SPEARBIT

Sudoswap LSSVM2 Security Review

Auditors

Gerard Persoon, Lead Security Researcher

Rajeev, Lead Security Researcher

Shodan, Security Researcher

Lucas Goiriz, Junior Security Researcher

David Chaparro, Junior Security Researcher

Report prepared by: Pablo Misirov

April 10, 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

Sudoswap is a minimal, gas-efficient automated market maker (AMM) protocol that facilitates NFT-to-token swaps (and vice versa) using customizable bonding curves. Sudoswap supports ERC721 NFTs, as well as all ETH and ERC20 tokens.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of LSSVM2 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 15 days in total, Sudorandom Labs engaged with Spearbit to review the Sudoswap lssvm2 protocol. In this period of time a total of **114** issues were found.

Summary

Project Name	Sudoswap
Repository	lssvm2
Commit	5c1a0c...23d6
Type of Project	P2P NFT Swaps, DeFi
Audit Timeline	Feb 6 - Feb 22
Two week fix period	Feb 22 - March 8

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	9	8	1
Medium Risk	20	12	8
Low Risk	29	8	21
Gas Optimizations	15	6	9
Informational	41	20	21
Total	**114**	**54**	**60**

5 Findings

5.1 High Risk

5.1.1 Partial fills for buy orders in ERC1155 swaps will fail when pair has insufficient balance

Severity: High Risk

Context: VeryFastRouter.sol#L189-L198

Description: Partial fills are currently supported for buy orders in `VeryFastRouter.swap()`. When `_findMaxFillableAmtForBuy()` determines `numItemsToFill`, it is not guaranteed that the underlying pair has so many items left to fill. While ERC721 swap handles the scenario where pair balance is less than `numItemsToFill` in the logic of `_findAvailableIds()` (`maxIdsNeeded` vs `numIdsFound`), ERC1155 swap is missing a similar check and reduction of item numbers when required.

Partial fills for buy orders in ERC1155 swaps will fail when the pair has a balance less than `numItemsToFill` as determined by `_findMaxFillableAmtForBuy()`. Partial filling, a key feature of `VeryFastRouter`, will then not work as expected and would lead to an early revert which defeats the purpose of `swap()`.

Recommendation: Check for `numItemsToFill` against pair balance and use the smaller of the two for partial filling, i.e. calculate `min(numItemsToFill, erc1155.balanceOf(pair))` as the amount of NFTs to transfer.

Sudorandom Labs: This has been fixed after the review started as part of PR#26.

Spearbit: Verified that this is fixed by PR#26.

5.1.2 Function `token()` of `cloneERC1155ERC20Pair()` reads from wrong location

Severity: High Risk

Context: LSSVMPairERC20.sol#L26-L31, LSSVMPairCloner.sol#L359-L436

Description: The function `token()` loads the token data from position 81. However on ERC1155 pairs it should load it from position 93. Currently, it doesn't retrieve the right values and the code won't function correctly.

```
LSSVMPair.sol:         _factory         := shr(0x60, calldataload(sub(calldatasize(), paramsLength)))
LSSVMPair.sol:         _bondingCurve    := shr(0x60, calldataload(add(sub(calldatasize(), paramsLength),
↪   20)))
LSSVMPair.sol:         _nft             := shr(0x60, calldataload(add(sub(calldatasize(), paramsLength),
↪   40)))
LSSVMPair.sol:         _poolType        := shr(0xf8, calldataload(add(sub(calldatasize(), paramsLength),
↪   60)))
LSSVMPairERC1155.sol:  id               := calldataload(add(sub(calldatasize(), paramsLength), 61))
LSSVMPairERC721.sol:   _propertyChecker := shr(0x60, calldataload(add(sub(calldatasize(), paramsLength),
↪   61)))
LSSVMPairERC20.sol:    _token           := shr(0x60, calldataload(add(sub(calldatasize(), paramsLength),
↪   81)))

function cloneERC1155ERC20Pair(... ) ... {
    assembly {
        ...
        mstore(add(ptr, 0x3e), shl(0x60, factory))       // position 0  - 20 bytes
        mstore(add(ptr, 0x52), shl(0x60, bondingCurve)) // position 20 - 20 bytes
        mstore(add(ptr, 0x66), shl(0x60, nft))           // position 40 - 20 bytes
        mstore8(add(ptr, 0x7a), poolType)                // position 60 - 1 bytes
        mstore(add(ptr, 0x7b), nftId)                    // position 61 - 32 bytes
        mstore(add(ptr, 0x9b), shl(0x60, token))         // position 93 - 20 bytes
        ...
    }
}
```

Recommendation: After the review has started, the function `token()` has been updated to read the last 20 bytes. See PR#21.

Sudorandom Labs: Solved in PR#21.

Spearbit: Verified that this is fixed by PR#21.

5.1.3 Switched order of update leads to incorrect partial fill calculations

Severity: High Risk

Context: VeryFastRouter.sol#L260-L264

Description: In the binary search, the order of updation of `start` and `numItemsToFill` is switched with `start` being updated before `numItemsToFill` which itself uses the value of `start`:

```
start = (start + end)/2 + 1;
numItemsToFill = (start + end)/2;
```

This leads to incorrect partial fill calculations when binary search recurses on the right half.

Recommendation: This was found by the project team after the start of the review and fixed (by switching the order of updations) in PR#27.

Sudorandom Labs: Solved in PR#27.

Spearbit: Verified that this is fixed by PR#27.

5.1.4 Swap functions with sell orders in `LSSVMRouter` will fail for property-check enforced pairs

Severity: High Risk

Context: LSSVMRouter.sol, VeryFastRouter.sol#L36-L37, VeryFastRouter.sol#L118-L140

Description: Swap functions with sell orders in `LSSVMRouter` will revert for property-check enforced pairs. While `VeryFastRouter` swap function supports sell orders to specify property check parameters for pairs enforcing them, none of the swap functions in `LSSVMRouter` support the same.

Recommendation: Deprecate `LSSVMRouter` or add support to it for feature-parity with `VeryFastRouter`.

Sudorandom Labs: The `VeryFastRouter` is intended to be a router rewrite that addresses many of the concerns with the `LSSVMRouter`. The `LSSVMRouter` has been modified very little from the audit in v1, with most of the new features (e.g. property check support, multiple token output support, buying with ETH and ERC20, and partial fill) being all moved to the new router. Acknowledged, as mentioned above, property checking is supported in the new `VeryFastRouter`. `LSSVMRouter` will be deprecated once full test coverage for VeryFastRouter is achieved.

Spearbit: Acknowledged.

5.1.5 `pairTransferERC20From()` only supports ERC721 NFTs

Severity: High Risk

Context: LSSVMRouter.sol#L491-L543, VeryFastRouter.sol#L344-L407

Description: Function `pairTransferERC20From()` which is is present in both `LSSVMRouter` and `VeryFastRouter`, only checks for `ERC721_ERC20`. This means ERC1155 NFTs are not supported by the routers.

The following code is present in both `LSSVMRouter` and `VeryFastRouter`.

```
function pairTransferERC20From(...) ... {
    require(factory.isPair(msg.sender, variant), "Not pair");
    ...
    require(variant == ILSSVMPairFactoryLike.PairVariant.ERC721_ERC20, "Not ERC20 pair");
    ...
}
```

Recommendation: After the start of this review, the function `pairTransferERC20From()` has already been updated in PR#21.

Also see issue: *"Use of `isPair()` is not intuitive"*

Sudorandom Labs: Solved in PR#21 and PR#30.

Spearbit: Verified that this is fixed by PR#21 and PR#30.

5.1.6 Insufficient application of trading fee leads to 50% loss for LPs in `swapTokenForAnyNFTs()`

Severity: High Risk

Context: LSSVMPairERC1155.sol#L114, LSSVMPairERC1155.sol#L61-L63, LSSVMPairERC721.sol#L51-L59

Description: The protocol applies a trading fee of `2*tradeFee` on NFT buys from pairs (to compensate for 0 fees on NFT sells as noted in the comment:

> "*// We pull twice the trade fee on buys but don't take trade fee on sells if assetRecipient is set*").

While this is enforced in `LSSVMPairERC721.swapTokenForSpecificNFTs()` and `LSSVMPairERC1155.swapTokenForSpecificNFTs()`, `LSSVMPairERC1155.swapTokenForAnyNFTs()` enforces a trading fee of only `tradeFee` (instead of `2*tradeFee`).

Affected LPs of pairs targeted by `LSSVMPairERC1155. swapTokenForAnyNFTs()` will unexpectedly lose 50% of the trade fees.

Recommendation: This entire function `LSSVMPairERC1155. swapTokenForAnyNFTs()` has been removed in a recent PR#22 (after the start of this review). So there is nothing to fix related to this issue.

Sudorandom Labs: Acknowledged, as function has been removed, this should no longer be a concern.

Spearbit: Verified that this is fixed by PR#22.

5.1.7 Royalty not always being taken into account leads to incorrect protocol accounting

Severity: High Risk

Context: LSSVMPair.sol#L225-L474, LSSVMRouter.sol#L281-L315, LSSVMPairERC1155.sol#L136-L184, StandardSettings.sol#L227-L294, VeryFastRouter.sol#L78-L95

Description: The function `getSellNFTQuoteWithRoyalties()` is similar to `getSellNFTQuote()`, except that it also takes the royalties into account. When the function `robustSwapNFTsForToken()` of the `LSSVMRouter` is called, it first calls `getSellNFTQuote()` and checks that a sufficient amount of tokens will be received. Then it calls `swapNFTsForToken()` with 0 as `minExpectedTokenOutput`; so it will accept any amount of tokens. The `swapNFTsForToken()` does subtract the Royalties which will result in a lower amount of tokens received and might not be enough to satisfy the requirements of the seller.

The same happens in

- `robustSwapETHForSpecificNFTsAndNFTsToToken` and
- `robustSwapERC20ForSpecificNFTsAndNFTsToToken`.

Note: Function `getSellNFTQuote()` of `StandardSettings.sol` also uses `getSellNFTQuote()`. However there it is compared to the results of `getBuyInfo()`; so this is ok as both don't take the royalties into account.

Note: `getNFTQuoteForSellOrderWithPartialFill()` also has to take royalties into account.

```
function getSellNFTQuote(uint256 numNFTs) ... { (
    (...., outputAmount, ) = bondingCurve().getSellInfo(...);
}
function getSellNFTQuoteWithRoyalties(uint256 assetId, uint256 numNFTs) ... {
    (..., outputAmount, ... ) = bondingCurve().getSellInfo(...);
    (,, uint256 royaltyTotal) = _calculateRoyaltiesView(assetId, outputAmount);
    ...
    outputAmount -= royaltyTotal;
}
function robustSwapNFTsForToken(...) ... {
    ...
    (error,,, pairOutput,) =
↪   swapList[i].swapInfo.pair.getSellNFTQuote(swapList[i].swapInfo.nftIds.length);
    ...
    if (pairOutput >= swapList[i].minOutput) {
        ....swapNFTsForToken(..., 0, ...);
    }
    ...
 }
function swapNFTsForToken(...) ... {
    ...
    (protocolFee, outputAmount) = _calculateSellInfoAndUpdatePoolParams(numNFTs[0], _bondingCurve,
↪   _factory);
    (... royaltyTotal) = _calculateRoyalties(nftId(), outputAmount);
    ...
    outputAmount -= royaltyTotal;
    ...
    _sendTokenOutput(tokenRecipient, outputAmount);
}
```

Recommendation: Preferably combine the functions `getSellNFTQuote()` and `getSellNFTQuoteWithRoyalties()`. Also consider integrating the royalty calculations in `_calculateSellInfoAndUpdatePoolParams()`. Alternatively call `swapNFTsForToken()` with the appropriate `minExpectedTokenOutput`.

Make sure `changeSpotPriceAndDelta()` keeps functioning with the updates.

Double check the comment of LSSVMPair.sol#L62.

Update `getNFTQuoteForSellOrderWithPartialFill()` to take royalties into account.

Sudorandom Labs: Solved in PR#29 and PR#27.

Spearbit: Verified that this is fixed by PR#29 and PR#27.

5.1.8 Error return codes of `getBuyInfo()` and `getSellInfo()` are sometimes ignored

Severity: High Risk

Context: ICurve.sol#L38-L87, LSSVMPair.sol#L206-L266

Description: The functions `getBuyInfo()` and `getSellInfo()` return an error code when they detect an error. The rest of the returned parameters then have an unusable/invalid value (0). However, some callers of these functions ignore the error code and continue processing with the other unusable/invalid values. The functions `getBuyNFTQuote()`, `getSellNFTQuote()` and `getSellNFTQuoteWithRoyalties()` pass through the error code, so their callers have to check the error codes too.

```
function getBuyInfo(...) ... returns (CurveErrorCodes.Error error, ... ) {
}
function getSellInfo(...) ... returns (CurveErrorCodes.Error error, ... ) {
}
function getBuyNFTQuote(...) ... returns (CurveErrorCodes.Error error, ... ) {
    (error, ... ) = bondingCurve().getBuyInfo(...);
}
function getSellNFTQuote(...) ... returns (CurveErrorCodes.Error error, ... ) {
    (error, ... ) = bondingCurve().getSellInfo(...);
}
function getSellNFTQuoteWithRoyalties(...) ... returns (CurveErrorCodes.Error error, ... ) {
    (error, ... ) = bondingCurve().getSellInfo(...);
}
```

Recommendation: Always check the return code of the functions `getBuyInfo()`, `getSellInfo()`, `getBuyNFTQuote()`, `getSellNFTQuote()` and `getSellNFTQuoteWithRoyalties()`.

Sudorandom Labs: Solved in PR#94.

Spearbit: Verified that this is fixed by PR#94.

5.1.9 `changeSpotPriceAndDelta()` only uses ERC721 version of `balanceOf()`

Severity: High Risk

Context: StandardSettings.sol#L227-L294

Description: The function `changeSpotPriceAndDelta()` uses `balanceOf()` with one parameter. This is the ERC721 variant. In order to support ERC1155, a second parameter of the NFT `id` has to be supplied.

```
function changeSpotPriceAndDelta(address pairAddress, ...) public {
    ...
    if ((newPriceToBuyFromPair < priceToBuyFromPair) && pair.nft().balanceOf(pairAddress) >= 1) {
        ...
    }
}
```

Recommendation: Detect the use of ERC1155 and use the appropriate `balanceOf()` version.

Sudorandom Labs: Solved in PR#30.

Spearbit: Verified that this is fixed by PR#30.

5.2 Medium Risk

5.2.1 `_pullTokenInputAndPayProtocolFee()` doesn't check that tokens are received

Severity: Medium Risk

Context: LSSVMPairERC20.sol#L34-L115

Description: The function `_pullTokenInputAndPayProtocolFee()` doesn't verify that it actually received the tokens after doing `safeTransferFrom()`. This can be an issue with fee on transfer tokens. This is also an issue with (accidentally) non-existing tokens, as `safeTransferFrom()` won't revert on that, see POC below.

Note: also see issue *"Malicious router mitigation may break for deflationary tokens"*.

```
function _pullTokenInputAndPayProtocolFee(...) ... {
    ...
    _token.safeTransferFrom(msg.sender, _assetRecipient, saleAmount);
   ...
}
```

Proof Of Concept:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.18;
import "hardhat/console.sol";
import {ERC20} from
↪   "https://raw.githubusercontent.com/transmissions11/solmate/main/src/tokens/ERC20.sol";
import {SafeTransferLib} from
↪   "https://raw.githubusercontent.com/transmissions11/solmate/main/src/utils/SafeTransferLib.sol";
contract test {
    using SafeTransferLib for ERC20;

function t() public {
    ERC20 _token = ERC20(address(1));
    _token.safeTransferFrom(msg.sender, address(0), 100);
    console.log("after safeTransferFrom");
}

}
```

Recommendation: Check balance before and after `safeTransferFrom()`. Consider checking the tokens exist in `LSSVMPairFactory`.

Sudorandom Labs: Acknowledged, pair owners should verify the token addresses for tokens they want to list. Separately, fee on transfer tokens are beyond the current scope of the protocol, so undefined behavior is an acceptable risk.

Spearbit: Acknowledged.

5.2.2 A malicious `settings` contract can call `onOwnershipTransferred()` to take over pair

Severity: Medium Risk

Context: StandardSettings.sol#L118-L158

Description: The function `onOwnershipTransferred()` can be called from a pair via `call()`. This can be done either before `transferOwnership()` or after it. If it is called before then it updates the `AssetRecipient`. It can only be called after the `transferOwnership()` when an alternative (malicious) settings contract is used. In that situation `pairInfos[]` is overwritten and the original owner is lost; so effectively the pair can be taken over.

Note: if the settings contract is malicious then there are different ways to take over the pair, but using this approach the vulnerabilities can be hidden.

```
function onOwnershipTransferred(address prevOwner, bytes memory) public payable {
    ILSSVMPair pair = ILSSVMPair(msg.sender);
    require(pair.poolType() == ILSSVMPair.PoolType.TRADE, "Only TRADE pairs");
    ...
}
```

Recommendation:

1. In `onOwnershipTransferred()` check that `address(this)` is the owner of the pair.
2. In `onOwnershipTransferred()` check that `pairInfos[]` hasn't been used before.
3. As an extra protection, in function `call()` of `LSSVMPair`, disallow calling `onOwnershipTransferred()`

Note: also see related issue "Function `call()` is risky and can be restricted further".

Sudorandom Labs: Solved in PR#34.

Spearbit: Verified that PR#34 implements Recommendation (3).

5.2.3 One can attempt to steal a pair's ETH

Severity: Medium Risk

Context: StandardSettings.sol#L301-L305, Splitter.sol#L26-L29, LSSVMPairETH.sol#L103-L105

Description: Anyone can pass the enrolled pair's address instead of a splitter address in `bulkWithdrawFees()` to effectively call the pair's `withdrawAllETH()` instead of a splitter's `withdrawAllETH()`.

Anyone can attempt to steal/drain all the ETH from a pair. However, the pair's `withdrawAllETH()` sends ETH to the owner, which in this case is the settings contract. The settings contract is unable to receive ETH as currently implemented. So the attempt reverts.

Recommendation: This was fixed by the project team, after the review started, by changing the Splitter's function signature from `withdrawAllETH()` to `withdrawAllETHInSplitter()` in PR#36.

Additionally:

1. Check that none of the addresses in `splitterAddresses` of `bulkWithdrawFees()` are pair addresses
2. Change the access control for `bulkWithdrawFees()` from permissionless to `onlyOwner` which adds an extra layer of defense
3. Ensure that the settings contract cannot receive ETH to prevent a malicious setting from draining the pair
4. Given *PropertyCheckers and Settings not sufficiently restricted*, any arbitrary settings contract may accidentally be allowed which makes PR#36 ineffective, we need to ensure that a new settings contract should be audited to make sure it doesn't call any of the withdraw functions

Sudorandom Labs: Addressed as the function signature for `withdrawAllETH` has now been changed.

Spearbit: Verified that the Splitter's function signature is changed from `withdrawAllETH()` to `withdrawAllETHInSplitter()` in PR#36. `StandardSettings` having no `receive` function addresses Recommendation (3).

5.2.4 `swap()` could mix tokens with ETH

Severity: Medium Risk

Context: VeryFastRouter.sol#L102-L212

Description: The function `swap()` adds the output of `swapNFTsForToken()` to the `ethAmount`. Although this only happens when `order.isETHSell == true` , this value could be set to the wrong value accidentally or on purpose. Then the number of received ERC20 tokens could be added to the `ethAmount`, which is clearly unwanted. The resulting `ethAmount` is returned to the user. Luckily the router (normally) doesn't have extra ETH so the impact should be limited.

```
function swap(Order calldata swapOrder) external payable {
    uint256 ethAmount = msg.value;
    if (order.isETHSell && swapOrder.recycleETH) {
        ...
        outputAmount = pair.swapNFTsForToken(...);
        ...
        ethAmount += outputAmount;
        ...
    }
    ...
    // Send excess ETH back to token recipient
    if (ethAmount > 0) {
        payable(swapOrder.tokenRecipient).safeTransferETH(ethAmount);
    }
}
```

Recommendation: Verify the parameters supplied to `swap()` are compatible with the type of pair.

Sudorandom Labs: As the router is not intended to hold ETH, this is an acceptable risk.

Spearbit: Acknowledged.

5.2.5 Using a single `tokenRecipient` in `VeryFastRouter` could result in locked NFTs

Severity: Medium Risk

Context: VeryFastRouter.sol#L45, VeryFastRouter.sol#L134-L139, VeryFastRouter.sol#L158-L210, LSSVMRouter.sol#L42-L43

Description: `VeryFastRouter` uses a single `tokenRecipient` address for both ETH/tokens and NFTs, unlike `LSSVMRouter` which uses a separate `tokenRecipient` and `nftRecipient`.

It is error-prone to have a single `tokenRecipient` receive both tokens and NFTs, especially when the other/existing `LSSVMRouter` has a separate `nftRecipient`. `VeryFastRouter.swap()` sends both sell order tokens to `tokenRecipient` and buy order NFTs to `tokenRecipient`. Front-ends integrating with both routers (or migrating to the new one) may surprise users by sending both tokens+NFTs to the same address when interacting with this router. This coupled with the use of `nft.transferFrom()` may result in NFTs being sent to contracts that are not ERC-721 receivers and get them locked forever.

Recommendation: Consider a separate `nftRecipient` in orders similar to `LSSVMRouter`.

Sudorandom Labs: Solved in PR#57.

Spearbit: Verified that this is fixed by PR#57.

5.2.6 Owner can mislead users by abusing `changeSpotPrice()` and `changeDelta()`

Severity: Medium Risk

Context: LSSVMPair.sol#L584-L604

Description: A malicious owner could set up a pair which promises to buy NFTs for high prices. As soon as someone tries to trade, the owner could frontrun the transaction by setting the spotprice to `0` and gets the NFT for free. Both `changeSpotPrice()` and `changeDelta()` can be used to immediately change trade parameters where the aftereffects depends on the curve being used.

Note: The `swapNFTsForToken()` parameter `minExpectedTokenOutput` and `swapTokenForSpecificNFTs()` parameter `maxExpectedTokenInput` protect users against sudden price changes. But users might not always set them in an optimal way.

A design goal of the project team is that the pool owner can quickly respond to changing market conditions, to prevent unnecessary losses.

```
function changeSpotPrice(uint128 newSpotPrice) external onlyOwner {
    ...
}
function changeDelta(uint128 newDelta) external onlyOwner {
    ...
}
```

Recommendation: Consider introducing a small delay of 1 or 2 blocks to prevent frontrunning. It could be done, for example, with 2 functions : `announce(newSpotPrice)` which registers timestamp + new price and emits an event (for transparency) followed by the existing `changeSpotPrice()` which checks if current timestamp > n blocks after the previously announced timestamp.

Sudorandom Labs: Acknowledged, callers should use `maxInput` and `minOutput` to protect themselves.

Spearbit: Acknowledged.

5.2.7 Pair may receive less ETH trade fees than expected under certain conditions

Severity: Medium Risk

Context: LSSVMPairETH.sol#L48-L55

Description: Depending on the values of protocol fee and royalties, if `_feeRecipient == _assetRecipient`, the pair will receive less trade fees than expected.

Assume a scenario where `inputAmount == 100`, `protocolFee == 30`, `royaltyTotal == 60` and `tradeFeeAmount == 20`. This will result in a revert because of underflow in `saleAmount -= tradeFeeAmount;` when `_feeRecipient != _assetRecipient`. However, when `_feeRecipient == _assetRecipient`, the pair will receive trade fees of `100 - 30 - 60 = 10`, whereas it normally would have expected 20.

Recommendation: One option is to only skip the transfer of trade fees when `_feeRecipient == _assetRecipient` but allow the subtraction to revert on any underflows.

Sudorandom Labs: Solved in PR#59.

Spearbit: Verified that this is fixed by PR#59.

5.2.8 Swapping tokens/ETH for NFTs may exhibit unexpected behavior for certain values of input amount, trade fees and royalties

Severity: Medium Risk

Context: LSSVMPairERC20.sol#L34-L115, LSSVMPairETH.sol#L22-L73

Description: The `_pullTokenInputAndPayProtocolFee()` function pulls ERC20/ETH from caller/router and pays protocol fees, trade fees and royalties proportionately. Trade fees have a threshold of `MAX_FEE == 50%`, which allows `2*fee` to be 100%. Royalty amounts could technically be any percentage as well. This allows edge cases where the protocol fee, trade fee and royalty amounts add up to be > `inputAmount`.

In `LSSVMPairERC20`, the ordering of subtracting/transferring the `protocolFee` and `royaltyTotal` first causes the final attempted transfer of `tradeFeeAmount` to either revert because of unavailable funds or uses any balance funds from the pair itself. In `LSSVMPairETH`, the ordering of subtracting/transferring the `tradeFees` and `royaltyTotal` first causes the final attempted transfer of `protocolFee` to either revert because of unavailable funds or uses any balance funds from the pair itself.

Recommendation: Check that `protocolFee + royaltyTotal + tradeFeeAmount < inputAmount`. Consider making the order of operations/transfers the same between `LSSVMPairERC20` and `LSSVMPairETH` to make their behavior/failure modes consistent.

Sudorandom Labs: Acknowledged, no change beyond PR#59 to address the specific trade fee issue. The cases here deal with situations when the royalty percentage is very high (e.g. 50% or more), which we are fine acknowledging but leaving out of scope.

Spearbit: Acknowledged.

5.2.9 NFTs may be exchanged for `0` tokens when price decreases too much

Severity: Medium Risk

Context: LinearCurve.sol#L98-L161

Description: The sale of multiple NFTs, in combination with linear curves, results in a price decrease. When the resulting price is below `0`, then `getSellInfo()` calculates how many NFTs are required to reach a price of `0`. However, the complete number of NFTs is transferred from the originator of the transaction, even while the last few NFTs are worth `0`. This might be undesirable for the originator.

```
function getSellInfo(..., uint256 numItems, ... ) ... {
    ...
    uint256 totalPriceDecrease = delta * numItems;
    if (spotPrice < totalPriceDecrease) {
        ...
        uint256 numItemsTillZeroPrice = spotPrice / delta + 1;
        numItems = numItemsTillZeroPrice;
    }
}
```

Recommendation: Consider letting function `getSellInfo()` return the number of NFTs that are rational to sell. Then transfer only that many NFTs from the originator of the transaction.

Sudorandom Labs: Acknowledged, users intending to sell should use the `minExpectedTokenOutput` to lower bound the amount of tokens they receive. The routing / pricing calculation is intended to be done by callers beforehand (either on-chain or through a client). No change to the `ICurve` interface at this time.

Spearbit: Acknowledged.

5.2.10 `balanceOf()` can be circumvented via reentrancy and two pairs

Severity: Medium Risk

Context: LSSVMPairERC1155.sol#L222-L246

Description: A reentrancy issue can occur if two pairs with the same ERC1155 NFTid are deployed. Via a call to swap NFTs, the ERC1155 callback `onERC1155BatchReceived()` is called. This callback can start a second NFT swap via a second pair. As the second pair has its own reentrancy modifier, this is allowed.

This way the `balanceOf()` check of `_takeNFTsFromSender()` can be circumvented. If a reentrant call, to a second pair, supplies a sufficient amount of NFTs then the `balanceOf()` check of the original call can be satisfied at the same time.

We haven't found a realistic scenario to abuse this with the current routers.

Permissionless routers will certainly increase the risk as they can abuse `isRouter == true`. If the router is malicious then it also has other ways to steal the NFTs; however with the reentrancy scenario it might be less obvious this is happening.

Note: ERC777 tokens also contain such a callback and have the same interface as ERC20 so they could be used in an ERC20 pair.

```
function _takeNFTsFromSender(IERC1155 _nft, uint256 numNFTs, bool isRouter, address routerCaller) ... {
    ...
    if (isRouter) {
        ...
        uint256 beforeBalance = _nft.balanceOf(_assetRecipient, _nftId);
        ...
        router.pairTransferERC1155From(...); // reentrancy with other pair
        require((_nft.balanceOf(_assetRecipient, _nftId) - beforeBalance) == numNFTs, ...); // circumvented
    } else {
    ...
    }
}
```

Recommendation:

1. Thoroughly verify routers before whitelisting them.
2. To protect against reentrancy issues involving multiple pairs, consider putting the reentrancy storage variable on a common location, for example in the `LSSVMPairFactory`.

Sudorandom Labs: Solved in PR#83 and PR#93.

Spearbit: Verified that this is fixed by PR#83 and PR#93.

5.2.11 Function `call()` is risky and can be restricted further

Severity: Medium Risk

Context: LSSVMPair.sol#L640-L645

Description: The function `call()` is powerful and thus risky. To reduce the risk it can be restricted further by disallowing potentially dangerous function selectors. This is also a step closer to introducing permissionless routers.

```
function call(address payable target, bytes calldata data) external onlyOwner {
    ILSSVMPairFactoryLike _factory = factory();
    require(_factory.callAllowed(target), "Target must be whitelisted");
    (bool result,) = target.call{value: 0}(data);
    require(result, "Call failed");
 }
```

Recommendation: Filter out unwanted function selectors, for example for the following functions: `pairTransferERC20From()`, `pairTransferNFTFrom()`, `pairTransferERC1155From()`, `onOwnershipTransferred()`

Sudorandom Labs: Solved in PR#44.

Spearbit: Verified that this is fixed by PR#44.

5.2.12 Incorrect `newSpotPrice` and `newDelta` may be obtained due to unsafe downcasts

Severity: Medium Risk

Context: XykCurve.sol#L83 and XykCurve.sol#L130

Description: When calculating `newSpotPrice` in `getBuyInfo()`, an unsafe downcast from `uint256` into `uint128` may occur and silently overflow, leading to much less value for `newSpotPrice` than expected.

```
function getBuyInfo(
    uint128 spotPrice, uint128 delta, uint256 numItems,
    uint256 feeMultiplier, uint256 protocolFeeMultiplier
) external pure override returns (
    Error error, uint128 newSpotPrice, uint128 newDelta,
    uint256 inputValue, uint256 tradeFee, uint256 protocolFee
)
{
...
    // get the pair's virtual nft and token reserves
    uint256 tokenBalance = spotPrice;
    uint256 nftBalance = delta;
    ...
    // calculate the amount to send in
    uint256 inputValueWithoutFee = (numItems * tokenBalance) / (nftBalance - numItems);
    ...
    // set the new virtual reserves
    newSpotPrice = uint128(spotPrice + inputValueWithoutFee); // token reserve
    ...
}
```

Same happens when calculating `newDelta` in `getSellInfo()`:

```
function getSellInfo(
    uint128 spotPrice, uint128 delta, uint256 numItems,
    uint256 feeMultiplier, uint256 protocolFeeMultiplier
) external pure override returns (
    Error error, uint128 newSpotPrice, uint128 newDelta,
    uint256 outputValue, uint256 tradeFee, uint256 protocolFee
)
{
    ...
    // get the pair's virtual nft and eth/erc20 balance
    uint256 tokenBalance = spotPrice;
    uint256 nftBalance = delta;
    ...
    // set the new virtual reserves
    newDelta = uint128(nftBalance + numItems); // nft reserve
    ...
```

PoC

Proof of concept about how this wouldn't revert but silently overflow:

```
import "hardhat/console.sol";
contract test{
    constructor() {
        uint256 a = type(uint128).max;
        uint256 b = 2;
        uint128 c = uint128(a + b);
        console.log(c); // c == 1, no error
    }
}
```

Recommendation: Check if the value would overflow before casting as is already done in other places. This can also be done with libraries such as OpenZeppelin SafeCast

```
// set the new virtual reserves
+ uint256 _newDelta = nftBalance + numItems
+ if (_newDelta > type(uint128).max) {
+     return (Error.SPOT_PRICE_OVERFLOW, 0, 0, 0, 0, 0);
+ }
-   newDelta = uint128(nftBalance + numItems); // nft reserve
+   newDelta = uint128(_newDelta ); // nft reserve
```

Sudorandom Labs: Solved in PR#42.

Spearbit: Verified that this is fixed by PR#42.

5.2.13 Fewer checks in `pairTransferNFTFrom()` and `pairTransferERC1155From()` than in `pairTransferERC20From()`

Severity: Medium Risk

Context: LSSVMRouter.sol#L491-L543, VeryFastRouter.sol#L344-L407

Description: The functions `pairTransferNFTFrom()` and `pairTransferERC1155From()` don't verify that the correct type of pair is used, whereas `pairTransferERC20From()` does. This means actions could be attempted on the wrong type of pairs. These could succeed for example if a NFT is used that supports both ERC721 and ERC1155.

Note: also see issue *"`pairTransferERC20From` only supports ERC721 NFTs"*

The following code is present in both `LSSVMRouter` and `VeryFastRouter`.

```
function pairTransferERC20From(...) ... {
    require(factory.isPair(msg.sender, variant), "Not pair");
    ...
    require(variant == ILSSVMPairFactoryLike.PairVariant.ERC721_ERC20, "Not ERC20 pair");
    ...
}
function pairTransferNFTFrom(...) ... {
    require(factory.isPair(msg.sender, variant), "Not pair");
    ...
}
function pairTransferERC1155From(...) ... {
    require(factory.isPair(msg.sender, variant), "Not pair");
    ...
}
```

Recommendation: Add comparable checks as in `pairTransferERC20From()` to the functions `pairTransferNFTFrom()` and `pairTransferERC1155From()`.

Sudorandom Labs: Solved in PR#30.

Spearbit: Verified that this is fixed by PR#30.

5.2.14 A malicious collection admin can reclaim a pair at any time to deny enhanced setting royalties

Severity: Medium Risk

Context: StandardSettings.sol#L164-L178

Description: A collection admin can forcibly/selectively call `reclaimPair()` prematurely (before the advertised and agreed upon lockup period) to unilaterally break the settings contract at any time. This will effectively lead to a DoS to the pair owner for the enhanced royalty terms of the setting despite paying the upfront fee and agreeing to a fee split in return. This is because the `unlockTime` is enforced only on the previous pair owner and not on collection admins.

A malicious collection admin can advertise very attractive setting royalty terms to entice pair owners to pay a high upfront fee to sign-up for their settings contract but then force-end the contract prematurely. This will lead to the pair owner losing the paid upfront fee and the promised attractive royalty terms. A lax pair owner who may not be actively monitoring `SettingsRemovedForPair` events before the lockup period will be surprised at the prematurely forced settings contract termination by the collection admin, loss of their earlier paid upfront fee and any payments of default royalty instead of their expected enhanced amounts.

Recommendation: Enforce `unlockTime` on collection admins authorized by `authAllowedForToken`.

Sudorandom Labs: Addressed in PR#85.

Spearbit: Verified that this is fixed by PR#85.

5.2.15 PropertyCheckers and Settings not sufficiently restricted

Severity: Medium Risk

Context: LSSVMPairFactory.sol#L120-L201, LSSVMPairFactory.sol#L430-L433, LSSVMPairFactory.sol#L485-L492, StandardSettingsFactory.sol, PropertyCheckerFactory.sol

Description: The `LSSVMPairFactory` accepts any address for external contracts which contain critical logic but there are no sanity checks done on them. These are the `_bondingCurve`, `_propertyChecker` and `settings` contracts. The contracts could perhaps be updated later via a proxy pattern or a create2/selfdestruct pattern which means that it's difficult to completely rely on them. Both `_propertyChecker` and `settings` contracts have a factory associated: `PropertyCheckerFactory` and `StandardSettingsFactory`. It is straightforward to enforce that only contracts created by the factory can be used. For the `bondingCurve`s there is a whitelist that limits the risk.

```
function createPairERC721ETH(..., ICurve _bondingCurve, ..., address _propertyChecker, ...) ... {
    ... // no checks on _bondingCurve and _propertyChecker
}
function toggleSettingsForCollection(address settings, address collectionAddress, bool enable) public {
    ... // no checks on settings
}
function setBondingCurveAllowed(ICurve bondingCurve, bool isAllowed) external onlyOwner {
    bondingCurveAllowed[bondingCurve] = isAllowed;
    emit BondingCurveStatusUpdate(bondingCurve, isAllowed);
}
```

Recommendation: Enforce that only contracts created by the factories `PropertyCheckerFactory` and `StandardSettingsFactory` can be used. This can be done by keeping a mapping in these contracts which stores all the generated contracts and can then be queried by the factories to verify their origin.

Note: this requires that the `LSSVMPairFactory` is aware of the address of the factories.

Sudorandom Labs: Acknowledged, this is intended behavior. The factories for property checking and settings are intended to be open-ended for e.g. future types of property checkers or settings. The property checker factory and settings factory included in the audit are designed to be the recommended ones at start, but not the only choices available (clients may decide filter pairs to show only pairs with values created from the factories).

Spearbit: Acknowledged.

5.2.16 A malicious router can skip transfer of royalties and protocol fee

Severity: Medium Risk

Context: LSSVMPairERC20.sol#L59-L91

Description: A malicious router, if accidentally/intentionally whitelisted by the protocol, may implement `pairTransferERC20From()` functions which do not actually transfer the number of tokens as expected. This is within the protocol's threat model as evidenced by the use of before-after balance checks on the `_assetRecipient` for `saleAmount`. However, similar before-after balance checks are missing for transfers of royalties and protocol fee payments.

Royalty recipients do not receive their royalties from the malicious router if the protocol/factory intentionally/accidentally whitelists one. The protocol/factory may also accidentally whitelist a malicious router that does not transfer even the protocol fee.

Recommendation: Add before-after balance checks for royalty and protocol fee transfers.

Sudorandom Labs: Talked internally, we're going to hold-off on this one for now: The factory owner has no incentive to not add routers which don't pay the fee, and if they do (e.g. by accident), they can always disable / add a new one. The gas trade-off here is one we're potentially willing to make.

Spearbit: Verified that this is partially fixed in PR#40. Acknowledged the part about `protocolFee`.

5.2.17 Malicious front-end can sneak intermediate ownership changes to perform unauthorized actions

Severity: Medium Risk

Context: LSSVMPair.sol#L653-L667

Description: `LSSVMPair` implements an `onlyOwner multicall` function to allow owner to batch multiple calls. Natspec indicates that this is "*Intended for withdrawing/altering pool pricing in one tx, only callable by owner, cannot change owner.*" The check `require(owner() == msg.sender, "Ownership cannot be changed in multicall");` with a preceding comment "*Prevent multicall from malicious frontend sneaking in ownership change*" indicates the intent of the check and that a malicious front-end is within the threat model.

While the post-loop check prevents malicious front-ends from executing ownership changing calls that attempt to persist beyond the `multicall`, this still allows one to sneak in an intermediate ownership change during a call -> perform malicious actions under the new unauthorized malicious owner within `onOwnershipTransferred()` callback -> change ownership back to originally authorized `msg.sender` owner before returning from the callback and successfully executing any subsequent (`onlyOwner`) calls and the existing check.

While a malicious front-end could introduce many attack vectors that are out-of-scope for detecting/preventing in backend contracts, an unauthorized ownership change seems like a critical one and warrants additional checks for `onlyOwner multicall` to prevent malicious actions from being executed in the context of any newly/temporarily unauthorized owner.

Recommendation: Prevent `transferOwnership()` call in `multicall`. Consider adding sufficient warnings/documentation for this privileged `multicall` usage from front-ends.

Sudorandom Labs: Solved in PR#41.

Spearbit: Verified that this is fixed by PR#41.

5.2.18 Missing override in `authAllowedForToken` prevents authorized admins from toggling settings and reclaiming pairs

Severity: Medium Risk

Context: LSSVMPairFactory.sol#L330-L377, RoyaltyEngine.sol#L38-L46

Description: Manifold admins are incorrectly not allowed by `authAllowedForToken` to toggle settings and reclaim their authorized pairs in the protocol context. `authAllowedForToken` checks for different admin overrides including admin interfaces of NFT marketplaces Nifty, Foundation, Digitalax and ArtBlocks. However, the protocol supports royalties from other marketplaces of Manifold, Rarible, SuperRare and Zora. Of those, Manifold does have `getAdmins()` interface which is not considered in `authAllowedForToken`. And it is not certain that the others don't.

Recommendation: Add admin support for Manifold and other marketplaces (Rarible, SuperRare and Zora), if available, that are recognized by the protocol.

Sudorandom Labs: Acknowledged, no change for now. Adherence to the manifold code is preferred over extending the admin surface for now. The Manifold implementation contract uses AdminControlUpgradeable, which contains an isAdmin function. This function is covered by line LSSVMPairFactory.sol#L338.

Spearbit: Acknowledged.

5.2.19 Misdirected transfers to invalid pair variants or non-pair recipients may lead to loss/lock of NFTs/tokens

Severity: Medium Risk

Context: LSSVMPairFactory.sol#L650-L663, LSSVMPairFactory.sol#L668-L676

Description: Functions `depositNFTs()` and `depositERC20()` allow deposits of ERC 721 NFTs and ERC20 tokens after pair creation. While they check that the deposit recipient is a valid pair/variant for emitting an event, the deposit transfers happen prior to the check and without the same validation. With dual home tokens (see weird-erc20), the `emit` could be skipped when the "other" token is transferred. Also, the `isPair()` check in `depositNFTs()` does not specifically check if the pair variant is `ERC721_ERC20` or `ERC721_ETH`.

This allows accidentally misdirected deposits to invalid pair variants or non-pair recipients leading to loss/lock of NFTs/tokens.

Recommendation: For functions `depositNFTs()` and `depositERC20()` apply the specific pair variant check for both events and transfers and check the right tokens/NFT are deposited.

Sudorandom Labs: We'll acknowledge the finding, but no additional changes at this time. Only event emission is important to be tracked with the pool type, as pool owners can always withdraw any erc20/721/1155 sent to their pool (in the event they e.g. deposit to a pool they own for a different asset type).

Spearbit: Acknowledged.

5.2.20 `authAllowedForToken()` returns prematurely in certain scenarios causing an authentication DoS

Severity: Medium Risk

Context: LSSVMPairFactory.sol#L330-L377

Description: Tokens listed on Nifty or Foundation (therefore returning a valid `niftyRegistry` or `foundationTreasury`) where the `proposedAuthAddress` is *not* a valid Nifty sender or a valid Foundation Treasury admin will cause an authentication DoS if the token were also listed on Digitalax or ArtBlocks and the `proposedAuthAddress` had admin roles on those platforms.

This happens because the return values of `valid` and `isAdmin` for `isValidNiftySender(proposedAuthAddress)` and `isAdmin(proposedAuthAddress)` respectively are returned as-is instead of returning only if/when they are true but continuing the checks for authorization otherwise (if/when they are false) on Digitalax and ArtBlocks.

`toggleSettingsForCollection` and `reclaimPair` (which utilize `authAllowedForToken`) would incorrectly fail for valid `proposedAuthAddress` in such scenarios.

Recommendation: Check the return values of `valid` and `isAdmin` for `isValidNiftySender(proposedAuthAddress)` and `isAdmin(proposedAuthAddress)` and return if they are true (as done for Digitalax). Continue with the authorization checks otherwise, if/when they are false.

Sudorandom Labs: Addressed in PR#64.

Spearbit: Verified that this is fixed by PR#64.

5.3 Low Risk

5.3.1 Partial fills don't recycle ETH

Severity: Low Risk

Context: VeryFastRouter.sol#L211-L425

Description: After several fixes are applied, the following code exists. If the sell can be filled completely then ETH is recycled, however when a partial fill is applied then ETH is not recycled. This might lead to a `revert` and would require doing the trade again. This costs extra gas and the trading conditions might be worse then.

```
function swap(Order calldata swapOrder) external payable returns (uint256[] memory results) {
    ...
    // Go through each sell order
   ...
   if (pairSpotPrice == order.expectedSpotPrice) {
        // If the pair is an ETH pair and we opt into recycling ETH, add the output to our total accrued
        if (order.isETHSell && swapOrder.recycleETH) {
            ...
            ... order.pair.swapNFTsForToken(... , payable(address(this)), ... );
        } // Otherwise, all tokens or ETH received from the sale go to the token recipient
        else {
            ... order.pair.swapNFTsForToken(..., swapOrder.tokenRecipient, ... );
        }
    }  // Otherwise we need to do some partial fill calculations first
    else {
        ...
        ... order.pair.swapNFTsForToken(...,  swapOrder.tokenRecipient, ... ); // ETH not recycled
    }
    // Go through each buy order
    ...
}
```

Recommendation: Consider also recycling ETH in the partial fill case.

Sudorandom Labs: There are a few situations where this could happen:

- The user intends to sell for more than they are trying to buy, and they are sending 0 ETH. They encounter a partial fill, and they still receive enough ETH to cover their buys. The ETH received from selling is recycled, and the buy succeeds.
- The user intends to sell for more than they are trying to buy, and they are sending 0 ETH. They encounter a partial fill, and they do not receive enough ETH to cover their buys. The ETH received from selling is recycled, but the buy still fails because there is not enough ETH.
- The user intends to sell for less than they are trying to buy, and they send only enough ETH to cover a buy assuming the sell succeeds. They encounter a partial fill, and they do not receive enough ETH to cover their buys. The ETH received from selling is recycled, but the buy still fails because there is not enough ETH.

Given that I expect situations 2 and 3 to be common, I don't think it's worth complicating the code more to handle the case of 1. If this becomes a larger issue in the future, there is always the choice of writing a new router to handle it.

But for now, I think I will leave it as-is.

Spearbit: Acknowledged.

5.3.2 Wrong allowances can be abused by the owner

Severity: Low Risk

Context: LSSVMPair.sol#L640-L645

Description: The function `call()` allows transferring tokens and NFTs that have an allowance set to the pair. Normally, allowances should be given to the router, but they could be accidentally given to the pair.

Although `call()` is protected by `onlyOwner`, the pair is created permissionless and so the owner could be anyone.

Recommendation: In function `call()` disallow the targets `nft()` and `erc20()` (for ERC20 pairs).

Sudorandom Labs: This has been solved by the project team after the audit started in PR#34 and PR#52.

Spearbit: Verified that this is fixed by PR#34 and PR#52.

5.3.3 Malicious router mitigation may break for deflationary tokens

Severity: Low Risk

Context: LSSVMPairERC20.sol#L72-L75

Description: ERC20 `_pullTokenInputAndPayProtocolFee()` for routers has a mitigation for malicious routers by checking if the before-after token balance difference is equal to the transferred amount.

This will break for any ERC20 pairs with fee-on-transfer deflationary tokens (see weird-erc20). Note that there has been a real-world exploit related to this with Balancer pool and STA deflationary tokens.

Recommendation: Evaluate the feasibility of disallowing ERC20 deflationary tokens vis-a-vis the likelihood of pairs using deflationary tokens leading to this issue.

Sudorandom Labs: Acknowledged, no change for now. Pair creators specify their own quote token, i.e. no allowlist for quote tokens. So the risk is there and pair creators will need to select their quote tokens appropriately.

Spearbit: Acknowledged.

5.3.4 Inconsistent royalty threshold checks allow some royalties to be equal to sale amount

Severity: Low Risk

Context: RoyaltyEngine.sol#L197-L210, RoyaltyEngine.sol#L225-L231

Description: Threshold checks on royalty amounts are implemented both in `_getRoyaltyAndSpec()` and its caller `_calculateRoyalties()`. While `_calculateRoyalties()` implements an inclusive check with `require(saleAmount >= royaltyTotal, "Royalty exceeds sale price");`, (allowing royalty to be equal to sale amount) the different checks in `_getRoyaltyAndSpec()` on the returned amounts or in the calculations on bps in `_computeAmounts()` exclude the `saleAmount` forcing royalty to be less than the `saleAmount`. However, only Known Origin and SuperRare are missing a similar threshold check in `_getRoyaltyAndSpec()`.

This allows only the Known Origin and SuperRare royalties to be equal to the sale amount as enforced by the check in `_calculateRoyalties()`.

Recommendation: Consider adding checks for Known Origin and SuperRare royalties in `_getRoyaltyAndSpec()` and remove the redundant/inclusive check from `_calculateRoyalties()` which allows their royalties to be equal to sale amount.

Sudorandom Labs: This PR aims to centralize all the input vs royalty amount checking in `LSSVMPair` rather than the RoyaltyEngine. We need to check that Settings don't override the royalty percentage to be too high anyway, which is already check in `_calculateRoyalties`, so the decision is to remove the total amount checking in the `RoyaltyEngine` and instead centralize the check in the Pair itself.

Spearbit: Verified that this is fixed by PR#60 and PR#78.

5.3.5 Numerical difference between `getNFTQuoteForBuyOrderWithPartialFill()` and `_findMaxFillableAmtForBuy()` may lead to precision errors

Severity: Low Risk

Context: VeryFastRouter.sol#L56-L95, VeryFastRouter.sol#L228-L266

Description: There is a slight numerical instability between the partial fill calculation and the first client side calculation (i.e. `getNFTQuoteForSellOrderWithPartialFill()` / `getNFTQuoteForBuyOrderWithPartialFill()`, `_findMaxFillableAmtForBuy()`). This is because `getNFTQuoteForSellOrderWithPartialFill()` first assumes a buy of 1 item, updates `spotPrice/delta`, and then gets the next subsequent quote to buy the next item. Whereas `_findMaxFillableAmtForBuy()` assumes buying multiple items at one time. This can for e.g. `ExponentialCurve.sol` and `XykCurve.sol` lead to minor numerical precision errors.

```
function getNFTQuoteForBuyOrderWithPartialFill(LSSVMPair pair, uint256 numNFTs) external view returns
↪   (uint256[] memory) {
    ...
    for (uint256 i; i < numNFTs; i++) {
        ...
        (, spotPrice, delta, price,,) = pair.bondingCurve().getBuyInfo(spotPrice, delta, 1, fee, ...);
        ...
    }
}
function getNFTQuoteForSellOrderWithPartialFill(LSSVMPair pair, uint256 numNFTs) external view returns
↪   (uint256[] memory) {
    ...
    for (uint256 i; i < numNFTs; i++) {
        ...
        (, spotPrice, delta, price,,) = pair.bondingCurve().getSellInfo(spotPrice, delta, 1, fee, ... );
        ...
    }
    ...
}
function _findMaxFillableAmtForBuy(LSSVMPair pair, uint256 maxNumNFTs, uint256[] memory
↪   maxCostPerNumNFTs, uint256
    ...
    while (start <= end) {
        ...
       (...) = pair.bondingCurve().getBuyInfo(spotPrice, delta, (start + end)/2, feeMultiplier,
↪   protocolFeeMultiplier);
       ...
    }
}
```

Recommendation: This has been solved after the review started in PR#27. `getNFTQuoteForSellOrderWithPartialFill()` now accepts an optional slippage parameter which scales up the buy quotes by that amount. As long as the slippage amount is kept to a minimum amount (e.g. 0.00000001%), this should be acceptable.

Sudorandom Labs: Solved in PR#27.

Spearbit: Verified that this is fixed by PR#27.

5.3.6 Differences with Manifold version of `RoyaltyEngine` may cause unexpected behavior

Severity: Low Risk

Context: Manifold RoyaltyEngineV1.sol#L170-L189, Manifold RoyaltyEngineV1.sol#L91-L109, RoyaltyEngine.sol#L132-L159, RoyaltyEngine.sol#L94-L108

Description: Sudoswap has forked `RoyaltyEngine` from Manifold; however there are some differences. The Manifold version of `_getRoyaltyAndSpec()` also queries `getRecipients()`, while the Sudoswap version doesn't. This means the Sudoswap will not spread the royalties over all recipients.

```
function _getRoyaltyAndSpec(...) // Manifold
    ...
    try IEIP2981(royaltyAddress).royaltyInfo(tokenId, value) returns (address recipient, uint256
↪   amount) {
        ...
        try IRoyaltySplitter(royaltyAddress).getRecipients() returns (Recipient[] memory
↪   splitRecipients) {
            ...
        }
   }
}
function _getRoyaltyAndSpec(...) // Sudoswap
    ...
        try IEIP2981(royaltyAddress).royaltyInfo(tokenId, value) returns (address recipient, uint256
↪   amount) {
            ...
        } ...
    }
}
```

The Manifold version of `getRoyalty()` has an extra `try/catch` compared to the Sudoswap version. This protects against reverts in the cached functions. Note: adding an extra `try/catch` requires the function `_getRoyaltyAndSpec()` to be external.

```
function getRoyalty(address tokenAddress, uint256 tokenId, uint256 value) ... {  // Manifold
    ...
    try this._getRoyaltyAndSpec{gas: 100000}(tokenAddress, tokenId, value) returns ( ... ) ....
}
function getRoyalty(address tokenAddress, uint256 tokenId, uint256 value) ... { // Sudoswap
    ...
    ...
    (...) = _getRoyaltyAndSpec(tokenAddress, tokenId, value);
}
```

Recommendation: Check the latest version of Manifold code to determine if anything needs updating.

Sudorandom Labs: Acknowledged, we don't feel strongly about the added code so we'll leave ours as is.

Spearbit: Acknowledged.

5.3.7 Swaps with property-checked ERC1155 sell orders in `VeryFastRouter` will fail

Severity: Low Risk

Context: VeryFastRouter.sol#L120, VeryFastRouter.sol#L134

Description: Any swap batch of transactions which has a property-checked sell order for ERC1155 will revert. Given that property checks are not supported on ERC1155 pairs (but only ERC721), swap sell orders for ERC1155 in `VeryFastRouter` will fail if `order.doPropertyCheck` is accidentally set because the logic thereafter assumes it is an ERC721 order.

Recommendation: Check if the order pair is ERC721 before checking property or allow users to explicitly specify in an order if it is ERC721 or ERC1155 specific.

Sudorandom Labs: No change for now. Callers using the router should set the appropriate parameters for the asset type of the swap. The downside is bounded as a revert (rather than e.g. callers losing their funds) which is an acceptable risk.

Spearbit: Acknowledged.

5.3.8 `changeSpotPriceAndDelta()` reverts when there is enough liquidity to support 1 sell

Severity: Low Risk

Context: StandardSettings.sol.sol#L287

Description: `changeSpotPriceAndDelta()` reverts when there is enough liquidity to support 1 sell because it uses > instead of >= in the check `pairBalance > newPriceToSellToPair`.

Recommendation: Use >= instead of >:

```
// If the new sell price is higher, and there is enough liquidity to support at least 1 sell, then make
↪  the change
-  if ((newPriceToSellToPair > priceToSellToPair) && pairBalance > newPriceToSellToPair) {
+  if ((newPriceToSellToPair > priceToSellToPair) && pairBalance >= newPriceToSellToPair) {
```

Sudorandom Labs: Solved in PR#56.

Spearbit: Verified that this is fixed by PR#56.

5.3.9 Lack of support for per-token royalties may lead to incorrect royalty payments

Severity: Low Risk

Context: LSSVMPair.sol#L259, LSSVMPairERC721.sol#L52

Description: The protocol currently lacks complete support for per-token royalties, assumes that all NFTs in a pair have the same royalty and so considers the first `assetId` to determine royalties for all NFT token Ids in the pair. If not, the pair owner is expected to make a new pair for NFTs that have different royalties.

A pair with NFTs that have different royalties will lead to incorrect royalty payments for the different NFTs.

Recommendation: Evaluate adding complete support for per-token royalties.

Sudorandom Labs: This is a design decision to balance trade-off between gas and utility. If different NFT IDs have different royalty amounts / receivers (e.g. Artblocks), the intent is for pool owners to separate them into different pools.

Spearbit: Acknowledged.

5.3.10 Missing additional safety for `multicall` may lead to lost ETH in future

Severity: Low Risk

Context: LSSVMPair.sol#L653-L663

Description: If the function `multicall()` would be `payable`, then multiple `delegated-to` functions could use the same `msg.value` , which could result in losing ETH from the pair. A future upgrade of Solidity might change the default setting for function to `payable`. See Solidity issue#12539.

```
function multicall(bytes[] calldata calls, bool revertOnFail) external onlyOwner {
    for (uint256 i; i < calls.length;) {
        (bool success, bytes memory result) = address(this).delegatecall(calls[i]);
        ...
    }
}
```

Recommendation: Consider adding a check for `msg.value` to be future proof and extra safe. Alternatively make a comment about the risk.

```
function multicall(bytes[] calldata calls, bool revertOnFail) external onlyOwner {
    require(msg.value == 0);
    for (uint256 i; i < calls.length;) {
        (bool success, bytes memory result) = address(this).delegatecall(calls[i]);
        ...
    }
}
```

Sudorandom Labs: Acknowledged, no change for now as solidity has not made functions payable by default.

Spearbit: Acknowledged.

5.3.11 Missing zero-address check may allow re-initialization of pairs

Severity: Low Risk

Context: LSSVMPair.sol#L118-L126

Description: `LSSVMPair. initialize()` checks if it is already initialized using `require(owner() == address(0), "Initialized");`. However, without a zero-address check on `_owner`, this can be true even later if the pair is initialized accidentally with `address(0)` instead of `msg.sender`. This is because `__Ownable_init` in `OwnableWithTransferCallback` does not disallow `address(0)` unlike `transferOwnership`. This is however not the case with the current implementation where `LSSVMPair.initialize()` is called from `LSSVMPairFactory` with `msg.sender` as argument for `_owner`.

Therefore, `LSSVMPair.initialize()` may be called multiple times.

Recommendation: Add a zero-address check on `_owner` parameter of `initialize()`.

Sudorandom Labs: Acknowledged, no change as at the moment, we pass in caller to be the owner of pairs.

Spearbit: Acknowledged.

5.3.12 Trade pair owners are allowed to change asset recipient address when it has no impact

Severity: Low Risk

Context: LSSVMPair.sol#L627-L632, LSSVMPair.sol#L310-L328

Description: Trade pair owners are allowed to change their asset recipient address using `changeAssetRecipient()` while `getAssetRecipient()` always returns the pair address itself for Trade pairs as expected.

Trade pair owners mistakenly assume that they can change their asset recipient address using `changeAssetRecipient()` because they are allowed to do so successfully, but may be surprised to see that it has no effect. They may expect assets at the new address but that will not be the case.

Recommendation: `changeAssetRecipient()` could exclude `PoolType.TRADE` owners from being allowed to change the asset recipient.

Sudorandom Labs: It's intended to let this value be changed for `TRADE` pools. To avoid an extra storage slot, `getFeeRecipient` reads from this value for `TRADE` pools.

Spearbit: Acknowledged.

5.3.13 NFT projects with custom settings and multiple royalty receivers will receive royalty only for first receiver

Severity: Low Risk

Context: LSSVMPair.sol#L489-L503, LSSVMPair.sol#L523-L538

Description: `_calculateRoyalties()` and its view equivalent only consider the first royalty receiver when custom settings are enabled.

If non-ERC-2981 compliant NFT projects on Manifold/ArtBlocks or other platforms that support multiple royalty receivers come up with custom settings that pair owners subscribe to, then all the royalty will go to the first recipient. Other receivers will not receive any royalties.

Recommendation: Evaluate splitting of enhanced setting royalties evenly/proportionally across all receivers.

Sudorandom Labs: To give you more context on this logic, most NFT projects only have a single royalty receiver as they follow the ERC2981 standard. However, Manifold and ArtBlocks will sometimes have multiple receivers. In these cases, we choose to use the first receiver for simplicity's sake if the pair has custom settings. It is unlikely that a project creator will both Manifold and also have custom settings.

Acknowledged, no change as this is only relevant for a small subset of collections, and is an acceptable risk.

Spearbit: Acknowledged.

5.3.14 Missing non-zero checks allow event emission spamming

Severity: Low Risk

Context: LSSVMPairFactory.sol#L650-L663, LSSVMPairFactory.sol#L668-L676

Description: Functions `depositNFTs()` and `depositERC20()` are meant to allow deposits into the pair post-creation. However, they do not check if non-zero NFTs or tokens are being deposited. The event emission only checks if the pair recipient is valid. Given their permissionless nature, this allows anyone to grief the system with zero NFT/token deposits causing emission of events which may hinder indexing/monitoring systems.

Recommendation: Add non-zero checks for `numNFTs` and `amount` before event emission.

Sudorandom Labs: Solved in PR#63.

Spearbit: Verified that this is fixed by PR#63.

5.3.15 Missing sanity zero-address checks may lead to undesired behavior or lock of funds

Severity: Low Risk

Context: StandardSettings.sol#L38-L42, StandardSettings.sol#L164-L206, LSSVMPairFactory.sol#L82-L98, VeryFastRouter.sol#L48-L50, StandardSettings.sol#L132, Splitter.sol#L34, VeryFastRouter.sol#L210, LSSVMRouter.sol#L597, LSSVMRouter.sol#L362, LSSVMRouter.sol#L232, LSSVMPairFactory.sol#L393, LSSVMPairETH.sol#L114, LSSVMPairETH.sol#L95, LSSVMPairETH.sol#L63

Description: Certain logic requires zero-address checks to avoid undesired behavior or lock of funds. For example, in Splitter.sol#L34 users can permanently lock ETH by mistakenly using `safeTransferETH` with default/zero-address value.

Recommendation: Check if an address-type variable is `address(0)` and revert when `true` with an appropriate error message. In particular, see:

- StandardSettings.sol#L38-L42: consider adding `require(_settingsFeeRecipient != address(0))`.
- StandardSettings.sol#L164-L206: consider adding `require(pairInfo.prevOwner != address(0))`.
- LSSVMPairFactory.sol#L82-L98: consider adding `require(_protocolFeeRecipient != address(0))`.
- VeryFastRouter.sol#L48-L50: consider adding `require(_factory != address(0))`.
- StandardSettings.sol#L132, Splitter.sol#L34, VeryFastRouter.sol#L210, LSSVMRouter.sol#L597, LSSVMRouter.sol#L362, LSSVMRouter.sol#L232, LSSVMPairFactory.sol#L393, LSSVMPairETH.sol#L114, LSSVMPairETH.sol#L95, LSSVMPairETH.sol#L63: consider adding a zero-address check on the caller of `safeTransferETH`.

Sudorandom Labs: We're going to pass on this since the exploit scope is low impact.

Spearbit: Acknowledged.

5.3.16 Legacy NFTs are not compatible with protocol pairs

Severity: Low Risk

Context: LSSVMPair.sol#L15

Description: Pairs support ERC721 and ERC1155 NFTs. However, users of NFT marketplaces may also expect to find OG NFTs such as Cryptopunks, Etherrocks or Cryptokitties, which do not adhere to these ERC standards.

For example, Cryptopunks have their own internal marketplace which allows users to trade their NFTs with other users. Given that Cryptopunks does not adhere to the ERC721 standard, it will always fail when the protocol attempts to trade them.

Even with wrapped versions of these NFTs, people who aren't aware or have the original version won't be able to trade them in a pair.

Recommendation: Consider adding compatibility as it's a competitive feature. Typical way of supporting them is to add a flow for their addresses as shown here.

Sudorandom Labs: Yes, this is a known incompatibility. There are various wrapped variants (e.g. for mooncats/punks) which have support for ERC721. The design goal here is to adhere more towards the asset standards when possible. I am okay that certain legacy assets may be out of scope.

Spearbit: Acknowledged.

5.3.17 Unnecessary `payable` specifier for functions may allow ETH to be sent and locked/lost

Severity: Low Risk

Context: LSSVMRouter.sol#L410-L415, LSSVMPair.sol#L118-L124

Description: Functions `LSSVMRouter.robustSwapERC20ForSpecificNFTsAndNFTsToToken()` and `LSSVMPair.initialize()` which do not expect to receive and process Ether have the `payable` specifier which allows interacting users to accidentally send them Ether which will get locked/lost.

Recommendation: Remove `payable` specifier on these functions which do not expect to receive and process Ether.

Sudorandom Labs: Solved in PR#66.

Spearbit: Verified that this is fixed by PR#66.

5.3.18 Obsolete `Splitter` contract may lead to locked ETH/tokens

Severity: Low Risk

Context: Splitter.sol#L31-L60, StandardSettings.sol#L89-L91, StandardSettings.sol#L164-L206

Description: After a pair has be reclaimed via `reclaimPair()`, `pairInfos[]` will be emptied and `getPrevFeeRecipientForPair()` will return `0`. The obsolete `Splitter` will however remain present, but any ETH or tokens that are sent to the contract can't be completely retrieved via `withdrawETH()` and `withdrawTokens()`. This is because `getPrevFeeRecipientForPair()` is `0` and the tokens would be send to `address(0)`.

It is unlikely though that ETH or tokens are sent to the `Splitter` contract as it is not used anymore.

```
function withdrawETH(uint256 ethAmount) public {
    ISettings parentSettings = ISettings(getParentSettings());
    ...
    payable(parentSettings.getPrevFeeRecipientForPair(getPairAddressForSplitter())).safeTransferETH(... );
}
function withdrawTokens(ERC20 token, uint256 tokenAmount) public {
    ISettings parentSettings = ISettings(getParentSettings());
    ...
    token.safeTransfer(parentSettings.getPrevFeeRecipientForPair(getPairAddressForSplitter()), ... );
}
function getPrevFeeRecipientForPair(address pairAddress) public view returns (address) {
        return pairInfos[pairAddress].prevFeeRecipient;
}
function reclaimPair(address pairAddress) public {
    ...
    delete pairInfos[pairAddress];
    ...
}
```

Recommendation: Evaluate if it is worth the trouble to do anything with the obsolete `Splitter` contract.

Sudorandom Labs: Acknowledged, no change for now.

Spearbit: Acknowledged.

5.3.19 Divisions in `getBuyInfo()` and `getSellInfo()` may be rounded down to `0`

Severity: Low Risk

Context: XykCurve.sol#L42-L134

Description: In extreme cases (e.g. tokens with a few decimals, see this example), divisions in `getBuyInfo()` and `getSellInfo()` may be rounded down to `0`. This means `inputValueWithoutFee` and/or `outputValueWithoutFee` may be `0`.

```
function getBuyInfo(..., uint256 numItems, ... ) ... {
    ...
    uint256 inputValueWithoutFee = (numItems * tokenBalance) / (nftBalance - numItems);
    ...
}
function getSellInfo(..., uint256 numItems, ... ) ... {
    ...
    uint256 outputValueWithoutFee = (numItems * tokenBalance) / (nftBalance + numItems);
    ...
}
```

Recommendation: The rounding down could be ignored as it involves very low amounts of tokens. Alternatively round up, for example, in the following way:

```
uint256 outputValueWithoutFee = (numItems * tokenBalance + (nftBalance - numItems - 1) ) / (nftBalance
↪   - numItems);
```

Sudorandom Labs: This is an acceptable risk when decimals and prices are both very low.

Spearbit: Acknowledged.

5.3.20 Last NFT in an `XykCurve` cannot be sold

Severity: Low Risk

Context: XykCurve.sol#L42-L88, StandardSettings.sol#L227-L294, LSSVMPair.sol#L206-L219

Description: The function `getBuyInfo()` of `XykCurve` enforces `numItems < nftBalance`, which means the last NFT can never be sold.

One potential solution as suggested by the Sudoswap team is to set `delta` (`=nftBalance`) one higher than the real amount of NFTs. This could cause problems in other parts of the code.

For example, once only one NFT is left, if we try to use `changeSpotPriceAndDelta()`, `getBuyNFTQuote(1)` will error and thus the prices (`tokenBalance`) and delta (`nftBalance`) can't be changed anymore.

If `nftBalance` is set to one higher, then it won't satisfy `pair.nft().balanceOf(pairAddress) >= 1`.

```
contract XykCurve ... {
    function getBuyInfo(..., uint256 numItems, ... ) ... {
        ...
        uint256 tokenBalance = spotPrice;
        uint256 nftBalance = delta;
        ...
        // If numItems is too large, we will get divide by zero error
        if (numItems >= nftBalance) {
            return (Error.INVALID_NUMITEMS, 0, 0, 0, 0, 0);
        }
        ...
    }
}

function changeSpotPriceAndDelta(...) ... {
     ...
    (,,, uint256 priceToBuyFromPair,) = pair.getBuyNFTQuote(1);
    ...
    if (... && pair.nft().balanceOf(pairAddress) >= 1) {
            pair.changeSpotPrice(newSpotPrice);
            pair.changeDelta(newDelta);
            return;
        }
    ...
}
function getBuyNFTQuote(uint256 numNFTs) ... {
     (error, ...) = bondingCurve().getBuyInfo(..., numNFTs, ...);
}
```

Recommendation: Evaluate what to do with the last NFT in a `XykCurve`. Perhaps it is unsolvable due to the nature of Xyk curves.

Sudorandom Labs: Acknowledged, the last item is an open question for now. As long as pair creators understand the pricing dynamic, this is an acceptable risk.

Spearbit: Acknowledged.

5.3.21 Allowing different ERC20 tokens in `LSSVMRouter` swaps will affect accounting and lead to undefined behavior

Severity: Low Risk

Context: LSSVMRouter.sol#L109-L135

Description: As commented "*Note: All ERC20 swaps assume that a single ERC20 token is used for all the pairs involved. * Swapping using multiple tokens in the same transaction is possible, but the slippage checks * & the return values will be meaningless and may lead to undefined behavior.*"

This assumption may be risky if users end up mistakenly using different ERC20 tokens in different swaps. Summing up their `inputAmount` and `remainingValue` will not be meaningful and lead to accounting errors and undefined behavior (as noted).

Recommendation: Consider adding checks to enforce (instead of assuming/warning) that ERC20 tokens used in all the swap list pairs are the same.

Sudorandom Labs: Acknowledged, `VeryFastRouter` is intended to be the full-featured router for interacting with v2 of the Pairs that has support for separate `minOutput` values for different tokens.

Spearbit: Acknowledged.

5.3.22 Missing array length equality checks may lead to incorrect or undefined behavior

Severity: Low Risk

Context: LSSVMPairERC721.sol#L292-L298, LSSVMPairERC1155.sol#L273-L301, StandardSettings.sol#L301-L313, RoyaltyEngine.sol#L78-L89, MerklePropertyChecker.sol#L18-L28

Description: Functions taking two array type parameters and not checking that their lengths are equal may lead to incorrect or undefined behavior when accidentally passing arrays of unequal lengths.

Recommendation: Check if the lengths of the two array parameters are equal before use.

Sudorandom Labs: Acknowledged, no change for the following: the `LSSVMPair` functions are owner operated, so if any mismatches arise, the owner can just call it again. For the `RoyaltyEngine`, the `bulkCache` call is also user initiated, so it can also be retried.

Spearbit: Acknowledged.

5.3.23 Owners may have funds locked if `newOwner` is EOA in `transferOwnership()`

Severity: Low Risk

Context: OwnableWithTransferCallback.sol#L45

Description: In `transferOwnership()`, if `newOwner` has zero `code.length` (i.e. EOA), `newOwner.isContract()` will be `false` and therefore, `if` block will be ignored. As the function is `payable`, any `msg.value` from the call would get locked in the contract.

Note: ERC20 pairs and `StandardSettings` don't have a method to recover ETH.

Recommendation: Create an `else` block where `msg.value` is checked for not being `0`. If it is not `0`, transfer `msg.value` back to the `msg.sender` or revert.

Sudorandom Labs: Acknowledged, no change for now. Certain owners may wish to both transfer ownership and send funds to the new recipient. Any issues with the recipient e.g. being unable to accept ETH are an acceptable risk.

Spearbit: Acknowledged.

5.3.24 Use of `transferFrom` may lead to NFTs getting locked forever

Severity: Low Risk

Context: LSSVMPairERC721.sol#L199, LSSVMPairERC721.sol#L253

Description: ERC721 NFTs may get locked forever if the recipient is not aware of ERC721 for some reason. While `safeTransferFrom()` is used for ERC1155 NFTs (which has the `_doSafeTransferAcceptanceCheck` check on recipient and does not have an option to avoid this), `transferFrom()` is used for ERC721 NFTs presumably for gas savings and reentrancy concerns over its `safeTransferFrom` variant (which has the `_checkOnERC721Received` check on the recipient).

Recommendation: Evaluate using ERC721 `safeTransferFrom()` to avoid NFTs getting stuck vis-a-vis its reentrancy risk and gas costs.

Sudorandom Labs: Acknowledged, this is an acceptable risk for us given the gas savings and reduced reentrancy surface.

Spearbit: Acknowledged.

5.3.25 Single-step ownership change introduces risks

Severity: Low Risk

Context: LSSVMPairFactory.sol#L39, OwnableWithTransferCallback.sol#L68-L71

Description: Single-step ownership transfers add the risk of setting an unwanted owner by accident (this includes `address(0)`) if the ownership transfer is not done with excessive care. The ownership control library `Owned` by Solmate implements a simple single-step ownership transfer without zero-address checks.

Recommendation: Consider employing 2 step ownership transfer mechanisms for this critical ownership, such as Open Zeppelin's `Ownable2Step` or Synthetic's `Owned`.

Sudorandom Labs: Acknowledged, risk is acceptable for us here as ownership is transferred to callers on factory call. As it is caller-initiated, risks of unintended owners being set are reduced.

Spearbit: Acknowledged.

5.3.26 `getAllPairsForSettings()` may run out of gas

Severity: Low Risk

Context: LSSVMPairFactory.sol#L531-L541

Description: The function `getAllPairsForSettings()` has a loop over `pairsForSettings`. As the creation of pairs is permissionless, that array could get arbitrarily large. Once the array is large enough, the function will run out of gas. Note: the function is only called from the outside.

```
function getAllPairsForSettings(address settings) external view returns (address[] memory) {
    uint256 numPairs = pairsForSettings[settings].length();
    ...
    for (uint256 i; i < numPairs;) {
        ...
        unchecked { ++i;  }
    }
    ...
}
```

Recommendation: Make sure alternative ways exist to retrieve this information in the (unlikely) case that function runs out of gas.

Sudorandom Labs: This function is meant to be used externally only so we're not worried about that.

Spearbit: Acknowledged.

5.3.27 Partially implemented `SellOrderWithPartialFill` functionality may cause unexpected behavior

Severity: Low Risk

Context: VeryFastRouter.sol#L276-L291, VeryFastRouter.sol#L78-L95

Description: `SellOrderWithPartialFill` cannot be performed and sell orders will only be executed if `pair.spotPrice() == order.expectedSpotPrice` in a swap. This may be confusing to users who expect partial fills in both directions but notice unexpected behavior if deployed as-is. While the `BuyOrderWithPartialFill` functionality is fully implemented, the corresponding `SellOrderWithPartialFill` feature is partially implemented with `getNFTQuoteForSellOrderWithPartialFill`, an incomplete `_findMaxFillableAmtForSell` (placeholder comment: "*// TODO: implement*") and other supporting logic required in `swap()`.

Recommendation: Complete implementation of `SellOrderWithPartialFill` feature.

Sudorandom Labs: Acknowledged, now addressed in this open PR#27.

Spearbit: Verified that this is fixed by PR#27.

5.3.28 Lack of deadline checks for certain swap functions allows greater exposure to volatile market prices

Severity: Low Risk

Context: LSSVMRouter.sol#L46-L49, LSSVMRouter.sol#L552-L554, LSSVMRouter.sol#L327-L332, LSSVMRouter.sol#L410-L415

Description: Many swap functions in `LSSVMRouter` use the `checkDeadline` modifier to prevent swaps from executing beyond a certain user-specified deadline. This is presumably to reduce exposure to volatile market prices on top of the thresholds of `maxCost` for buys and `minOutput` for sells. However two router functions `robustSwapETHForSpecificNFTsAndNFTsToToken` and `robustSwapERC20ForSpecificNFTsAndNFTsToToken` in `LSSVMRouter` and all functions in `VeryFastRouter` are missing this modifier and the user parameter required for it.

Users attempting to swap using these two swap functions do not have a way to specify a deadline for their execution unlike the other swap functions in this router. If the front-end does not highlight or warn about this, then the user swaps may get executed after a long time depending on the tip included in the transaction and the network congestion. This causes greater exposure for the swaps to volatile market prices.

Recommendation: Add a `deadline` field to `struct RobustPairNFTsFoTokenAndTokenforNFTsTrade` and include that with the modifier `checkDeadline` similar to other swap functions. Consider adding the same feature for swap functions in `VeryFastRouter`.

Sudorandom Labs: Acknowledged, no change. The intent is for users to use the minInput/maxOutput amounts (or e.g. update their nonce) to prevent against prices changes. Execution over a longer time frame (but within the specified price range) is an acceptable end result.

Spearbit: Acknowledged.

5.3.29 Missing function to deposit ERC1155 NFTs after pair creation

Severity: Low Risk

Context: LSSVMPairFactory.sol#L650-L676

Description: Functions `depositNFTs()` and `depositERC20()` are apparently used to deposit ERC721 NFTs and ERC20s into appropriate pairs after their creation. According to the project team, this is used "*for various UIs to consolidate approvals + emit a canonical event for deposits.*" However, an equivalent function for depositing ERC1155 NFTs is missing.

This prevents ERC1155 NFTs from being deposited into pairs after creation for scenarios anticipated similar to ERC721 NFTs and ERC20 tokens.

Recommendation: Add a `depositNFTs()` for ERC1155 NFTs.

Sudorandom Labs: Solved in PR#65.

Spearbit: Verified that this is fixed by PR#65.

5.4 Gas Optimization

5.4.1 Reading from state is more gas expensive than using msg.sender

Severity: Gas Optimization

Context: LSSVMPairETH.sol#L113-L123

Description: Solmate's Owned.sol contract implements the concept of ownership (by saving during contract construction the deployer in the owner state variable) and owner-exclusive functions via the onlyOwner() modifier. Therefore, within functions protected by the onlyOwner() modifier, the addresses stored in msg.sender and owner will be equal. So, if a function of said characteristics has to make use of the address of the owner, it is cheaper to use msg.sender than owner, because the latter reads from the contract state (using SLOAD opcode) while the former doesn't (address is directly retrieved via the cheaper CALLER opcode).

Reading from state (SLOAD opcode which costs either 100 or 2100 gas units) costs more gas than using the msg.sender environmental variable (CALLER opcode which costs 2 units of gas). Note: withdrawERC20() already uses msg.sender

```
function withdrawETH(uint256 amount) public onlyOwner {
    payable(owner()).safeTransferETH(amount);
    ...
}
function withdrawERC20(ERC20 a, uint256 amount) external override onlyOwner {
    a.safeTransfer(msg.sender, amount);
}
```

Recommendation: Consider replacing owner() by msg.sender to avoid reading from storage:

```
function withdrawETH(uint256 amount) public onlyOwner {
-   payable(owner()).safeTransferETH(amount);
+   payable(msg.sender).safeTransferETH(amount);
    ...
}
```

Sudorandom Labs: Solved in PR#53.

Spearbit: Verified that this is fixed by PR#53.

5.4.2 pair.factory().protocolFeeMultiplier() is read from storage on every iteration of the loop wasting gas

Severity: Gas Optimization

Context: VeryFastRouter.sol#L67, VeryFastRouter.sol#L93

Description: Not caching storage variables that are accessed multiple times within a loop causes waste of gas. If not cached, the solidity compiler will always read the value of protocolFeeMultiplier from storage during each iteration. For a storage variable, this implies extra SLOAD operations (100 additional gas for each iteration beyond the first). In contrast, for a memory variable, it implies extra MLOAD operations (3 additional gas for each iteration beyond the first).

Recommendation: Consider caching pair.factory().protocolFeeMultiplier() to save gas.

Sudorandom Labs: Acknowledged, these functions are intended to be used off chain.

Spearbit: Acknowledged.

5.4.3 The use of `factory` in `ERC1155._takeNFTsFromSender()` can be via a parameter rather than calling `factory()` again

Severity: Gas Optimization

Context: LSSVMPairERC1155.sol#L181, LSSVMPairERC721.sol#L179

Description: `factory` is being sent as a parameter to `_takeNFTsFromSender` in LSSVMPairERC721.sol#L179, which is saving gas because it is not required to read the value again.

```
 _takeNFTsFromSender(IERC721(nft()), nftIds, _factory, isRouter, routerCaller);
```

However, in LSSVMPairERC1155.sol#L181, the similar function `_takeNFTsFromSender()` gets the value by calling `factory()` instead of using a parameter.

```
_takeNFTsFromSender(IERC1155(nft()), numNFTs[0], isRouter, routerCaller);
```

This creates an unnecessary asymmetry between the two contracts which are expected to be similar and also a possible gas optimization by avoiding a call to the factory getter.

Recommendation: Add a `_factory` parameter to LSSVMPairERC1155.sol#L222.

Sudorandom Labs: Solved in PR#51.

Spearbit: Verified that this is fixed by PR#51.

5.4.4 Variables only set at construction time could be made `immutable`

Severity: Gas Optimization

Context: RoyaltyEngine.sol#L50

Description: `immutable` variables can be assigned either at construction time or at declaration time, and only once. The contract creation code generated by the compiler will modify the contract's runtime code before it is returned by replacing all references to `immutable` variables by the values assigned to the them; so the compiler does not reserve a storage slot for these variables.

Declaring variables only set at construction time as `immutable` results in saving one call per variable to `SSTORE` (`0x55`) opcode, thus saving gas during construction.

Recommendation: `royaltyRegistry` could be declared as `immutable` because it is only set in the constructor, saving some gas in the process:

```
- address public royaltyRegistry;
+ address public immutable royaltyRegistry;
```

Sudorandom Labs: Solved in PR#56.

Spearbit: Verified that this is fixed by PR#56.

5.4.5 Hoisting check out of loop will save gas

Severity: Gas Optimization

Context: VeryFastRouter.sol#L312-L320

Description: The check `numIdsFound == maxIdsNeeded` will never be true before the outer `for` loop finishes iterating over `maxIdsNeeded` because `numIdsFound` is conditionally incremented only by 1 in each iteration.

Recommendation: Hoist the check and accompanying `return idsThatExist;` outside the `for` loop to save gas.

Sudorandom Labs: Solved in PR#56.

Spearbit: Verified that this is fixed by PR#56.

5.4.6 Functionality of `safeBatchTransferFrom()` is not used

Severity: Gas Optimization

Context: LSSVMRouter.sol#L531-L543

Description: The function `pairTransferERC1155From()` allow that transfer of multiple id's of ERC1155 NFTs. The rest of the code only uses one id at a time. Using `safeTransferFrom()` instead of `safeBatchTransferFrom()`, might be better as it only accesses one id and uses less gas because no for loop is necessary. However future version of Sudoswap might support multiple ids. In that case its better to leave as is.

```
function pairTransferERC1155From(..., uint256[] calldata ids, uint256[] calldata amounts,...) ... {
    ...
    nft.safeBatchTransferFrom(from, to, ids, amounts, bytes(""));
}
```

Recommendation: Consider using `safeTransferFrom()` instead of `safeBatchTransferFrom()`.

Sudorandom Labs: No change for now.

Spearbit: Acknowledged.

5.4.7 Using `!= 0` instead of `> 0` can save gas

Severity: Gas Optimization

Context: LSSVMPairERC721.sol#L42, LSSVMPairERC721.sol#L149, LSSVMPairERC1155.sol#L52, LSSVMPairERC1155.sol#L103, LSSVMPairERC1155.sol#L151, LSSVMPairERC20.sol#L109, LSSVMPairETH.sol#L48, LSSVMPairETH.sol#L86, LSSVMPairFactory.sol#L588, LSSVMRouter.sol#L231, LSSVMRouter.sol#L361, LSSVMRouter.sol#L596, LSSVMRouter2.sol#L83, LSSVMRouter2.sol#L185, LSSVMRouter2.sol#L381, LSSVMRouter2.sol#L452, LSSVMRouter2.sol#L485, VeryFastRouter.sol#L146, VeryFastRouter.sol#L161, VeryFastRouter.sol#L170, VeryFastRouter.sol#L174, VeryFastRouter.sol#L201, VeryFastRouter.sol#L209.

Description: When dealing with unsigned integer types, comparisons with `!= 0` are 3 gas cheaper than `> 0`.

Recommendation: Use `!= 0` instead of `> 0` for comparison with unsigned integer types where appropriate.

Sudorandom Labs: Addressed in PR#92. `SSVMRouter` changes not addressed because it is intended to be deprecated.

Spearbit: Verified that this is fixed by PR#92.

5.4.8 Using `>>1` instead of `/2` can save gas

Severity: Gas Optimization

Context: LinearCurve.sol#L76, LinearCurve.sol#L144, LSSVMRouter2.sol#L246, LSSVMRouter2.sol#L280, VeryFastRouter.sol#L251, VeryFastRouter.sol#L256,VeryFastRouter.sol#L257, VeryFastRouter.sol#L261.

Description: A division by `2` can be calculated by shifting one to the right (`>>1`). While the `DIV` opcode uses 5 gas, the `SHR` opcode only uses 3 gas.

Recommendation: Consider using shift right for tiny gas optimization.

```
- end = (start + end)/2 - 1;
+ end = (start + end) >> 1 - 1;
```

Sudorandom Labs: Acknowledged, no change for now as we only use `/2` sparingly, and the readability is preferred.

Spearbit: Acknowledged.

5.4.9 Retrieval of ether balance of contract can be gas optimized

Severity: Gas Optimization

Context: Splitter.sol#L27

Description: The retrieval of the ether balance of a contract is typically done with `address(this).balance`. However, by using an assembly block and the `selfbalance()` instruction, one can get the balance with a discount of 15 units of gas.

Recommendation: Consider using an assembly block to retrieve the internal balance. See below:

```
- uint256 ethBalance = address(this).balance;
+ uint256 ethBalance;
+ assembly {
+        ethBalance := selfbalance()
+    }
```

Sudorandom Labs: Acknowledged, no change for now.

Spearbit: Acknowledged.

5.4.10 Function parameters should be validated at the very beginning for gas optimizations

Severity: Gas Optimization

Context: LSSVMPair.sol#L587, LSSVMPair.sol#L600, LSSVMPair.sol#L616

Description: Function parameters should be validated at the very beginning of the function to allow typical execution paths and revert on the exceptional paths, which will lead to gas savings over validating later.

Recommendation: Add the function parameter checks at the very beginning. For example

```
function changeSpotPrice(uint128 newSpotPrice) external onlyOwner {
+   if (spotPrice != newSpotPrice) {
    ICurve _bondingCurve = bondingCurve();
    require(_bondingCurve.validateSpotPrice(newSpotPrice), "Invalid new spot price for curve");
-   if (spotPrice != newSpotPrice) {
        spotPrice = newSpotPrice;
        emit SpotPriceUpdate(newSpotPrice);
    }
+   else {
+       revert(CustomError);
+   }
}
```

Sudorandom Labs: Acknowledged, no change for now.

Spearbit: Acknowledged.

5.4.11 Loop counters are not gas optimized in some places

Severity: Gas Optimization

Context: MerklePropertyChecker.sol#L22, RangePropertyChecker.sol#L28, LSSVMRouter2.sol#L88, LSSVMRouter2.sol#L96, RoyaltyEngine.sol#L81, RoyaltyEngine.sol#L180, RoyaltyEngine.sol#L268, RoyaltyEngine.sol#L321, VeryFastRouter.sol#L61, VeryFastRouter.sol#L69, VeryFastRouter.sol#L83, VeryFastRouter.sol#L91, VeryFastRouter.sol#L106, VeryFastRouter.sol#L151, VeryFastRouter.sol#L312.

Description: Loop counters are optimized in many parts of the code by using an `unchecked {++i}` (unchecked + prefix increment). However, this is not done in some places where it is safe to do so.

Recommendation: Use the following style for loop counter increments in all places consistently:

```
for (uint256 i; i < cachedArrayLength;) {
    ...
    unchecked {
        ++i;
    }
}
```

Sudorandom Labs: Solved in PR#68 and PR#93.

Spearbit: Verified that this is fixed by PR#68 and PR#93.

5.4.12 Mixed use of custom errors and revert strings is inconsistent and uses extra gas

Severity: Gas Optimization

Context: LSSVMPairERC1155.sol, LSSVMPairERC721.sol#L149, LSSVMPair.sol, LSSVMPairETH.sol, VeryFastRouter.sol#, LSSVMPairERC20.sol, LSSVMPairFactory.sol, LSSVMRouter.sol, LSSVMRouter2.sol#L75, RoyaltyEngine.sol

Description: In some parts of the code, custom errors are declared and later used (CurveErrorCodes and Ownable Errors), while in other parts, classic revert strings are used in `require` statements. Instead of using error strings, custom errors can be used, which would reduce deployment and runtime costs.

Using only custom errors would improve consistency and gas cost. This would also avoid long revert strings which consume extra gas. Each extra memory word of bytes past the original 32 incurs an MSTORE which costs 3 gas. This happens at LSSVMPair.sol#L133, LSSVMPair.sol#L666 and LSSVMPairFactory.sol#L505.

Recommendation: Consider using one type of error message consistently, preferably custom errors for gas efficiency and other benefits.

Sudorandom Labs: Acknowledged, but no change at this time.

Spearbit: Acknowledged.

5.4.13 Array length read in each iteration of the loop wastes gas

Severity: Gas Optimization

Context: LSSVMPairERC1155.sol, LSSVMPairERC721.sol, LSSVMPair.sol, LSSVMPairETH.sol, VeryFastRouter.sol, LSSVMPairERC20.sol, LSSVMPairFactory.sol, LSSVMRouter.sol, LSSVMRouter2.sol, RoyaltyEngine.sol

Description: If not cached, the Solidity compiler will always read the length of the array from storage during each iteration. For storage array, this implies extra `SLOAD` operations (100 additional gas for each iteration beyond the first). In contrast, for a memory array, it implies extra `MLOAD` operations (3 additional gas for each iteration beyond the first).

Recommendation: Cache the array length outside of the loop and use that variable in the loop.

```
+ uint256 _royaltyRecipientsLength = royaltyRecipients.length;
- for (uint256 i; i < royaltyRecipients.length;) {
+ for (uint256 i; i < _royaltyRecipientsLength;) {
...
}
```

Sudorandom Labs: Acknowledged, partially addressed in PR#82. The rest would either lead to stack too deep issues or are usually called with 1-3 arguments, so caching will not save too much on gas.

Spearbit: Acknowledged.

5.4.14 Not tightly packing struct variables consumes extra storage slots and gas

Severity: Gas Optimization

Context: VeryFastRouter.sol.sol#L22-L30

Description: Gas efficiency can be achieved by tightly packing structs. Struct variables are stored in 32 bytes each and so you can group smaller types to occupy less storage.

Recommendation: Pack bools together so they use less slots.

Sudorandom Labs: Acknowledged, no change for now.

Spearbit: Acknowledged.

5.4.15 Variables that are redeclared in each loop iteration can be declared once outside the loop

Severity: Gas Optimization

Context: VeryFastRouter.sol#L62

Description: `price` is redefined in each iteration of the loop and right after declaration is set to a new value.

```
    for (uint256 i; i < numNFTs; i++) {
       uint256 price;
       (, spotPrice, delta, price,,) =
           pair.bondingCurve().getBuyInfo(spotPrice, delta, 1, fee,
       pair.factory().protocolFeeMultiplier());
        ...
    }
```

Recommendation: Declaration for `price` can be moved before the loop for gas optimization.

```
+   uint256 price;
    for (uint256 i; i < numNFTs; i++) {
-     uint256 price;
       (, spotPrice, delta, price,,) =
           pair.bondingCurve().getBuyInfo(spotPrice, delta, 1, fee,
       pair.factory().protocolFeeMultiplier());
        ...
    }
```

Sudorandom Labs: Acknowledged, no change as this is inteded to be called only by clients.

Spearbit: Acknowledged.

5.5 Informational

5.5.1 Caller of `swapTokenForSpecificNFTs()` must be able to receive ETH

Severity: Informational

Context: LSSVMPairETH.sol#L76-L81

Description: The function `_refundTokenToSender()` sends ETH back to the caller. If this caller is a contract then it might not be able to receive ETH. If it can't receive ETH then the transaction will revert.

```
function _refundTokenToSender(uint256 inputAmount) internal override {
    // Give excess ETH back to caller
    if (msg.value > inputAmount) {
        payable(msg.sender).safeTransferETH(msg.value - inputAmount);
    }
}
```

Recommendation: Document the requirement that the caller must be able to receive ETH.

Sudorandom Labs: Addressed in PR#76.

Spearbit: Verified that this is fixed by PR#76.

5.5.2 `order.doPropertyCheck` could be replaced by the pair's `propertyChecker()`

Severity: Informational

Context: VeryFastRouter.sol#L120, VeryFastRouter.sol#L134

Description: The field+check for a separate `order.doPropertyCheck` in `struct SellOrder` is unnecessary because this can already be checked via the pair's `propertyChecker()` without relying on the user to explicitly specify it in their order.

Recommendation: Consider removing the field+check for `order.doPropertyCheck` and replacing it with the pair's `propertyChecker()` to save a struct field and to avoid relying on the user's input.

Sudorandom Labs: Acknowledged, no change for now. The current approach is intended to save a bit on gas (no need to call the `propertyChecker()`), and the risk is bounded (the transaction just reverts).

Spearbit: Acknowledged.

5.5.3 `_payProtocolFeeFromPair()` could be replaced with `_sendTokenOutput()`

Severity: Informational

Context: LSSVMPairERC20.sol#L123-L129, LSSVMPairERC20.sol#L132-L137, LSSVMPairETH.sol#L84-L89, LSSVMPairETH.sol#L92-L97

Description: Both ERC20 and ETH versions of `_payProtocolFeeFromPair()` and `_sendTokenOutput()` are identical in their parameters and logic.

Recommendation: `_payProtocolFeeFromPair()` could be replaced with the more generic `_sendTokenOutput()` for simplicity and readability.

Sudorandom Labs: Solved in PR#50.

Spearbit: Verified that this is fixed by PR#50.

5.5.4 False positive in test_getSellInfoWithoutFee() when delta == FixedPointMathLib.WAD due to wrong implementation

Severity: Informational

Context: ExponentialCurve.t.sol#L92, ExponentialCurve.sol#L181

Description: In test_getSellInfoWithoutFee, delta is not validated via validateDelta, which causes a false positive in the current test when delta == FixedPointMathLib.WAD. This can be tried with the following proof of concept

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.15;
import {FixedPointMathLib} from
↪  "https://raw.githubusercontent.com/transmissions11/solmate/main/src/utils/FixedPointMathLib.sol";

contract test{
    using FixedPointMathLib for uint256;
    constructor() {
        uint256 delta = FixedPointMathLib.WAD;
        uint256 invDelta = FixedPointMathLib.WAD.divWadDown(delta);
        uint outputValue = delta.divWadDown(FixedPointMathLib.WAD - invDelta); // revert
    }
}
```

Recommendation: Consider the following change within the test

```
- if (delta < FixedPointMathLib.WAD || spotPrice < MIN_PRICE || numItems == 0) {
+ if (delta <= FixedPointMathLib.WAD || spotPrice < MIN_PRICE || numItems == 0) {
    return;
}
```

Sudorandom Labs: Solved in PR#48.

Spearbit: Verified that this is fixed by PR#48.

5.5.5 Checks-Effects-Interactions pattern not used in swapNFTsForToken()

Severity: Informational

Context: LSSVMPairERC1155.sol#L181, LSSVMPairERC721.sol#L179

Description: It is a defensive programming pattern to first take NFTs and then send the tokens (i.e. the Checks-Effects-Interactions pattern).

```
function swapNFTsForToken(...) ... {
   ...
   _sendTokenOutput(tokenRecipient, outputAmount);
    ...
    _sendTokenOutput(royaltyRecipients[i], royaltyAmounts[i]);
    ...
    _payProtocolFeeFromPair(_factory, protocolFee);
    ...
    _takeNFTsFromSender(...);
    ...
}
```

Recommendation: In both versions of swapNFTsForToken() change the order in the following way:

```
function swapNFTsForToken(...) ... {
    ...
+   _takeNFTsFromSender(...);
   ...
    _sendTokenOutput(tokenRecipient, outputAmount);
    ...
    _sendTokenOutput(royaltyRecipients[i], royaltyAmounts[i]);
    ...
    _payProtocolFeeFromPair(_factory, protocolFee);
    ...
-   _takeNFTsFromSender(...);
    ...
}
```

Sudorandom Labs: Addressed in PR#73.

Spearbit: Verified that this is fixed by PR#73.

5.5.6 Two versions of `withdrawERC721()` and `withdrawERC1155()`

Severity: Informational

Context: LSSVMPairERC721.sol#L272-L299, LSSVMPairERC1155.sol#L257-L301

Description: Both the contracts `LSSVMPairERC721` and `LSSVMPairERC1155` contain the functions `withdrawERC721()` and `withdrawERC1155()` with slightly different implementations. This is more difficult to maintain.

Recommendation: Consider integrating the different versions of the functions `withdrawERC721()` and `withdrawERC1155()` and move them to a library. Note: moving the functions to `LSSVMPair` doesn't work due to multiple inheritance issues.

Sudorandom Labs: Acknowledged, no change for now.

Spearbit: Acknowledged.

5.5.7 Missing sanity/threshold checks may cause undesirable behavior and/or waste of gas

Severity: Informational

Context: LSSVMPairERC1155.sol#L49-53

Description: Numerical user inputs and external call returns that are subject to thresholds due to the contract's logic should be checked for sanity to avoid undesirable behavior or reverts in later logic and wasting unnecessary gas in the process.

Recommendation: Consider adding `require(numNFTs <= _nft.balanceOf(address(this), nftId()), "Ask for <= balanceOf NFTs")`

Sudorandom Labs: Acknowledged, no change for now.

Spearbit: Acknowledged.

5.5.8 Deviation from standard/uniform naming convention affects readability

Severity: Informational

Context: LSSVMPairFactory.sol#L471, LSSVMRouter.sol#L69-L75, LSSVMRouter.sol#L128-L135, LSSVMRouter.sol#L145-L152, LSSVMPairERC20.sol#L34-L115, LSSVMPairERC721.sol#L71-L83, LSSVMPairERC721.sol#L99-L115, PropertyCheckerFactory.sol#L32-L41, RangePropertyChecker.sol#L7-L36, LSSVMPairFactory.sol#L67, LSSVMPairERC1155.sol#86, LSSVMPairERC1155.sol#L212

Description: Following standard/uniform naming conventions are essential to make a codebase easy to read and understand.

Recommendation: Here are some suggestions that would improve the codebase readability and quality:

- LSSVMRouter.sol#L69-L75, LSSVMRouter.sol#L128-L135, LSSVMRouter.sol#L145-L152, LSSVMPairERC721.sol#L71-L83, LSSVMPairERC721.sol#L99-L115: for consistency/clarity, assign to the named return variable instead of an explicit return.
- MerklePropertyChecker.sol#L18-L28 and RangePropertyChecker.sol#L24-L35: for consistency/clarity, assign to the named return variable instead of an explicit return.
- LSSVMPair.sol#L315-L328: for consistency/clarity, assign to the named return variable instead of an explicit return.
- LSSVMPairFactory.sol#L471: the function could be named more specifically (e.g. `getSettingsRoyaltyForPair`) because this is specifically returning the royalty aspect of setting benefits and not setting requirements of upfront-fee, fee-splits and lockup duration.
- LSSVMPairERC20.sol#L34-L115: the function could be named more specifically as it also pulls royalties.
- PropertyCheckerFactory.sol#L32-L41: the function arguments `lowerBound/upperBound` could be renamed to `start/end`.
- RangePropertyChecker.sol#L7-L36: variables and function names referencing `lowerBound/upperBound` could be accordingly renamed so that they reference `start/end`.
- LSSVMPairFactory.sol#L67: the field `wasEverAllowed` of struct `RouterStatus` could be named more specifically (e.g. `wasEverTouched`). The current name can be confusing because the `RouterStatus` struct can be declared with `isAllowed == False`.
- LSSVMPairERC1155.sol#86: the function `swapTokenForAnyNFTs()` should be renamed to `swapTokenForSpecificNFTs()` for consistency with its ERC721 counterpart.
- LSSVMPairERC1155.sol#L212: the function `_sendAnyNFTsToRecipient()` should be renamed to `_sendSpecificNFTsToRecipient()` for consistency with its ERC721 counterpart.

Sudorandom Labs: One suggestion addressed in PR#82, rest acked.

Spearbit: Verified that `wasEverAllowed` is replaced with `wasEverTouched`. Rest is acknowledged.

5.5.9 Function `_getRoyaltyAndSpec()` contains code duplication which affects maintainability

Severity: Informational

Context: RoyaltyEngine.sol#L132-L313

Description: The function `_getRoyaltyAndSpec()` is rather long and contains code duplication. This makes it difficult to maintain.

```
function _getRoyaltyAndSpec(address tokenAddress, uint256 tokenId, uint256 value)
    ...
    if (spec <= NOT_CONFIGURED && spec > NONE) {
        try IArtBlocksOverride(royaltyAddress).getRoyalties(tokenAddress, tokenId) returns (...) {
            // Support Art Blocks override
            require(recipients_.length == bps.length);
            return (recipients_, _computeAmounts(value, bps), ARTBLOCKS, royaltyAddress, addToCache);
        } catch {}
        ...
    } else { // Spec exists, just execute the appropriate one
        ...
        ... if (spec == ARTBLOCKS) {
            // Art Blocks spec
            uint256[] memory bps;
            (recipients, bps) = IArtBlocksOverride(royaltyAddress).getRoyalties(tokenAddress, tokenId);
            require(recipients.length == bps.length);
            return (recipients, _computeAmounts(value, bps), spec, royaltyAddress, addToCache);
        } else ...
    }
```

Recommendation: Consider splitting up the `_getRoyaltyAndSpec()` in smaller pieces where similar code is combined in one smaller function. The use of function pointers might be helpful here.

Sudorandom Labs: Acknowledged, no change. The decision here is to align more closely with the original Manifold code.

Spearbit: Acknowledged.

5.5.10 `getSellInfo` always adds `1` rather than rounding which leads to last item being sold at `0`

Severity: Informational

Context: LinearCurve.sol.sol#L130

Description: Based on the comment `// We calculate how many items we can sell into the linear curve until the spot price reaches 0, rounding up.`

In cases where `delta == spotPrice && numItems > 1`, the last item would be sold at `0`:

```
delta = 100;
spotPrice = 100;
numItems = 2;

uint256 totalPriceDecrease = delta * numItems = 200;
```

Therefore succeeds at:

```
if (spotPrice < totalPriceDecrease)
```

Later calculated:

```
uint256 numItemsTillZeroPrice = spotPrice / delta + 1;
```

That would result in `2`, while the division was an exact `1`, therefore is not rounded up in case where `spotPrice == delta` but increased always by `1`.

Recommendation: Correct the difference between the comment and the implemented logic.

Sudorandom Labs: Yes the comment is inaccurate like you said. The logic is still correct though, in the PoC the first item is sold for 100, the second is sold for 0, and `outputValue = numItems * spotPrice - (numItems * (numItems - 1) * delta) / 2;` equals 100 which is correct.

Spearbit: Acknowledged.

5.5.11 Natspec for robustSwapETHForSpecificNFTs() is slightly misleading

Severity: Informational

Context: LSSVMRouter.sol#L193

Description: The function robustSwapETHForSpecificNFTs() has this comment:

```
* @dev We assume msg.value >= sum of values in maxCostPerPair
```

This doesn't have to be the case. The transaction just reverts if msg.value isn't sufficient.

Recommendation: Consider changing the comment to something like:

```
* @dev Supply msg.value >= sum of values in maxCostPerPair to make sure the transaction doesn't revert
```

Sudorandom Labs: Solved in PR#92 .

Spearbit: Verified that this is fixed by PR#92 .

5.5.12 Two copies of pairTransferERC20From(), pairTransferNFTFrom() and pairTransferERC1155From() are present

Severity: Informational

Context: LSSVMRouter.sol#L491-L543, VeryFastRouter.sol#L344-L407

Description: Both contracts LSSVMRouter and VeryFastRouter contain the functions pairTransferERC20From(), pairTransferNFTFrom() and pairTransferERC1155From(). This is more difficult to maintain as both copies have to stay in synch.

Recommendation: Consider using only one copy of the functions. This can be done via a library or a template from which both contracts inherit.

Sudorandom Labs: We are going to be deprecating LSSVMRouter in favor of VeryFastRouter. Yes, I agree a library would eventually be useful for e.g. making other routers. I'm classifying it as out of scope for now (but something to revisit in the future) as the VeryFastRouter is intended to be the primary router moving forward.

Spearbit: Acknowledged.

5.5.13 Not using error strings in require statements obfuscates monitoring

Severity: Informational

Context: LSSVMPairETH.sol#L139, RoyaltyEngine.sol#L53, RoyaltyEngine.sol#L165, RoyaltyEngine.sol#L172, RoyaltyEngine.sol#L192, RoyaltyEngine.sol#L215, RoyaltyEngine.sol#L222, RoyaltyEngine.sol#L229, RoyaltyEngine.sol#L253, RoyaltyEngine.sol#L259, RoyaltyEngine.sol#L280, RoyaltyEngine.sol#L286, RoyaltyEngine.sol#L304.

Description: require statements should include meaningful error messages to help with monitoring the system.

Recommendation: Consider adding a descriptive reason in an error string / custom error.

Sudorandom Labs: Acknowledged, no change.

Spearbit: Acknowledged.

5.5.14 prices and balances in the curves may not be updated after calls to depositNFTs() and depositERC20()

Severity: Informational

Context: LSSVMPairFactory.sol#L650-L676

Description: The functions depositNFTs() and depositERC20() allow anyone to add NFTs and/or ERC20 to a pair but do not update the prices and balances in the curves. And if they were to do so, then the functions might be abused to update token prices with irrelevant tokens and NFTs.

However, it is not clear if/how the prices and balances in the curves are updated to reflect this. The owner can't fully rely on emits.

Recommendation: Consider making the functions onlyOwner for the pair owner.

Sudorandom Labs: The only curve at the moment which cares about balances is the XYK curve which uses virtual balances (and doesn't read from state). So anyone depositing into a pool wouldn't affect pricing. Adding onlyOwner wouldnt' stop e.g. people from directly calling transferFrom anyway. The intended steps for pool owners who want to modify their pool pricing for an XYK curve with respect to additional funds deposited is to either modify the price first, then deposit funds, or deposit funds first, then modify the price. In general, as depositing funds is going to increase the reserves (and thus decrease the slippage), the recommended procedure would be to deposit additional funds first, then update the virtual reserves.

Spearbit: Acknowledged.

5.5.15 Functions enableSettingsForPair() and disableSettingsForPair() can be simplified

Severity: Informational

Context: LSSVMPairFactory.sol#L501-L524

Description: The functions enableSettingsForPair() and disableSettingsForPair() define a temporary variable pair. This could also be used earlier in the code to simplify the code.

```
function enableSettingsForPair(address settings, address pairAddress) public {
    require(isPair(pairAddress, LSSVMPair(pairAddress).pairVariant()), "Invalid pair address");
    LSSVMPair pair = LSSVMPair(pairAddress);
    ...
    }
function disableSettingsForPair(address settings, address pairAddress) public {
    require(isPair(pairAddress, LSSVMPair(pairAddress).pairVariant()), "Invalid pair address");
    ...
    LSSVMPair pair = LSSVMPair(pairAddress);
    ...
}
```

Recommendation: Consider changing the code to:

```
+       LSSVMPair pair = LSSVMPair(pairAddress);
-       require(isPair(pairAddress, LSSVMPair(pairAddress).pairVariant()), "Invalid pair address");
+       require(isPair(pairAddress, pair.pairVariant()), "Invalid pair address");
        ...
-       LSSVMPair pair = LSSVMPair(pairAddress);
```

Sudorandom Labs: This is no longer an issue because we refactored isPair to take in an address now.

Spearbit: Verified. Now isValidPair() checks whether the address provided is a valid pair without the need of casting the address to a LSSVMPair, unlike what isPair() needed.

5.5.16 Design asymmetry decreases code readability

Severity: Informational

Context: LSSVMPair.sol#L1

Description: The function `_calculateBuyInfoAndUpdatePoolParams()` performs a check on `maxExpectedTokenInput` inside its function.

On the other hand, the comparable check for `_calculateSellInfoAndUpdatePoolParams()` is done outside of the function:

```
function _swapNFTsForToken(...) ... {   // LSSVMPairERC721.sol
    ...
    (protocolFee, outputAmount) = _calculateSellInfoAndUpdatePoolParams(...)
    require(outputAmount >= minExpectedTokenOutput, "Out too few tokens");
    ...
}
```

The asymmetry in the design of these functions affects code readability and may confuse the reader.

Recommendation: Implementing the aforementioned checks in a more symmetric manner yields a cleaner code.

Sudorandom Labs: Solved in PR#76.

Spearbit: Verified that this is fixed by PR#76.

5.5.17 Providing the same `_nftID` multiple times will increase `numPairNFTsWithdrawn` multiple times to potentially cause confusion

Severity: Informational

Context: LSSVMPairERC1155.sol#L285

Description: If one accidentally (or intentionally) supplies the same `id == _nftID` multiple times in the array `ids[]`, then `numPairNFTsWithdrawn` is increased multiple times. Assuming this value is used via indexing for the user interface, this could be misleading.

Recommendation: Potential solution is to add a `break` once an occurrence of `_nftId` has been found. This also saves some gas.

```
for (uint256 i; i < numNFTs;) {
    if (ids[i] == _nftId) {
        numPairNFTsWithdrawn += amounts[i];
+       break;
    }

    unchecked {
        ++i;
    }
}

if (numPairNFTsWithdrawn != 0) {
    // only emit for the pair's NFT
    emit NFTWithdrawal(numPairNFTsWithdrawn);
}
```

Sudorandom Labs: This does change the behavior of the function - in theory a user could pass in ids [1, 1, 1] and amounts [10, 20, 30] and we'd expect the batchTransfer to transfer 60 tokens. We're ok skipping over this change.

Spearbit: Acknowledged.

5.5.18 Dual interface NFTs may cause unexpected behavior if not considered in future

Severity: Informational

Context: LSSVMPairCloner.sol#L90-L94, LSSVMPairCloner.sol#L341-L345

Description: Some NFTs support both the ERC721 and the ERC1155 standard. For example NFTs of the Sandbox project.

Additionally, the internal layout of the parameters of `cloneETHPair` and `cloneERC1155ETHPair` are very similar: | cloneETHPair | cloneERC1155ETHPair | | --- | --- | | mstore(add(ptr, 0x3e), shl(0x60, factory)) | mstore(add(ptr, 0x3e), shl(0x60, factory)) | | mstore(add(ptr, 0x52), shl(0x60, bondingCurve)) | mstore(add(ptr, 0x52), shl(0x60, bondingCurve)) | | mstore(add(ptr, 0x66), shl(0x60, nft)) | mstore(add(ptr, 0x66), shl(0x60, nft)) | | mstore8(add(ptr, 0x7a), poolType) | mstore8(add(ptr, 0x7a), poolType) | | mstore(add(ptr, 0x7b), shl(0x60, propertyChecker)) | mstore(add(ptr, 0x7b), nftId) |

In case there is a specific function that only works on ERC721, and that can be applied to ERC1155 pairs, in combination with an NFT that supports both standards, then an unexpected situation could occur. Currently, this is not the case, but that might occur in future iterations of the code.

Recommendation: Be aware of the risk while maintaining the code. If necessary the type of NFT can be retrieved via `supportInterface()`. By checking for both NFT types, the NFT supporting both standards can be detected.

Sudorandom Labs: Acknowledged.

Spearbit: Acknowledged.

5.5.19 Missing event emission in `multicall`

Severity: Informational

Context: LSSVMPair.sol#L653-L663

Description: Not emitting events on success/failure of calls within a multicall makes debugging failed multicalls difficult. There are several actions that should always emit events for transparency such as ownership change, transfer of ether/tokens etc. In the case of a multicall function, it is recommended to emit an event for succeeding (or failing) calls.

Recommendation: Consider emitting an event after each succeeding/failing call of multicall.

Sudorandom Labs: Acknowledged, no change at this time.

Spearbit: Acknowledged.

5.5.20 Returning only one type of fee from `getBuyNFTQuote()`, `getSellNFTQuote()` and `getSellNFTQuoteWithRoyalties()` could be misleading

Severity: Informational

Context: LSSVMPair.sol#L206-L266

Description: The functions `getBuyNFTQuote()`, `getSellNFTQuote()` and `getSellNFTQuoteWithRoyalties()` return a `protocolFee` variable. There are also other fees like `tradeFee` and `royaltyTotal` that are not returned from these functions. Given that these functions might be called from the outside, it is not clear why other fees are not included here.

Recommendation: Double-check the usefulness of the `protocolFee` variable. If it is never used it could be removed. If it is used from the outside, then it might be useful to also add `tradeFee` and `royaltyTotal`, or return the sum of all fees.

Sudorandom Labs: Acknowledged, no change for now. Users can call `calculateRoyaltiesView` or use the `tradeFee` to calculate values they need from pairs of interest.

Spearbit: Acknowledged.

5.5.21 Two ways to query the assetRecipient could be confusing

Severity: Informational

Context: LSSVMPair.sol#L77, LSSVMPair.sol#L315-L328

Description: The contract LSSVMPair has two ways to query the assetRecipient. On via the getter assetRecipient() and one via getAssetRecipient(). Both give different results and generally getAssetRecipient() should be used. Having two ways could be confusing.

```
address payable public assetRecipient;

function getAssetRecipient() public view returns (address payable _assetRecipient) {
    ... // logic to determine _assetRecipient
}
```

Recommendation: Change the variable assetRecipient to internal. This way no getter function is generated.

Sudorandom Labs: Solved in PR#62 and PR#84.

Spearbit: Verified that this is fixed by PR#62 and PR#84.

5.5.22 Functions expecting NFT deposits can validate parameters for sanity and optimization

Severity: Informational

Context: LSSVMPairFactory.sol#L603-L621, LSSVMPairFactory.sol#L623-L645

Description: Functions expecting NFT deposits in their typical flows can validate parameters for sanity and optimization.

Recommendation: Add _initialNFTBalance != 0 to prevent empty transfers in _nft.safeTransferFrom()

Sudorandom Labs: Solved in PR#49.

Spearbit: Verified that this is fixed in PR#49.

5.5.23 Functions expecting ETH deposits can check msg.value for sanity and optimization

Severity: Informational

Context: LSSVMPairFactory.sol#L547-L571, LSSVMPairFactory.sol#L603-L621

Description: Functions that expect ETH deposits in their typical flows can check for non-zero values of msg.value for sanity and optimization.

Recommendation: Add msg.value > 0 checks.

Sudorandom Labs: Solved in PR#79.

Spearbit: Verified that this is fixed by PR#79.

5.5.24 LSSVMPairs can be simplified

Severity: Informational

Context: LSSVMPairERC1155ERC20.sol#L19, LSSVMPairERC1155ETH.sol#L18, LSSVMPairERC721ERC20.sol#L18, LSSVMPairERC721ETH.sol#L18

Description: At the different LSSVMPairs, PairVariant and IMMUTABLE_PARAMS_LENGTH can be passed to LSSVMPair, which could store them as immutable. Then functions pairVariant() and _immutableParamsLength() can also be moved to LSSVMPair, which would simplify the code.

Recommendation: Consider passing the values to LSSVMPair in the constructor and move the functions pairVariant() and _immutableParamsLength() to LSSVMPair. For example, at LSSVMPairERC1155ERC20.sol#L19:

```
- constructor(IRoyaltyEngineV1 royaltyEngine) LSSVMPair(royaltyEngine) {}
+ constructor(IRoyaltyEngineV1 royaltyEngine)
↪  LSSVMPair(ILSSVMPairFactoryLike.PairVariant.ERC1155_ERC20, 133, royaltyEngine) {}
```

Sudorandom Labs: Good suggestion. Acknowledged, no change for now.

Spearbit: Acknowledged.

5.5.25 Unused values in catch can be avoided for better readability

Severity: Informational

Context: StandardSettings.sol#L137

Description: Employing a catch clause with higher verbosity may reduce readability. Solidity supports different kinds of catch blocks depending on the type of error. However, if the error data is of no interest, one can use a simple catch statement without error data.

Recommendation: Consider removing the arguments between the catch statement for clarity. See below

```
try pairFactory.enableSettingsForPair(address(this), msg.sender) {}
- catch(bytes memory ) {
+ catch {
    revert("Pair verification failed");
}
```

Sudorandom Labs: Solved in PR#72.

Spearbit: Verified that this is solved by PR#72.

5.5.26 Stale constant and comments reduce readability

Severity: Informational

Context: RoyaltyEngine.sol#L35, RoyaltyEngine.sol#L146

Description: After some updates, the logic was added ~2 years ago when enum was changed to int16. Based on the comments and given that was upgradeable, it was expected that one could add new unconfigured specs with negative IDs between NONE (by decrementing it) and NOT_CONFIGURED. In this non-upgradeable fork, the current constants treat only the spec ID of 0 as NOT_CONFIGURED.

```
// Anything > NONE and <= NOT_CONFIGURED is considered not configured
int16 private constant NONE = -1;
int16 private constant NOT_CONFIGURED = 0;
```

Recommendation: Logic and comments that considers value > NONE && value <= NOT_CONFIGURED can be refactored into value <= NOT_CONFIGURED

Sudorandom Labs: Solved in PR#72.

Spearbit: Verified that this is fixed by PR#72.

5.5.27 Different `MAX_FEE` value and comments in different places is misleading

Severity: Informational

Context: LSSVMPairFactory.sol#L46, LSSVMPair.sol#L46-L47

Description: The same `MAX_FEE` constant is declared in different files with different values, while comments indicate that these values should be the same.

```
// 50%, must <= 1 - MAX_PROTOCOL_FEE (set in LSSVMPairFactory)
uint256 internal constant MAX_FEE = 0.5e18;
```

```
uint256 internal constant MAX_PROTOCOL_FEE = 0.1e18; // 10%, must <= 1 - MAX_FEE`
```

Recommendation: Fix the inconsistency between comment/values in the two files.

Sudorandom Labs: Solved in PR#79 and PR#92.

Spearbit: Verified that this is fixed by PR#79 and PR#92.

5.5.28 Events without indexed event parameters make it harder/inefficient for off-chain tools

Severity: Informational

Context: PropertyCheckerFactory.sol#L11, PropertyCheckerFactory.sol#L12, LSSVMPair.sol#L83, LSSVMPair.sol#L84, LSSVMPair.sol#L85, LSSVMPair.sol#L86, LSSVMPair.sol#L87, LSSVMPair.sol#L88, LSSVMPair.sol#L89, LSSVMPair.sol#L90, LSSVMPair.sol#L91, LSSVMPair.sol#L92, LSSVMPair.sol#L93, LSSVMPair.sol#L94, LSSVMPairFactory.sol#L72, LSSVMPairFactory.sol#L73, LSSVMPairFactory.sol#L74, LSSVMPairFactory.sol#L75, LSSVMPairFactory.sol#L76, LSSVMPairFactory.sol#L77, LSSVMPairFactory.sol#L78, LSSVMPairFactory.sol#L79, LSSVMPairFactory.sol#L80

Description: Indexed event fields make them quickly accessible to off-chain tools that parse events. However, note that each indexed field costs extra gas during emission; so it's not necessarily best to index the maximum allowed per event (three fields).

Recommendation: Consider which event parameters could be particularly useful for off-chain tools and index them.

Sudorandom Labs: Solved in PR#74.

Spearbit: Verified that this is fixed by PR#74.

5.5.29 Some functions included in `LSSVMPair` are not found in `ILSSVMPair.sol` and `ILSSVMPairFactoryLike.sol`

Severity: Informational

Context: ILSSVMPair.sol#L10

Description: `LSSVMPair` contract defines the following functions which are missing from interface `ILSSVMPair`:

```
ROYALTY_ENGINE()
spotPrice()
delta()
assetRecipient()
pairVariant()
factory()
swapNFTsForToken()  (2 versions)
swapTokenForSpecificNFTs()
getSellNFTQuoteWithRoyalties()
call()
withdrawERC1155()
```

Recommendation: Consider including the above functions in interface `ILSSVMPair` so that other contracts can import the interface and use them.

Sudorandom Labs: Acknowledged.

Spearbit: Acknowledged.

5.5.30 Absent/Incomplete Natspec affects readability and maintenance

Severity: Informational

Context: IOwnershipTransferReceiver.sol#L6, OwnableWithTransferCallback.sol#L39-L42, RangePropertyChecker.sol#L24, MerklePropertyChecker.sol#L11-L13, MerklePropertyChecker.sol#L18, IPropertyChecker.sol#L18, LSSVMPairERC1155.sol#L216-L221, VeryFastRouter.sol#L53-L55, VeryFastRouter.sol#L75-L77, VeryFastRouter.sol#L220-L227, LSSVMRouter.sol#L317-L326, LSSVMRouter.sol#L400-L409, LSSVMPair.sol#L202-L205, LSSVMPair.sol#L221-L224, LSSVMPair.sol#L240-L243, LSSVMPair.sol#L268-L270

Description: Comments are key to understanding the codebase logic. In particular, Natspec comments provide rich documentation for functions, return variables and more. This documentation aids users, developers and auditors in understanding what the functions within the contract are meant to do.

However, some functions within the codebase contain issues with respect to their comments with either no Natspec or incomplete Natspec annotations, leading to partial descriptions of the functions.

Recommendation: Add and fix comments where needed. For Natspec comments, include the missing parameters, returns and descriptions where needed.

- Missing Natspec:
  - IOwnershipTransferReceiver.sol#L6
  - RangePropertyChecker.sol#L24
  - MerklePropertyChecker.sol#L18
  - IPropertyChecker.sol#L18
- Incomplete Natspec:
  - OwnableWithTransferCallback.sol#L39-L42: annotations missing for `newOwner` and `data`.
  - MerklePropertyChecker.sol#L11-L13: wrong description for `return`.
  - LSSVMPairERC1155.sol#L216-L221: annotations missing for `isRouter` and `routerCaller`.
  - VeryFastRouter.sol#L53-L55: although it can be inferred from the comment, there are missing annotations for `pair`, `numNFTs` and the `return`.
  - VeryFastRouter.sol#L75-L77: natspec is incorrect (copy-paste from another function).
  - VeryFastRouter.sol#L220-L227: annotations missing for `protocolFeeMultiplier` and `return` values.
  - LSSVMRouter.sol#L317-L326: annotations missing for `return` values.

  - LSSVMRouter.sol#L400-L409: annotations missing for `return` values.
  - LSSVMPair.sol#L202-L205: annotations missing for `return` values.
  - LSSVMPair.sol#L221-L224: annotations missing for `return` values.
  - LSSVMPair.sol#L240-L243: annotations missing for `assetId` and `return` values.
  - LSSVMPair.sol#L268-L270: annotations missing for `return` values.
  - LSSVMPairERC721.sol#L220 annotations missing for `_factory` value.
- Unclear comment
  - LSSVMPairCloner.sol#L51 not clear if the value it's hexadecimal or decimal.
- Incorrect comment
  - LSSVMPairCloner.sol#L299 should be EXTRA DATA (93 bytes).
  - LSSVMPairCloner.sol#L386 should be EXTRA DATA (113 bytes).
  - StandardSettings.sol#L257 should say "sell to pair".
  - LSSVMPairERC721.sol#L94 should say "ERC721 tokens will be transfered" rather than "ERC20 tokens will be transferred".
  - LSSVMPairERC721.sol#L268 should be `onlyOwner` not `onlyOwnable`.
  - StandardSettings.sol#L257 should say "sell to pair".
  - LSSVMPairERC721.sol#L94 it says "ERC20 tokens will be transfered", should be "ERC721 tokens will be transferred".
  - LSSVMPairERC721.sol#L268, LSSVMPair.sol#L559, LSSVMPair.sol#L566 should be `onlyOwner` not `onlyOwnable`.
  - LSSVMRouter.sol#L61-L68 `The Unix timestamp (in seconds) at/after which the swap will revert` should be replaced by `The Unix timestamp (in seconds) after which the swap will revert`.
- Missing comment
  - `cloneERC1155ETHPair`, `cloneERC1155ERC20Pair` are missing Total length comment like the ones found at `cloneETHPair()` and `cloneERC20Pair()`, `cloneERC1155ERC20Pair`.
  - ExponentialCurve.sol#L24 Other contract have a comment about the unused variable name. For example: `function validateSpotPrice(uint128 /*newSpotPrice*/ ) external pure override returns (bool) {`.
- Misplaced comment
  - ReentrancyGuard.sol#L3, `cloneERC1155ERC20Pair` comment is related to `Ownable` not to `ReentrancyGuard`.
  - MerklePropertyChecker.sol#L12 comment not relevant (copy/paste from `RangePropertyChecker`).
  - StandardSettings.sol#L239 comments from what variables are make more sense to stay between `,`s.
- `@inheritdoc` can be used for inherited Natspec
  - LinearCurve.sol#L15, LinearCurve.sol#L23, LinearCurve.sol#L31, LinearCurve.sol#L96, XykCurve.sol#L24, XykCurve.sol#L32, XykCurve.sol#L40, XykCurve.sol#L91, ExponentialCurve.sol#L15, ExponentialCurve.sol#L22, ExponentialCurve.sol#L29, ExponentialCurve.sol#L96 can use `@inheritdoc ICurve`

Sudorandom Labs: Acknowledged, some larger ones addressed in PR#82, the rest are acked but not changed.

Spearbit: Acknowledged.

5.5.31 MAX_SETTABLE_FEE value does not follow a standard notation

Severity: Informational

Context: StandardSettings.sol#L22

Description: The protocol establishes several constant hard-coded MAX_FEE-like variables across different contracts. The percentages expressed in those variables should be declared in a standard way all over the codebase. In StandardSettings.sol#L22, the standard followed by the rest of the codebase is not respected. Not respecting the standard notation may confuse the reader.

Recommendation: Consider adapting a standard notation for fee amounts on the codebase. For example, set 100% as 1e18, 99% as 0.99e18 and so on.

```
- uint96 constant MAX_SETTABLE_FEE = 2e17; // Max fee of 20%
+ uint96 constant MAX_SETTABLE_FEE = 0.2e18; // Max fee of 20%
```

Sudorandom Labs: Solved in PR#79.

Spearbit: Verified that this is fixed by PR#79.

5.5.32 No modifier for __Ownable_init

Severity: Informational

Context: OwnableWithTransferCallback.sol#L24-L26

Description: Usually __Ownable_init also has a modifier like initializer or onlyInitializing, see OwnableUpgradeable.sol#L29. The version in OwnableWithTransferCallback.sol doesn't have this. It is not really necessary as the function is internal but it is more robust if it has.

```
function __Ownable_init(address initialOwner) internal {
     _owner = initialOwner;
}
```

Recommendation: Consider adding a modifier like initializer to __Ownable_init().

Sudorandom Labs: Acknowledged, no change as the function is internal.

Spearbit: Acknowledged.

5.5.33 Wrong value of seconds in year slightly affects precision

Severity: Informational

Context: StandardSettingsFactory.sol#L12

Description: Calculation of ONE_YEAR_SECS takes into account leap years (typically 365.25 days), looking for most exact precision.

However as can be seen at NASA and stackoverflow, the value is slightly different.

Current case:

365.2425 days = 31_556_952 / (24 * 3600)

NASA case:

365.2422 days = 31_556_926 / (24 * 3600)

Recommendation: Use 31_556_926 in ONE_YEAR_SECS for maximum precision

Sudorandom Labs: Solved in PR#79.

Spearbit: Verified that this is fixed by PR#79.

5.5.34 Missing idempotent checks may be added for consistency

Severity: Informational

Context: LSSVMPairFactory.sol#L409-L413, LSSVMPairFactory.sol#L419-L423, LSSVMPairFactory.sol#L430-L433, StandardSettings.sol#L79-L81

Description: Setter functions could check if the value being set is the same as the variable's existing value to avoid doing a state variable write in such scenarios and they could also revert to flag potentially mismatched offchain-onchain states. While this is done in many places, there are a few setters missing this check.

Recommendation: Add idempotent checks and consider reverting when the same value is being set.

Sudorandom Labs: Acknowledged, no change.

Spearbit: Acknowledged.

5.5.35 Missing events affect transparency and monitoring

Severity: Informational

Context: LSSVMPair.sol#L640-L645, LSSVMPairFactory.sol#L485-L492, LSSVMPairFactory.sol#L501-L508, LSSVMPairFactory.sol#L517-L524, LSSVMPairERC1155.sol#L257-L265, LSSVMPairERC1155.sol#L273-L301, LSSVMPairERC721.sol#L272-L284, LSSVMPairERC721.sol#L292-L300, LSSVMPairETH.sol#L113-L118, LSSVMPairETH.sol#L121-L123, LSSVMPairERC20.sol#L140-L147

Description: Missing events in critical functions, especially privileged ones, reduce transparency and ease of monitoring. Users may be surprised at changes affected by such functions without being able to observe related events.

Recommendation: Add appropriate events to emit in critical/privileged functions.

Sudorandom Labs: Acknowledged, Settings enable/disable event now tracked in PR#86, but not other changes.

Spearbit: Acknowledged.

5.5.36 Wrong error returned affects debugging and off-chain monitoring

Severity: Informational

Context: XykCurve.sol#L71

Description: `Error.INVALID_NUMITEMS` is declared for `0` case, but is returned twice in the same function: first time for `numItems == 0` and second time for `numItems >= nftBalance`. This can make hard to know why it is failing.

Recommendation: Define a new error type for this case and use accordingly.

```diff
enum Error {
        OK, // No error
        INVALID_NUMITEMS, // The numItem value is 0
-       SPOT_PRICE_OVERFLOW // The updated spot price doesn't fit into 128 bits
+       SPOT_PRICE_OVERFLOW, // The updated spot price doesn't fit into 128 bits
+       TOO_MANY_NUMITEMS // The numItem >= nftBalance
    }
```

Sudorandom Labs: Acknowledged, no change.

Spearbit: Acknowledged.

5.5.37 Functions can be renamed for clarity and consistency

Severity: Informational

Context: LSSVMPairCloner.sol#L22 LSSVMPairCloner.sol#L112

Description: Since both functions `cloneETHPair()` and `cloneERC20Pair()` use `IERC721 nft` as a parameter, renaming them to `cloneERC721ETHPair()` and `cloneERC721ERC20Pair()` respectively makes it clearer that the functions process ERC721 tokens. This also provides consistency in the naming of functions considering that we already have function cloneERC1155ETHPair() using this nomenclature.

Recommendation: Consider renaming `cloneETHPair()` and `cloneERC20Pair()` to `cloneERC721ETHPair()` and `cloneERC721ERC20Pair()` respectively.

Sudorandom Labs: Addressed in PR#82.

Spearbit: Verified that this is fixed by PR#82.

5.5.38 Two events `TokenDeposit()` with different parameters

Severity: Informational

Context: LSSVMPairFactory.sol#L74, LSSVMPair.sol#L88

Description: The event `TokenDeposit()` of `LSSVMPairFactory` has an `address` parameter while the event `TokenDeposit()` of `LSSVMPair` has an `uint256` parameter. This might be confusing.

```
contract LSSVMPairFactory {
    ...
    event TokenDeposit(address poolAddress);
    ...
}
abstract contract LSSVMPair ... {
    ...
    event TokenDeposit(uint256 amount);
    ...
}
```

Recommendation: Consider renaming one of the event names or expand the events to both include `address` and `amount`.

Sudorandom Labs: Solved in PR#81.

Spearbit: Verified that this is fixed by PR#81.

5.5.39 Unused imports affect readability

Severity: Informational

Context: XykCurve.sol#L4-L13, LSSVMPairERC20.sol#L4-L13, LSSVMPairETH.sol#L4-L11

Description: The following imports are unused in

- `XykCurve.sol`

```
import {IERC721} from "@openzeppelin/contracts/token/ERC721/IERC721.sol";
import {LSSVMPair} from "../LSSVMPair.sol";
import {LSSVMPairERC20} from "../LSSVMPairERC20.sol";
import {LSSVMPairCloner} from "../lib/LSSVMPairCloner.sol";
import {ILSSVMPairFactoryLike} from "../LSSVMPairFactory.sol";
```

- `LSSVMPairERC20.sol`

```
import {IERC721} from "@openzeppelin/contracts/token/ERC721/IERC721.sol";
import {ICurve} from "./bonding-curves/ICurve.sol";
import {CurveErrorCodes} from "./bonding-curves/CurveErrorCodes.sol";
```

- `LSSVMPairETH.sol`

```
import {IERC721} from "@openzeppelin/contracts/token/ERC721/IERC721.sol";
import {ICurve} from "./bonding-curves/ICurve.sol";
```

Recommendation: Remove the unused imports.

Sudorandom Labs: Solved in PR#76.

Spearbit: Verified that this is fixed by PR#76.

5.5.40 Use of `isPair()` is not intuitive

Severity: Informational

Context: LSSVMPairFactory.sol#L309-L322

Description: There are two usecases for `isPair()`

1) To check if the contract is a pair of any of the 4 types. Here the type is always retrieved via `pairVariant()`.
2) To check if a pair is `ETH` / `ERC20` / `ERC721` / `ERC1155`. Each of these values are represented by two different pair types. Using `isPair()` this way is not intuitive and some errors have been made in the code where only one value is tested. Note: also see issue *"`pairTransferERC20From` only supports ERC721 NFTs"*.

Function `isPair()` could be refactored to make the code easier to read and maintain.

```
function isPair(address potentialPair, PairVariant variant) public view override returns (bool) {
    ...
}
```

These are the occurrences of use case 1:

```
LSSVMPairFactory.sol:  require(isPair(pairAddress, LSSVMPair(pairAddress).pairVariant()), "Invalid pair
↪   address");
LSSVMPairFactory.sol:  require(isPair(pairAddress, LSSVMPair(pairAddress).pairVariant()), "Invalid pair
↪   address");
LSSVMPairFactory.sol:  if (isPair(recipient, LSSVMPair(recipient).pairVariant())) {

// router interaction, which first queries `pairVariant()`
LSSVMPairERC20.sol:            router.pairTransferERC20From(..., pairVariant());
LSSVMPairERC20.sol:            router.pairTransferERC20From(..., pairVariant()
LSSVMPairERC20.sol:            router.pairTransferERC20From(..., pairVariant());
erc721/LSSVMPairERC721.sol:    router.pairTransferNFTFrom(..., pairVariant());
erc721/LSSVMPairERC721.sol:    router.pairTransferNFTFrom(..., pairVariant());
erc1155/LSSVMPairERC1155.sol:  router.pairTransferERC1155From(..., pairVariant());
// router and VeryFastRouter
function pairTransferERC20From(..., ILSSVMPairFactoryLike.PairVariant variant) ... {
    ...     require(factory.isPair(msg.sender, variant), "Not pair");    ...
}
function pairTransferNFTFrom(..., ILSSVMPairFactoryLike.PairVariant variant ... {
    ...     require(factory.isPair(msg.sender, variant), "Not pair");    ...
}
function pairTransferERC1155From(..., ILSSVMPairFactoryLike.PairVariant variant) ... {
    ...    require(factory.isPair(msg.sender, variant), "Not pair");    ...
}
```

These are the occurrences of use case 2:

```
LSSVMPairFactory.sol:    (isPair(...ERC721_ERC20)    || isPair(...ERC1155_ERC20))
StandardSettings.sol:    ...isPair(....ERC721_ETH)   || ...isPair(...ERC1155_ETH)
StandardSettings.sol:    ...isPair(...ERC721_ERC20)  || ...isPair(...ERC1155_ERC20)
StandardSettings.sol:    ...isPair(...ERC721_ETH)    || ...isPair(...ERC1155_ETH)
StandardSettings.sol:    ...isPair(...ERC721_ERC20)  || ...isPair(...ERC1155_ERC20)
```

Recommendation: Consider creating two (possibly overloaded) versions of `isPair()`, dedicated for the both use cases. This could be something like:

```
function isPair(address potentialPair)    // use case 1
enum PoolTypeSelect { ETH, ERC20, ERC721, ERC1155  }
function isPair(address potentialPair, PoolTypeSelect  toCheck)  // use case 2
```

Sudorandom Labs: Solved in PR#30.

Spearbit: Verified that this is fixed by PR#30.

5.5.41 Royalty related code spread across different contracts affects readability

Severity: Informational

Context: LSSVMPairFactory.sol#L330-L377, RoyaltyEngine.sol

Description: The contract `LSSVMPairFactory` contains the function `authAllowedForToken()`, which has a lot of interactions with external contracts related to royalties. The code is rather similar to code that is present in the `RoyaltyEngine` contract. Combining this code in `RoyaltyEngine` contract would make the code cleaner and easier to read.

Recommendation: Consider moving the function `authAllowedForToken()` to the `RoyaltyEngine` contract.

Sudorandom Labs: Acknowledged, though no change at the moment. Moving it to the `Engine` itself would incur additional gas for the `CALL`, as the `authAllowed` functions are only used for the PairFactory. So this is a gas trade-off we are willing to make in exchange for scattering the logic.

Spearbit: Acknowledged.