PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For Smardex.io

19 July 2023

paladinsec.co   info@paladinsec.co

Table of Contents

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains the right to re-use any and all knowledge and expertise gained during the audit process, including, but not limited to, vulnerabilities, bugs, or new attack vectors. Paladin is therefore allowed and expected to use this knowledge in subsequent audits and to inform any third party, who may or may not be our past or current clients, whose projects have similar vulnerabilities. Paladin is furthermore allowed to claim bug bounties from third-parties while doing so.

1   Overview

This report has been prepared for Smardex.io on the Ethereum network. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1   Summary

Project Name	Smardex.io
URL	https://smardex.io/
Network	Ethereum, Polygon, BNB Smart Chain, Arbitrum
Language	Solidity
Preliminary	https://github.com/SmarDex-Dev/smart-contracts/tree/15e4dea57745b30a1f65083930300442b1661a85
Resolution 1	https://github.com/SmarDex-Dev/smardex-contract-fix-paladin-09.06.2023/tree/e07745bcad54252060f115db56b8eba50206ccf9
Resolution 2	Re-Audit Upgradable fees https://github.com/SmarDex-Dev/smart-contracts-updatable-fees/commit/23045fa2fef2e7a03f98b6632d520f393a904213 Audit L2 Contracts https://github.com/SmarDex-Dev/smart-contracts-updatable-fees/blob/3f73b2298e438fdb879c451bcb5c404cd56b7642 Resolution 2 fixes https://github.com/SmarDex-Dev/smart-contracts-updatable-fees/commit/dc05e390fbc86cd5ca9919a44f14dabd300389c4

Paladin Blockchain Security

1.2 Contracts Assessed

Name	Contract	Live Code Match
RewardManager	ETH: 0xC049c7fE5EAa024095861212BD8a8E8Df88a2b1f	✓ MATCH
FarmingRange	ETH: 0x7d85C0905a6E1Ab5837a0b57cD94A419d3a77523 Polygon: 0x7DB73A1e526db36c40e508b09428420c1fA8e46b BSC: 0xb891Aeb2130805171796644a2af76Fc7Ff25a0b9	✓ MATCH
Staking	0x80497049b005Fd236591c3CD431DBD6E06eB1A31	✓ MATCH
AutoSwapper	0x865d61582abc2CCd85549774CEe171280fE82e3e	✓ MATCH
Router	ETH: 0xEf2f9b48d7EC80440Ab4573dF1A2aBDBE06D3f60 Polygon: 0xA8EF6FEa013034E62E2C4A9Ec1CDb059fE23Af33 BSC: 0x391BeCc8DAaf32b9ba8e602e9527Bf9DA04C8deb Arbitrum: 0xdd4536dD9636564D891c919416880a3e250f975A	✓ MATCH
BytesLib	Dependency	✓ MATCH
Path	Dependency	✓ MATCH
PoolAddress	Dependency	✓ MATCH
PoolHelpers	Dependency	✓ MATCH
SmardexPair	Deployed by SmardexFactory	✓ MATCH

SmardexFactory	ETH: 0x7753F36E711B66a0350a753aba9F5651BAE76A1D Polygon: 0x9A1e1681f6D59Ca051776410465AfAda6384398f BSC: 0xA8EF6FEa013034E62E2C4A9Ec1CDb059fE23Af33 Arbitrum: 0x41A00e3FbE7F479A99bA6822704d9c5dEB611F22	✓ MATCH
SmardexLibrary	Dependency	✓ MATCH
TransferHelper	Dependency	✓ MATCH
SmardexToken	ETH: 0x5DE8ab7E27f6E7A1fFf3E5B337584Aa43961BEeF Polygon: 0x6899fAcE15c14348E1759371049ab64A3a06bFA6 BSC: 0xFdc66A08B0d0Dc44c17bbd471B88f49F50CdD20F Arbitrum: 0xabD587f2607542723b17f14d00d99b987C29b074	✓ MATCH
AutoSwapperL2	Polygon: 0x8ef71fDe0681f567d241A2Cb7f2aE46d6D709D30 BSC: 0x41A00e3FbE7F479A99bA6822704d9c5dEB611F22 Arbitrum: 0xd978bb7a7aE5b01116D1bD714fEDd1B77Ca9e75b	✓ MATCH
FarmingRangeL2Arbitr um	0x53D165DF0414bD02E91747775450934BF2257f69	✓ MATCH
RewardManagerL2	Polygon: 0x391BeCc8DAaf32b9ba8e602e9527Bf9DA04C8deb BSC: 0x8ef71fDe0681f567d241A2Cb7f2aE46d6D709D30	✓ MATCH
RewardManagerL2Arbit rum	0x5C622Dcc96b6D96ac6c154f99CF081815094CBC9	✓ MATCH

Paladin Blockchain Security

1.3   Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
🔴 High	5	3	1	1
🟠 Medium	11	7	1	3
🟡 Low	17	12	-	5
🟣 Informational	17	13	-	4
Total	**50**	**35**	**2**	**13**

Classification of Issues

Severity	Description
🔴 High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
🟠 Medium	Bugs or issues that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
🟡 Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
🟣 Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1   RewardManager

ID	Severity	Summary	Status
01	LOW	`safeApprove` should be used within the `RewardManager`	✓ RESOLVED
02	LOW	`_startFarmingCampaign` can be set to a value in the past	✓ RESOLVED
03	INFO	Typographical error	✓ RESOLVED

1.3.2   FarmingRange

ID	Severity	Summary	Status
04	HIGH	The contract owner can steal all staking tokens	✓ RESOLVED
05	HIGH	First depositor can steal reward tokens	✓ RESOLVED
06	HIGH	Contract does not support tokens with a fee on transfer	ACKNOWLEDGED
07	MEDIUM	Owner can steal all rewards	✓ RESOLVED
08	MEDIUM	Adding too many reward epochs can result in a DoS	PARTIAL
09	MEDIUM	Consecutive calls of `upgradePrecision` can result in a DoS	✓ RESOLVED
10	MEDIUM	`setRewardInfoLimit` can cause DoS	✓ RESOLVED
11	MEDIUM	Rewards can get stuck in the contract	✓ RESOLVED
12	LOW	Reentrancy call allows owner to add false `rewardInfo`	✓ RESOLVED
13	LOW	`totalRewards` is not decreased for next round's refund	✓ RESOLVED
14	LOW	Lack of validation	✓ RESOLVED
15	LOW	Lack of `safeTransfer` usage	✓ RESOLVED
16	LOW	`startBlock` can be set to a value in the past	✓ RESOLVED
17	INFO	Unlisted campaigns can be updated	✓ RESOLVED
18	INFO	The CEI pattern is not adhered to	✓ RESOLVED
19	INFO	Gas optimizations	✓ RESOLVED
20	INFO	Lack of events for `upgradePrecision`	✓ RESOLVED

Paladin Blockchain Security

1.3.3 Staking

ID	Severity	Summary	Status
21	HIGH	The first depositor can steal the shares of subsequent depositors	PARTIAL
22	HIGH	Funds can be stuck in the contract forever	✔ RESOLVED
23	MEDIUM	Admin injections can be frontran	✔ RESOLVED
24	LOW	Lack of validation	✔ RESOLVED
25	INFO	Contract does not adhere to the CEI pattern	✔ RESOLVED
26	INFO	`_sharesToTokens` is slightly flawed	ACKNOWLEDGED

1.3.4 AutoSwapper

ID	Severity	Summary	Status
27	MEDIUM	The slippage calculation is flawed	ACKNOWLEDGED
28	LOW	Lack of validation	✔ RESOLVED
29	LOW	Arbitrage opportunities in increased trading periods	ACKNOWLEDGED
30	INFO	Lack of events for `executeWork`	✔ RESOLVED

1.3.5 SmardexRouter

ID	Severity	Summary	Status
31	MEDIUM	Ether is refunded to the `_to` address	✔ RESOLVED
32	MEDIUM	Router does not support tokens with a fee on transfer	ACKNOWLEDGED
33	MEDIUM	`getAmountIn` and `getAmountOut` might lead to wrong values	✔ RESOLVED
34	LOW	Funds can be stuck in the router	✔ RESOLVED
35	INFO	Unprivileged ETH can be stolen	ACKNOWLEDGED

1.3.6   BytesLib

ID	Severity	Summary	Status
36	INFO	`toUint24` is unused	✔ RESOLVED

1.3.7   Path

ID	Severity	Summary	Status
37	INFO	Gas optimization	✔ RESOLVED
38	INFO	Typographical error	✔ RESOLVED

1.3.8   PoolAddress

No issues found.

1.3.9   PoolHelpers

No issues found.

1.3.10　SmardexPair

ID	Severity	Summary	Status
39	MEDIUM	The protection for `_feeToSwap` does not suffice	ACKNOWLEDGED
40	LOW	`_feeToSwap` can be subjected to sandwich attacks	ACKNOWLEDGED
41	INFO	`_feeToSwap` can be subjected to gas griefing	ACKNOWLEDGED
42	INFO	Gas optimization	✓ RESOLVED
43	INFO	Typographical issue	✓ RESOLVED

1.3.11　SmardexFactory

ID	Severity	Summary	Status
44	LOW	Missing safeguards on `setFeeToSetter`	✓ RESOLVED
45	INFO	Owner can redirect all fees to their own address	ACKNOWLEDGED
46	INFO	Lack of events for `setFeeTo` and `setFeeToSetter`	✓ RESOLVED

1.3.12　SmardexLibrary

ID	Severity	Summary	Status
47	LOW	Swap logic limits swaps	ACKNOWLEDGED

1.3.13 TransferHelper

No issues found.

1.3.14 SmardexToken

ID	Severity	Summary	Status
48	LOW	Outdated ERC20 `Permit` implementation	✓ RESOLVED

1.3.15 AutoSwapperL2

No issues found.

1.3.16 FarmingRangeL2Arbitrum

No issues found.

1.3.17 RewardManagerL2

ID	Severity	Summary	Status
49	LOW	The Staking contract should not be deployed to L2 chains	ACKNOWLEDGED

1.3.18   RewardManagerL2Arbitrum

ID	Severity	Summary	Status
50	LOW	The Staking contract should not be deployed to L2 chains	ACKNOWLEDGED

2   Findings

2.1   Rewards/RewardManager

`RewardManager` is a simple initiator contract which handles the deployment of the `FarmingRange` and `Staking` contracts. During the contract deployment, it automatically deploys both of these contracts and creates the first campaign within the `FarmingRange` contract which uses the `Staking` contract as the staking token and the SmarDex tokens as the reward token.

Besides this deployment, this contract is meant to store all reward tokens for the `FarmingRange` contract and eventually distributes them upon each campaign configuration via `resetAllowance`.

2.1.1 Issues & Recommendations

Issue #01	safeApprove should be used within the RewardManager
Severity	🟡 LOW SEVERITY
Description	RewardManager may not be able to approve tokens that do not follow the ERC20 standard. For example, USDT on mainnet would revert as it does not return a boolean as expected.
Recommendation	Consider using safeApprove instead. Note that safeApprove needs to first be set to 0 before changing its value to a value greater than 0.
Resolution	✅ RESOLVED

Issue #02	_startFarmingCampaign can be set to a value in the past
Severity	🟡 LOW SEVERITY
Description	There is no check that _startFarmingCampaign is set to a value in the future. If this is accidentally set to a value in the past, users will receive rewards retroactively, especially since the _startTime cannot be updated anymore within the addRewardInfo function.
Recommendation	Consider validating that _startFarmingCampaign can only be set to a value in the future.
Resolution	✅ RESOLVED

Issue #03	Typographical error
Severity	● INFORMATIONAL
Description	*/** * @title RewardManager * @notice RewardManager handles de creation of the contract staking and farming, automatically create a campaignInfo * in the farming for the staking, at slot 0 and initialize farming. The RewardManager is the owner of the funds in * the FarmingRange, only the RewardManager is capable of sending funds to be farmed and only the RewardManager will get * the funds back when updating of removing campaigns. */* *de* should be corrected to *the*.
Recommendation	Consider fixing the typographical error.
Resolution	✔ RESOLVED

2.2   Rewards/FarmingRange

FarmingRange is a highly customized masterchef-like contract where users can stake different staking tokens in order to receive different reward tokens. The main logic is handled by different campaigns.

The contract owner can add different campaignInfo[s] where each campaign represents a pool with a staking token and a corresponding reward token. After a campaign is successfully added, the owner can then assign rewards to this campaign via addRewardInfo — this can be done in multiple block ranges whereas the reward amount to be distributed is calculated via rewardPerBlock * (endBlock - startBlock). Each block range can have a totally different rewardPerBlock and block duration assigned, however, these block ranges must be sequential, i.e., the following block can only start after the current block has ended.

The owner can change rewardsPerBlock and endTime of the current round or any future rounds for any campaign, which will then refund tokens to the RewardManager or require more tokens to be transferred in. When the endTime is increased, the block range for the next round will be decreased, however, rewards will not be decreased, as the logic simply increases the rewardPerBlock for the next round in order to accommodate the block range decrease.

Users can participate in these campaigns by depositing in the different campaignIds which start at slot 1 (the zero slot is already occupied for the Staking contract). Initially, each campaign can have up to 52 rounds, however, the owner can change the rewardInfoLimit arbitrarily which potentially allows for more rounds.

After the owner has added a campaign, users can stake in it, however, they will not receive a reward if there are no added reward rounds.

2.2.1　Privileged Functions

- transferOwnership

- renounceOwnership

- upgradePrecision

- setRewardManager

- setRewardInfoLimit

- addCampaignInfo

- addRewardInfo

- addRewardInfoMultiple

- updateRewardInfo

- updateRewardMultiple

- updateCampaignsReward

- removeLastRewardInfo

2.2.2 Issues & Recommendations

Issue #04	The contract owner can steal all staking tokens
Severity	🔴 HIGH SEVERITY
Description	The contract owner has the ability to steal staking tokens via a consecutive process of function calls.

At first glance, it seems impossible for the owner to steal any tokens because even if any `stakingToken` is used as the `rewardToken`, the `RewardManager` would need to transfer in the correct amount in order to allocate rewards to a pool. Additionally, the removal or update of a reward round would only transfer the same / leftover amount out, which was transferred in by the admin.

However, we found two methods to exploit this flow in order to steal any staking token from the user.

Consider the first PoC:

1. The owner sends a specific amount of USDT-USDC LP to the `RewardManager`.

2. The owner creates a dummy pool with USDT-USDC as a reward token.

3. The owner adds a campaign which takes exact the same amount as rewards as sent to the `RewardManager` and deposits into this pool.

4. The owner now calls `upgradePrecision` which increases the `accRewardPerShare` by 1e8, essentially increasing the accumulated USDT-USDC reward as well.

5. The owner has now successfully increased the allocated rewards, which means that USDT-USDC from users will be used as reward for the allocated share of the owner.

Consider the second PoC which is a bit more sophisticated:

1. The owner creates a standard campaign with an LP-Token and a reward token. For the sake of simplicity, let's just take USDC/USDT-LP as staking token and SmarDex token as reward token. This campaign has ID 1.

2. Charles deposits 500_000 USDC/USDT LP into `campaignID` 1 in order to accumulate SmarDex tokens.

3. The contract owner creates a new campaign with a dummy token and USDC/USDT-LP as reward token.

4. The owner deposits 500_000 USDC/USDT-LP into the RewardManager contract in order to seed the reward distribution.

5. The owner creates a reward round for this dummy campaign with an arbitrary block range and an arbitrary `rewardPerBlock` parameter, let's say 100 blocks range and 5000 tokens per block. This now transfers 500_000 LP tokens into the `FarmingRange` contract in order to distribute them as a reward.

6. The owner now changes the `RewardManager` to a malicious contract that has a) a withdrawal method and b) an external call within the `resetAllowance` function. This `RewardManager` has exactly 500 LP tokens as a balance but has none of them approved to the `FarmingRange` contract.

7. The owner now calls `updateRewardInfo` with the same amount of `rewardsPerBlock` and an `endTime` which is exactly 1 block extended. Now the function calls `_transferFromWithAllowance` which attempts to transfer 500 tokens from the `RewardManager` in, however it does not work because no approval is granted. Now an external call to the malicious `RewardManager` is being executed in order to grant the approval:

```
rewardManager.call(abi.encodeWithSignature("resetAllow
ance(uint256)", _campaignID));
```

8. During the resetAllowance call, the RewardManager executes a call to the owner contract which then executes another updateRewardInfo which uses the same end time as the initial value but rewardPerBlock is zero. Due to these parameters, the FarmingRange contract will now refund all distributed tokens back to the RewardManager contract because the rewardPerToken is now zero.

9. After the successful transfer out, it sets the endBlock to the same endBlock as before and the rewardPerToken to zero, effectively mitigating any potential reward distribution.

10. However, the call is not finished, it now continues with the resetAllowance function and approves the StakingToken to the RewardManager because the first call attempted to receive 500 staking tokens in order to meet the desired extension of 1 block.

11. After the tokens have been successfully transferred in:
    _rewardToken.safeTransferFrom(rewardManager, address(this), _amount);

    the new endBlock and rewardPerBlock is now set:
    selectedRewardInfo.endBlock = _endBlock;selectedRewardInfo.rewardPerBlock = _rewardPerBlock;

    The previous settings have now been effectively overriden.

12. The admin now withdraws 500_000 LP tokens while still having the rewardsPerBlock set for his dummy pool, and they can simply drain Charles' staked LP tokens over the following 100 blocks.

Recommendation	Consider making the RewardManager immutable and removing the upgradePrecision function.
Resolution	✔ RESOLVED

Issue #05	First depositor can steal reward tokens
Severity	🔴 HIGH SEVERITY
Description	Campaigns are not updated if there are no staked tokens:

```
if (campaign.totalStaked == 0) {

// if there is no total supply, return and use the campaign's
start block as the last reward block
// so that ALL reward will be distributed.
// however, if the first deposit is out of reward period,
last reward block will be its block number
// in order to keep the multiplier = 0

    if (block.number > _endBlockOf(_campaignID,
block.number)) {
        campaign.lastRewardBlock = block.number;
    }
    return;
}
```

While the `natspec` indicates that the intention is to distribute **all** rewards, there will be a severe issue in the following PoC:

1. The owner adds a new campaign which has ID 3 and the `startBlock=17062006` — the `lastRewardBlock` for this campaign is `17062006`.

2. The owner now adds multiple reward rounds — the first round starts from block 17062006 and runs until block 17064006 with 100e18 tokens per block.

3. Several blocks has passed and nobody has deposited since then, let's say 200 blocks.

4. Charles deposits 1 nominal token, and during this deposit, `campaign.TotalStaked` value is zero. Due to that, Charles' `rewardDebt` will be set as follows:

    ```
    (user.amount * campaign.accRewardPerShare) / (1e20);
    ```

    Which is effectively zero because pools were not updated.

5. Charles now withdraws his 1 nominal token but now, pools are updated because `campaign.TotalStaked` is 1.

6. `accRewardPerShare` is now exactly `100e18 * 200`, which means that Charles will receive `20_000e18` tokens.

This issue occurs every time where no tokens are staked for this pool. Charles now received `20_000e18` tokens retroactively for staking 1 nominal token for 1 block.

Recommendation	Consider always updating the pools when any deposit or withdrawal happens in order to prevent such attacks unless this behavior is desired.
Resolution	✅ **RESOLVED** However, a bug was introduced which is described under Issue #11.

Issue #06	Contract does not support tokens with a fee on transfer
Severity	🔴 HIGH SEVERITY
Description	The contract contains multiple spots which are not designed for tokens with a fee on transfer.
Recommendation	Consider not using such tokens, and if still desired, consider fixing every spot where tokens are being transferred in and adjust the accounting logic accordingly.
Resolution	⚫ ACKNOWLEDGED
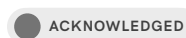
Issue #07	Owner can steal all rewards
Severity	🟠 MEDIUM SEVERITY
Description	Within `updateRewardInfo` and `removeLastRewardInfo`, the owner can decrease or remove a reward round which then transfers the excess reward amount back to the `RewardManager` contract. However, the owner can at any time change the `RewardManager` contract via `setRewardManager`, which allows the owner to steal these tokens.
Recommendation	Consider removing this function. If the team wishes to keep it, consider setting the owner under a strict and reputable multi-signature set-up. It might also make sense to implement a timelock mechanism and a push-pull pattern for changing the `RewardManager` contract.
Resolution	✅ RESOLVED `RewardManager` is immutable.

Issue #08	Adding too many reward epochs can result in a DoS
Severity	🟠 MEDIUM SEVERITY
Description	The contract previously had a limit on the number of reward epochs. This limit has been removed in the latest fix and thus can be used to DoS the pair by adding too many epochs.
Recommendation	Consider adding a limit that prevents any DoS. We recommend setting it to around 50.
Resolution	🔵 PARTIALLY RESOLVED The cap is configurable and set by default to 52. The admin can still DoS if the cap is set too high. Consider doing various fork-tests before changing this variable.

Issue #09	Consecutive calls of `upgradePrecision` can result in a DoS
Severity	🟠 MEDIUM SEVERITY
Description	`upgradePrecision` increases the `accRewardPerShare` by 1e18. While this function itself should never be present in a contract since it allows tokens to be drained, there is another flaw which can occur due to it: all arithmetic operations that use the `accRewardPerShare` variable are then at risk of overflows which essentially breaks the whole contract logic irreversibly.
Recommendation	Consider removing this function, and if under all circumstances the team still wishes to keep it, consider implementing a time-restriction, i.e., allow this function only be callable once every 3 days.
Resolution	✅ RESOLVED The function has been removed.

Issue #10	`setRewardInfoLimit` can cause DoS
Severity	🟠 MEDIUM SEVERITY
Description	`rewardInfoLimit` is set to 52 during contract deployment. However, the owner can freely set it via the `setRewardInfoLimit` function which can ultimately lead to a DoS state within the `updateCampaign` function when too many reward rounds are being added.
Recommendation	Consider determining a reasonable upper value for `rewardInfoLimit` within the `setRewardInfoLimit` function.
Resolution	✅ RESOLVED The function has been removed.

Issue #11	Rewards can get stuck in the contract
Severity	🔴 MEDIUM SEVERITY
Description	If a round is added and rewards have been allocated, these will be distributed amongst all stakers - however, if there is no staker for this round the rewards will simply remain forever in the contract.
Recommendation	Consider if that will become an issue, if yes consider transferring them out for this special case.
Resolution	✅ RESOLVED

Issue #12	Reentrancy call allows owner to add false `rewardInfo`
Severity	🟡 LOW SEVERITY

| **Description** | `addReward` allows the owner of the contract to add reward rounds to a campaign. With the usual business logic, one round can only be followed by another subsequent round which means that each round has its own block range and this will never interfere with the previous or following round.

However, `_transferFromWithAllowance` opens the opportunity for a potential reentrancy call if the `RewardManager` is set to another contract where the `resetAllowance` function executes arbitrary logic.

This can for example be used to add another round with the same `_endBlock` as the previous round or other misbehaviors like a subsequent round having a lower `_endBlock` than the previous round. This would then result clearly in an unexpected behavior with potential other-side effects. |
|---|---|
| **Recommendation** | In order to mitigate any potential undesirable side-effects that can be set by a malicious owner, we recommend adding a `nonReentrant` modifier to the `addRewardInfo` function. This issue potentially also applies to the `updateRewardInfo` function where we recommend the same. |
| **Resolution** | ✅ RESOLVED |

Issue #13	totalRewards is not decreased for next round's refund
Severity	🟡 LOW SEVERITY
Description	totalRewards is decreased as follows: `campaign.totalRewards = _refund ? campaign.totalRewards - _diff : campaign.totalRewards + _diff;` However, the potential decrease is missing: `campaign.rewardToken.safeTransfer(rewardManager, _initialNextTotal - _nextTotal);`
Recommendation	Consider decreasing it for the latter case as well.
Resolution	✅ RESOLVED

Issue #14	Lack of validation
Severity	🟡 LOW SEVERITY
Location	Line 53 `rewardManager = _rewardManager;`
Description	There should be a check that rewardManager cannot be address(0) as it would break important functionality.
Recommendation	Consider validating the variable appropriately.
Resolution	✅ RESOLVED

Issue #15	Lack of safeTransfer usage
Severity	🟡 LOW SEVERITY
Description	The contract uses `safeTransfer` everywhere except in the first attempt to transfer tokens in from the `RewardManager`:
	`try _rewardToken.transferFrom(rewardManager, address(this), _amount) {}`
	This will not work for tokens that return `false` on transfer or tokens that do not have a return value, i.e. USDT on mainnet.
Recommendation	Consider using `safeTransferFrom`.
Resolution	✅ RESOLVED

Issue #16	startBlock can be set to a value in the past
Severity	🟡 LOW SEVERITY
Description	addCampaignInfo does not check whether `startBlock` is in the future. If the `startBlock` is accidentally set to a time in the past, this will then accumulate rewards retroactively.
Recommendation	Consider validating that `startBlock` is in the future.
Resolution	✅ RESOLVED

FarmingRange

Paladin Blockchain Security

Issue #17	Unlisted campaigns can be updated
Severity	INFORMATIONAL
Description	It is possible to call `updateCampaign` with a `campaignID` that has not been added yet. This will set the `lastRewardBlock` of this campaign to the current block.
	While this does not expose any risk because the addition of this campaign will reset `lastRewardBlock` anyway, it might still make sense to prevent users from calling this.
Recommendation	Consider validating that `updateCampaign` can only be called with valid `campaignID`s.
Resolution	RESOLVED

Issue #18	The CEI pattern is not adhered to
Severity	INFORMATIONAL
Description	The contract contains multiple spots where the CEI pattern is not adhered to (https://fravoll.github.io/solidity-patterns/checks_effects_interactions.html).
	As most potential issues have already been mentioned within this report (ie. theft of staking tokens), this issue will has been marked as informational.
Recommendation	Consider adhering to the CEI pattern wherever possible in the contract.
Resolution	RESOLVED

Issue #19	Gas optimizations
Severity	● INFORMATIONAL

Description	L216

`block.number > _lastRewardInfoEndBlock ? block.number : _lastRewardInfoEndBlock,`

The check of `block.number > _lastRewardInfoEndBlock` is redundant as the require on Line 211 makes the function accept the last reward info to be removed only if `_lastRewardInfoEndBlock > block.number`.

L157 - 158
```
nextRewardInfo.rewardPerBlock =
(nextRewardInfo.rewardPerBlock * _initialBlockRange) /
_nextBlockRange;
uint256 _nextTotal = _nextBlockRange *
nextRewardInfo.rewardPerBlock;
```

`nextRewardInfo.rewardPerBlock` should be cached to save some gas.

———

```
for (uint256 _i = 0; _i < _len; ++_i) {
```

Assigning a variable with default value at declaration consumes gas (`uint256 _i = 0`).

Recommendation	Consider implementing the gas optimizations mentioned above.
Resolution	✔ RESOLVED

Issue #20	Lack of events for `upgradePrecision`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the function.
Resolution	RESOLVED The function has been removed.

2.3   Rewards/Staking

`Staking` is a yield-bearing staking contract which receives its rewards via three methods:

1. Incentives from `campaignId 0` within the `FarmingRewards` contract

2. Fees from the liquidity pairs

3. Admin injections

Users can simply deposit `SmarDex` tokens and receive a staking receipt which represents the underlying position. Unlike most other vaults, this receipt is not transferable by the user and can only be redeemed by the actual initiator of the corresponding position.

During each deposit and withdraw, the contract will automatically harvest rewards from the `FarmingRewards` contract which will then be distributed amongst all pool participants.

2.3.1   Issues & Recommendations

Issue #21	The first depositor can steal the shares of subsequent depositors
Severity	🔴 HIGH SEVERITY
Description	The first depositor can steal the shares of all subsequent depositors. Consider the following PoC:

1. Deposit 1 nominal token, receiving 1e18 shares.

2. Withdraw 999999999999999999 shares.

3. Transfer 1000e18 tokens to the staking contract

4. The subsequent depositors' shares will be rounded down to zero.

Recommendation	There are multiple methods for remediation:

1. Burn 1000 shares during the first deposit.

2. Revert in case shares become zero.

3. Follow the virtual shares principle from OZ (https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/extensions/ERC4626.sol)

Resolution	🔵 PARTIALLY RESOLVED

1000 shares have been burned but there is still wiggle room if enough tokens are transferred to the vault. We recommend implementing a mechanism where shares round down to zero. However, the most ideal scenario would be in fact following OpenZeppelin's new method.

Issue #22	Funds can be stuck in the contract forever
Severity	🔴 HIGH SEVERITY
Description	Each deposit and withdrawal executes a harvest, which is a `withdraw(0)` on `campaign[0]` within the `FarmingRange` contract. While this works without any issues, this function can revert if the the `FarmingRange` owner sets `rewardInfoLimit` to a very high value followed by adding several reward rounds which then results in `_updateCampaign[0]` reverting. The same issue can occur if the owner calls `upgradePrecision` often to artificially provoke an overflow.
Recommendation	Consider implementing logic which allows users to emergency withdraw their funds without harvesting in case of emergency. It is important to not do the same for `deposit` because that could be exploited in order to steal yield.
Resolution	✅ RESOLVED

Issue #23	Admin injections can be frontran
Severity	🟠 MEDIUM SEVERITY
Description	The natspec documents the following: Pool will receive SDEX rewards fees by external transfer from admin This special behavior can be abused by a frontrunner in order to steal yield.
Recommendation	While there is no code-specific recommendation for this, we would recommend to do these injections in very small steps in order to mitigate the potential damage by frontrunners.
Resolution	⚫ ACKNOWLEDGED

Issue #24	Lack of validation
Severity	🟡 LOW SEVERITY
Description	During the contract creation, both variables `_smardexToken` and `farming` are set. However, there is no check that these addresses are not `address(0)`.
Recommendation	Consider implementing an `address(0)` check for both variables.
Resolution	✅ RESOLVED

Issue #25	Contract does not adhere to the CEI pattern
Severity	🟣 INFORMATIONAL
Description	Within the `deposit` function, the token transfer is being executed before the accounting logic. While at this point there is no possibility of abuse, it is still considered as best-practice to adhere to the CEI pattern. (https://fravoll.github.io/solidity-patterns/checks_effects_interactions.html)
Recommendation	Consider executing the `transfer` after all effects have been applied.
Resolution	✅ RESOLVED

Issue #26	_sharesToTokens is slightly flawed
Severity	● INFORMATIONAL
Description	_sharesToTokens checks for the condition that `totalShares > 0` and then returns based on that value. However, due to the business logic, it does not make any sense to check for this condition because it is not possible to convert any shares to tokens if there are no shares in existence.
Recommendation	Consider simply removing this check.
Resolution	⬤ ACKNOWLEDGED

2.4   Rewards/AutoSwapper

`AutoSwapper` is a simple swapper contract which regularly receives funds during each mint and burn call from all pairs. A pair simply sends all accumulated fees directly to the swapper contract and then calls `executeWork` which swaps one or both tokens to the Smardex token and transfers it directly to the Staking contract where it is then distributed amongst all vault participants.

2.4.1   Issues & Recommendations

Issue #27	The slippage calculation is flawed
Severity	🟠 MEDIUM SEVERITY
Description	The slippage calculation is done as follows: `uint256 _amountOutWithSlippage = (_params.balanceIn * _params.newPriceAvOut * (AUTOSWAP_SLIPPAGE_BASE - AUTOSWAP_SLIPPAGE)) / (_params.newPriceAvIn * AUTOSWAP_SLIPPAGE_BASE);` However, this calculation assumes that the constant k formula is always applied on the price averages. While this is correct for the first swap, it will not work for consecutive swaps. Below is an example:

1. The fictive reserves are `Xfr = 500e18; Yfr = 500e18`

2. Charles swaps `50e18` `tokenX` to `tokenY`

3. `priceAverage` was not set before, therefore it becomes `Xpa = 500e18; Ypa = 500e18`

4. The swap is executes based on the constant k formula which changes `fictiveReserves` to `Xfr = 550e18; Yfr = 455e18`

5. 150 seconds passes and Charles adds liquidity which proportionally increases the `fictiveReserves`. In order to keep this example simple, we will use the value of 10%
`-> Xfr = 605e18; Yfr = 500.5e18`

6. Now the `executeWork` function is triggered

7. The current `priceAverage` is being calculated based on the aforementioned `priceAverage` and the current fictive reserves which leads to: `Xpa = 605e18, Ypa = 552e18`

8. These `priceAverages` lead to a minimum required output of `44.7e18`

9. However, during the swap, the constant k formula is applied on the `fictiveReserves` which was `Xfr = 605e18; Yfr = 500.5e18`, which leads to an output of `41.3e18` (ignoring the slippage)

This PoC clearly outlines the issue behind this calculation: the PA did not reach the FR ratio yet which results in a higher output when the calculation is done with the PA.

Recommendation	Consider executing the calculation based on the fictive reserves, however, one must potentially compute them beforehand if the PA ratio is near the FR ratio.
Resolution	⬤ ACKNOWLEDGED

Issue #28	Lack of validation
Severity	🟡 LOW SEVERITY
Description	`factory`, `smardexToken` and `stakingAddress` are set during the contract deployment. However, there is no safeguard which ensures that these addresses cannot be set to `address(0)`.
Recommendation	Consider implementing an appropriate validation.
Resolution	✅ RESOLVED

Issue #29	Arbitrage opportunities in increased trading periods
Severity	🟡 LOW SEVERITY
Description	When a swap is executed, the fees are accumulated within the pair. Once a mint or burn of liquidity is called, the fees are transferred to the `feeTo` parameter from the factory which we assume is the `AutoSwapper` contract. This contract has a method that is publicly available which converts the fees into `SmardexToken` and sends it to the staking contract.
	An MEV searcher can take advantage of this opportunity by bundling the flow of:
	1. Looking for a pair that has accumulated a significant amount of fees
	2. Mint a small amount to trigger `_mintFee` from the pair which sends the tokens to `AutoSwapper`
	3. Deposit a significant amount of tokens into the staking contract
	4. Call `executeWork` on the `AutoSwapper`
	5. Withdraw from the Staking contract
	The bundle can yield a good amount of reward for the MEV searcher.
Recommendation	Consider monitoring the fee amounts of the pairs and call `executeWork` frequently, and consider adding a method that lets `feeToSetter` transfer the fees from a pair to `AutoSwapper`, otherwise it is necessary to mint/burn small amounts of liquidity to trigger the transfer.
	An alternative solution would be to just set `feeTo` to an EOA which can then be automatically called once a day to convert them manually and deposit into the staking contract.
Resolution	⬤ ACKNOWLEDGED

Issue #30	Lack of events for `executeWork`
Severity	● INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the function.
Resolution	✔ RESOLVED

2.5   Periphery/SmardexRouter

SmardexRouter allows users to add liquidity, swap tokens and remove liquidity. Unlike UniswapV2Router, it follows the mechanism of UniswapV3 where tokens to be added as liquidity or to be swapped are transferred from the user only via a callback function to the pair.

Users can swap tokens via an exact input amount or via an exact output amount — the internal functions in the libraries getAmountIn and getAmountOut will then calculate the corresponding amounts.

2.5.1   Issues & Recommendations

Issue #31	Ether is refunded to the `_to` address
Severity	🟠 MEDIUM SEVERITY
Description	swapETHForExactTokens allows a user to swap Ether to any arbitrary token while the `_to` address is intended to receive the output amount. However, due to the nature of this function the input amount will most probably not be exact which means that the user might receive a leftover amount back, which is transferred to the `_to` address.
Recommendation	Consider refunding Ether to `msg.sender`.
Resolution	✅ RESOLVED

Issue #32	Router does not support tokens with a fee on transfer
Severity	🔴 MEDIUM SEVERITY
Description	In the case of swapping tokens to exact tokens, the pair calculates how many tokens are necessary to receive the exact output amount and then requests during a callback to transfer this amount from the initiator to the pair. Afterwards, the pair checks that the received amount is in fact sufficient. However, for tokens that have a fee on transfer, the balance will not be be sufficient, effectively reverting the transaction. In the case of swapping exact tokens to tokens, the pair calculates the output amount based on the input amount and then during the callback requests the input amount to be transferred from the initiator to the pair. However, similar to the above scenario, the pair will receive less tokens than expected, effectively reverting the whole transaction.
Recommendation	Consider implementing logic that accounts for the case of the swapping tokens with a fee on transfer.
Resolution	⚫ ACKNOWLEDGED

Issue #33	getAmountIn and getAmountOut might lead to wrong values
Severity	🔴 MEDIUM SEVERITY
Description	getAmountIn and getAmountOut call the corresponding functions within SmardexLibrary. However, care should be taken when doing this as users could accidentally use outdated values from the pair as input parameters. In fact, priceAverage should be always updated before calling any of these functions. This issue is declared as medium since it can result in DoS for external contracts that rely on the correct return amount.
Recommendation	Consider always fetching the updated price-average for these functions — this is especially important if these functions are used by third parties as it may lead to DoS. Fortunately, the Pair already has a getter function for the updated price average: getUpdatedPriceAverage. This function could be used to fetch the return values and then use the return values for getAmountOut and getAmountIn.
Resolution	✅ RESOLVED

Issue #34	Funds can be stuck in the router
Severity	🟡 LOW SEVERITY

Description	_swapExactOut has the following check:
	`if (_to == address(0)) {`
	`_to = address(this); }`
	
	However, for that case, the funds would simply be stuck in the router without any possibility of retrieval. The same applies to _swapExactOutIn.

Recommendation	Consider explicitly reverting for _to = address(0).
Resolution	✅ RESOLVED

Issue #35	Unprivileged ETH can be stolen

Severity	🟣 INFORMATIONAL

Description	The contract disabled the ability to receive Ether from any sources other than the WETH address. However, Ether can be still sent via various methods including a self-destruct transaction.
	If the contract ever receives any Ether, a malicious user can simply drain it by calling `swapExactTokensForETH` with a very small or null-ish token amount. The desired Ether amount will be calculated and transferred from the pair to the router.
	Afterwards, `_unwrapETH` is called which transfers the whole contract balance to the user, effectively taking all Ether in the contract while the user should only receive a very small amount.

Recommendation	This issue is only informational due the low likelihood of it happening. At this point, we do not recommend any change to the contract, however, this case should be kept in mind.

Resolution	⚫ ACKNOWLEDGED

2.6   Periphery/BytesLib

The `BytesLib` library is used within the Path library and is responsible for slicing and type casting bytes arrays.

This contract was forked from https://github.com/GNSPS/solidity-bytes-utils/blob/master/contracts/BytesLib.sol.

2.6.1 Issues & Recommendations

Issue #36	`toUint24 is unused`
Severity	● INFORMATIONAL
Description	While all other functions within this library are used, we could not determine a use case for this function.
	The `uint24` returned may have some ghost bits that could lead to errors when using this value in an assembly block.
Recommendation	Consider removing the `toUint24` function.
Resolution	✔ RESOLVED

2.7 Periphery/Path

The `Path` library is used for manipulating path data, i.e., reversing paths, getting the first pool from a path or skipping one token from a path.

This contract was forked from https://github.com/Uniswap/v3-periphery/blob/main/contracts/libraries/Path.sol.

2.7.1   Issues & Recommendations

Issue #37	Gas optimization
Severity	INFORMATIONAL
Description	encodeTightlyPacked and encodeTightlyPackedReversed use dynamic length arrays as parameters which are declared as memory. Declaring them as calldata instead would save some gas.
Recommendation	Consider declaring the above variables as calldata.
Resolution	RESOLVED

Issue #38	Typographical error
Severity	INFORMATIONAL
Location	Line 36 *// Ignore the first token address. From then on every fee and token offset indicates a pool.*
Description	This comment is outdated.
Recommendation	Consider fixing the typographical error.
Resolution	RESOLVED

2.8 Periphery/PoolAddress

PoolAddress is a simple helper contract that computes a pair address based on the init code hash
=b477a06204165d50e6d795c7c216306290eff5d6015f8b65bb46002a8775b548, the factory and both tokens (sorted). It also has a function to retrieve the pair address directly from the factory (if it already exists).

2.8.1 Issues & Recommendations

No issues found.

2.9 Periphery/PoolHelpers

`PoolHelpers` is a simple helper contract which is responsible for fetching different values like `priceAverage` or `fictiveReserves` from the pair.

2.9.1 Issues & Recommendations

No issues found.

2.10   Core/SmardexPair

`SmardexPair` is the standard pair contract that is deployed by `SmardexFactory`. Usually, it is called by the user via the router to swap tokens, add liquidity and remove liquidity. The logic behind the swaps and the swap fee is handled within the `SmardexLibrary`.

Users can swap tokens by providing the desired output amount via `amountSpecified` as a positive value or providing the desired input amount via amountSpe `amountSpecified` cified as a negative value. The swap logic will then calculate the corresponding input/output amount automatically based on the sign of the value.

The swap logic is described under the `SmardexLibrary` section.

2.10.1   Issues & Recommendations

Issue #39	The protection for `_feeToSwap` does not suffice

Severity	🔴 MEDIUM SEVERITY

Location	<u>Line 512-520</u>
	`_feeTo.call(abi.encodeWithSelector(AUTOSWAP_SELECTOR, token0, token1));`
	// After the _feeTo call, we check if gas is not equal to 0. Though seemingly redundant
	// (as running out of gas would fail the transaction anyway), this require prevents the
	// compiler from ignoring a simple gasleft() expression, which it may perceive as
	// useless if not used elsewhere. The aim here is to consume some gas to ensure the
	// transaction isn't out of gas at this point. This acts as a safeguard against potential
	// exploitation where a low gasLimit prevents the _feeTo call from fully executing.
	`require(gasleft() != 0, "");`

Description	`_feeToSwap` has a check at the end of the function to make sure that the right amount of gas was sent in the `_feeTo.call`. However, the check is not sufficient as the 1/64 rule states that 1 out of the 64th of the gas sent within this function still remains in the parent function, rendering that call to always be true and the attack still possible.

Recommendation	Consider implementing a correct check so that enough gas is left. An example can be seen in the following article: https://medium.com/@wighawag/ethereum-the-concept-of-gas-and-its-dangers-28d0eb809bb2

Resolution	⚫ ACKNOWLEDGED

Issue #40	_feeToSwap can be subjected to sandwich attacks
Severity	🟡 LOW SEVERITY

Description	Each `mint` and `burn` transaction transfers the `fee` and the `feeTo` to the AutoSwapper contract respectively. At the end of each transaction, the internal `_feeToSwap` function is called which calls `executeWork` within the `AutoSwapper` contract.

If a large number of swaps are being executed within a period where no liquidity is added or burned, a malicious user can benefit from the following scenario:

1. Experience a lot of fee accumulation

2. Buy `SmardexToken`

3. Deposit `SmardexToken` in the `Staking` contract

4. Add liquidity in order to trigger `_feeToSwap`

5. `SmardexToken` now gains value due to the swap and the deposited amount within the Staking contract accumulates tokens due to the transfer of the `SmardexToken` to the deposit contract

6. Withdraw from the Staking contract and sell the `SmardexToken`

Due to the low likelihood of periods where no liquidity is added / burned we will classify this issue only as low severity.

Recommendation	Consider if this becomes an issue in practice, and if so, consider implementing an external function which clears the fees and calls `executeWork` in the AutoSwapper contract. Consider implementing a call via a third party that is regularly executed every few hours.

Resolution	⚫ ACKNOWLEDGED

A bot will regularly call the `_feeToSwap` function to prevent the accumulation of too much fees.

Issue #41	_feeToSwap can be subjected to gas griefing
Severity	● INFORMATIONAL

Description

_feeToSwap does an external call to the `feeTo` address from the factory contract.

L488
_feeTo.call(abi.encodeWithSelector(AUTOSWAP_SELECTOR, token0, token1));

The return value of the call is unchecked, meaning that even if it fails, the call will continue to execute.

An attacker can make the external call silently fail by forwarding the exact gas when calling `mint` so that the execution of the `mint` function executes but there is not enough gas to handle the external call. This is possible with EIP-150 and its gas rule where it forwards 63 out of 64 gas left for execution. Therefore that 1/64 gas left finshes executing the function before the external call happens.

As mentioned previously, the attacker can make the external call fail by sending the specific amount of gas needed for the execution of the first call, not using the function correctly. It is a low severity issue because even if the attacker can brick the execution of the external call, the fees are deducted beforehand and the next liquidity event will simply trigger the function

Recommendation

Due to the current design, it is definitely worse to explicitly check the return value of the function call because this might expose a way for DoS within the pair, which can result in the funds stuck within the pair.

We recommend simply acknowledging this issue and keep it in mind.

Resolution ● ACKNOWLEDGED

Issue #42	Gas optimization
Severity	🟣 INFORMATIONAL
Location	L261 `require(_to != _params.token0 && _to != _params.token1, "SmarDex: INVALID_TO");`
Description	This check should be moved to line 182 to save some gas if this requirement was making the transaction to revert.
Recommendation	Consider implementing the gas optimization mentioned above.
Resolution	✔️ RESOLVED

Issue #43	Typographical issue
Severity	🟣 INFORMATIONAL
Location	L355 `function _mintFee() private returns (bool feeOn_)`
Description	The name of the function is ambiguous as it is in fact just transferring the fee amount to `_feeTo`. Consider renaming it to `_transferFee` or `_sendFee`.
Recommendation	Consider fixing the typographical issue.
Resolution	✔️ RESOLVED

2.11   Core/SmardexFactory

`SmardexFactory` is a slightly modified fork of `UniswapV2Factory`. Users can create deterministic pairs and the `feeToSetter` address can determine the `feeTo` address.

2.11.1   Privileged Functions

- setFeeTo

2.11.2   Issues & Recommendations

Issue #44	Missing safeguards on `setFeeToSetter`
Severity	🟡 LOW SEVERITY
Description	The function is missing safeguards for the new value to not be address(0) as setFeeTo cannot be called anymore if setFeeToSetter is set to address(0).
Recommendation	Consider adding checks for address(0).
Resolution	✅ RESOLVED feeToSetter has been removed, now feeTo can be set by the owner.

Issue #45	Owner can redirect all fees to their own address
Severity	🟣 INFORMATIONAL
Description	The fees are either completely disabled or redirected to the Staking address as SmardexToken. However, the owner can simply set the feeTo address to any other address than the AutoSwapper contract which results in the owner receiving all fees.
Recommendation	Consider setting the feeToSetter address to a multi-signature contract.
Resolution	⬤ ACKNOWLEDGED The owner is already a multi-signature contract, and will become a DAO. This is done on purpose to let the DAO decide on eventual future updates regarding AutoSwapper, Stake or fees Strategies.

Issue #46	Lack of events for `setFeeTo` and `setFeeToSetter`
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications.
Recommendation	Add events for the above functions.
Resolution	RESOLVED

2.12   Core/SmardexLibrary

`SmardexLibrary` is a library that is used to calculate amounts during a swap or a liquidity event. The library allows to calculate the fictive and real reserves of the pair during a deposit, withdrawal and a swap.

The calculation of the fictive reserves is a crucial point of SmarDex as it is what provides the Impermanent Loss mitigation and in certain cases, can even provide Impermanent Gain.

The idea behind SmarDex is that a trade that drives the price closer to the average price should have less liquidity while a trade that drives the price away from the average price should have more liquidity. Essentially, the price impact of a swap is lower when swapping away from the average price. This aligns with the desire of "selling high, buying low".

When a swap exceeds the price average, for example if the price was lower or higher than the average price, and that the swap increases or decreases the price but crosses the average price, the swap will be calculated in two steps:

- First, the share of the swap that gets closer to the `priceAverage` will be done with the current fictive reserves in order to reach the `priceAverage`.

- Second, the fictive reserves will now be recomputed and the last share of the swap, that drives the price away from the average price will be done with recomputed fictive reserves.

All swaps use the constant product formula, but with fictive reserves that may be different from the real reserves of the pair, and that may be different from the different steps of the swap.

The library also takes care of the fee calculation — FEES_LP is the fees that the liquidity provider will receive and they are set to 0.05%, while FEES_POOL is the fees for the protocol and they are set to 0.02%. The swap fees are thus set to 0.07%.

The average price logic was introduced to prevent the pair from being exploited by a malicious user that could use the liquidity flow to steal funds from the pool. The average price is a weighted average of the last average price and the current price, weighted by the time. If the last swap was done during the same block, the price is not updated, if it has been more than 300 seconds, the average price is set to the current price, and if it is between 0 and 300 seconds, it returns the weighted average price.

2.12.1 Issues & Recommendations

Issue #47	Swap logic limits swaps
Severity	🟡 LOW SEVERITY
Description	The swap prices are based on `fictiveReserves`, which means that the slippage will be higher than with a standard `UniswapV2` swap due to `fictiveReserves` being significantly lower than the real reserves. Moreover, `applyKConstRuleOut` calculates the new fictive reserves as follows: `newFictiveReserveOut_ = _fictiveReserveOut - amountOut_` This means even if a user decides to accept a huge slippage, it would not work because the fictive reserves would underflow.
Recommendation	A fix for this issue is non-trivial as it would require changing the whole swap logic.
Resolution	⬤ ACKNOWLEDGED

2.13   Core/TransferHelper

TransferHelper is a simple helper contract which is responsible for safely calling approve, `transfer`, `transferFrom` and transferring the native token.

2.13.1   Issues & Recommendations

No issues found.

2.14   Core/SmardexToken

SmardexToken is a simple ERC20 token with `Permit` functionality — it follows the OpenZeppelin standard which is a well-known standard in the industry.

2.14.1   Issues & Recommendations

Issue #48	Outdated ERC20 `Permit` implementation
Severity	🟡 LOW SEVERITY
Description	The contract currently implements an outdated ERC20 Permit implementation from OpenZeppelin. Upgrading to the latest version would get the ERC20 permit from Draft status to production.
Recommendation	Consider upgrading the OpenZeppelin libraries to the latest version.
Resolution	✅ RESOLVED

2.15 AutoSwapperL2

`AutoSwapperL2` is a stripped version of `AutoSwapper` that will be deployed on mainnet. The L2 version sends SDEX to a dead address instead of a staking contract as the L2 version will not support the staking of SDEX.

2.15.1 Issues & Recommendations

No issues found.

2.16　FarmingRangeL2Arbitrum

`FarmingRangeL2Arbitrum` is a copy of the `FarmingRange` contract that modifies the way the `block.number` is retrieved to adapt to the Arbitrum network.

2.16.1　Privileged Functions

- `transferOwnership`
- `renounceOwnership`
- `upgradePrecision`
- `setRewardManager`
- `setRewardInfoLimit`
- `addCampaignInfo`
- `addRewardInfo`
- `addRewardInfoMultiple`
- `updateRewardInfo`
- `updateRewardMultiple`
- `updateCampaignsReward`
- `removeLastRewardInfo`

2.16.2　Issues & Recommendations

No issues found.
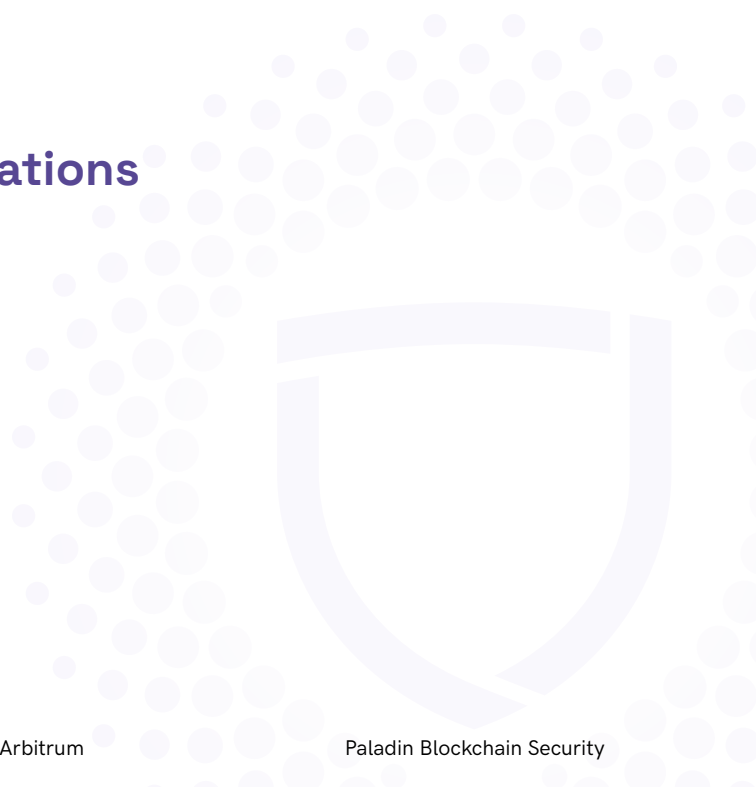
2.17   RewardManagerL2

RewardManagerL2 is a stripped version of `RewardManager`, specially designed to be deployed on a L2 network. This contract does not deploy a staking contract and does not set a campaign with id 0 (the campaign designed for the staking contract in the normal RewardManager) as there is no staking contract present on L2.

2.17.1   Issues & Recommendations

Issue #49	The Staking contract should not be deployed to L2 chains
Severity	🟡 LOW SEVERITY
Description	As the staking contract expects pool id 0 to be the staking reward, if the Staking contract is ever deployed with L2 contracts, this assumption will be false and the Staking contract will malfunction.
Recommendation	Consider carefully never deploying the Staking contract with the L2 contracts.
Resolution	⚫ ACKNOWLEDGED

2.18   RewardManagerL2Arbitrum

RewardManagerL2Arbitrum is a stripped version of RewardManager, specially designed to be deployed on the Arbitrum network. This contract does not deploy a staking contract and does not set a campaign with id 0 (the campaign designed for the staking contract in the normal RewardManager) as there is no staking contract present on Arbitrum.

2.18.1   Issues & Recommendations

Issue #50	The Staking contract should not be deployed to L2 chains
Severity	🟡 LOW SEVERITY
Description	As the staking contract expects pool id 0 to be the staking reward, if the Staking contract is ever deployed with L2 contracts, this assumption will be false and the Staking contract will malfunction.
Recommendation	Consider carefully never deploying the Staking contract with the L2 contracts.
Resolution	⚫ ACKNOWLEDGED