SudoAMM v2

Security Testing and Assessment

February 20, 2023

Prepared for Sudorandom Labs

Narya.ai

Table of Content

Summary

Overview

From February 6, 2023, to February 20, 2023, Sudorandom Labs engaged Narya.ai to test the security of its AMM (SudoAMM) v2 in the GitHub repository: https://github.com/sudoswap/lssvm2 (commit 43828e4a393c48a7d6b401dcbfc3bb92574b6c14).

SudoAMM v2 is a DEX protocol for trading ERC721 and ERC1155 NFTs with ETH or ERC20 tokens. It is an upgraded version of sudoAMM v1 with the following new features:

- On-chain royalty support for all collections
- Property checkers for pairs to allow for specific NFT IDs
- Project-controlled setting for each pair that enforces specific requirements
- ERC1155 support
- A new VeryFastRouter which allows for handling all swap types (i.e. ERC721<>ETH, ERC721<>ERC20, ERC1155<>ETH, ERC1155<>ERC20)

Project Scope

We reviewed and tested the smart contracts that implement the different swapping implementations but also their interactions with royalties payments and Settings contracts. There are other security-critical components of SudoAMM V2, such as off-chain services and the web front end, that are not included in the scope of this smart contract security assessment. We recommend a further review of those components.

Summary of Findings

Severity	# of Findings
High	0
Medium	3
Low	6
Informational	0
Total	9

Throughout the testing, we identified 9 issues of medium and low severity:

- 1 issue of double trade fee on NFT buys
- 1 issue of duplicate function signature leading to user NFT loss
- 2 issues of wrong royalty calculation
- 1 issue of unchecked call target
- 2 issues of NFT deposit checks
- 1 issue of pair ownership management
- 1 issue of missing check on asset/token recipient

Disclaimer

This testing report should not be used as investment advice.

Narya.ai uses an automatic testing technique to test smart contracts' security properties and business logic rapidly and continuously. However, we do not provide any guarantees on eliminating all possible security issues. The technique has its limitations: for example, it may not generate a random edge case that violates an invariant during the allotted time. Its use is also limited by time and resource constraints.

Unlike time-boxed security assessment, Narya.ai advises continuing to create and update security tests throughout the project’s lifetime. In addition, Narya.ai recommends proceeding with several other independent audits and a public bug bounty program to ensure smart contract security.

Project Overview

SudoAMM V2 is divided into multiple components:

- A Router that implements different types of swaps: directly or indirectly (using ETH/ERC20 intermediaries), and non-reverting swaps.
- A very fast router that is able to bulk orders together and supports partial fills.
- A Royalty engine capable of handling different types of royalties and computing the amount.
- Property checkers allow a Pair to filter the NFTs being swapped.
- Settings that offer the ability to enforce specific requirements on the pool. For example, a lockup period, an upfront fee, or a separate royalty amount.

Tests

Tested Invariants and Properties

We relied on the Narya engine that used a smart fuzzing approach to test the following 18 invariants and properties of the smart contracts.

Table 1 Tested Invariants

ID	Invariant/Property Description	Found Bug(s)
01	A deployed pair's owner cannot be changed by an arbitrary user.	Passed
02	A deployed factory's owner cannot be changed by an arbitrary user.	Passed
03	For a pool using the linear bonding curve, the swapped amount should be the same as the quoted amount.	Passed
04	For a pool using the XYK bonding curve, the swapped amount should be the same as the quoted amount.	Passed
05	For a pool using the exponential bonding curve, the swapped amount should be the same as the quoted amount.	Passed
06	For a pool using the linear bonding curve, the swap enforces the bonding curve correctly.	Passed
07	For a pool using the XYK bonding curve, the swap enforces the bonding curve correctly.	Passed

08	For a pool using the exponential bonding curve, the swap enforces the bonding curve correctly.	Passed
09	Regardless of how an ETH Pair is created, an expected successful swap should succeed.	1. Double trade fee on buys conflicts with royalties
10	Regardless of how an ERC20 Pair is created, a successful swap of tokens for NFTs should not make the Pair lose ERC20 tokens.	1. Double trade fee on buys conflicts with royalties
11	When calling bulkWithdrawFees(), regardless of what addresses are being passed to the function as Splitter objects, a Pair's liquidity shouldn't be drained.	2. Duplicate signature of withdrawAllETH can be used to steal assets from pairs stealthily
12	If a user approves a transaction but doesn't swap due to a change of mind, it is validated that the user's NFTs cannot be stolen.	4. LSSVMPair::call should not allow the call target to be the current NFT
13	The expected amount of royalties is correctly received for ETH pairs.	Passed
14	The expected amount of royalties is correctly received for ERC20 pairs.	5. Royalty calculation error
15	The NFT information emitted by the NFTDeposit events matches that of the deposited NFTs.	6. Wrong NFTDeposit event can be emitted
16	A collection owner should always be able to reclaim ownership using reclaimPair().	7. Collection owner can be rejected in reclaimPair
17	A user shouldn't be able to call swap and have their share of tokens being received by the Pair while no NFTs were sent to the pair.	8. Incorrect NFT balance check in LSSVMPairERC721::_takeNFTsFromSender
18	A user shouldn't be able to call swap and have their share of tokens/NFTs being received by the Pair, resulting in a potential loss of funds.	9. nftRecipient and tokenRecipient are not checked to ensure assets don't go to the pool

Key Findings and Recommendations

1. Double trade fee on buys conflicts with royalties

Severity: **Medium** Difficulty: **Low**
File(s): LSSVMPairERC721.sol, LSSVMPairERC1155.sol

Description

In the `swapTokenForSpecificNFTs` function of the `LSSVMPairERC721` contract, the `_pullTokenInputAndPayProtocolFee` function is invoked to pull the required number of input tokens from the buyer or the current `msg.value`:

Code 1 https://github.com/sudoswap/lssvm2/blob/main/src/erc721/LSSVMPairERC721.sol#L51

```
        _pullTokenInputAndPayProtocolFee(
            nftIds[0],
            inputAmount,
            2 * tradeFee, // We pull twice the trade fee on buys but don't take
trade fee on sells if assetRecipient is set
            isRouter,
            routerCaller,
            _factory,
            protocolFee
        );
```

As shown above, double trade fees are pulled for buying NFTs. And `inputAmount` represents the maximum number of tokens required for the swap, calculated from bonding curves:

```
inputAmount = cost + cost * protocolFeeBps + cost * tradeFeeBps
```

Meanwhile, both double trade fees and royalties are pulled from `inputAmount`. Based on the calculation in the `_pullTokenInputAndPayProtocolFee` function,

```
royalty = (inputAmount - protocolFee) * royaltyBps
```

The total amount of double trade fees and royalties can be more significant than `inputAmount` when the following condition is satisfied:

```
tradeFeeBps + royaltyBps + tradeFeeBps * royaltyBps > 1
```

In this case, a more significant number of input tokens than `inputAmount` is required for a swap.

Impact

When the pair is an ETH one and the aforementioned condition is satisfied, all the trades will fail due to an insufficient amount of ETH (`msg.value`) being sent. A user needs to send (no refund) more than `maxExpectedTokenInput` amount of ETH to complete a swap.
ERC20 pairs are different from ETH pairs. The trade fees are sent to `assetRecipient` at the end when `assetRecipient` is set. Since the required number of ERC20 tokens is larger than inputAmount, not enough tokens are pulled from the user. The tokens in the pair will be sent to `assetRecipient`, which decreases the liquidity. If there is setting for splitting trade fee, more tokens will be paid to the collection owner.

Failed Invariant

```
    function invariantUnderflow() public {
        if (address(pair721) == address(0)) return;

        (bool success, bytes memory data) =
address(this).call(abi.encodeWithSignature("nonRevertingInvariantOverflow()"));
        require(success, "Swap failed when it should succeed");
    }

    function nonRevertingInvariantOverflow() public {
        uint256[] memory nftIds = new uint256[](1);
        nftIds[0] = 1;
        LSSVMRouter.PairSwapSpecific[] memory swapList = new
LSSVMRouter.PairSwapSpecific[](1);
        swapList[0] = LSSVMRouter.PairSwapSpecific({pair: pair721, nftIds:
nftIds});
        (,,, uint256 inputAmount,) = pair721.getBuyNFTQuote(1);

        vm.startPrank(user);
        this.swapTokenForSpecificNFTs{value: modifyInputAmount(inputAmount)}(
            router, swapList, payable(address(this)), address(this),
block.timestamp, inputAmount
        );
        vm.stopPrank();
    }
```

```
    function invariantUnderflow() public {
        if (address(pair721) == address(0)) return;

        uint256[] memory nftIds = new uint256[](1);
```

```
        nftIds[0] = 1;
        LSSVMRouter.PairSwapSpecific[] memory swapList = new
LSSVMRouter.PairSwapSpecific[](1);
        swapList[0] = LSSVMRouter.PairSwapSpecific({pair: pair721, nftIds:
nftIds});
        (,,, uint256 inputAmount, uint256 protocolFee) =
pair721.getBuyNFTQuote(1);

        uint256 balanceBefore = this.getBalance(address(pair721));

        this.swapTokenForSpecificNFTs{value: modifyInputAmount(inputAmount)}(
            router, swapList, payable(address(this)), address(this),
block.timestamp, inputAmount
        );

        require(balanceBefore < this.getBalance(address(pair721)), "Pair lost
tokens when it is supposed to receive tokens");
    }
```

Recommendation

- Limit the royalty fee to be smaller than 33.3% as long as the trade fee is limited to under 50%.
- Split a part of the trade fees for royalty.

Remediation

This issue has been acknowledged by Sudorandom Labs and fixed at commit 1ec3e31a4b2600fb2b32286156270e11ebb1ae80. The maximum royalty amount is limited to 25%.

2. Duplicate signature of withdrawAllETH can be used to steal assets from pairs stealthily

Severity: **Medium** Difficulty: **High**
File(s): StandardSettings.sol

Description

The default setting template `StandardSettings` has an external function called `bulkWithdrawFees` which invokes the `withdrawAllETH` function to a list of input addresses (splitter addresses):

Code 2 https://github.com/sudoswap/lssvm2/blob/main/src/settings/StandardSettings.sol#L302

```
    function bulkWithdrawFees(address[] calldata splitterAddresses, bool[] calldata
isETHPair) external {
        for (uint256 i; i < splitterAddresses.length;) {
            Splitter splitter = Splitter(payable(splitterAddresses[i]));
            if (isETHPair[i]) {
                splitter.withdrawAllETH();
            } else {
                splitter.withdrawAllBaseQuoteTokens();
            }
            unchecked {
                ++i;
            }
        }
    }
```

But the `LSSVMPairETH` contract also has the `withdrawAllETH` function with the same signature. There is a type confusion between splitters and pairs.

Impact

An attacker can release a setting implementation that simply adds an empty fallback function to the default `StandardSettings`, which looks harmless. Later, the attacker can input pair addresses as the splitter addresses to the `bulkWithdrawFees` function to transfer all the ETH from the pairs which use the setting.

Failed Invariant

```
function invariantPairBalance() public {
        for (uint i = 0; i < pnmLogs.length; ++i) {
            LogInfo memory log = pnmLogs[i];
            require(log.pairBefore == log.pairAfter, "Pair liquidity drained");
        }
        delete pnmLogs;
    }
```

Recommendation

Use different method signatures for the `withdrawAllETH` function in the `LSSVMPairETH` and `Splitter` contracts.

Remediation

This issue has been acknowledged by Sudorandom Labs and fixed at commit 184693b25d3b82e433aebf4b8e8fb5f7bc8bdf17.

3. Ignorance of different royalty settings for different NFT IDs

Severity: **Medium** Difficulty: **Low**
File(s): StandardSettings.sol

Description

When trading multiple numbers of ERC721 tokens, the royalty calculation only uses the first NFT in the array.

Code 3 https://github.com/sudoswap/lssvm2/blob/main/src/erc721/LSSVMPairERC721.sol#L51

```
        _pullTokenInputAndPayProtocolFee(
            nftIds[0], // ← the first NFT ID
            inputAmount,
            2 * tradeFee, // We pull twice the trade fee on buys but don't take
trade fee on sells if assetRecipient is set
            isRouter,
            routerCaller,
            _factory,
            protocolFee
        );
```

However, according to ERC-2981 (NFT Royalty Standard), different NFT IDs can have different royalty settings. This will result in wrong royalty calculation in total and thus wrong swap results.

Impact

For the same group of NFTs, if their order in the array is different, the swap result will be different.

Failed Invariant

This issue was found by manual code review.

Recommendation

Get the royalty fee for each NFT ID and use the average value for calculating the total royalty amount for the swap.

Remediation

This issue has been acknowledged by Sudorandom Labs. It is considered a known limitation of the existing code that makes this trade-off for gas optimization. This issue will not put the customer funds of SudoAMM v2 at risk.

4. LSSVMPair::call should not allow the call target to be the current NFT

Severity: **Low** Difficulty: **High**
File(s): StandardSettings.sol

Description

A user can directly call a pair where the user will directly approve the pair to use her own NFTs. If the factory adds an NFT address to the whitelist, the pair owner can leverage the `call` function below to call the `transferFrom` function to transfer the user's NFTs.

Code 4 https://github.com/sudoswap/lssvm2/blob/main/src/LSSVMPair.sol#L625

```
    function call(address payable target, bytes calldata data) external onlyOwner {
        ILSSVMPairFactoryLike _factory = factory();
        require(_factory.callAllowed(target), "Target must be whitelisted");

        // ensure the call isn't calling a banned function
        bytes4 sig = bytes4(data[:4]);
        require(sig != IOwnershipTransferReceiver.onOwnershipTransferred.selector,
"Banned function");

        (bool result,) = target.call{value: 0}(data);
        require(result, "Call failed");
    }
```

Note that this issue also exists in SudoAMM v1.

Impact

The pair owner is able to steal NFTs from users.

Failed Invariant

```
   function invariantApproveRaceCondition() public {
       require(test721.balanceOf(user) == 1, "NFTs were stolen");
   }
```

Recommendation

Add the following check in the `call` function of the `LSSVMPair` contract:
`require(target != address(nft()));`

Remediation

This issue has been acknowledged by Sudorandom Labs and fixed at commit 60d0f068d2eb759feb86cb9b7c072e496f347967.

5. Royalty calculation error

Severity: **Low** Difficulty: **High**
File(s): LSSVMPair.sol, StandardSettingsFactory.sol, RoyaltyEngine.sol

Description

The project mostly uses 1e18 as the base to calculate the fee bps except for the royalty bps, where 10000 is used, such as

Code 5 https://github.com/sudoswap/lssvm2/blob/main/src/LSSVMPair.sol#L522

```
        if (recipients.length != 0) {
            // If a pair has custom Settings, use the overridden royalty
amount and only use the first receiver
            (bool settingsEnabled, uint96 bps) =
factory().getSettingsForPair(address(this));
            if (settingsEnabled) {
                royaltyRecipients = new address payable[](1);
                royaltyRecipients[0] = recipients[0];
                royaltyAmounts = new uint256[](1);
                royaltyAmounts[0] = (saleAmount * bps) / 10000;
            } else {
                royaltyRecipients = recipients;
                royaltyAmounts = amounts;
            }
        }
```

Code 6 https://github.com/sudoswap/lssvm2/blob/main/src/settings/StandardSettingsFactory.sol#L28

```
    function createSettings(
        address payable settingsFeeRecipient,
        uint256 ethCost,
        uint64 secDuration,
        uint64 feeSplitBps,
        uint64 royaltyBps
    ) public returns (StandardSettings settings) {
        require(royaltyBps <= (BASE / 10), "Max 10% for modified royalty
bps");
        require(feeSplitBps <= BASE, "Max 100% for trade fee bps split");
        require(secDuration <= ONE_YEAR_SECS, "Max lock duration 1 year");
        bytes memory data = abi.encodePacked(ethCost, secDuration,
feeSplitBps, royaltyBps);
```

```
        settings =
StandardSettings(address(standardSettingsImplementation).clone(data));
        settings.initialize(msg.sender, settingsFeeRecipient);
        emit NewSettings(address(settings));
    }
```

Code 7 https://github.com/sudoswap/lssvm2/blob/main/src/RoyaltyEngine.sol#L322

```
    function _computeAmounts(uint256 value, uint256[] memory bps) private
pure returns (uint256[] memory amounts) {
        amounts = new uint256[](bps.length);
        uint256 totalAmount;
        for (uint256 i = 0; i < bps.length; i++) {
            amounts[i] = value * bps[i] / 10000;
            totalAmount += amounts[i];
        }
        require(totalAmount < value, "Invalid royalty amount");
        return amounts;
    }
```

Although 10000 is used in the example in EIP-2981, there will be rounding errors when trading with ERC20 tokens that have smaller decimals.

Impact

Considering trading with WBTC whose decimal is 8, when using 10000 as a base for the NFT whose price is lower than 0.0001 BTC ($2.1), the royalty amount will be 0.

Failed Invariant

```
function invariantRoyaltyReceived() public {
       for (uint i = 0; i < pnmLogs.length; ++i) {
           LogInfo memory log = pnmLogs[i];
           require(log.royaltyAfter > log.royaltyBefore,
               "didn't receive any royalty");
           require(log.royaltyAfter - log.royaltyBefore == log.expectedRoyalty,
               "unexpected royalty amount");
       }
   }
```

Recommendation

Adjust the royalty bps base to be larger than 1e6.

Remediation

This issue has been acknowledged by Sudorandom Labs. For most relevant tokens, the trade size will be very small when the decimals are this low. Therefore, Sudorandom Labs will not deploy any fix for this issue.

6. Wrong NFTDeposit event can be emitted

Severity: **Low** Difficulty: **Low**
File(s): LSSVMPairFactory.sol

Description

The depositNFT function does not check if the input NFT address matches the NFT address of the recipient pair:

Code 8 https://github.com/sudoswap/lssvm2/blob/main/src/LSSVMPairFactory.sol#L648

```
    /**
     * @dev Used to deposit NFTs into a pair after creation and emit an event for
indexing (if recipient is indeed a pair)
     */
    function depositNFTs(IERC721 _nft, uint256[] calldata ids, address recipient)
external {
        // transfer NFTs from caller to recipient
        uint256 numNFTs = ids.length;
        for (uint256 i; i < numNFTs;) {
            _nft.transferFrom(msg.sender, recipient, ids[i]);

            unchecked {
                ++i;
            }
        }
        if (isPair(recipient, LSSVMPair(recipient).pairVariant())) {
            emit NFTDeposit(recipient, ids);
        }
    }
```

Impact

The NFTDeposit event emitted by the depositNFT function can be meaningless affecting indexing.

Failed Invariant

```
    function invariantNFTDepositEvent() public {
        Vm.Log[] memory entries = vm.getRecordedLogs();
        for (uint i = 0; i < entries.length; ++i) {
```

```
            if (entries[i].topics[0] ==
keccak256("NFTDeposit(address,uint256[])")) {
                (address recipient, uint256[] memory idList) =
this.recoverArguments(entries[i].data);
                for (uint j = 0; j < pnmLogs.length; ++j) {
                    LogInfo memory log = pnmLogs[j];
                    if (log.recipient == recipient && log.idList.length ==
idList.length) {
                        bool identical = true;
                        for (uint k = 0; k < idList.length; ++k) {
                            if (log.idList[k] != idList[k]) {
                                identical = false;
                                break;
                            }
                        }

                        if (identical) {
                            require(LSSVMPair(recipient).nft() == log.nft,
                                "Wrong event fired, deposited nft doesnt belong
to pair");
                        }
                    }
                }
            }
        }
    }

    function recoverArguments(bytes calldata data) public
        returns (address recipient, uint256[] memory idList)
    {
        require(data.length % 32 == 0);
        uint256 len = data.length / 32;
        uint256[] memory _data = new uint256[](len);

        for (uint i = 0; i < len; ++i) {
            for (uint j = 0; j < 32; ++j) {
                _data[i] |= (uint256(uint8(data[i*32+j])) << (248 - (8 * j)));
            }
        }

        recipient = address(uint160(_data[0]));
        uint256 idListLen = _data[2];
        idList = new uint256[](idListLen);
        for (uint i = 0; i < idListLen; ++i) {
            idList[i] = _data[3+i];
```

```
        }
    }
```

Recommendation

Check if the input NFT address equals `pair.nft()` and only emit the indexed event when it is true.

Remediation

This issue has been acknowledged by Sudorandom Labs and fixed at commit 836181d6d913d86e65adc08d0e7e8528b4a9dc47.

7. Collection owner can be rejected in reclaimPair

Severity: **Low** Difficulty: **Low**
File(s): StandardSettings.sol

Description

The `reclaimPair` function checks:

(1) If the `msg.sender` is the original pair owner, then the time must pass the unlock time.
(2) If the `msg.sender` is the collection owner, then there is no further check.

Code 9 https://github.com/sudoswap/lssvm2/blob/main/src/settings/StandardSettings.sol#L164

```
    function reclaimPair(address pairAddress) public {
        PairInfo memory pairInfo = pairInfos[pairAddress];

        ILSSVMPair pair = ILSSVMPair(pairAddress);

        // Verify that the caller is the previous pair owner or admin of the NFT
collection
        if (msg.sender == pairInfo.prevOwner) { // (1)
            // If previous owner, verify that the current time is past the unlock
time
            require(block.timestamp > pairInfo.unlockTime, "Lockup not over");
        }
        // Otherwise, if not an authorized address, revert
        else if (!pairFactory.authAllowedForToken(address(pair.nft()), msg.sender))
{ // (2)
            revert("Not prev owner or collection admin");
        }
        // ...
```

However, if the collection owner is the original pair owner, then it will go into the first case where there is still time check.

Impact

The collection owner may still get the unlock time check when calling the `reclaimPair` function.

Failed Invariant

```
    function invariantReclaimAsCollectionOwner() public {
        if (IOwnable(address(pair)).owner() != address(settings)) return;
```

```
        (bool success, bytes memory data) =
address(this).call(abi.encodeWithSignature("reclaimAsCollectionOwner"));

        require(success, "Failed to reclaim the pair as the collection owner and
previous pair owner");
    }
```

Recommendation

We suggest fixing the code as below:

```
// if not an authorized address, revert
        if (!pairFactory.authAllowedForToken(address(pair.nft()),
msg.sender)) {
            if (msg.sender == pairInfo.prevOwner) {
                // If previous owner, verify that the current time is past
the unlock time
                require(block.timestamp > pairInfo.unlockTime, "Lockup not
over");
            }
            revert("Not prev owner or collection admin");
        }
```

Remediation

This issue has been acknowledged by Sudorandom Labs and fixed at commit 93282a1ca74723a228834a2eaf1cc8a95698bc5e.

8. Incorrect NFT balance check in LSSVMPairERC721::_takeNFTsFromSender

Severity: **Low** Difficulty: **High**
File(s): LSSVMPairERC721.sol

Description

If more than 1 NFT is being transferred, the _takeNFTsFromSender function does a balance check:

Code 10 https://github.com/sudoswap/lssvm2/blob/main/src/erc721/LSSVMPairERC721.sol#L236

```
    // Call router to pull NFTs
    // If more than 1 NFT is being transfered, we can do a balance check instead of
an ownership check, as pools are indifferent between NFTs from the same collection
    if (numNFTs > 1) {
        uint256 beforeBalance = _nft.balanceOf(_assetRecipient);
        for (uint256 i = 0; i < numNFTs;) {
            router.pairTransferNFTFrom(_nft, routerCaller, _assetRecipient,
nftIds[i], pairVariant());

            unchecked {
                ++i;
            }
        }
        require((_nft.balanceOf(_assetRecipient) - beforeBalance) == numNFTs, "NFTs
not transferred");
    } else {
        router.pairTransferNFTFrom(_nft, routerCaller, _assetRecipient, nftIds[0],
pairVariant());
        require(_nft.ownerOf(nftIds[0]) == _assetRecipient, "NFT not transferred");
    }
```

However, if the pair has configured a property check which only accepts particular NFTs, the balance check will become invalid.

Impact

The balance check cannot guarantee that all the required NFTs have been pull by the router.

Failed Invariant

```
    function invariantNFTSent() public {
```

```
        // our NFTs should have been sent
        for (uint i = 0; i < pnmLogs.nftSent.length; ++i)
            require(test721.ownerOf(pnmLogs.nftSent[i]) != user, "nft wasn't
sent!");
        delete pnmLogs.nftSent;
    }
```

Recommendation

Add the condition check: `propertyChecker() != address(0)`. If the condition is true and more than 1 NFT is being transferred, the function needs to check whether each NFT is received by ID.

Remediation

This issue has been acknowledged by Sudorandom Labs and fixed at commit 6d23976a16463010ceb3b06944857310683e8adc.

9. nftRecipient and tokenRecipient are not checked to ensure assets don’t go to the pool

Severity: **Low**
File(s): LSSVMPairERC721.sol, LSSVMPairERC1155.sol

Difficulty: **Low**

Description

When `nftRecipient` or `tokenRecipient` is set to the pair’s address, the user will send NFTs or tokens to the pair using the swap function but will not receive anything in return.

Code 11 https://github.com/sudoswap/lssvm2/blob/main/src/erc721/LSSVMPairERC721.sol#L199

```
    function _sendSpecificNFTsToRecipient(IERC721 _nft, address nftRecipient,
uint256[] calldata nftIds)
        internal
        virtual
    {
        // Send NFTs to recipient
        uint256 numNFTs = nftIds.length;
        for (uint256 i; i < numNFTs;) {
            _nft.transferFrom(address(this), nftRecipient, nftIds[i]);

            unchecked {
                ++i;
            }
        }
    }
```

Code 12 https://github.com/sudoswap/lssvm2/blob/main/src/erc1155/LSSVMPairERC1155.sol#L162

```
    function _sendAnyNFTsToRecipient(IERC1155 _nft, address nftRecipient, uint256
numNFTs) internal virtual {
        _nft.safeTransferFrom(address(this), nftRecipient, nftId(), numNFTs,
bytes(""));
    }
```

Code 13 https://github.com/sudoswap/lssvm2/blob/main/src/LSSVMPairERC20.sol#L135

```
   function _sendTokenOutput(address payable tokenRecipient, uint256 outputAmount)
internal override {
        // Send tokens to caller
        if (outputAmount != 0) {
            token().safeTransfer(tokenRecipient, outputAmount);
```

```
        }
    }
```

Code 14 https://github.com/sudoswap/lssvm2/blob/main/src/LSSVMPairETH.sol#L95

```
    function _sendTokenOutput(address payable tokenRecipient, uint256 outputAmount)
internal override {
        // Send ETH to caller
        if (outputAmount != 0) {
            tokenRecipient.safeTransferETH(outputAmount);
        }
    }
```

Impact

A user that mistakenly put the pair's address will have its swap succeed without getting anything in return.

Failed Invariant

```
function invariantRecipientNotPair() public {
       for (uint i = 0; i < pnmLogs.length; ++i) {
           LogInfo memory log = pnmLogs[i];
           if (log.isBuy) {
               require(log.newNFTBalance == log.oldNFTBalance + 1, "didn't
receive any NFT from swap");
           }
       }
   }
```

Recommendation

Add the condition check to make sure that neither `nftRecipient` nor `tokenRecipient` is the pair's address.

Remediation

This issue has been acknowledged by Sudorandom Labs. Sudorandom Labs will mitigate the issue by cautioning the swap callers to be careful who they specify as recipients.

Fix Log

Table 6 Fix Log

ID	Title	Severity	Status
01	Double trade fee on buys conflicts with royalties	**Medium**	**Fixed at commit 1ec3e31a4b2600fb2b32286156270e11ebb1ae80**
02	Duplicate signature of withdrawAllETH can be used to steal assets from pairs stealthily	**Medium**	**Fixed at commit 184693b25d3b82e433aebf4b8e8fb5f7bc8bdf17**
03	Ignorance of different royalty settings for different NFT IDs	**Medium**	**Acknowledged***
04	LSSVMPair::call should not allow the call target to be the current NFT	**Low**	**Fixed at commit 60d0f068d2eb759feb86cb9b7c072e496f347967**
05	Royalty calculation error	**Low**	**Acknowledged***
06	Wrong NFTDeposit event can be emitted	**Low**	**Fixed at commit 836181d6d913d86e65adc08d0e7e8528b4a9dc47**
07	Collection owner can be rejected in reclaimPair	**Low**	**Fixed at commit 93282a1ca74723a228834a2eaf1cc8a95698bc5e**
08	Incorrect NFT balance check in LSSVMPairERC721::_takeNFTsFromSender	**Low**	**Fixed at commit 6d23976a16463010ceb3b06944857310683e8adc**
09	nftRecipient and tokenRecipient are not checked to ensure assets don't go to the pool	**Low**	**Acknowledged and mitigated**

* The issues have been acknowledged by Sudorandom Labs but won't be fully fixed or mitigated at the time of this report. Check the former description of each issue for details.

Appendix

Severity Categories

Severity	Description
Undetermined	The extent of the risk can not be determined during this assessment
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or Defense in Depth
Low	The risk is relatively small or is not a risk that the customer indicated as important
Medium	Individual user's information is at risk, exploitation would be bad for client's reputation, moderate financial impact, possibly legal implication for client
High	Affects large number of users, very bad for clients reputation or posses great financial or legal risk

Difficulty Levels

Difficulty	Description
Undetermined	The difficulty of the exploit is undetermined during this assessment
Low	Commonly exploited, existing tools can be leveraged or can be easily scripted
Medium	Attacker must write a dedicated exploit, or need in depth knowledge of a complex system
High	The attacker must have privileged insider access or need to know extremely complicated technical details or must discover other issues to exploit this