Quantstamp

Talentir Token & Marketplace

Executive Summary

This audit report was prepared by Quantstamp, the leader in blockchain security.

Type	NFT Marketplace
Timeline	2023-04-11 through 2023-04-11
Language	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Source Code	• Talentir/talentir-contracts #313bd6f
Auditors	• Hytham Farah Auditing Engineer • Julio Aguliar Auditing Engineer I • Zeeshan Meghji Auditing Engineer • Roman Rohleder Senior Research Engineer

Documentation quality	Medium
Test quality	Medium
Total Findings	19 Fixed: 11 Acknowledged: 5 Mitigated: 3
High severity findings	2 Fixed: 2
Medium severity findings	2 Mitigated: 2
Low severity findings	7 Fixed: 5 Acknowledged: 1 Mitigated: 1
Undetermined severity findings	1 Fixed: 1
Informational findings	7 Fixed: 3 Acknowledged: 4

Summary of Findings

The Talentir Marketplace is for content creators to publish tokens representing their media and for their fans to trade them. The tokens that can be minted on the Talentir platform conform to the ERC-1155 standard so that creators can release multiple copies of the same token. This results in a marketplace that combines elements of fungible and non-fungible marketplaces.

The DEX is designed as an order book where users specify a price that gets executed whenever a willing counter-party submits the other side of the order. The order book is a tuple consisting of a RB-tree whose nodes represent the prices and a mapping that takes a price to a linked list of orders that were submitted at that price level. One high severity issue was found involving insecure use of the methods in this data structure (TAL-1).

Overall the code is well-written and the NatSpec and documentation helped clarify the inner workings of the code. Most of the critical findings center around the external calls made in order to transfer ether. There are many opportunities that can lead to malicious users degrading or denying protocol service using these external calls (TAL-2). However, these can be mitigated by implementing a withdrawal pattern.

UPDATE: The team has addressed all the relevant issues or acknowledged and clarified them.

ID	DESCRIPTION	SEVERITY	STATUS
TAL-1	Incorrect Accounting in `_removeOrder()` Leading to Loss of Orders	High	Fixed
TAL-2	Denial of Service Through Expensive ETH Receiver	High	Fixed
TAL-3	Approval Not Updated when Changing the Marketplace Address in `TalentirTokenV1`	Medium	Mitigated
TAL-4	REDACTED	Undetermined	Fixed
TAL-5	Denial of Service Through Small Order Spamming	Medium	Mitigated

ID	DESCRIPTION	SEVERITY	STATUS
TAL-6	Loss Of Precision May Lead To Unfavourable Execution	Low	Fixed
TAL-7	Gas Usage / Loop Concerns	Low	Mitigated
TAL-8	Missing Input Validation	Low	Fixed
TAL-9	Content Identifiers Can Be Duplicates	Low	Fixed
TAL-10	Unchecked Return Value	Low	Fixed
TAL-11	Privileged Roles and Ownership	Low	Acknowledged
TAL-12	ERC-1155 Tokens May Be Locked In the MarketPlace	Low	Fixed
TAL-13	Incorrect Computation in `TalentirMarketplaceV1.calcTalentirFee()`	Informational	Fixed
TAL-14	`TalentirMarketplaceV1.cancelOrders()` Front-Runnable	Informational	Acknowledged
TAL-15	Unnecessary State Variable	Informational	Fixed
TAL-16	Missing Event Emissions	Informational	Fixed
TAL-17	Creator Fees May Still Be Avoided	Informational	Acknowledged
TAL-18	Unnecessary Approval For Marketplace	Informational	Acknowledged
TAL-19	Unlucky Users Could Waste Gas on Orders that Do Not Get Filled.	Informational	Acknowledged

Assessment Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Disclaimer

Only features that are contained within the repositories at the commit hashes specified on the front page of the report are within the scope of the audit and fix review. All features added in future revisions of the code are excluded from consideration in this report.

In particular, the libraries `LinkedListLibrary.sol` and `RBTLibrary.sol` were outside of the scope of this audit. The marketplace relies heavily on the proper functioning of these libraries and our working assumption throughout the audit is that these libraries work as intended with no unforeseen bugs or errors. Any bugs in these libraries will likely have a notable impact on the Talentir contracts.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage

- Arbitrary token minting

Methodology

1. Code review that includes the following
    1. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
    2. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
    3. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
    1. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    2. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarity, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Findings

TAL-1
Incorrect Accounting in `_removeOrder()` Leading to Loss of Orders

• High Fixed

> **Update**
> The Talentir team fixed the issue in commit 29e5bc9a49f3af07714752fc458398b1052e8459 by removing the specific order ID from the list.

File(s) affected: `TalentirMarketplaceV1`

Description: The function `TalentirMarketplaceV1._addOrder()` only ever appends new orders (in increasing order ID number) to the front of the linked list `markets[_tokenId][_side].orderList[_price]`.

```
markets[_tokenId][_side].orderList[_price].push(nextOrderId, true);
```

However, the function `TalentirMarketplaceV1._removeOrder()` always removes the last element/order from the linked list `markets[tokenId][side].orderList[price]` in `L378`, without confirming it being the to-be-deleted element:

```
markets[tokenId][side].orderList[price].pop(false);
```

This could lead to the loss of order IDs other than the desired one. While other "copies" of such deleted order IDs exist (`orders` and `userOrders`), the function `getBestOrder()` always takes the first inserted element of that linked list, which in turn is used when filling existing orders when creating a new one. Deleting and reading order IDs this way would lead to a deteriorated order book, where orders of other users may be (maliciously) removed from the order book.

Exploit Scenario: Suppose a user places two orders chronologically, A and B, that are with the same token, with the same price, and on the same side. If this user goes to cancel order B using the `cancelOrder()` function, order `A` will be canceled instead.

Recommendation: We recommend redesigning the use of the `orderList` linked list, i.e. such that function `_removeOrder()` traverses the list until it finds the matching ID.

TAL-2 Denial of Service Through Expensive ETH Receiver

• High Fixed

> **Update**
> PR-37 implements The withdrawal pattern for both the native cryptocurrency (e.g. ETH) and Talentir ERC-1155 tokens.

File(s) affected: `TalentirMarketplaceV1`

Description: A malicious seller can submit a sell order which causes a revert on any subsequent buy transaction. The attack is made possible by two components. Firstly, the addition of a very cheap sell order guarantees its execution the next time a user submits a buy order. Secondly, when the seller receives ETH, the fallback function in their contract can be used to expend the transaction's gas such that it does not have enough to complete the order.

The `TalentirMarketplaceV1` allows users to either buy Talentir tokens with the chain's native currency, such as ETH or sell Talentir tokens in exchange for the chain's native currency. If a user submits a buy order, `TalentirMarketplaceV1` first tries to find the cheapest sell order to

fill the buy order with. Thus, if a malicious seller is able to submit a sell order with the lowest price, this order is always guaranteed to be used to fill the next order. This can be seen in the function which is used to retrieve the appropriate sell order:

```
function getBestOrder(uint256 _tokenId, Side _side) public view returns (uint256, uint256) {
    uint256 price = _side == Side.BUY
        ? markets[_tokenId][_side].priceTree.last()
        : markets[_tokenId][_side].priceTree.first();
    uint256 bestOrderId;
    (, bestOrderId, ) = markets[_tokenId][_side].orderList[price].getNode(0);
    return (bestOrderId, price);
}
```

When the order actually executes, ETH (or another native coin) is sent to the seller through a low-level `call()`. According to EIP-150, by default `63/64` of the remaining gas can be used by default in a low-level call. Several actions happen after this call, including the emission of an event and ETH transfers to other addresses. If the `call()` to the seller expends too much gas, then these other actions will fail as there is not enough gas left to execute them:

```
function _executeOrder(
    address _sender,
    uint256 _orderId,
    uint256 _quantity
) internal returns (uint256 ethQuantity) {
    ...
    (locals.success, ) = locals.seller.call{value: locals.payToSeller}("");
    (locals.success, ) = locals.royaltiesReceiver.call{value: locals.royalties}("");
    (locals.success, ) = talentirFeeWallet.call{value: locals.talentirFee}("");

    emit OrderExecuted(
    ...
}
```

By placing a very cheap sell order which would invoke an expensive fallback function, the malicious seller is able to stop the marketplace from functioning for that `tokenId`. All subsequent buy orders would revert. Subsequent sell orders would not revert but would never get executed due to the buy orders reverting.

Note that the same exploit is also possible to perform on the buy side, by offering a high price and then implementing a malicious ERC-1155 receiver function.

Exploit Scenario:

1. Suppose we have a `TalentirMarketplaceV1` contract with no orders yet.
2. An attacker creates an attack contract with the following functions and transfers `1` Talentir token to it:
    1. `placeCheapSellOrder()`: Call `TalentirMarketplaceV1.makeSellOrder()` to place a sell order at a very low price.
    2. `fallback()`: Has an infinite loop of expensive operations.
3. The attacker calls `placeCheapSellOrder()` on the attack contract.
4. A user called `user_1`, places a normal sell order.
5. Another user called `user_2` places a buy order which matches the sell order placed by `user_1`.
    1. The hacker's sell order is executed first, causing a revert due to the transaction running out of gas.

Recommendation: Do not return the ETH (or other native currency) to the seller when the corresponding buy order is placed and executed. Instead, add a function to the marketplace contract which allows the seller to retrieve the proceeds from the sale. This would be an implementation of the withdrawal pattern.

The same pattern should be implemented for withdrawing the NFTs since the same exploit is possible on the buy side.

TAL-3
Approval Not Updated when Changing the Marketplace Address in `TalentirTokenV1`

Medium | Mitigated

Update

In PR-41, the team added functionality that allows the `owner` to remove approval of an address for a list of user wallets. This functionality adds some centralization risk, as the owner is free to revoke approval of any address for any user, but solves the issue.

File(s) affected: `TalentirTokenV1`

Description: When the marketplace address is changed in `TalentirTokenV1.setMarketplace()` only newly minted tokens will have it approved for all, while previously minted tokens will still have the old marketplace address approved for all. This would lead the tokens to be at risk of token owners of previously minted tokens until they manually revoke/update the approval via `setApprovalForall()`.

Recommendation: Consider adding a second blocklist address that would block all outdated marketplaces from interacting with users' ERC-1155 tokens, or at the very least, ensure that users are reminded to revoke old permissions.

TAL-4 REDACTED

• **Undetermined** Fixed

Description: Flase positive removed from report.

TAL-5 Denial of Service Through Small Order Spamming

• **Medium** Mitigated

Update

The issue has been mitigated by modifying the `cancelOrder()` function to be callable by the contract owner. This will allow the Talentir team to cancel problematic orders. Note that this also adds an element of centralization, namely, the team can cancel any order at their discretion. Since there are very few scenarios where the team would benefit from canceling legitimate orders on their platform we are not too concerned by the centralization risk introduced by this added functionality. While this will not prevent the issue from happening, the added functionality provides a workaround which when combined with monitoring and off-chain code will greatly limit the effect. The change was made in PR-37.

Description: A hacker may submit many tiny orders, either cheap sell orders or very expensive buy orders. In doing so, they would guarantee that these small orders would be executed first. If there is a sufficient number of these small orders, then normal buy or sell orders would run out of gas before they fill all the small orders. We will explain the problem of having many tiny expensive buy orders. However, the corresponding problem with sell orders is almost the same.

The `TalentirMarketplaceV1` allows users to either buy Talentir tokens with the chain's native currency, such as ETH or sell Talentir tokens in exchange for the chain's native currency. If a user submits a sell order, `TalentirMarketplaceV1` first tries to find the most expensive buy order to fill the sell order with. Thus, if a malicious buyer is able to submit many tiny buy orders with the highest price, these orders are always guaranteed to be used to fill the next sell order. This can be seen in the function which is used to retrieve the appropriate buy order:

```
function getBestOrder(uint256 _tokenId, Side _side) public view returns (uint256, uint256) {
    uint256 price = _side == Side.BUY
        ? markets[_tokenId][_side].priceTree.last()
        : markets[_tokenId][_side].priceTree.first();
    uint256 bestOrderId;
    (, bestOrderId, ) = markets[_tokenId][_side].orderList[price].getNode(0);
    return (bestOrderId, price);
}
```

The function for executing a normal sell order will loop over the buy orders to fill it, starting from the most expensive buy order. Thus if the malicious user places enough expensive buy orders, this function is forced to loop through all of them, potentially reaching the gas limit and failing:

```
function _makeOrder(
    address _sender,
    uint256 _tokenId,
    Side _side,
    uint256 _ethQuantity,
    uint256 _tokenQuantity,
    bool _addOrderForRemaining
) internal {
    ...
    (bestOrderId, bestPrice) = getBestOrder(_tokenId, oppositeSide);
    // If possible, buy up to the specified price limit
    uint256 remainingQuantity = _tokenQuantity;
    while (
        (remainingQuantity > 0) &&
        ((_side == Side.BUY) ? price >= bestPrice : price <= bestPrice) &&
        (bestOrderId > 0)
    ) {
        ...
        if (remainingQuantity > 0) {
```

```
            (bestOrderId, bestPrice) = getBestOrder(_tokenId, oppositeSide);
        }
    }
    ...
}
```

Exploit Scenario:

1. Suppose we have a `TalentirMarketplaceV1` contract with no buy orders yet.
2. An attacker places `1000` sell orders to sell `1` Talentir token for `1` Wei.
3. A normal user tries to buy `1000` Talentir tokens.
    1. This will result in an attempt to fill all `1000` sell orders placed by the attacker.
    2. The transaction may hit the gas limit and revert.

Recommendation: Consider introducing minimum amounts for the orders to make an attack like this more expensive and less feasible. Alternatively, the team can monitor and remove spam orders if they occur to free up the marketplace.

Finally, further care can be taken to minimize the gas consumption of the `_makeOrder()` function as outlined in TAL-7.

TAL-6 Loss Of Precision May Lead To Unfavourable Execution

Low Fixed

Update

The Talentir team fixed the issue in PR-38 by using a scaling factor of one million in the price calculation. Note that the scaling factor being one million is based on the number of ERC-1155 tokens for a particular token ID. If that number changes due to an updated Talentir token contract, then this scaling factor should change as well (perhaps through a new version of the marketplace contract).

File(s) affected: `TalentirMarketplaceV1`

Description: The `TalentirMarketplaceV1` contract divides the amount of ETH by the amount of Talentir tokens to sell or buy to store the price. However, this will result in an immediate loss of precision in the price and an indirect loss of precision in subsequent calculations. The loss of precision will at the very least result in sellers' getting less ETH (or other native currency) than they should.

Regardless of whether a user places a sell or a buy order, the price is always calculated using the following code:

```
unction _makeOrder(
    address _sender,
    uint256 _tokenId,
    Side _side,
    uint256 _ethQuantity,
    uint256 _tokenQuantity,
    bool _addOrderForRemaining
) internal {
    ...
    uint256 price = _ethQuantity / _tokenQuantity;
    ...
}
```

This price is then used to determine which orders are suitable for filling orders coming in on the opposite side. The loss of precision in the price calculation can result in an unfavorable result for the Talentir token seller, as shown in the exploit scenario.

Exploit Scenario:

1. A seller places an order for selling `1_000_000` Talentir tokens for a minimum price of `1_999_999` Wei.
    1. The price calculated and stored is `1_999_999/1_000_000==1`.
2. A buyer places an order to buy `1_000_000` Talentir tokens for a maximum price of `1_000_000` Wei
    1. The initial sell order gets executed, sending just `1_000_000` Wei to the seller.
3. The seller had asked for `1_999_999` Wei minimum but received only `1_000_000` Wei due to a loss of precision in the price calculation.

Recommendation: Multiply the `_ethQuantity` by the maximum supply for the Talentir token Id before dividing by the `_tokenQuantity`. In the current implementation, the maximum supply would be one million.

TAL-7 Gas Usage / Loop Concerns

Low Mitigated

Update

Acknowledged by the developers with:

> In TAL-5 we added the ability for the owner() to remove orders. In case of too many orders, our backend has the ability to clean up the order book by removing the most expensive sell orders and the cheapest buy orders.

File(s) affected: `TalentirMarketplaceV1`

Related Issue(s): SWC-126, SWC-134

Description: Gas usage is a main concern for smart contract developers and users, since high gas costs may prevent users from wanting to use the smart contract. For example, if a loop requires too much gas to finish processing, then it may prevent the contract from functioning correctly entirely.

When creating an order the while loop in `_makeOrder()` could run out of gas if there is a very large amount of orders on the opposite side. Even in the case where the orders are benign, some gas concerns may occur.

Recommendation: If possible, we recommend breaking loops into individual functions and/or adding function arguments that allow users to continue loop processing in a separate transaction.

We recommend performing a gas usage analysis in cases where many (small) orders are created/exist and are tried to be filled by an incoming big order, documenting the limitation if necessary, or considering a redesign of the data structures used.

TAL-8 Missing Input Validation

Low Fixed

> **Update**
> The Talentir team added all recommended validations. The changes were made in PR-32.

File(s) affected: `TalentirTokenV1`, `TalentirMarketplaceV1`

Related Issue(s): SWC-123

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error. We have listed all missing validations below:

1. `TalentirTokenV1`
    1. `updateTalent()`
        1. Validate that `talent` is not the zero address.
    2. `mint()`
        1. Validate that `talent` is not the zero address.
        2. Validate that `cid` and `contentID` are not empty strings.
    3. `setGlobalPresaleAllowance()`: Validate that `user` is not the zero address.
    4. `setTokenPresaleAllowance()`:
        1. Validate that `user` is not the zero address.
        2. Validate that `tokenId` already exists.
    5. `setMinterRole()`: Validate that `minterAddress` is not the zero address.
    6. `setMarketplace()`: Validate that `marketplace` is not the zero address.
2. `TalentirMarketplaceV1`
    1. `constructor()`: Validate that `_talentirNFT` is not the zero address.
    2. `setTalentirFee()`: Validate that `_wallet` is not the zero address.
    3. `_makeOrder()`: Validate that the `_tokenQuantity` is greater than the maximum number of fractions a token can have, i.e. `1_000_000`.

Recommendation: We recommend adding the relevant checks.

TAL-9 Content Identifiers Can Be Duplicates

Low Fixed

> **Update**
> The Talentir team fixed the issue in PR-25 by tracking the content IDs within a mapping in the contract and ensuring that new tokens have a unique content ID.

File(s) affected: `TalentirTokenV1`

Description: The `mint()` function makes sure that the `contentID` converted to `tokenId` is unique. However, the IPFS `cid` can be duplicated, which would allow multiple `tokenId`s to have the same URI.

Recommendation: We recommend adding another variable (i.e. `mapping(string => bool)`) to keep track of all the `cid`s, and to add a check that fails if the `cid` was already included.

TAL-10 Unchecked Return Value

Low | Fixed

> **Update**
> The issue was fixed in PR-37 by using the new `_ethTransfer()` internal function which ensures that the ether transfer calls are successful.

File(s) affected: `TalentirMarketplaceV1`

Related Issue(s): SWC-104

Description: Most functions will return a `True` or `False` value upon success. Some functions, like `send()`, are more crucial to check than others. It is important to ensure that every necessary function is checked.

In particular, consider the following instances:

1. `TalentirMarketplaceV1.sol#L186` in function `cancelOrders()`.
2. `TalentirMarketplaceV1.sol#L279` in the function `_makeOrder()`.
3. `TalentirMarketplaceV1.sol#L323` in the function `_executeOrder()`.
4. `TalentirMarketplaceV1.sol#L324` in the function `_executeOrder()`.
5. `TalentirMarketplaceV1.sol#L325` in the function `_executeOrder()`.

Recommendation: We recommend adding checking the return value of external calls to make sure they function properly. However, this should be implemented in tandem with a withdrawal pattern in order to prevent users with malicious `fallback()` and `receive()` functions.

TAL-11 Privileged Roles and Ownership

Low | Acknowledged

> **Update**
> The Talentir team acknowledged the issue. Furthermore we note that the owner role now has some additional permissions:
> - The owner can cancel orders.
> - The owner can now revoke approvals from the old marketplace/minter when changing the marketplace/minter.

File(s) affected: `TalentirTokenV1`, `TalentirMarketplaceV1`

Description: Certain contracts have state variables, e.g. `owner`, which provide certain addresses with privileged roles. Such roles may pose a risk to end-users.

The `TalentirTokenV1.sol` contract contains the following privileged roles:

1. An administrative/Owner role (`_owner`, `onlyOwner()` modifier), as initialized during the `constructor()` execution to `msg.sender`:
    1. Assign a new `_owner` address by calling `transferOwnership()`.
    2. Renounce the role (**and thereby preventing any future calls to the followingly listed funcitons!**) by calling `renounceOwnership()`.
    3. Pause/Unpause certain contract functionality by calling `pause()` / `unpause()`:
        1. The minting of new tokens (`mint()`).
        2. Approving addreses for all token transfers (`setApprovalForAll()`).
        3. Transfer multiple amounts of the same token id to different addresses (`batchTransfer()`).
        4. Perform token transfers (`safeTransferFrom()`).
        5. Perform token transfers of multiple token id's and amounts(`safeBatchTransferFrom()`).
    4. Change the royalty percentage for all tokes (any value up to `10%`) by calling `setRoyalty()`.
    5. Change the minter role address by calling `setMinterRole()`.
    6. Change the marketplace address by calling `setMarketplace()`.
2. A minter role (`_minterAddress`, `onlyMinter()` modifier) as set through the owner role (`_owner`) by calling `setMinterRole()`:
    1. Mint new tokens by calling `mint()`.
    2. **Arbitrarily transfer tokens on others behalf as being approved for all transfers until manually revoked by the talent/token owner by calling** `setApprovalForAll(_minterAddress, false)`.
    3. Add/Remove addresses from being able to buy any tokens during presales (`hasGlobalPresaleAllowance[addr]`) by calling `setGlobalPresaleAllowance()`.
    4. Add/Remove addresses from being able to buy specific tokens during presales (`hasTokenPresaleAllowance[addr][token]`) by calling `setTokenPresaleAllowance()`.
    5. End the presale phase for any token at any time (thereby opening up the transfer to all addresses) by calling `endPresale()`.
3. Token transfers may be blocked (`batchTransfer()`, `safeTransferFrom()` and `safeBatchTransferFrom()`), due to the use of "Operators filterer" as defined/governed by OpenSea's operator-filter-registry (with address 0×3cc6CddA760b79bAfa08dF41ECFA224f810dCeB6 being the filter contract address at version 1.4.0, at the time of publishing this report).
4. A marketplace address, as set through `setMarketplace()` by the owner role (`_owner`):
    1. **Arbitrarily transfer tokens on others behalf as being approved for all transfers until manually revoked by the talent/token owner by calling** `setApprovalForAll(_minterAddress, false)`.

The `TalentirTokenV1.sol` contract contains the following privileged roles:

1. An administrative/Owner role (`_owner`, `onlyOwner()` modifier), as initialized during the `constructor()` execution to `msg.sender`:
    1. Assign a new `_owner` address by calling `transferOwnership()`.

2. Renounce the role (**and thereby preventing any future calls to the followingly listed funcitons!**) by calling `renounceOwnership()`.
3. Pause/Unpause certain contract functionality by calling `pause()` / `unpause()`:
    1. Creating sell orders (`makeSellOrder()`).
    2. Creating buy orders (`makeBuyOrder()`).
4. Change the Talentir fees (order fees) up to 10% at any time (`setTalentirFee()`).
5. Change the Talentir fee receiving address (`talentirFeeWallet`) at any time (`setTalentirFee()`).

Recommendation: Clarify the impact of these privileged actions to the end-users via publicly facing documentation.

TAL-12 ERC-1155 Tokens May Be Locked In the MarketPlace

Low Fixed

> **Update**
>
> The Talentir team fixed the issue by implementing the recommendation in PR-34.

File(s) affected: `TalentirMarketplaceV1`

Description: The `TalentirMarketplaceV1` inherits from `ERC1155Holder`, which unconditionally accepts any ERC-1155 tokens as shown below;

```
contract ERC1155Holder is ERC1155Receiver {
    function onERC1155Received(
        address,
        address,
        uint256,
        uint256,
        bytes memory
    ) public virtual override returns (bytes4) {
        return this.onERC1155Received.selector;
    }

    function onERC1155BatchReceived(
        address,
        address,
        uint256[] memory,
        uint256[] memory,
        bytes memory
    ) public virtual override returns (bytes4) {
        return this.onERC1155BatchReceived.selector;
    }
}
```

This can result in non-Talentir ERC-1155 tokens becoming stuck in the contract. It may also result in Talentir tokens being locked in the marketplace contract if they are not transferred only through the use of the `makeSellOrder()` function.

Recommendation: Override `onERC1155Received()` and `onERC1155BatchReceived()` in `TalentirMarketplaceV1` such that both functions:

1. Validate that the ERC-1155 token being transferred is a Talentir token
2. Validate that the Talentir token is being transferred through the `makeSellOrder()` function. This could be done by setting a flag `makingSellOrder` on the contract when `makeSellOrder()` has been called and then unsetting the flag at the end of `makeSellOrder()`

TAL-13 Incorrect Computation in `TalentirMarketplaceV1.calcTalentirFee()`

Informational Fixed

> **Update**
>
> The Talentir team fixed the issue in PR-28 by removing the division and multiplication by 100.

File(s) affected: `TalentirMarketplaceV1`

Description: The computation of the talentir fee amount for a given total price as computed in `TalentirMarketplaceV1.calcTalentirFee()` unnecessarily multiplies and then at the end divides by `100`:

```
return ((100 * talentirFeePercent * _totalPaid) / PERCENT) / 100;
```

As the contract is using solidity version 0.8.17, which implicitly performs underflow/overflow checks on arithmetic operations, providing a total price higher than

```
0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff / (100 * 100_000)
```

which is equal to:

```
0x10c6f7a0b5ed8d36b4c7f34938583621fafc8b0079a2834d26fa3fcc9ea9
```

will lead to overflows and revert the operations and thereby preventing the execution on such orders.

Recommendation: We recommend removing the multiplication and division by `100` to save on gas and increase the range of allowed total order prices.

TAL-14 `TalentirMarketplaceV1.cancelOrders()` Front-Runnable

• Informational ⓘ Acknowledged

Update

The Talentir team updated the documentation surrounding the cancel order function to inform users. We recommend also informing users on the front-end of the application. The change was made in PR-40.

File(s) affected: `TalentirMarketplaceV1`

Description: Function `TalentirMarketplaceV1.cancelOrders()` may be front-runnable. This may be abused when the order owner wants to cancel one or more unfavorable market orders, while the front-runner observes pending transactions in the mempool, determines their profitability and executes a corresponding filling order with higher gas costs than that of the `cancelOrders()` call in order to have it executed before.

Recommendation: We recommend documenting this possibility in user-facing documentation. Consider informing users of private mempools, i.e. flashbots, which can be used to mitigate against front-running.

TAL-15 Unnecessary State Variable

• Informational ⓘ Fixed

Update

The issue was fixed by removing the `userOrders` mapping and all the associated code. The change was made in PR-30.

Description: The state variable `userOrders` is being updated when creating and removing orders. However, it is not used for any important task inside the contract. However, assuming it is for the front-end, the variable lacks the correct visibility modifier or a getter function for easier access.

Recommendation: We recommend changing the visibility modifier to `public` or adding a getter function. Using `userOrders` together with `orders`, the front-end can allow users to quickly verify their orders. However, if it is not used outside of the contract either, we recommend removing it.

TAL-16 Missing Event Emissions

• Informational ⓘ Fixed

Update

The recommended events have been emitted. The change was made in PR-29.

File(s) affected: `TalentirMarketplaceV1`, `TalentirTokenV1`

Description: In order to validate the proper deployment and initialization of the contracts, it is a good practice to emit events. Also, any important state transitions can be logged, which is beneficial for monitoring the contract, and also tracking eventual bugs or hacks. Below we present a non-exhaustive list of events that could be emitted to improve application management.

1. `TalentirTokenV1`
    1. Emit an event from `setMinterRole()`.
    2. Emit an event from `setMarketplace()`.
2. `TalentirMarketplaceV1`

1. `_removeOrder()` : Further information should be included in this event such as `_orderId` `orders[_orderId].sender` , `price` , `side` and `tokenId` .

Recommendation: Emit the recommended events.

TAL-17 Creator Fees May Still Be Avoided

Informational Acknowledged

> **Update**
> The Talentir team acknowledged the issue.

File(s) affected: `TalentirMarketplaceV1` , `TalentirTokenV1`

Description: The `TalentirTokenV1` contract uses OpenSea's operator filter registry in order to blacklist the addresses and bytecode of markets that do not respect creator royalties. However, it is important to realize that the blacklist is not exhaustive, and it is still technically possible to circumvent creator royalties through a market that has not yet been blacklisted.

Recommendation: There is no known comprehensive solution to this problem. We recommend acknowledging the issue and ensuring that the users know about the risks through documentation.

TAL-18 Unnecessary Approval For Marketplace

Informational Acknowledged

> **Update**
> The Talentir team acknowledged the approval for the marketplace and explained that it was necessary in order to provide a gasless experience for the token minters/talents when posting their first sale.

File(s) affected: `TalentirTokenV1`

Description: Within the `TalentirTokenV1.mint()` function, approval is given for all tokens belonging to the receiver of the minted tokens to both the `_approvedMarketplace` and the `minterAddress` . The Talentir team explained that the `minterAddress` represents their backend which should automatically place the first order on the market. However, there appears to be no reason for the approval of the `_approvedMarketplace` .

Recommendation: Consider removing the auto-approval for `_approvedMarketPlace` upon minting. Alternatively, add a code comment explaining why this is done.

TAL-19 Unlucky Users Could Waste Gas on Orders that Do Not Get Filled.

Informational Acknowledged

> **Update**
> The Talentir indicated that this is the intended behavior. They stated:
>
> > We would consider this a feature, not a bug: The user can ensure the transaction isn't executed if the price changes between submitting the transaction and it being executed.

File(s) affected: `TalentirMarketplaceV1`

Description: If a user makes a buy/sell order with `_addOrderForRemaining == False` , then if the user input price is greater/lower than the `bestPrice` at the moment, their transaction will go through and and they will have effectively accomplished nothing. This could be particularly frustrating if the matching `bestPrice` existed at the time of submitting their transaction.

Recommendation: Ensure that users are aware of the risk of a transaction that results in no buying or selling if there are no matches at the time the transaction is being processed.

Definitions

- **High severity** – High-severity issues usually put a large number of users' sensitive information at risk, or are reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.

- **Medium severity** – Medium-severity issues tend to put a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or are reasonably likely to lead to moderate financial impact.
- **Low severity** – The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
- **Informational** – The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
- **Undetermined** – The impact of the issue is uncertain.
- **Fixed** – Adjusted program implementation, requirements or constraints to eliminate the risk.
- **Mitigated** – Implemented actions to minimize the impact or likelihood of the risk.
- **Acknowledged** – The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).

Code Documentation

1. Spurious code comment in `TalentirERC2981.sol#L14`, which should be removed.
2. Unfinished sentence (`A contentID`) in code comment in `TalentirTokenV1.sol#L51`.
3. The following typographical errors have been noted:
    1. `TalentirTokenV1.sol#L172` : `param` → `@param`.
4. Missing or incorrect NatSpec comments:
    1. `TalentirMarketplaceV1.getBestOrder()` : Missing NatSpec comment for parameter `_side`.
    2. `TalentirMarketplaceV1.calcTalentirFee()` : Spurious spaces in front of NatSpec keywords.
    3. `TalentirMarketplaceV1.sol#L136` : `token` → `Ether`.
    4. `TalentirMarketplaceV1.sol#L154` : `Ether` → `ERC1155`.
5. Functions `getBestOrder()` and `calcTalentirFee()` in contract `TalentirMarketplaceV1.sol` are publicly callable but not listed under the `/// PUBLIC FUNCTIONS ///` code comment.
6. Make sure to document state variables, data structures, events and functions. In the `TalentirTokenV1` as well as in the `TalentirMarketplaceV1` contract, all member variables and events are missing their documentation. It is especially important for `mappings` to better understand what the key and value types represent.
7. The code comments inside the `TalentirTokenV1.mint()` state that the approvals can be revoked by the `talent`. However, that is not true. Only the `owner` of the token is able to.
8. Although some functions are documented according to the NatSpec standard, some functions remain undocumented such as `TalentirTokenV1.batchTransfer`.
9. Events should also be documented using the NatSpec standard.
10. The code comment for the `_for` parameter of the `TalentirMarketplaceV1` token may be misleading. It states `recipient who will receive the token`. This may mislead the reader into thinking the `_for` parameter will receive the Talentir token. However, in the case of a sell order, the `_for` parameter is where the token is transferred from.

Adherence to Best Practices

1. Since the following functions are never called by the contract they are declared in, consider changing their visibility to `external`:
    - `TalentirTokenV1` :
        - `updateTalent()`
        - `getTalent()`
        - `mint()`
        - `setGlobalPresaleAllowance()`
        - `setTokenPresaleAllowance()`
        - `endPresale()`
        - `setRoyalty()`
        - `setMinterRole()`
        - `setMarketplace()`
        - `batchTransfer()`
    - `TalentirERC2981` :
        - `royaltyInfo()`
2. Consider using custom errors, as they save gas, allow for better error handling, and data can be passed from contract to contract.
3. Remove dead or unused code:
    - Event `TalentirTokenV1.MarketplaceApproved` is not used.
4. Gas Optimizations:
    - `TalentirMarketplaceV1.talentirNFT` is set only once, and is marked as immutable to save gas. Also consider using the contract type, `TalentirTokenV1` , as data type instead of address.
    1. The function `TalentirMarketplaceV1._executeOrder()` computes `order.price * _quantity` 5 times. Storing the value in a temporary local variable will make the function less expensive.
5. Code readability:
    - In the function `TalentirMarketplaceV1._makeOrder()`, the instruction `(bestOrderId, bestPrice) = getBestOrder(_tokenId, oppositeSide);` can be replaced by `(uint256 bestOrderId, uint256 bestPrice) =`

`getBestOrder(_tokenId, oppositeSide);`, which would improve the code readability around it. The same approach can be used in `TalentirMarketplaceV1.getBestOrder()` when getting the `bestOrderId`.

6. To facilitate logging it is recommended to index address parameters within events. Therefore the `indexed` keyword should be added to the (other) address parameters in
   1. `TalentirTokenV1.MarketplaceApproved()`,
   2. `TalentirTokenV1.TalentChanged()`,
   3. `TalentirTokenV1.GlobalPresaleAllowanceSet()`,
   4. `TalentirTokenV1.TokenPresaleAllowanceSet()`,
   5. `TalentirMarketplaceV1.TalentirFeeSet()`.
7. For clarity and usability it is advised to use expressive code elements (i.e. revert/error messages, variable names, ...). In this regard, consider changing the following instances:
   1. The revert message in `TalentirTokenV1.sol#34` to be more expressive (i.e. `Not minter`).
   2. `TalentirERC2981.sol#L12`: `PERCENT` → `ONE_HUNDRED_PERCENT`.
   3. `TalentirMarketplaceV1.sol#L71`: `PERCENT` → `ONE_HUNDRED_PERCENT`.
8. To improve readability and lower the risk of introducing errors when making code changes, it is advised to not use magic constants throughout code, but instead declare them once (as constant and commented) and use these constant variables instead. The following instances should therefore be changed accordingly:
   1. `TalentirTokenV1.sol#L160`: `10_000` (Consider declaring a global constant, derived from `TalentirERC2981.PERCENT`, i.e. `uint256 internal constant PERCENT = PERCENT / 10;`).
9. Consider incrementing the counter within for-loops within an `unchecked` block to save gas.
10. Cache the length of an array in memory and then reference the memory variable within for-loop for gas savings.

Adherence to Specification

1. The specification states that the token contract has a set of approved marketplaces. However, the implementation only contains a single approved marketplace.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.9.3

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Automated Analysis

Slither

Slither results were either false positives or included in the report.

Test Suite Results

```
Talentir Marketplace Tests
  ✔ deploy (81ms)
  ✔ should open and close a single order (no fees) (231ms)
  ✔ should distribute fees correctly (96ms)
  ✔ should handle multiple orders (384ms)
  ✔ should pause and cancel (124ms)
  ✔ make orders on behalf of other accounts (68ms)
  ✔ async transfer / pull payment (194ms)
  ✔ owner can cancel orders (48ms)
  ✔ order removed in the correct order (108ms)
  ✔ precision

Talentir Token Tests
  ✔ Uri
```

```
    ✔ disallows interactions from unpermitted accounts
    ✔ mints (48ms)
    ✔ can approve a marketplace & minter to transfer tokens (66ms)
    ✔ can approve an account to transfer tokens
    ✔ pays out royalties (43ms)
    ✔ allows pausing (73ms)
    ✔ responds with its interfaces
    ✔ allows setting talents
    ✔ makes batch transfers (61ms)
    ✔ handles minting with presale (201ms)


  21 passing (3s)
```

Code Coverage

```
----------------------------|----------|----------|----------|----------|----------------|
File                        |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
----------------------------|----------|----------|----------|----------|----------------|
 contracts/                 |      100 |    93.41 |      100 |      100 |                |
  TalentirMarketplaceV1.sol |      100 |     90.7 |      100 |      100 |                |
  TalentirTokenV1.sol       |      100 |    95.83 |      100 |      100 |                |
 contracts/libraries/       |    52.42 |    42.45 |    53.57 |    45.49 |                |
  LinkedListLibrary.sol     |    48.72 |    41.67 |    63.64 |    47.92 |... 189,191,193 |
  RBTLibrary.sol            |    54.12 |    42.68 |    47.06 |    44.86 |... 326,327,328 |
 contracts/utils/           |      100 |      100 |      100 |      100 |                |
  ERC1155PullTransfer.sol   |      100 |      100 |      100 |      100 |                |
  TalentirERC2981.sol       |      100 |      100 |      100 |      100 |                |
----------------------------|----------|----------|----------|----------|----------------|
All files                   |    79.15 |    75.17 |    82.67 |    71.27 |                |
----------------------------|----------|----------|----------|----------|----------------|
```

Changelog

- 2023-04-11 - Initial report

About Quantstamp

Quantstamp is a global leader in blockchain security. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its best-in-class Web3 security products and services.

Quantstamp's team consists of cybersecurity experts hailing from globally recognized organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Quantstamp engineers hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 500 audits and secured over $200 billion in digital asset risk from hackers. Quantstamp has worked with a diverse range of customers, including startups, category leaders and financial institutions. Brands that Quantstamp has worked with include Ethereum 2.0, Binance, Visa, PayPal, Polygon, Avalanche, Curve, Solana, Compound, Lido, MakerDAO, Arbitrum, OpenSea and the World Economic Forum.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by Quantstamp; however, Quantstamp does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication or other making available of the report to you by Quantstamp.

Notice of confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites&aspo; owners. You agree that Quantstamp are not responsible for the content or operation of such web sites, and that Quantstamp shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that Quantstamp endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. Quantstamp assumes no responsibility for the use of third-party software on any website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any output generated by such software.

Disclaimer

The review and this report are provided on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Quantstamp disclaims all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. You agree that your access and/or use of the report and other results of the review, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE. This report is based on the scope of materials and documentation provided for a limited review at the time provided. You acknowledge that Blockchain technology remains under development and is subject to unknown risks and flaws and, as such, the report may not be complete or inclusive of all vulnerabilities. The review is limited to the materials identified in the report and does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. The report does not indicate the endorsement by Quantstamp of any particular project or team, nor guarantee its security, and may not be represented as such. No third party is entitled to rely on the report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. Quantstamp does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, or any related services and products, any hyperlinked websites, or any other websites or mobile applications, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third party. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.