Sudoswap Security Review

SPEARBIT

Reviewers
Gerard Persoon, Lead Security Researcher
Mudit Gupta, Lead Security Researcher
Max Goodman, Security Researcher

January 31st, 2022

1 Executive Summary

The Spearbit security researchers reviewed the Sudoswap lssvm contracts and have discovered numerous high, medium, and low risk issues. Using a combination of manual review and static analysis tools, we identified 1 critical risk issue, 1 high risk issue, 7 medium risk issues, and 9 low risk issues. In addition to these issues, we've included 9 informational level issues and 7 gas optimizations for the Sudoswap Team's consideration.

Overall, we recommend the Sudoswap team review these findings in order of severity (high to low) and pursue fixes, according to their specific remediations within this report.

Repository	**Commit**
sudoswap/lssvm	5dbe06188a53ef8e6dfd70d3179411ec08bd8d64

Summary

Type of Project	DeFi
Timeline	Jan 24th, 2022 - Jan 31st, 2022
Methods	Computer-Aided Verification, Manual Review
Documentation	Medium
Testing Coverage	Medium

Total Issues

Critical Risk	1
High Risk	1
Medium Risk	7
Low Risk	9
Gas Optimizations	7
Informational	9

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of Sudoswap according to the specific commit by a three person team. Any modifications to the code will require a new security review.

Contents

1 Executive Summary 1

2 Spearbit 3

3 Findings 3

3.1 Critical Risk . . . 3

3.1.1 Clones with malicious `extradata` are also considered valid clones . . . 3

3.2 High Risk . . . 5

3.2.1 Factory Owner can steal user funds approved to the Router 5

3.3 Medium Risk . . . 6

3.3.1 Missing check in the number of Received Tokens when tokens are transferred directly . . . 6

3.3.2 Malicious `assetRecipient` could get an unfair amount of tokens . . . 7

3.3.3 Malicious Router can exploit `cacheAssetRecipientNFT-Balance` to drain pair funds . . . 8

3.3.4 Malicious Router can steal NFTs via Re-Entrancy attack . 9

3.3.5 `getAllHeldIds()` of `LSSVMPairMissingEnumerable` is vulnerable to a denial of service attack . . . 10

3.3.6 With NFT pools the protocol fees end up in `assetRecipi-ent` instead of `_factory` . . . 11

3.3.7 Error codes of `Quote` functions are unchecked . . . 15

3.4 Low Risk . . . 16

3.4.1 Swaps can be front run by `Pair` Owner to extract any profit from slippage allowance . . . 16

3.4.2 Add check for `numItems == 0` . . . 16

3.4.3 Disallow arbitrary function calls to `LSSVMPairETH` . . . 18

3.4.4 Only transfer relevant funds for `PoolType` . . . 18

3.4.5 Check for `0` parameters . . . 19

3.4.6 Potentially undetected underflow In assembly . . . 20

3.4.7 Check number of NFTs is not `0` . . . 21

3.4.8 Avoid utilizing inside knowledge of functions . . . 23

3.4.9 Add Reentrancy Guards . . . 25

3.5 Gas Optimization . . . 27

3.5.1 Saving 1 byte off the `constructor()` code . . . 27

3.5.2 Decode `extradata` in `calldata` in one go . . . 28

3.5.3 Transfer last NFT instead of first . . . 28

3.5.4 Simplify the connection between `Pair` and `Router` 31
3.5.5 Cache array length 33
3.5.6 Use Custom Errors 35
3.5.7 Alternatives for the `immutable` Proxy variables 36
3.6 Informational 38
3.6.1 Pair implementations may not be Proxies 38
3.6.2 `NFT` and `Token` Pools can be signed orders instead 38
3.6.3 Remove Code Duplication 39
3.6.4 Unclear Function Name 39
3.6.5 Inaccurate Message About `MAX_FEE` 40
3.6.6 Inaccurate comment for `assetRecipientNFTBalanceAtTransferStart` 41
3.6.7 `IERC1155` not utilized 42
3.6.8 Use Fractions 42
3.6.9 Two families of token libraries used 43

2 Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem.

Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Find more about us on Spearbit.com.

3 Findings

3.1 Critical Risk

3.1.1 Clones with malicious `extradata` are also considered valid clones

Severity: Critical Risk

Context: `LSSVMPairCloner.sol#L121`, `LSSVMPair.sol#L687-695`, `LSSVMRouter.sol#L574-594`, `LSSVMPairFactory.sol#L223-257`, `LSSVMPairCloner.sol#L206-232`

Description: Spearbit discovered that the functions verifying if a contract is a pair do so by only checking the first 54 bytes (i.e. the `Proxy` code). An attacker could deploy a contract that starts with the first 54 bytes of proxy code but have a malicious payload, and these functions will still verify it as a legitimate clone. We have found this to be a critical issue based on the feasibility of a potential exploit.

Consider the following scenario:

1. An attacker creates a malicious pair by making a copy of the source of `cloneETHPair()` supplying malicious values for `factory`, `bondingCurve`, `nft` and `poolType` using a valid template for the connected contract.
2. The attacker has a contract with valid `proxy` code, connected to a valid template, but the rest of the parameters are invalid.
3. The `Pair` is initialized via a copy of `initialize()` of `LSSVMPair`, which calls `__Ownable_init()` to set a malicious owner.

4. The malicious owner calls `call()`, with `target` equal to the router contract and the `calldata` for the function `pairTransferERC20From()`:

```
// Owner is set by pair creator
function call(address payable target, bytes calldata data) external  onlyOwner
↪  {
    // Factory is malicious
    LSSVMPairFactoryLike _factory = factory();
    // `callAllowed()` is malicious and returns true
    require(_factory.callAllowed(target), "Target must be whitelisted");
    (bool result, ) = target.call{value: 0}(data);
    require(result, "Call failed");
}
```

5. The check for `onlyOwner` and `require` pass, therefore `pairTransferERC20From()` is called with the malicious `Pair` as `msg.sender`.

6. The router checks if it is called from a valid pair via `isPair()`:

```
function pairTransferERC20From(...) external {
    // Verify caller is a trusted pair contract
    // The malicious pair passed this test
    require(factory.isPair(msg.sender, variant), "Not pair");
    ...
    token.safeTransferFrom(from, to, amount);
}
```

7. Because the function `isPair()` only checks the first 54 bytes (the *runtime* code including the implementation address), `isPair()` does not check for extra parameters `factory`, `bondingCurve`, `nft` or `poolType`:

```
function isPair(address potentialPair, PairVariant variant) ... {
    ...
    } else if (variant == PairVariant.ENUMERABLE_ETH) {
        return
 ↪  LSSVMPairCloner.isETHPairClone(address(enumerableETHTemplate),potentialPair);
    }
    ...
}
function isETHPairClone(address implementation, address query) ... {
    ...
    // Compare expected bytecode with that of the queried contract
    let other := add(ptr, 0x40)
    extcodecopy(query, other, 0, 0x36)
    result := and(
        eq(mload(ptr), mload(other)),
        // Checks 32 + 22 = 54 bytes
        eq(mload(add(ptr, 0x16)), mload(add(other, 0x16)))
    )
}
```

8. Now the malicious pair is considered valid, the `require` statement in `pairTransferERC20From()` has passed and tokens can be transferred to the attacker from anyone who has set an allowance for the router.

Recommendation: Spearbit recommends Sudoswap to verify more values when checking if a pair is valid - especially the `factory` value. We also recommend to consider the removal of all trust between pairs and routers, as well as the function `call()`.

Sudoswap: Added factory check to `isPair` functions here.

Spearbit: Acknowledged. Please double-check with the changes for the finding "Saving 1 byte Off The Constructor() Code" - especially the amount of bytes checked at the end of `isETHPairClone()` and `isERC20PairClone()`.

3.2 High Risk

3.2.1 Factory Owner can steal user funds approved to the Router

Severity: High Risk

Context: `LSSVMPair.sol#L687-695`, `LSSVMRouter.sol#L574`

Description: A pair owner can make arbitrary calls to any contract that has been approved by the factory owner. The code in the factory intends to prevent router contracts from being approved for calls because router contracts can have access to user funds. An example includes the `pairTransferERC20From()` function, that can be used to steal funds from any account which has given it approval.

The router contracts can nevertheless be whitelisted by first being removed as a router and then being whitelisted. This way anyone can deploy a pair and use the `call` function to steal user funds.

Recommendation: Spearbit recommends Sudoswap to consider changing the architecture such that the router simply sends the NFTs to the pair when it calls the swap function. If you want to remove the trust from the router, make the pair store reserve balances and check tokens received against it.

Sudoswap: The immediate issue of adding/removing routers are addressed in this branch here. Every time a new router is added or removed, we only toggle the allowed flag, while `wasEverAllowed` is always true. `LSSVMPair.call()` now checks if we've ever approved a `Router`. The broader issue of factory owner being able to potentially to steal pool funds is acknowledged, with other specific vectors mentioned in the audit addressed in other branches.

Spearbit: Acknowledged.

3.3 Medium Risk

3.3.1 Missing check in the number of Received Tokens when tokens are transferred directly

Severity: Medium Risk

Context: `LSSVM\contracts`, `LSSVMPairERC20.sol#L41-78`

Description: Within the function `_validateTokenInput()` of `LSSVMPairERC20`, two methods exist to transfer tokens. In the first method via `router.pairTransferERC20From()` a check is performed on the number of received tokens. In the second method no checks are done.

Recent hacks (e.g. Qubit finance) have successfully exploited `safeTransferFrom()` functions which did not revert nor transfer tokens. Additionally, with malicious or re-balancing tokens the number of transferred tokens might be different from the amount requested to be transferred.

```
function _validateTokenInput(...) ... {
    ...
    if (isRouter) {
        ...
        // Call router to transfer tokens from user
        uint256 beforeBalance = _token.balanceOf(_assetRecipient);
        router.pairTransferERC20From(...)
        // Verify token transfer (protect pair against malicious router)
        require( _token.balanceOf(_assetRecipient) - beforeBalance ==
↪   inputAmount, "ERC20 not transferred in");
    } else {
        // Transfer tokens directly
        _token.safeTransferFrom(msg.sender, _assetRecipient, inputAmount);
    }
}
```

Recommendation: Spearbit recommends Sudoswap to verify the number of tokens received when these are transferred directly.

Sudoswap: Risks acknowledged but no changes at this time. `Pair` creators would have to willingly create deploy an `NFT/Token` pair for a Token using non-standard `ERC20` token behavior to be at risk.

Spearbit: Acknowledged.

3.3.2 Malicious `assetRecipient` could get an unfair amount of tokens

Severity: Medium Risk

Context: `LSSVMRouter.sol#L754-789`

Description: The function `_swapNFTsForToken()` of `LSSVMRouter` calls `safeTransferFrom()`, which then calls `ERC721Received` of `assetRecipient`. A malicious `assetRecipient` could manipulate its NFT balance by buying additional NFTs via the `Pair` and sending or selling them back to the `Pair`, enabling the malicious actor to obtain an unfair amount of tokens via `routerSwapNFTsForToken()`.

```
function _swapNFTsForToken(...) ... {
    ...
    swapList[i].pair.cacheAssetRecipientNFTBalance();
    ...
        for (uint256 j = 0; j < swapList[i].nftIds.length; j++) {
↪   nft.safeTransferFrom(msg.sender,assetRecipient,swapList[i].nftIds[j]); //
↪   call to onERC721Received of assetRecipient
        }
        ...
        outputAmount +=
↪   swapList[i].pair.routerSwapNFTsForToken(tokenRecipient); // checks the
↪   token balance of assetRecipient
    }
}
```

Recommendation: Spearbit recommends Sudoswap to implement re-entrancy modifiers (*see finding "Add Reentrancy Guards" in the "Low Risk" section of this report*). We also recommend that Sudoswap implements the recommendation "Simplify the Connection Between `Pair` and `Router`" in the "Gas Optimizations" section of this report to reduce the attack surface. Finally, we recommend that Sudoswap make sure the `assetRecipient` is trusted.

Sudoswap: We've also accepted the recommendation to simplify the connection between the Router and the Pair, provided fix is in the issue here.

Spearbit: Acknowledged.

3.3.3 Malicious Router can exploit `cacheAssetRecipientNFTBalance` to drain pair funds

Severity: Medium Risk

Context: `LSSVMPair.sol#L371-379`, `LSSVMPair.sol#L318-366`

Description: A malicious router could be whitelisted by an inattentive or a malicious factory owner and drain pair funds in the following exploit scenario:

1. Call the `cache` function. Suppose that the current balance is 10, so it gets cached.
2. Sell 5 NFTs to the pair and get paid using `swapNFTsForToken`. Total balance is now 15 but the cached balance is still 10.
3. Call `routerSwapNFTsForToken`. This function will compute `total_balance`

- cached_balance, assume 5 NFTs have been sent to it and pay the user. However, no new NFTs have been sent and it already paid for them in Step 2.

Recommendation: Spearbit recommends Sudoswap to implement reentrancy modifiers (*see finding "Add Reentrancy Guards" in the "Low Risk" section of this report*). We also suggest implementing recommendations in the "Simplify the Connection between `Pair` and `Router`" found in the "Gas Optimization" section of this report to reduce attack surface.

Sudoswap: Removed this flow through the implementation of the recommendation here.

Spearbit: Acknowledged.

3.3.4 Malicious Router can steal NFTs via Re-Entrancy attack

Severity: Medium Risk

Context: `LSSVMPair.sol`, `LSSVMPairERC20.sol`

Description: If the factory owner approves a malicious `_router`, it is possible for the malicious router to call functions like `swapTokenForAnyNFTs()` and set `isRouter` to true. Once that function reaches `router.pairTransferERC20From()` in `_validateTokenInput()`, they can re-enter the pair from the `router` and call `swapTokenForAnyNFTs()` again.

This second time the function reaches `router.pairTransferERC20From()`, allowing the malicious router to execute a token transfer so that the `require` of `_validateTokenInput` is satisfied when the context returns to the pair. When the context returns from the reentrant call back to the original call, the `require` of `_validateTokenInput` would still pass because the balance was cached before the reentrant call. Therefore, an attacker will receive 2 NFTs by sending tokens only once.

Recommendation: Spearbit recommends Sudoswap to implement reentrancy modifiers (*see finding "Add Reentrancy Guards" in the "Low Risk" section of this report*). Sudoswap should also consider checking if the NFT balance before and after `router.pairTransferERC20From()` is the same. Finally, we recommend to make sure the following contracts and addresses are trusted: the NFT contract, the ERC-20 tokens, the `assetRecipient`, the bonding curve, the factory, the factory owner, and the `protocolFeeRecipient`.

Sudoswap: The immediate issue is addressed in this branch. We now validate NFT balances after the `router.pairTransferERC20From` call to mitigate re-entrant balance changes by a malicious router. The broader issue of the `cache` function being exploitable is addressed in the GitHub Issue about simplifying the connection between the pair and the router.

Spearbit: Acknowledged.

3.3.5 `getAllHeldIds()` of `LSSVMPairMissingEnumerable` is vulnerable to a denial of service attack

Severity: Medium Risk

Context: `LSSVMPairMissingEnumerable.sol#L90-97`, `LSSVMPair.sol#L125`

Description: The contract `LSSVMPairMissingEnumerable` tries to compensate for NFT contracts that do not have `ERC721Enumerable` implemented. However, this cannot be done for everything as it is possible to use `transferFrom()` to send an NFT from the same collection to the `Pair`. In that case the callback `onERC721Received()` will not be triggered and the `idSet` administration of `LSSVMPairMissingEnumerable` will not be updated. This means that `nft().balanceOf(address(this));` can be different from the elements in `idSet`. Assuming an actor accidentally, or on purpose, uses `transferFrom()` to send additional NFTs to the `Pair`, `getAllHeldIds()` will fail as `idSet.at(i)` for unregistered NFTs will fail. This can be used in a griefing attack.

`getAllHeldIds()` in `LSSVMPairMissingEnumerable`:

```
function getAllHeldIds() external view override returns (uint256[] memory) {
    uint256 numNFTs = nft().balanceOf(address(this)); // returns the registered
↪   + unregistered NFTs
    uint256[] memory ids = new uint256[](numNFTs);
    for (uint256 i; i < numNFTs; i++) {
        ids[i] = idSet.at(i); // will fail at the unregistered NFTs
    }
    return ids;
}
```

The following checks performed with `_nft.balanceOf()` might not be accurate in combination with `LSSVMPairMissingEnumerable`. Risk is low because any additional NFTs making later calls to `_sendAnyNFTsToRecipient()` and `_sendSpecificNFTsToRecipient()` will fail. However, this might make it more difficult to troubleshoot issues.

```
function swapTokenForAnyNFTs(...) .. {
    ...
    require((numNFTs > 0) && (numNFTs <= _nft.balanceOf(address(this))),"Ask
 ↪  for > 0 and <= balanceOf NFTs");
    ...
    _sendAnyNFTsToRecipient(_nft, nftRecipient, numNFTs); // could fail
    ...
}
function swapTokenForSpecificNFTs(...) ... {
    ...
    require((nftIds.length > 0) && (nftIds.length <=
 ↪  _nft.balanceOf(address(this))),"Must ask for > 0 and < balanceOf NFTs"); //
 ↪  '<' should be '<='
    ...
    _sendSpecificNFTsToRecipient(_nft, nftRecipient, nftIds); // could fail
    ...
}
```

Note: The error string `< balanceOf NFTs` is not accurate.

Recommendation: Spearbit recommends Sudoswap to use `idSet.length()` in order to determine the number of NFTs by changing the code in accordance with the following `diff`:

```
- uint256 numNFTs = nft().balanceOf(address(this));
+ uint256 numNFTs = idSet.length();
```

To access `idSet.length()` from `LSSVMPair`, an extra function is necessary in `LSSVMPairMissingEnumerable.sol` and `LSSVMPairEnumerable.sol`.

Spearbit also suggests to fix the error string in `swapTokenForSpecificNFTs`.

Sudoswap: Addressed in this branch here. `idSet` size is now used instead of `NFT balanceOf`.

Spearbit: Acknowledged.

3.3.6 With NFT pools the protocol fees end up in `assetRecipient` instead of `_factory`

Severity: Medium Risk

Context: `LSSVMPair.sol#L192-245`, `LSSVMPairERC20.sol#L90-105`, `LSSVMPairETH.sol#L53-66`

Description: Assume a scenario where an NFT pool with `assetRecipient` set have the received funds sent directly to the `assetRecipient`. Now, suppose a user executes the `swapTokenForSpecificNFTs()`.

The function `_validateTokenInput()` sends the required input funds, including fees to the `assetRecipient`. The function `_payProtocolFee()` tries to send the fees to the `_factory`. However, this function attempts to do so from the pair contract. The pair contract does not have any funds because they have been sent directly to the `assetRecipient`. So following this action the `payProtocolFee()` lowers the fees to `0` and sends this number to the `_factory` while fees end up at `assetRecipient'` instead of at the `_factory`.

The fees then end up at `assetRecipient` instead of at the `_factory`.

Note:

- The same issue occurs in `swapTokenForAnyNFTs()`.
- This issue occurs with both `ETH` and `ERC20` NFT Pools, although their logic is slightly different.
- This issue occurs both when `swapTokenForSpecificNFTs()` is called directly as well as indirectly via the `LSSVMRouter`.
- Although the pool fees are `0` with NFT pools, the factory fee is still present.
- Luckily, `TRADE` pools cannot have an `assetRecipient` as this would also create issues.

```
abstract contract LSSVMPair is Ownable, ReentrancyGuard {
    ...
    function swapTokenForSpecificNFTs(...) external payable virtual returns
↪   (uint256 inputAmount) {
        ...
        _validateTokenInput(inputAmount, isRouter, routerCaller, _factory); //
↪   sends inputAmount to assetRecipient
        _sendSpecificNFTsToRecipient(_nft, nftRecipient, nftIds);
        _refundTokenToSender(inputAmount);
        _payProtocolFee(_factory, protocolFee);
        ...
    }
}

abstract contract LSSVMPairERC20 is LSSVMPair {
    ...
    function _payProtocolFee(LSSVMPairFactoryLike _factory, uint256
↪   protocolFee) internal override {
        ...
        uint256 pairTokenBalance = _token.balanceOf(address(this));
        if (protocolFee > pairTokenBalance) {
            protocolFee = pairTokenBalance;
        }
        _token.safeTransfer(address(_factory), protocolFee);  // tries to send
↪   from the Pair contract
    }
}

abstract contract LSSVMPairETH is LSSVMPair {
    function _payProtocolFee(LSSVMPairFactoryLike _factory, uint256
↪   protocolFee) internal override {
        ...
        uint256 pairETHBalance = address(this).balance;
        if (protocolFee > pairETHBalance) {
            protocolFee = pairETHBalance;
        }
        payable(address(_factory)).safeTransferETH(protocolFee);  // tries to
↪   send from the Pair contract
    }
}
```

Recommendation: Spearbit recommends Sudoswap to first be aware of the following:

- In `ETH` NFT pools, `ETH` is first sent to the `Pair` contract, then to other parties.

- In ERC20 NFT pools, ERC20 tokens are sent directly from the original caller to other parties. This means that when an assetRecipient is set, the ERC20 does not touch the pair.

Second, we recommend that Sudoswap change swapTokenForSpecificNFTs() as such:

```
function swapTokenForSpecificNFTs(...) external payable virtual returns
 ↪  (uint256 inputAmount) {
    ...
-   _validateTokenInput(inputAmount, isRouter, routerCaller, _factory);
+   _validateTokenInput(inputAmount - protocolFee, isRouter, routerCaller,
 ↪  _factory);
    ...
-   _payProtocolFee(_factory, protocolFee);
+   _payProtocolFeeByOriginalCaller(_factory, protocolFee, isRouter,
 ↪  routerCaller);
}
```

We also recommend that Sudoswap perform the same update for swapToken-ForAnyNFTs().

Third, we recommend creating a new _payProtocolFeeByOriginalCaller() in LSSVMPairERC20() that combines the functionality of _payProtocolFee() and _validateTokenInput(). This is due to the assumption that the function should also be able to retrieve the ERC20 tokens from the original caller.

Note: _payProtocolFeeByOriginalCaller in LSSVMPairETH() can just do the same as _payProtocolFee().

Lastly, it may be wise for Sudoswap to consider renaming _payProtocolFee() to _payProtocolFeeFromPair(), clarifying the difference.

Note: The functionality of _payProtocolFeeFromPair() is still necessary for routerSwapNFTsForToken() and swapNFTsForToken() because in that case, the funds are indeed in the Pair.

Sudoswap: Addressed in the branch here. Pulling tokens and taking the protocol fee is now done in the same step when swapping from tokens to NFTs so there should always be tokens to send for the fee. The original _payProtocolFee function has been renamed _payProtocolFeeFromPair as suggested.

Spearbit: Acknowledged.

3.3.7 Error codes of `Quote` functions are unchecked

Severity: Medium Risk

Context: LSSVMPair.sol#L389-431, LSSVMRouter.sol

Description: The error return values from functions `getBuyNFTQuote()` and `getSellNFTQuote()` are not checked in contract `LSSVMRouter.sol`, whereas other functions in contract `LSSVMPair.sol` do check for `error==CurveErrorCodes.Error.OK`.

```
abstract contract LSSVMPair is Ownable, ReentrancyGuard {
    ...
    function getBuyNFTQuote(uint256 numNFTs) external view returns
 ↪  (CurveErrorCodes.Error error, ...) {
        (error, ...) = bondingCurve().getBuyInfo(...);
    }
    function getSellNFTQuote(uint256 numNFTs) external view returns
 ↪  (CurveErrorCodes.Error error, ...) {
        (error, ...) = bondingCurve().getSellInfo(...);
    }
    function swapTokenForAnyNFTs(...)  external payable virtual returns
 ↪  (uint256 inputAmount) {
        ...
        (error, ...) = _bondingCurve.getBuyInfo(...);
        require(error == CurveErrorCodes.Error.OK, "Bonding curve error");
        ...
    }
}
```

LSSVMRouter.sol#L526

```
(, , pairOutput, ) = swapList[i].pair.getSellNFTQuote(...);
```

The following contract lines contain the same code snippet below: LSSVMRouter.sol#L360, LSSVMRouter.sol#L407, LSSVMRouter.sol#L450, LSSVMRouter.sol#L493, LSSVMRouter.sol#L627, LSSVMRouter.sol#L664

```
(, , pairCost, ) = swapList[i].pair.getBuyNFTQuote(...);
```

Note: The current Curve contracts, which implement the `getBuyNFTQuote()` and `getSellNFTQuote()` functions, have a limited number of potential errors. However, future Curve contracts might add additional error codes.

Recommendation: Check the error code of functions `getBuyNFTQuote()` and `getSellNFTQuote()` in contract `LSSVMRouter.sol`.

Sudoswap: Addressed in this branch here. `LSSVMRouter` now reverts if the Error is not `Error.OK` for a normal swap, or it skips performing the swap during a robust swap operation.

Spearbit: Acknowledged.

3.4 Low Risk

3.4.1 Swaps can be front run by `Pair` Owner to extract any profit from slippage allowance

Severity: Low Risk

Context: `LSSVMPair.sol#L630`, `LSSVMPair.sol#L644`, `LSSVMPair.sol#L660`

Description: If the user adds a nonzero slippage allowance, the pair owner can front run the swap to increase the fee/spot price and steal all of the slippage allowance. This basically makes sandwich attacks much easier and cheaper to execute for the pair owner.

Recommendation: Spearbit recommends that Sudoswap add a time delay of a few hours between pair owner submitting a new price/fee/delta and the execution of such new changes coming into effect. If this is not ideal for Sudoswap, Spearbit alternatively recommends to disallow the change of these parameters and requiring the pair owner to deploy a new pair instead.

Sudoswap: Acknowledged, but no changes have been made to the pricing model at this time. Still talking interally about what sort of time delay would be acceptable for a `spotPrice` change and how to change pricing logic if so.

Spearbit: Acknowledged.

3.4.2 Add check for `numItems == 0`

Severity: Low Risk

Context: `LinearCurve.sol#L38-58`, `ExponentialCurve.sol#L45-65`, `LinearCurve.sol#L100-120`, `ExponentialCurve.sol#L108-129`

Description: Functions `getBuyInfo()` and `getSellInfo()` in `LinearCurve.sol` check that `numItems != 0`. However, the same `getBuyInfo()` and `getSellInfo()` functions in `ExponentialCurve.sol` do not perform this check.

```
contract LinearCurve is ICurve, CurveErrorCodes {
    function getBuyInfo(...) ... {
        // We only calculate changes for buying 1 or more NFTs
        if (numItems == 0) {
            return (Error.INVALID_NUMITEMS, 0, 0, 0);
        }
        ...
    }
    function getSellInfo(...) ... {
        // We only calculate changes for selling 1 or more NFTs
        if (numItems == 0) {
            return (Error.INVALID_NUMITEMS, 0, 0, 0);
        }
        ...
    }
}
```

```
contract ExponentialCurve is ICurve, CurveErrorCodes {
    function getBuyInfo(...) ... {
        // No check on `numItems`
        uint256 deltaPowN = delta.fpow(numItems, FixedPointMathLib.WAD);
        ...
    }
    function getSellInfo(... ) ... {
        // No check on `numItems`
        uint256 invDelta =
 ↪  FixedPointMathLib.WAD.fdiv(delta,FixedPointMathLib.WAD);
        ...
    }
}
```

If the code remains unchanged, an erroneous situation may not be caught and funds might be sent when selling `0` NFTs.

Luckily, when `numItems == 0` then result `outputValue` of the functions in `ExponentialCurve` is still `0`, so there is no real issue. However, it is still important to fix this because a derived version of these functions might be used by future developers.

Recommendation: Spearbit recommends adding a check for `numItems == 0` to `getBuyInfo()` and `getSellInfo()` of `ExponentialCurve.sol`.

Sudoswap: Addressed in branch here.

Spearbit: Acknowledged.

3.4.3 Disallow arbitrary function calls to `LSSVMPairETH`

Severity: Low Risk

Context: `LSSVMPairETH.sol#L133-139`

Description: The contract `LSSVMPairETH` contains an open `fallback()` function. The `fallback()` is most likely necessary because the `proxy` adds `calldata` and the `receive()` function, therefore not receiving the `ETH`. However, without additional checks any function call to an `ETH Pair` will succeed. This could result in unforseen scenarios which hackers could potentially exploit.

```
fallback() external payable {
    emit TokenDeposited(msg.value);
}
```

Recommendation: Spearbit recommends adapting the `fallback()` function in the following way to ameliorate this risk:

```
fallback() external payable {
+   require (msg.data.length == _immutableParamsLength()); // only allow calls
 ↪  without function selector
    emit TokenDeposited(msg.value);
}
```

Sudoswap: Addressed here.

Spearbit: Acknowledged.

3.4.4 Only transfer relevant funds for `PoolType`

Severity: Low Risk

Context: `LSSVMPairFactory.sol#L363-416`

Description: The functions `_initializePairETH()` and `_initializePairERC20()` allow for the transfer of `ETH/ERC20` and `NFTs` even when this is not relevant for the `PoolType`. Although funds can be rescued from the `Pair`, it is perhaps better to prevent these types of mistakes.

```
function _initializePairETH(...) ... {
    ...
    // Transfer initial `ETH` to `pair`
    // Only relevant for `PoolType.TOKEN` or `PoolType.TRADE`
    payable(address(_pair)).safeTransferETH(msg.value);
    ...
    // Transfer initial `NFT`s from `sender` to `pair`
    for (uint256 i = 0; i < _initialNFTIDs.length; i++) {
        // Only relevant for PoolType.NFT or PoolType.TRADE
        _nft.safeTransferFrom(msg.sender,address(_pair),_initialNFTIDs[i]);
    }
}
function _initializePairERC20(...) ... {
    ...
    // Transfer initial tokens to pair
    // Only relevant for PoolType.TOKEN or PoolType.TRADE
    _token.safeTransferFrom(msg.sender,address(_pair),_initialTokenBalance);
    ...
    // Transfer initial NFTs from sender to pair
    for (uint256 i = 0; i < _initialNFTIDs.length; i++) {
        // Only relevant for PoolType.NFT or PoolType.TRADE
        _nft.safeTransferFrom(msg.sender,address(_pair),_initialNFTIDs[i]);
    }
}
```

Recommendation: Spearbit recommends Sudoswap to only transfer the `ETH/ERC20/NFTs` that are relevant for the `Pair PoolType`.

Sudoswap: Acknowledged, but no change for now. Clients will be responsible for ensuring users deposit the correct assets, and the existence of rescue functions makes it possible to correct.

Spearbit: Acknowledged.

3.4.5 Check for `0` parameters

Severity: Low Risk

Context: `LSSVMPairFactory.sol#L291-356`

Description: Functions `setCallAllowed()` and `setBondingCurveAllowed()` do not check that `target != 0` while the comparable function `setRouterAllowed()` does check for `_router != 0`.

```
function setCallAllowed(address payable target, bool isAllowed) external
 ↪  onlyOwner {
    ... // No check on target
    callAllowed[target] = isAllowed;
}
function setBondingCurveAllowed(ICurve bondingCurve, bool isAllowed) external
 ↪  onlyOwner {
    ...  // No check on bondingCurve
    bondingCurveAllowed[bondingCurve] = isAllowed;
}
function setRouterAllowed(LSSVMRouter _router, bool isAllowed) external
 ↪  onlyOwner {
    require(address(_router) != address(0), "0 router address");
    ...
    routerAllowed[_router] = isAllowed;
}
```

Recommendation: Spearbit recommends Sudoswap to consider adding a check for `0` parameters in `setCallAllowed()` and `setBondingCurveAllowed()`. If `0` checks are considered to be unnecessary because these functions are protected by `onlyOwner`, then the `0` check could be removed from `setRouterAllowed()`.

Sudoswap: Removed zero address check in `setRouterAllowed` for consistency here.

Spearbit: Acknowledged.

3.4.6 Potentially undetected underflow In assembly

Severity: Low Risk

Context: `LSSVMPair.sol#L447-494`, `LSSVMPairERC20.sol#L24-32`

Description: Functions `factory()`, `bondingCurve()`, `nft()`, `poolType()`, and `token()` have an assembly based calculation where the `paramsLength` is subtracted from `calldatasize()`. Assembly underflow checks are disregarded and if too few parameters are supplied in calls to the functions in the `LSSVMPair` contract, this calculation may underflow, resulting in the values for `factory()`, `bondingCurve()`, `nft()`, `poolType()`, and `token()` to be read from unexpected pieces of memory. This will be usually zeroed therefore execution will stop at some point. However, it is safer to prevent this from ever happening.

```
function factory() public pure returns (LSSVMPairFactoryLike _factory) {
    ...
    assembly {_factory := shr(0x60,calldataload(sub(calldatasize(),
 ↪  paramsLength)))}
}
function bondingCurve() public pure returns (ICurve _bondingCurve) {
    ...
    assembly {_bondingCurve := shr(0x60,calldataload(add(sub(calldatasize(),
 ↪  paramsLength), 20)))}
}
function nft() public pure returns (IERC721 _nft) {
    ...
    assembly {_nft := shr(0x60,calldataload(add(sub(calldatasize(),
 ↪  paramsLength), 40)))}
}
function poolType() public pure returns (PoolType _poolType) {
    ...
    assembly {_poolType := shr(0xf8,calldataload(add(sub(calldatasize(),
 ↪  paramsLength), 60)))}
}
function token() public pure returns (ERC20 _token) {
    ...
    assembly {_token := shr(0x60,calldataload(add(sub(calldatasize(),
 ↪  paramsLength), 61)))}
}
```

Recommendation: Spearbit recommends Sudoswap to implement the following changes for all functions so that an underflow will be detected at the Solidity level.

```
+ uint256 offset = msg.data.length - paramsLength;

- assembly {_token := shr(... ,calldataload(add(sub(calldatasize(),
 ↪  paramsLength), ...)))}
+ assembly {_token := shr(... ,calldataload(add(offset, ...)))}
```

Sudoswap: Acknowledged, but no change for now, as we only use this pattern for the purpose of reading variables meant to be immutable, so all values are hard-coded, which makes the possibility of underflow unlikely.

Spearbit: Acknowledged.

3.4.7 Check number of NFTs is not 0

Severity: Low Risk

Context: `LSSVMPair.sol#L258-310`, `LSSVMPair.sol#L318-366`, `LSSVMPair.sol#L413-431`

Description: Functions `swapNFTsForToken()`, `routerSwapNFTsForToken()`, and `getSellNFTQuote()` in `LSSVMPair.sol` do not perform input verification on the number of `NFTs`. If `_bondingCurve.getSellInfo()` accidentally happens to return a non-zero value, then an unfair amount of tokens will be given back to the caller.

The current two versions of `bondingCurve` do return `0`, but a future version might accidentally return non-zero.

Note:

1. `getSellInfo()` is supposed to return an error when `numNFTs == 0`, but this does not always happen. This error code is not always checked.

```
function swapNFTsForToken(uint256[] calldata nftIds, ...) external virtual
↪  returns (uint256 outputAmount) {
    ...
    // No check on `nftIds.length`
    (error, newSpotPrice, outputAmount, protocolFee) =
↪  _bondingCurve.getSellInfo(...,  nftIds.length,..);
    ...
}
function routerSwapNFTsForToken(address payable tokenRecipient) ... {
    ...
    uint256 numNFTs = _nft.balanceOf(getAssetRecipient()) -
↪  _assetRecipientNFTBalanceAtTransferStart;
    ...
    // No check that `numNFTs > 0`
    (error, newSpotPrice, outputAmount, protocolFee) =
↪  _bondingCurve.getSellInfo(..., numNFTs, ...);
}
function getSellNFTQuote(uint256 numNFTs) ... {
    ...
    // No check that `numNFTs > 0`
    (error, newSpotPrice, outputAmount, protocolFee) =
↪  bondingCurve().getSellInfo(..., numNFTs,...);
    ...
}
```

2. For comparison, the function `swapTokenForSpecificNFTs()` does perform an entry check on the number of requested `NFTs`.

```
function swapTokenForSpecificNFTs(uint256[] calldata nftIds,...) ... {
    ...
    //There is a check on the number of requested `NFT`s
    require( (nftIds.length > 0) && (nftIds.length <=
↪   _nft.balanceOf(address(this))), "Must ask for > 0 and < balanceOf NFTs");
↪   // check is present
    ...
}
```

Recommendation: Spearbit recommends Sudoswap to implement checks and make sure the number of `NFT`s is greater than `0`.

Sudoswap: Added `0` check to the `swapNFTForTokens` function here.

Spearbit: Acknowledged.

3.4.8 Avoid utilizing inside knowledge of functions

Severity: Low Risk

Context: `LSSVMRouter.sol`, `LSSVMPair.sol`, `LSSVMPairETH.sol`

Description: `ETH` based swap functions use `isRouter==false` and `routerCaller==address(0)` as parameters to `swapTokenForAnyNFTs()` and `swapTokenForSpecificNFTs()`. These parameters end up in `_validateTokenInput()`. The `LSSVMPairETH` version of this function does not use those parameters, so it is not a problem at this point.

However, the call actually originates from the `Router` so functionally `isRouter` should be true.

Our concern is that using inside knowledge of the functions might potentially introduce subtle issues in the following scenarios:

```
function robustSwapETHForAnyNFTs(...) ... {
    ...
    remainingValue -= swapList[i].pair.swapTokenForAnyNFTs{value:
 ↪  pairCost}(swapList[i].numItems, nftRecipient, false, address(0));
    ...
}
function robustSwapETHForSpecificNFTs(...) ... {
    ...
    remainingValue -= swapList[i].pair.swapTokenForSpecificNFTs{value:
 ↪  pairCost}(swapList[i].nftIds, nftRecipient, false, address(0));
    ...
}
function _swapETHForAnyNFTs(...) ... {
    ...
    remainingValue -= swapList[i].pair.swapTokenForAnyNFTs{value:
 ↪  pairCost}(swapList[i].numItems, nftRecipient, false, address(0));
    ...
}
function _swapETHForSpecificNFTs(...) ... {
    ...
     remainingValue -= swapList[i].pair.swapTokenForSpecificNFTs{value:
 ↪  pairCost}(swapList[i].nftIds, nftRecipient, false, address(0));
    ...
}
function swapTokenForAnyNFTs(..., bool isRouter, address routerCaller) ... {
    ...
    _validateTokenInput(inputAmount, isRouter, routerCaller, _factory);
    ...
}
function swapTokenForSpecificNFTs(..., bool isRouter, address routerCaller) ...
 ↪  {
    ...
    _validateTokenInput(inputAmount, isRouter, routerCaller, _factory);
    ...
}

abstract contract LSSVMPairETH is LSSVMPair {
    function _validateTokenInput(..., bool, /*isRouter*/  address,
 ↪  /*routerCaller*/... ) {
        // doesn't use isRouter and routerCaller
    }
}
```

Recommendation: Spearbit recommends Sudoswap to consider making the changes listed below in the follwoing functions `_swapETHForSpecificNFTs`, `robustSwapETHForAnyNFTs`, `robustSwapETHForSpecificNFTs`, and `_swapETHForAnyN-`

FTs:

```
- remainingValue -= swapList[i].pair.swapTokenForSpecificNFTs{value:
↪  pairCost}(swapList[i].nftIds, nftRecipient, false, address(0));
+ remainingValue -= swapList[i].pair.swapTokenForSpecificNFTs{value:
↪  pairCost}(swapList[i].nftIds, nftRecipient, true, msg.sender);

- remainingValue -= swapList[i].pair.swapTokenForAnyNFTs{value:
↪  pairCost}(swapList[i].numItems, nftRecipient, false, address(0));
+ remainingValue -= swapList[i].pair.swapTokenForAnyNFTs{value:
↪  pairCost}(swapList[i].numItems, nftRecipient, true, msg.sender);
```

Sudoswap: Addressed in this branch here.

Spearbit: Acknowledged.

3.4.9 Add Reentrancy Guards

Severity: Low Risk

Context: All `LSSVM` Contracts.

Specifically, three categories of functions:

1. Functions withdrawing ETH
2. Functions sending ETH
3. Functions that use `safeTransferFrom()` to call external addresses.

Instances of functions withdrawing `ETH`: `LSSVMPairFactory.sol#L272`, `LSSVMPairETH.sol#L104`

Instances of functions sending `ETH`: `LSSVMPairETH.sol#L34`, `LSSVMPairETH.sol#L46`, `LSSVMPairETH.sol#L79`, `LSSVMRouter.sol#L376`, `LSSVMRouter.sol#L423`, `LSSVMRouter.sol#L640`, `LSSVMRouter.sol#L677`

Uses of `safeTransferFrom()` to external addresses:

`LSSVMRouter.sol#L544`, `LSSVMRouter.sol#L593`, `LSSVMPairFactory.sol#L773`, `LSSVMPairEnumerable.sol#L33`, `LSSVMPairEnumerable.sol#L52`, `LSSVMPairEnumerable.sol#L73`, `LSSVMPairEnumerable.sol#L114`, `LSSVMPairMissingEnumerable.sol#L37`, `LSSVMPairMissingEnumerable.sol#L58`, `LSSVMPairMissingEnumerable.sol#L82`, `LSSVMPairMissingEnumerable.sol#L133`, `LSSVMPairMissingEnumerable.sol#L143`

Description: The abovementioned permalinks and corresponding functions are listed for Sudoswap's consideration to introduce reentrancy guard modifiers.

Currently, there is only one function that uses a reentrancy guard modifier: `withdrawAllETH()` in `LSSVMPairETH.sol#L94`.

Other functions in the codebase may also require reentrancy guard modifiers.

We have only seen reentrancy problems when malicious routers, `assetRecipients`, curves, factory owner or `protocolFeeRecipient` are involved. Despite normal prohibitions on this occurence, it is better to protect one's codebase than regret leaving open vulnerabilities available for potential attackers. There are three categories of functions that Sudoswap should consider applying reentrancy guard modifiers to: functions withdrawing ETH, functions sending ETH, and uses of `safeTransferFrom()` to external addresses (which will trigger an `onERC1155Received()` callback to receiving contracts).

Examples of functions withdrawing `ETH` within `LSSVM`: `LSSVMPairFactory.sol#L272`

`LSSVMPairETH.sol#L104`

Instances of functions sending `ETH` within `LSSVM`: `LSSVMPairETH.sol#L34`

`LSSVMPairETH.sol#L46`

A couple of instances that use `safeTransferFrom()` to call external addresses, which will trigger an `onERC1155Received()` callback to receiving contracts: `LSSVMPairFactory.sol#L428`

`LSSVMRouter.sol#L544`

Recommendation: Spearbit recommends Sudoswap to consider adding reentrancy guards to the abovementioned functions and to the entire codebase where appropriate.

Sudoswap: We've addressed the specific issues linked above with regards to reentrancy. The checks in `_pairTransferERC20From` and `_takeNFTsFromSender` both verify token balance or ownership before and after each transfer which can help ensure that tokens are actually sent to the `Pair` (or its asset recipient). However, we acknowledge this does not mitigate the entire space of possible issues with future malicious `Routers`. The gas overhead for adding ReentrancyGuard can be quite significant when e.g. swapping across multiple pools in one tx, so we have opted to avoid it if possible. Pair owners will need to be aware of which new Routers become approved by the Factory owner (intended to be set to governance controlled timelock during deployment).

Spearbit: Acknowledged.

3.5 Gas Optimization

3.5.1 Saving 1 byte off the `constructor()` code

Severity: Gas Optimization

Context: LSSVMPairCloner#L21-45, LSSVMPairCloner#L113-147

Description: The `dup2` before the `return` in the code below indicates a possible optimization by rearranging the stack.

```
function cloneETHPair(...) ... {
    assembly {
        ...
        // 3d          | RETURNDATASIZE        | 0                          | -
        // 60 runtime  | PUSH1 runtime     (r) | r 0                        | -
        // 80          | DUP1                  | r r 0                      | -
        // 60 creation | PUSH1 creation (c)    | c r r 0                    | -
        // 3d          | RETURNDATASIZE        | 0 c r r 0                  | -
        // 39          | CODECOPY              | r 0                        |
↪   [0-2d]: runtime code
        // 81          | DUP2                  | 0 c  0                     |
↪   [0-2d]: runtime code
        // f3          | RETURN                | 0                          |
↪   [0-2d]: runtime code
        ...
    }
}
```

Recommendation: Spearbit recommends Sudoswap to consider replacing the constructor code in `cloneETHPair()` and `cloneERC20Pair()` with the following code. Also, Sudoswap should make sure to update any dependencies on the length.

```
// 60 runtime    | PUSH1 runtime (r)     | r
// 3d            | RETURNDATASIZE        | 0 r
// 81            | DUP2                  | r 0 r
// 60 creation   | PUSH1 creation (c)    | c r 0 r
// 3d            | RETURNDATASIZE        | 0 c r 0 r
// 39            | CODECOPY              | 0 r
// f3            | RETURN                |
```

Sudoswap: Addressed in this branch here.

Spearbit: Acknowledged. Please double-check with the changes for the finding "Clones With Malicious `extradata` Are Also Considered Valid Clones" - especially the amount of bytes checked at the end of `isETHPairClone()` and `isERC20PairClone()`.

3.5.2 Decode `extradata` in `calldata` in one go

Severity: Gas Optimization

Context: `LSSVMPair.sol#L131-133`

Description: Spearbit discovered that the functions `factory()`, `bondingCurve()` and `nft()` are called independently but in most use cases all of the data is required.

Recommendation: Spearbit recommends Sudoswap to create a function that decodes data to save some gas. We recommend creating a new function that calls `calldatasize` and `_immutableParamsLength` only once to decode all parameters. This function will also save `jump` operations.

Sudoswap: Acknowledged, but no change for now. A previous implementation decoding all parameters at once had inconclusive gas savings during gas profiling.

Spearbit: Acknowledged.

3.5.3 Transfer last NFT instead of first

Severity: Gas Optimization

Context: `LSSVMPairEnumerable.sol#L23-35`, `LSSVMPairMissingEnumerable#L28-40`, `OpenZeppelin's ERC721Enumerable`, `OpenZeppelin's EnumerableSet.sol`

Description: When executing `_sendAnyNFTsToRecipient()` NFTs are retrieved by taking the first available NFT and then sending it to `nftRecipient`. In (most) `ERC721` implementations as well as in the `EnumerableSet` implementation, the array that stores the ownership is updated by swapping the last element with the selected element, to be able to shrink the array afterwards. When you always transfer the last NFT instead of the first NFT, swapping isn't necessary so gas is saved.

Code related to `LSSVMPairEnumerable.sol`:

```
abstract contract LSSVMPairEnumerable is LSSVMPair {
    function _sendAnyNFTsToRecipient(IERC721 _nft, address nftRecipient,
 ↪  uint256 numNFTs) internal override {
        ...
        for (uint256 i = 0; i < numNFTs; i++) {
            uint256 nftId =
 ↪  IERC721Enumerable(address(_nft)).tokenOfOwnerByIndex(address(this), 0);  //
 ↪  take the first NFT
            _nft.safeTransferFrom(address(this), nftRecipient, nftId); // this
 ↪  calls _beforeTokenTransfer of ERC721Enumerable
        }
    }
}
abstract contract ERC721Enumerable is ERC721, IERC721Enumerable {
    function _beforeTokenTransfer(address from, address to, uint256 tokenId)
 ↪  internal virtual override {
        ...
        _removeTokenFromOwnerEnumeration(from, tokenId);
        ...
    }
    function _removeTokenFromOwnerEnumeration(address from, uint256 tokenId)
 ↪  private {
        ...
        uint256 lastTokenIndex = ERC721.balanceOf(from) - 1;
        uint256 tokenIndex = _ownedTokensIndex[tokenId];

        // When the token to delete is the last token, the swap operation is
 ↪  unnecessary ==> we can make use of this
        if (tokenIndex != lastTokenIndex) {
            uint256 lastTokenId = _ownedTokens[from][lastTokenIndex];
            _ownedTokens[from][tokenIndex] = lastTokenId; // Move the last
 ↪  token to the slot of the to-delete token
            _ownedTokensIndex[lastTokenId] = tokenIndex; // Update the moved
 ↪  token's index
        }
        // This also deletes the contents at the last position of the array
        delete _ownedTokensIndex[tokenId];
        delete _ownedTokens[from][lastTokenIndex];
    }
}
```

Code related to `LSSVMPairMissingEnumerable.sol`:

```
abstract contract LSSVMPairMissingEnumerable is LSSVMPair {
    function _sendAnyNFTsToRecipient(IERC721 _nft, address nftRecipient,
 ↪  uint256 numNFTs) internal override {
        ...
        for (uint256 i = 0; i < numNFTs; i++) {
            uint256 nftId = idSet.at(0); // take the first NFT
            _nft.safeTransferFrom(address(this), nftRecipient, nftId);
            idSet.remove(nftId); // finally calls _remove()
        }
    }
}
library EnumerableSet {
    function _remove(Set storage set, bytes32 value) private returns (bool) {
        ...
        uint256 toDeleteIndex = valueIndex - 1;
        uint256 lastIndex = set._values.length - 1;
        if (lastIndex != toDeleteIndex) {  // ==> we can make use of this
            bytes32 lastvalue = set._values[lastIndex];
            set._values[toDeleteIndex] = lastvalue;  // Move the last value to
 ↪  the index where the value to delete is
            set._indexes[lastvalue] = valueIndex;  // Replace lastvalue's index
 ↪  to valueIndex
        }
        set._values.pop();  // Delete the slot where the moved value was stored
        delete set._indexes[value];  // Delete the index for the deleted slot
        ...
    }
}
```

Recommendation: Spearbit recommends Sudoswap to consider changing the `_sendAnyNFTsToRecipient()` function as follows;

Note: Do consider, if worth the trouble, taking into account the expected average number of `NFT`s that are exchanged. Note that the dynamics of distributing `NFT`s do change once you implement this; the mechanism changes from FIFO (first in first out) to LIFO (last in first out).

For `LSSVMPairEnumerable.sol`:

```
function _sendAnyNFTsToRecipient(IERC721 _nft, address nftRecipient, uint256
↪  numNFTs) internal override {
    ...
+   uint256 lastTokenIndex = _nft.balanceOf(address(this)) - 1;
    for (uint256 i = 0; i < numNFTs; i++) {
-       uint256 nftId =
↪  IERC721Enumerable(address(_nft)).tokenOfOwnerByIndex(address(this), 0); //
↪  take the first NFT
+       uint256 nftId =
↪  IERC721Enumerable(address(_nft)).tokenOfOwnerByIndex(address(this),
↪  lastTokenIndex-- ); // take the last NFT
        _nft.safeTransferFrom(address(this), nftRecipient, nftId);
    }
}
```

For `LSSVMPairMissingEnumerable.sol`:

```
function _sendAnyNFTsToRecipient(IERC721 _nft, address nftRecipient, uint256
↪  numNFTs) internal override {
    ...
+   uint256 lastIndex = idSet.length() - 1;
    for (uint256 i = 0; i < numNFTs; i++) {
-       uint256 nftId = idSet.at(0); // take the first NFT
+       uint256 nftId = idSet.at(lastIndex-- ); // take the last NFT
        _nft.safeTransferFrom(address(this), nftRecipient, nftId);
        idSet.remove(nftId);
    }
}
```

Sudoswap: Partially addressed in this branch here. We tried both recommendations for both the `Enumerable` and `MissingEnumerable` `Pairs`. The gas snapshot for our test bench showed gas reductions when transferring the last `NFT` out first in the `MissingEnumerable` implementation, so we have followed the recommendation. However, the gas savings in the `Enumerable` case were inconclusive, so we have not followed the second recommendation.

Spearbit: Acknowledged.

3.5.4 Simplify the connection between `Pair` and `Router`

Severity: Gas Optimization

Context: `LSSVMPairERC20.sol#L41-78`, `LSSVMRouter.sol#L574-594`, `LSSVMRouter.sol#L754-789`, `LSSVMPair.sol#L371-379`

Description: There are two ways to interact between Pair and Router:

1. `LSSVMPairERC20.sol` calls `router.pairTransferERC20From`, where the goal is to transfer `ERC20`
2. `_swapNFTsForToken` calls `pair.cacheAssetRecipientNFTBalance` and `pair.routerSwapNFTsForToken`, where the goal is to transfer `NFTs` Using two different patterns to solve the same problem makes the code more complex and larger than necessary. Patterns with `cacheAssetRecipientNFTBalance()` are also error prone.

```
abstract contract LSSVMPairERC20 is LSSVMPair {
    function _validateTokenInput(..., bool isRouter, ...) ... {
        ...
        if (isRouter) {
            LSSVMRouter router = LSSVMRouter(payable(msg.sender));  // Verify
 ↪  if router is allowed
            require(_factory.routerAllowed(router), "Not router");
            ...
            router.pairTransferERC20From(
                _token,
                routerCaller,
                _assetRecipient,
                inputAmount,
                pairVariant()
            );
            ...
        }
    ...
    }
}
contract LSSVMRouter {
    function pairTransferERC20From(...) ... {
        // verify caller is a trusted pair contract
        require(factory.isPair(msg.sender, variant), "Not pair");
        ...
        // transfer tokens to pair
        token.safeTransferFrom(from, to, amount); // transfer ERC20 from the
 ↪  original caller
    }
}
```

```
contract LSSVMRouter {
    function _swapNFTsForToken(...) ... {
        ...
        // Cache current asset recipient balance
        swapList[i].pair.cacheAssetRecipientNFTBalance();
        ...
        for (uint256 j = 0; j < swapList[i].nftIds.length; j++) {

↪   nft.safeTransferFrom(msg.sender,assetRecipient,swapList[i].nftIds[j]); //
↪   transfer NFTs from the original caller
        }
        ...
        outputAmount +=
↪   swapList[i].pair.routerSwapNFTsForToken(tokenRecipient);
        ...
   }
}
abstract contract LSSVMPair is Ownable, ReentrancyGuard {
    function cacheAssetRecipientNFTBalance() external {
        require(factory().routerAllowed(LSSVMRouter(payable(msg.sender))),"Not
↪   router"); // Verify if router is allowed
        assetRecipientNFTBalanceAtTransferStart =
↪   nft().balanceOf(getAssetRecipient()) + 2;
    }
}
```

Recommendation: Spearbit recommends Sudoswap to let `_swapNFTsForToken` use the same pattern as `LSSVMPairERC20.sol` so that `cacheAssetRecipientNFTBalance()` will not be necessary any longer. In addition, the functions `routerSwapNFTsForToken()` and `swapNFTsForToken()` can be merged, simplifying the code.

Sudoswap: Addressed in this branch here. Balances are no longer `cached` and then used in a separate `routerSwap` function. Instead, we use the same pattern as `ERC20` tokens, with the router pulling from the user, with the same check that it can only come from `LSSVMPair`. `LSSVMPair` now checks `NFT` ownership before and after each `transfer` to guard against potentially malicious routers.

Spearbit: Acknowledged.

3.5.5 Cache array length

Severity: Gas Optimization

Context: `LSSVM Contracts`, `LSSVMPairEnumerable.sol`, `LSSVMPairFactory.sol`, `LSSVMPairMissingEnumerable.sol`, `LSSVMRouter.sol`

Specifically, the following lines in the corresponding contracts:

`LLSSVMPairEnumerable.sol`: `LSSVMPairEnumerable.sol#L51`, `LSSVMPairEnumerable.sol#L72`, `LSSVMPairEnumerable.sol#L113`

`LSSVMPairFactory.sol`: `LSSVMPairFactory.sol#L378`, `LSSVMPairFactory.sol#L409`, `LSSVMPairFactory.sol#L427`

`LSSVMPairMissingEnumerable.sol`: `LSSVMPairMissingEnumerable.sol#L57`, `LSSVMPairMissingEnumerable.sol#L81`, `LSSVMPairMissingEnumerable.sol#L132`, `LSSVMPairMissingEnumerable.sol#L142`

`LSSVMRouter.sol`: `LSSVMRouter.sol#L358`, `LSSVMRouter.sol#L405`, `LSSVMRouter.sol#L448`, `LSSVMRouter.sol#L491`, `LSSVMRouter.sol#L524`, `LSSVMRouter.sol#L543`, `LSSVMRouter.sol#L625`, `LSSVMRouter.sol#L662`, `LSSVMRouter.sol#L700`, `LSSVMRouter.sol#L772`

Description: An array length is frequently used in `for` loops. This value is an evaluation for every iteration of the loop.

Assuming the arrays are regularly larger than 1, it saves some gas to store the array length in a temporary variable.

The following snippets are samples of the above context for lines of code where this is relevant:

`LSSVMPairEnumerable.sol#L51`

`LSSVMPairFactory.sol#L378`

`LSSVMPairMissingEnumerable.sol#L57`

`LSSVMRouter.sol#L358`

For more examples, please see the context above for exact lines where this applies.

The following contains examples of the overusage of `nftIds.length`:

```
function swapTokenForSpecificNFTs(...) ... {
    ...
    require((nftIds.length > 0)
    && (nftIds.length <= _nft.balanceOf(address(this))),"Must ask for > 0 and <
↪   balanceOf NFTs");
    ...
    (error, newSpotPrice, inputAmount, protocolFee) = _bondingCurve
        .getBuyInfo(
            spotPrice,
            delta,
            nftIds.length,
            fee,
            _factory.protocolFeeMultiplier()
        );
    ...
}
```

Recommendation: Spearbit recommends Sudoswap to store array lengths in a temporary variable, especially with `for` loops.

Sudoswap: Acknowledged, but no change for now.

Spearbit: Acknowledged.

3.5.6 Use Custom Errors

Severity: Gas Optimization

Context: `LSSVM\Contracts`, specifically the following contracts: `LSSVMPair.sol`, `LSSVMPairERC20.sol`, `LSSVMPairETH.sol`, `LSSVMPairFactory.sol` `LSSVMRouter.sol`.

Specific locations within each contract:

`LSSVMPair.sol`: `LSSVMPair.sol#L79`, `LSSVMPair.sol#L86`, `LSSVMPair.sol#L91`, `LSSVMPair.sol#L92-95`, `LSSVMPair.sol#L99`, `LSSVMPair.sol#L100-103`, `LSSVMPair.sol#L138-141`, `LSSVMPair.sol#L142-145`, `LSSVMPair.sol#L161`, `LSSVMPair.sol#L205-208`, `LSSVMPair.sol#L209-213`, `LSSVMPair.sol#L229`, `LSSVMPair.sol#L271-274`, `LSSVMPair.sol#L290`, `LSSVMPair.sol#L298-301`, `LSSVMPair.sol#L354`, `LSSVMPair.sol#L372-375`, `LSSVMPair.sol#L632-635`, `LSSVMPair.sol#L646-649`, `LSSVMPair.sol#L662`, `LSSVMPair.sol#L663`, `LSSVMPair.sol#L677`, `LSSVMPair.sol#L692`, `LSSVMPair.sol#L694`

`LSSVMPairERC20.sol`: `LSSVMPairERC20.sol#L47`, `LSSVMPairERC20.sol#L55`, `LSSVMPairERC20.sol#L69-73`

LSSVMPairETH.sol: LSSVMPairETH.sol#L29

LSSVMPairFactory.sol: LSSVMPairFactory.sol#L51-54, LSSVMPairFactory.sol#L57-60, LSSVMPairFactory.sol#L63-66, LSSVMPairFactory.sol#L69-72, LSSVMPairFactory.sol#L75, LSSVMPairFactory.sol#L78, LSSVMPairFactory.sol#L111-114, LSSVMPairFactory.sol#L179-182, LSSVMPairFactory.sol#L295, LSSVMPairFactory.sol#L307, LSSVMPairFactory.sol#L335, LSSVMPairFactory.sol#L350, LSSVMPairFactory.sol#L353

LSSVMRouter.sol: LSSVMRouter.sol#L582, LSSVMRouter.sol#L585-590, LSSVMRouter.sol#L604, LSSVMRouter.sol#L788

Description: Strings are used to encode error messages. With the current Solidity versions it is possible to replace them with custom errors, which are more gas efficient.

Example of non-custom errors used in `LSSVM`:

LSSVMRouter.sol#L604

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

LSSVMRouter.sol#L788

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

Note: This pattern has been used in Ownable.sol#L6-L7

Recommendation: Spearbit recommends Sudoswap to use custom error messages. Custom error message usage is explained here on the Solidity Language Blog.

Sudoswap: Acknowledged, but no change for now. Because `Pair`s are minimal proxies, the gas savings from not having to deploy strings is less of a consideration as the implementation is only deployed once.

Spearbit: Acknowledged.

3.5.7 Alternatives for the `immutable` Proxy variables

Severity: Gas Optimization

Context: LSSVMPairCloner.sol#L113-137

Description: In the current `LSSVMPairClone`, the `immutable` variables stored in the `proxy` are sent along with every call. It may be possible to optimize this.

Recommendation: Spearbit recommends Sudoswap two different approaches to this optimization. The first is that Sudoswap store extra data in the contract code, accessing the extra data by using `extcodecopy` in the `Pair` contracts. It simplifies the `proxy` code a little, and arguably the `Pair` code as well.

Spearbit completed a potential implementation of this recommendation as follows:

Source snippet for proxy creation:

```
bytes memory ptr = abi.encodePacked(<proxy part1>, implementation, <proxy
 ↪  part2>,factory, bondingCurve, nft, poolType, token);
assembly {
    instance := create(0, ptr, ...)
    ...
}
```

Source snippet to retrieve (immutable) values from the proxy code:

```
function getFactory() internal view returns (address factory) {
    assembly {
        // Copies to "scratch space" 0 memory pointer
        extcodecopy(address(), 0, 0x28, 0x14)
        factory:= shr(0x60, mload(0))
    }
}
```

If this is not preferable for Sudoswap, our second recommendation is to improve the readability of the code by using Solidity only. However, this approach will cost some more gas. To implement this alternative, follow these steps:

1. Store the extra data as an immutable variable in the proxy.
2. Create a getter function `extraData()`. Because the address will always be hot when being called by the `Pair`, the call should cost only about 150 gas.
3. The `proxy` code can then be almost entirely in Solidity and the `Pair` will not require the use any low-level calls.

Sudoswap: Acknowledged, but no changes at this time. We previously used the `extcodecopy` method, and the current method of accessing the immutable variables appears to be cheaper, gas-wise.

Spearbit: Acknowledged.

3.6 Informational

3.6.1 Pair implementations may not be Proxies

Severity: Informational

Context: `LSSVMRouter.sol#L574-594`, `LSSVMPairFactory.sol#L223-257`, `LSSVMPairCloner.sol#L206-267`

Description: The security of function `pairTransferERC20From()` relies on `isPair()`. In turn, `isPair()` relies on both `isETHPairClone()` and `isERC20PairClone()`. These functions check that a valid proxy is used with a valid implementation address. However, if the implementation address itself is a proxy it could link to any other contract. In this case security could be undermined depending on the implementation details. This is not how the protocol is designed, but future developers or developers using a fork of the code might not be aware of this.

Recommendation: Spearbit recommends Sudoswap to make sure no proxies are used for the `Pair` implementation. This is a control that should be present at the deployment process (e.g. is outside of Solidity).

Sudoswap: Acknowledged, no external facing changes made. As mentioned above, our protocol does not use proxies for the implementation so no action is taken at this time.

Spearbit: Acknowledged.

3.6.2 `NFT` and `Token` Pools can be signed orders instead

Severity: Informational

Context: `LSSVMPair.sol`

Description: Currently if any actor wants to create a `buy/sell` order they would have to create a new pool and pay gas for it. However, the advantage of this is unclear. `TOKEN` and `NFT` type pools can really be `buy/sell` orders at a price curve using signed data. This is reminiscent of how similar limit orders implemented by OpenSea, 1Inch, and SushiSwap currently function. Amending this in the codebase would make creating `buy/sell` orders free and should attract more liquidity and/or orders to Sudoswap.

Recommendation: Spearbit recommends Sudoswap to consider the addition of a single `OrderBook` contract that only the taker has to call and provide the maker's signed order to fulfill an order.

Sudoswap: Acknowledged. The signature based model is well-served by e.g. 0x Protocol, Wyvern, and others with off-chain matching and on-chain settlement. The intent here behind having on-chain pools is precisely the benefits of being on-chain, e.g easier management by DAOs and other smart contract actors which would need to make a tx anyway to work with a signature-based order book anyway allowing projects to lock in buy-side liquidity in a trustless manner similar to how LP for normal AMM pools can be locked decentralized order book by design--anyone can query pools and build front-ends for swapping without relying on centralized off-chain API. Also to note: separately from this AMM protocol, Sudoswap already has an off-chain order book leveraging 0x Protocol v2 for limit buys/sells.

Spearbit: Acknowledged.

3.6.3 Remove Code Duplication

Severity: Informational

Context: `LSSVMPair.sol#L125`, `LSSVMPair.sol#L192`

Description: Functions like `swapTokenForAnyNFTs` and `swapTokenForSpecificNFTs` are nearly identical and can be deduplicated by creating a common internal function. On the other hand this will slightly increase gas usage due to an extra jump.

Recommendation: Spearbit recommends Sudoswap to considering the trade-off between code hygiene and minute gas savings. Our opinion is that the minor gas savings are not worth it in this case and the code should be deduplicated.

Sudoswap: Acknowledged, but no change at this time.

Spearbit: Acknowledged.

3.6.4 Unclear Function Name

Severity: Informational

Context: `LSSVMPairETH.sol#L23-36`, `LSSVMPairERC20.sol#L41-78`

Description: The functions `_validateTokenInput()` of both `LSSVMPairETH` and `LSSVMPairERC20` do not only validate the token input but also transfer `ETH/ERC20`. The function name does not reasonably imply this and therefore can create some confusion.

```
abstract contract LSSVMPairETH is LSSVMPair {
    function _validateTokenInput(...) ... {
        ...
        _assetRecipient.safeTransferETH(inputAmount);
        ...
    }
}
abstract contract LSSVMPairERC20 is LSSVMPair {
    function _validateTokenInput(...) ... {
        ...
        if (isRouter) {
            ...
            router.pairTransferERC20From(...); // transfer of tokens
            ...
        } else {
            // Transfer tokens directly
            _token.safeTransferFrom(msg.sender, _assetRecipient, inputAmount);
        }
    }
}
```

Recommendation: Spearbit recommends Sudoswap to consider renaming the function `_validateTokenInput()` to `_validateAndTransferTokenInput()`.

Sudoswap: Changed in response to this issue. `_validateTokenInput` is now called `_pullTokenInputAndPayProtocolFee`.

Spearbit: Acknowledged.

3.6.5 Inaccurate Message About `MAX_FEE`

Severity: Informational

Context: `LSSVMPair.sol`

Description: The function `initialize()` of `LSSVMPair` has an error message containing `less than 100%`. This is likely an error and should probably be less than 90%, as in the `changeFee()` function and because `MAX_FEE == 90%`.

```
// 90%, must <= 1 - MAX_PROTOCOL_FEE (set in LSSVMPairFactory)
uint256 internal constant MAX_FEE = 9e17;

function initialize(..., uint256 _fee, ...) external payable {
    ...
    require(_fee < MAX_FEE, "Trade fee must be less than 100%"); // 100% should
↪   be 90%
    ...
}
function changeFee(uint256 newFee) external onlyOwner {
    ...
    require(newFee < MAX_FEE, "Trade fee must be less than 90%");
    ...
}
```

Recommendation: Spearbit recommends Sudoswap to change the 100% to 90%.

Sudoswap: Addressed in branch here.

Spearbit: Acknowledged.

3.6.6 Inaccurate comment for `assetRecipientNFTBalanceAtTransferStart`

Severity: Informational

Context: `LSSVMPair.sol#L27-29`, `LSSVMPair.sol#L318-337`

Description: The comment in `LSSVMPair` notes that `assetRecipientNFTBalanceAtTransferStart` is `0`; however, in `routerSwapNFTsForToken()` the variable `assetRecipientNFTBalanceAtTransferStart` is set to `1`. As such, the below comment is probably inaccurate.

```
// Temporarily used during LSSVMRouter::_swapNFTsForToken to store the number
↪   of NFTs transferred
// directly to the pair. Should be 0 outside of the execution of
↪   routerSwapAnyNFTsForToken.
uint256 internal `assetRecipientNFTBalanceAtTransferStart`;

function routerSwapNFTsForToken(address payable tokenRecipient) ... {
    ...
    assetRecipientNFTBalanceAtTransferStart = 1;
    ...
}
```

Recommendation: Spearbit recommends Sudoswap to re-evaluate the accuracy of this comment.

Sudoswap: The comment is indeed incorrect. Addressed in the change to the GitHub Issue here (We no longer use the `cache` flow).

Spearbit: Acknowledged.

3.6.7 `IERC1155` not utilized

Severity: Informational

Context: `LSSVMPair.sol#L5`, `LSSVMRouter.sol#L35-38`

Description: The contract `LSSVMPair` references `IERC1155`, but does not utilitze the interface within `LSSVMPair.sol`.

```
import {IERC1155} from "@openzeppelin/contracts/token/ERC1155/IERC1155.sol";
```

The struct `TokenToTokenTrade` is defined in `LSSVMRouter`, but the contract does not utilize the interface either.

```
struct TokenToTokenTrade {
    PairSwapSpecific[] tokenToNFTTrades;
    PairSwapSpecific[] nftToTokenTrades;
}
```

It is better to remove unused code due to potential confusion.

Recommendation: Spearbit recommends Sudoswap to remove the import of `IERC1155` from `LSSVMPair.sol` and remove the struct `TokenToTokenTrade` from `LSSVMRouter.sol`.

Sudoswap: Addressed in branch here. Unused import and struct are removed.

Spearbit: Acknowledged.

3.6.8 Use Fractions

Severity: Informational

Context: `LSSVMPairFactory.sol#L28`, `LSSVMPair.sol#L25`

Description: In some occasions percentages are indicated in a number format ending in `e17`. It is also possible to use fractions of `e18`. Considering `e18` is the standard base format, using fractions might be easier to read.

`LSSVMPairFactory.sol#L28`

`LSSVMPair.sol#L25`

Recommendation: Spearbit recommends Sudoswap to update `LSSVMPairFactory.sol#L28` for better readability with fractions:

```
-   uint256 internal constant MAX_PROTOCOL_FEE = 1e17;
+   uint256 internal constant MAX_PROTOCOL_FEE = 0.10e18; // 10%
```

In addition, Sudoswap should update `LSSVMPair.sol#L25` to include better readibility with fractions, according to the `diff` below:

```
-   uint256 internal constant MAX_FEE = 9e17;
+   uint256 internal constant MAX_FEE = 0.90e18; // 90%
```

Sudoswap: Addressed in branch here.

Spearbit: Acknowledged.

3.6.9 Two families of token libraries used

Severity: Informational

Context: `LSSVMPairFactory.sol#L4-10`

Description: The Sudoswap contract imports token libraries from both OpenZeppelin and Solmate. If Sudoswap sticks within one library family, then it will not be necessary to track potential issues from two separate families of libraries.

Recommendation: Spearbit recommends Sudoswap to consider choosing either the Solmate or the OpenZeppelin family of libraries.

If there is a specific reason to use multiple libraries, add a comment in the contracts why multiple libraries are used.

Sudoswap: We use the solmate library for easier integration with their `safeTransferLib` which saves on some gas. Dev comment explaining this has been added in this commit here.

Spearbit: Acknowledged.