms)
  setBaseURI
    ✓ should set base uri correct
  getter
    ✓ should get version correct
    ✓ should return supportsInterface correct
  balanceOf
```

```
  ✓ should return balance of user correct
  ✓ shouldn't return balance for zero address
balanceOfBatch
  ✓ should return balanceOfBatch of user correct (77ms)
  ✓ shouldn't return balanceOfBatch if arrays length mismatch
setApprovalForAll & safeTransferFrom
  ✓ should setApprovalForAll correct (50ms)
  ✓ shouldn't set approval for itself
  ✓ should call safeTransferFrom correct (90ms)
  ✓ should call safeBatchTransferFrom correct (92ms)
  ✓ shouldn't call safeTransferFrom if not approved (80ms)
  ✓ shouldn't call safeBatchTransferFrom if not approved (84ms)
  ✓ should call safeTransferFrom by the owner of asset
    correct (64ms)
  ✓ should call safeBatchTransferFrom by the owner of asset
    correct (66ms)
  ✓ shouldn't call safeTransferFrom if destination address is
    zero (43ms)
  ✓ shouldn't call safeTransferFrom if account from has
    insufficient balance (55ms)
  ✓ shouldn't call safeBatchTransferFrom if destination address
    is zero (64ms)
  ✓ shouldn't call safeBatchTransferFrom if account from has
    insufficient balance (63ms)
  ✓ shouldn't call safeBatchTransferFrom if ids and amounts
    length mismatch (48ms)
safeTransfer to contract
  ✓ should reject token by the ERC1155Receiver contract while
    safeTransferFrom (104ms)
  ✓ should reject token by the ERC1155Receiver contract while
    safeBatchTransferFrom (111ms)
  ✓ should safeTransferFrom token to ERC1155Receiver
    contract (59ms)
  ✓ should safeBatchTransferFrom token to ERC1155Receiver
    contract (61ms)
  ✓ shouldn't safeTransferFrom token to ERC1155Receiver contract
    when it is not allowed (80ms)
  ✓ shouldn't safeBatchTransferFrom token to ERC1155Receiver
    contract when it is not allowed (97ms)
  ✓ shouldn't safeTransferFrom token to non ERC1155Receiver
    contract (61ms)
```

```
    ✓ shouldn't safeBatchTransferFrom token to non ERC1155Receiver
      contract (65ms)
  pauseable
    ✓ should pause correct
    ✓ should unpause correct
    ✓ shouldn't pause by not the pauser (88ms)
    ✓ shouldn't unpause by not the pauser (98ms)
  mint
    ✓ should mint correct (62ms)
    ✓ shouldn't mint to zero destination address
    ✓ shouldn't mint when paused (54ms)
  burn
    ✓ should burn correct (72ms)
    ✓ shouldn't burn by not the owner (161ms)
    ✓ shouldn't burn when paused (49ms)
    ✓ shouldn't burn if amount exceeds balance (63ms)
  mintBatch
    ✓ should mintBatch correct (56ms)
    ✓ shouldn't mint to zero destination address
    ✓ shouldn't mintBatch when paused (50ms)
    ✓ shouldn't mintBatch when arrays length is mismatch (157ms)
  burnBatch
    ✓ should burnBatch correct (110ms)
    ✓ shouldn't burnBatch by not the owner (118ms)
    ✓ shouldn't burnBatch when paused (38ms)
    ✓ shouldn't burnBatch when arrays length is mismatch (53ms)
    ✓ shouldn't burnBatch if amount exceeds balance (52ms)

Contract: Marketplace
  initialization
    ✓ should set collection correct
    ✓ should set beneficiary correct
    ✓ should grant initial roles correct
    ✓ should set initial fee correct
    ✓ shouldn't deploy with collection as zero address (89ms)
    ✓ shouldn't deploy with beneficiary as zero address (44ms)
  setBeneficiary
    ✓ should set beneficiary correct
    ✓ shouldn't set beneficiary as zero address
    ✓ shouldn't set beneficiary by not the admin (119ms)
  setFee
    ✓ should set fee correct
```

```
  ✓ shouldn't set fee bigger than 100% (63ms)
  ✓ shouldn't set fee by not the admin (114ms)
getters
  ✓ should get chainID correct
  ✓ should get timestamp correct
  ✓ should get version correct
  ✓ should recover order signer correct (49ms)
pauseable
  ✓ should pause correct
  ✓ should unpause correct
  ✓ shouldn't pause by not the admin (93ms)
  ✓ shouldn't unpause by not the admin (120ms)
matchOrders
  ✓ should swap orders with shigned ask order correct (94ms)
  ✓ should swap orders with shigned bid order correct (93ms)
  ✓ should swap orders if start == 0 correct (93ms)
  ✓ should swap orders if end == 0 correct (93ms)
  ✓ shouldn't swap orders if seller and buyer the same (89ms)
  ✓ shouldn't swap orders if nft is not the same in ask and
    bid orders (65ms)
  ✓ shouldn't swap orders if nftId is not matched (74ms)
  ✓ shouldn't swap orders if payment token does not match (68ms)
  ✓ shouldn't swap orders if ask price bigger than bid (88ms)
  ✓ shouldn't swap orders if there is no signed order (50ms)
  ✓ shouldn't swap orders if ask signer is incorrect (88ms)
  ✓ shouldn't swap orders if ask is signed and caller is
    not buyer (75ms)
  ✓ shouldn't swap orders if bid signer is incorrect (71ms)
  ✓ shouldn't swap orders if ask is signed and caller is
    not buyer (75ms)
  ✓ shouldn't swap orders with signed ask order if ask start
    date is bigger current timestamp (66ms)
  ✓ shouldn't swap orders with signed ask order if ask end
    date is less current timestamp (88ms)
  ✓ shouldn't swap orders with signed bid order if bid start
    date is bigger current timestamp (77ms)
  ✓ shouldn't swap orders with signed bid order if bid end
    date is less current timestamp (70ms)
  ✓ should swap orders with signed ask order and ask data
    execution correct (122ms)
  ✓ shouldn't swap orders with signed ask order and ask data
    execution incorrect (141ms)
```

```
    ✓ should swap orders with signed bid order and bid data
      execution correct (120ms)
    ✓ shouldn't swap orders with signed bid order and bid data
      execution incorrect (145ms)

  94 passing (17s)
```

We are delighted to have a chance to work with the VRJAM team and contribute to your company's success by reviewing and certifying the security of your smart contracts.

The statements made in this document should be interpreted neither as investment or legal advice, nor should its authors be held accountable for decisions made based on this document.

Vidma is a security audit company helping crypto companies ensure their code and products operate safely and as intended, enabling founders to sleep soundly at night. We specialize in auditing DeFi protocols, layer one protocols, and marketplace solutions. Our team consists of experienced and internationally trained specialists. Our company is based in Ukraine, known for its strong engineering, cryptography, and cybersecurity culture.

Website: **vidma.io**
Email: **security@vidma.io**

VIDMA