

DEX & Token Launch

Smart Contract Audit Report
Prepared for Arken Finance



Date Issued: Sep 1, 2022
Project ID: AUDIT2022045
Version: v1.0
Confidentiality Level: Public

Report Information

| | |
|-----------------------|---|
| Project ID | AUDIT2022045 |
| Version | v1.0 |
| Client | Arken Finance |
| Project | DEX & Token Launch |
| Auditor(s) | Peeraphut Punsuwan Phitchakorn Apiratisakul Sorawish Laovakul |
| Author(s) | Phitchakorn Apiratisakul |
| Reviewer | Natsasit Jirathammanuwat |
| Confidentiality Level | Public |

Version History

| Version | Date | Description | Author(s) |
|---------|-------------|-------------|--------------------------|
| 1.0 | Sep 1, 2022 | Full report | Phitchakorn Apiratisakul |

Contact Information

| | |
|----------|--|
| Company | Inspex |
| Phone | (+66) 90 888 7186 |
| Telegram | t.me/inspexco |
| Email | audit@inspex.co |

Table of Contents

| | |
|--|-----------|
| 1. Executive Summary | 1 |
| 1.1. Audit Result | 1 |
| 1.2. Disclaimer | 1 |
| 2. Project Overview | 2 |
| 2.1. Project Introduction | 2 |
| 2.2. Scope | 3 |
| 3. Methodology | 5 |
| 3.1. Test Categories | 5 |
| 3.2. Audit Items | 6 |
| 3.3. Risk Rating | 8 |
| 4. Summary of Findings | 9 |
| 5. Detailed Findings Information | 11 |
| 5.1. Centralized Control of State Variable | 11 |
| 5.2. Division Before Multiplication | 13 |
| 5.3. Outdated Compiler Version | 16 |
| 5.4. Insufficient Logging for Privileged Functions | 18 |
| 5.5. Improper Function Visibility | 20 |
| 6. Appendix | 22 |
| 6.1. About Inspex | 22 |

1. Executive Summary

As requested by Arken Finance, Inspex team conducted an audit to verify the security posture of the DEX & Token Launch smart contracts between Aug 26, 2022 and Aug 30, 2022. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of DEX & Token Launch smart contracts. Static code analysis, dynamic analysis, and manual review were done in conjunction to identify smart contract vulnerabilities together with technical & business logic flaws that may be exposed to the potential risk of the platform and the ecosystem. Practical recommendations are provided according to each vulnerability found and should be followed to remediate the issue.

1.1. Audit Result

In the initial audit, Inspex found 1 high, 1 medium, 1 very low, and 2 info-severity issues. With the project team's prompt response in resolving the issues found by Inspex, all issues were resolved or mitigated in the reassessment. Therefore, Inspex trusts that DEX & Token Launch smart contracts have high-level protections in place to be safe from most attacks.



1.2. Disclaimer

This security audit is not produced to supplant any other type of assessment and does not guarantee the discovery of all security vulnerabilities within the scope of the assessment. However, we warrant that this audit is conducted with goodwill, professional approach, and competence. Since an assessment from one single party cannot be confirmed to cover all possible issues within the smart contract(s), Inspex suggests conducting multiple independent assessments to minimize the risks. Lastly, nothing contained in this audit report should be considered as investment advice.

2. Project Overview

2.1. Project Introduction

Arken Finance platform is the Automated Market Maker (AMM) protocol, forked from Uniswap V2 on the Binance Smart Chain (BSC). Users can swap ERC20 tokens with the liquidity pool of the platform, which is also provided by the users, to gain a part of the swapping fee and the \$ARKEN. The platform serves the simplest way to provide liquidity by combining the swap, add liquidity, and stake LP all in one transaction.

Scope Information:

| | |
|-----------------------------|---|
| Project Name | DEX & Token Launch |
| Website | https://arken.finance/ |
| Smart Contract Type | Ethereum Smart Contract |
| Chain | BNB Smart Chain |
| Programming Language | Solidity |
| Category | AMM, Yield Farming, Token |

Audit Information:

| | |
|--------------------------|-----------------------------|
| Audit Method | Whitebox |
| Audit Date | Aug 26, 2022 - Aug 30, 2022 |
| Reassessment Date | Sep 1, 2022 |

The audit method can be categorized into two types depending on the assessment targets provided:

1. **Whitebox:** The complete source code of the smart contracts are provided for the assessment.
2. **Blackbox:** Only the bytecodes of the smart contracts are provided for the assessment.

2.2. Scope

The following smart contracts were audited and reassessed by Inspex in detail:

Initial Audit: (Commit: 4aacac74293a6864a8dc5d4fde9803816a04637c)

| Contract | Location (URL) |
|------------------|---|
| ArkenPair | https://github.com/arken-lab/arken-swap-protocol/blob/4aacac7429/contracts/dex/ArkenPair.sol |
| ArkenPairFactory | https://github.com/arken-lab/arken-swap-protocol/blob/4aacac7429/contracts/dex/ArkenPairFactory.sol |
| ArkenRouter | https://github.com/arken-lab/arken-swap-protocol/blob/4aacac7429/contracts/dex/ArkenRouter.sol |
| ArkenSmithy | https://github.com/arken-lab/arken-swap-protocol/blob/4aacac7429/contracts/token/ArkenSmithy.sol |
| ArkenSmithyPool | https://github.com/arken-lab/arken-swap-protocol/blob/4aacac7429/contracts/token/ArkenSmithyPool.sol |
| ArkenStaker | https://github.com/arken-lab/arken-swap-protocol/blob/4aacac7429/contracts/token/ArkenStaker.sol |
| ArkenToken | https://github.com/arken-lab/arken-swap-protocol/blob/4aacac7429/contracts/token/ArkenToken.sol |

Reassessment: (Commit: 54ee7192e9f9c71b4a84eade1fd2480c48ada215)

| Contract | Location (URL) |
|------------------|---|
| ArkenPair | https://github.com/arken-lab/arken-swap-protocol/blob/54ee7192e9/contracts/dex/ArkenPair.sol |
| ArkenPairFactory | https://github.com/arken-lab/arken-swap-protocol/blob/54ee7192e9/contracts/dex/ArkenPairFactory.sol |
| ArkenRouter | https://github.com/arken-lab/arken-swap-protocol/blob/54ee7192e9/contracts/dex/ArkenRouter.sol |
| ArkenSmithy | https://github.com/arken-lab/arken-swap-protocol/blob/54ee7192e9/contracts/token/ArkenSmithy.sol |
| ArkenSmithyPool | https://github.com/arken-lab/arken-swap-protocol/blob/54ee7192e9/contracts/token/ArkenSmithyPool.sol |
| ArkenStaker | https://github.com/arken-lab/arken-swap-protocol/blob/54ee7192e9/contracts/token/ArkenStaker.sol |

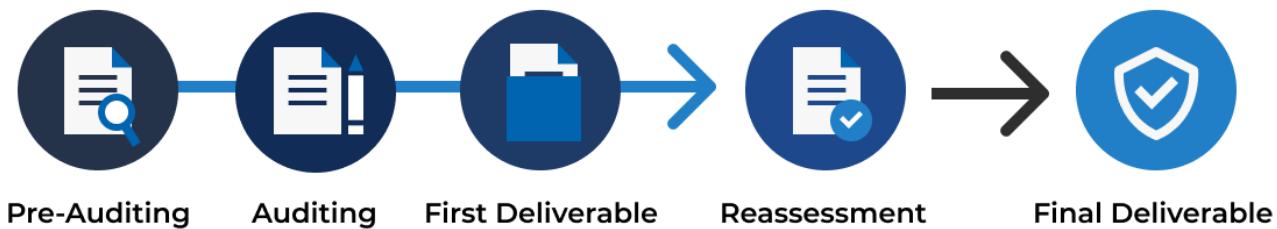
| | |
|------------|--|
| ArkenToken | https://github.com/arken-lab/arken-swap-protocol/blob/54ee7192e9/contracts /token/ArkenToken.sol |
|------------|--|

The assessment scope covers only the in-scope smart contracts and the smart contracts that they inherit from.

3. Methodology

Inspex conducts the following procedure to enhance the security level of our clients' smart contracts:

1. **Pre-Auditing:** Getting to understand the overall operations of the related smart contracts, checking for readiness, and preparing for the auditing
2. **Auditing:** Inspecting the smart contracts using automated analysis tools and manual analysis by a team of professionals
3. **First Deliverable and Consulting:** Delivering a preliminary report on the findings with suggestions on how to remediate those issues and providing consultation
4. **Reassessment:** Verifying the status of the issues and whether there are any other complications in the fixes applied
5. **Final Deliverable:** Providing a full report with the detailed status of each issue



3.1. Test Categories

Inspex smart contract auditing methodology consists of both automated testing with scanning tools and manual testing by experienced testers. We have categorized the tests into 3 categories as follows:

1. **General Smart Contract Vulnerability (General)** - Smart contracts are analyzed automatically using static code analysis tools for general smart contract coding bugs, which are then verified manually to remove all false positives generated.
2. **Advanced Smart Contract Vulnerability (Advanced)** - The workflow, logic, and the actual behavior of the smart contracts are manually analyzed in-depth to determine any flaws that can cause technical or business damage to the smart contracts or the users of the smart contracts.
3. **Smart Contract Best Practice (Best Practice)** - The code of smart contracts is then analyzed from the development perspective, providing suggestions to improve the overall code quality using standardized best practices.

3.2. Audit Items

The testing items checked are based on our Smart Contract Security Testing Guide (SCSTG) v1.0 (https://github.com/InspexCo/SCSTG/releases/download/v1.0/SCSTG_v1.0.pdf) which covers most prevalent risks in smart contracts. The latest version of the document can also be found at <https://inspex.gitbook.io/testing-guide/>.

The following audit items were checked during the auditing activity:

| Testing Category | Testing Items |
|-------------------------------|--|
| 1. Architecture and Design | 1.1. Proper measures should be used to control the modifications of smart contract logic 1.2. The latest stable compiler version should be used 1.3. The circuit breaker mechanism should not prevent users from withdrawing their funds 1.4. The smart contract source code should be publicly available 1.5. State variables should not be unfairly controlled by privileged accounts 1.6. Least privilege principle should be used for the rights of each role |
| 2. Access Control | 2.1. Contract self-destruct should not be done by unauthorized actors 2.2. Contract ownership should not be modifiable by unauthorized actors 2.3. Access control should be defined and enforced for each actor roles 2.4. Authentication measures must be able to correctly identify the user 2.5. Smart contract initialization should be done only once by an authorized party 2.6. tx.origin should not be used for authorization |
| 3. Error Handling and Logging | 3.1. Function return values should be checked to handle different results 3.2. Privileged functions or modifications of critical states should be logged 3.3. Modifier should not skip function execution without reverting |
| 4. Business Logic | 4.1. The business logic implementation should correspond to the business design 4.2. Measures should be implemented to prevent undesired effects from the ordering of transactions 4.3. msg.value should not be used in loop iteration |
| 5. Blockchain Data | 5.1. Result from random value generation should not be predictable 5.2. Spot price should not be used as a data source for price oracles 5.3. Timestamp should not be used to execute critical functions 5.4. Plain sensitive data should not be stored on-chain 5.5. Modification of array state should not be done by value 5.6. State variable should not be used without being initialized |

| Testing Category | Testing Items |
|------------------------|---|
| 6. External Components | 6.1. Unknown external components should not be invoked 6.2. Funds should not be approved or transferred to unknown accounts 6.3. Reentrant calling should not negatively affect the contract states 6.4. Vulnerable or outdated components should not be used in the smart contract 6.5. Deprecated components that have no longer been supported should not be used in the smart contract 6.6. Delegatecall should not be used on untrusted contracts |
| 7. Arithmetic | 7.1. Values should be checked before performing arithmetic operations to prevent overflows and underflows 7.2. Explicit conversion of types should be checked to prevent unexpected results 7.3. Integer division should not be done before multiplication to prevent loss of precision |
| 8. Denial of Services | 8.1. State changing functions that loop over unbounded data structures should not be used 8.2. Unexpected revert should not make the whole smart contract unusable 8.3. Strict equalities should not cause the function to be unusable |
| 9. Best Practices | 9.1. State and function visibility should be explicitly labeled 9.2. Token implementation should comply with the standard specification 9.3. Floating pragma version should not be used 9.4. Builtin symbols should not be shadowed 9.5. Functions that are never called internally should not have public visibility 9.6. Assert statement should not be used for validating common conditions |

3.3. Risk Rating

OWASP Risk Rating Methodology (https://owasp.org/www-community/OWASP_Risk_Rating_Methodology) is used to determine the severity of each issue with the following criteria:

- **Likelihood:** a measure of how likely this vulnerability is to be uncovered and exploited by an attacker
- **Impact:** a measure of the damage caused by a successful attack

Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

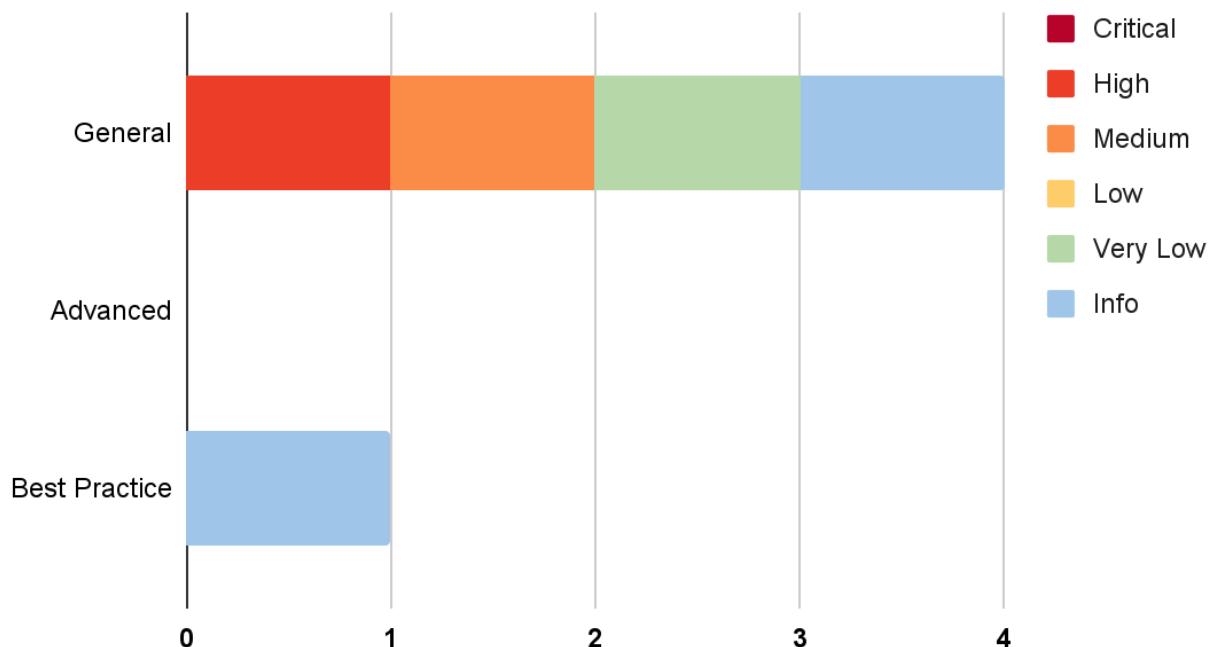
Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would be categorized as **Info**.

| Impact | Likelihood | Low | Medium | High |
|--------|------------|----------|--------|----------|
| Low | | Very Low | Low | Medium |
| Medium | | Low | Medium | High |
| High | | Medium | High | Critical |

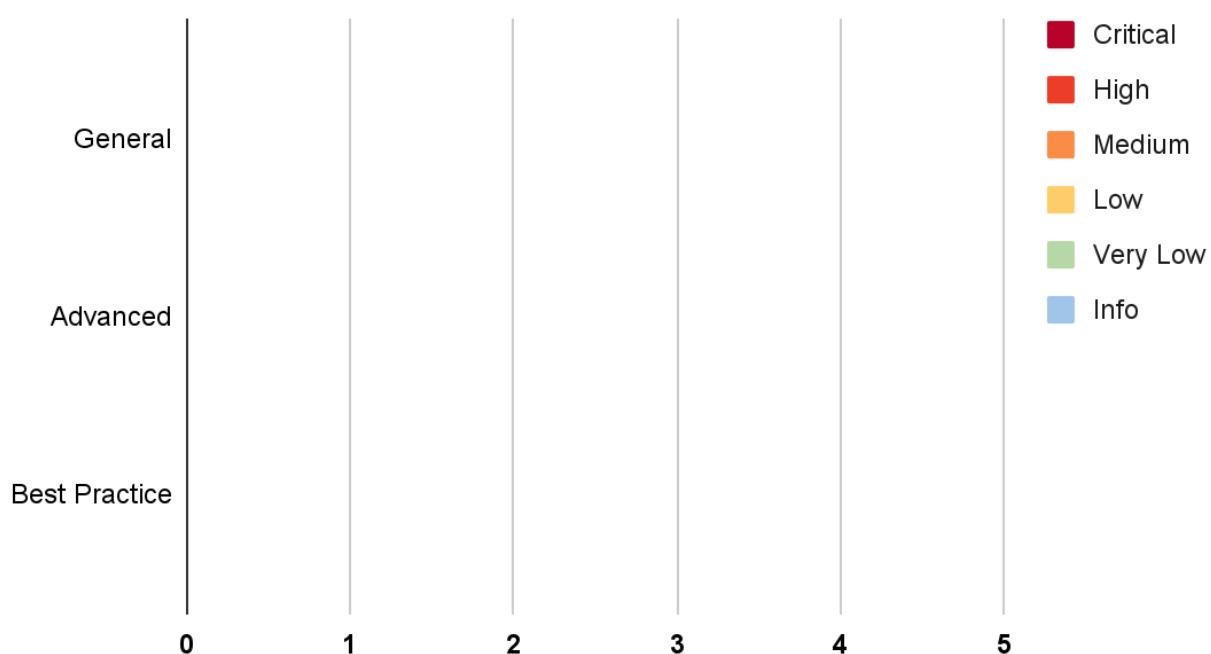
4. Summary of Findings

The following charts show the number of the issues found during the assessment and the issues acknowledged in the reassessment, categorized into three categories: **General**, **Advanced**, and **Best Practice**.

Assessment:



Reassessment:



The statuses of the issues are defined as follows:

| Status | Description |
|--------------------|---|
| Resolved | The issue has been resolved and has no further complications. |
| Resolved * | The issue has been resolved with mitigations and clarifications. For the clarification or mitigation detail, please refer to Chapter 5. |
| Acknowledged | The issue's risk has been acknowledged and accepted. |
| No Security Impact | The best practice recommendation has been acknowledged. |

The information and status of each issue can be found in the following table:

| ID | Title | Category | Severity | Status |
|---------|---|---------------|----------|------------|
| IDX-001 | Centralized Control of State Variable | General | High | Resolved * |
| IDX-002 | Division Before Multiplication | General | Medium | Resolved |
| IDX-003 | Outdated Compiler Version | General | Very Low | Resolved |
| IDX-004 | Insufficient Logging for Privileged Functions | General | Info | Resolved |
| IDX-005 | Improper Function Visibility | Best Practice | Info | Resolved |

* The mitigations or clarifications by Arken Finance can be found in Chapter 5.

5. Detailed Findings Information

5.1. Centralized Control of State Variable

| | |
|----------|---|
| ID | IDX-001 |
| Target | ArkenSmithy ArkenSmithyPool ArkenPairFactory |
| Category | General Smart Contract Vulnerability |
| CWE | CWE-284: Improper Access Control |
| Risk | <p>Severity: High</p> <p>Impact: High The controlling authorities can change the critical state variables to gain additional profit. Thus, it is unfair to the other users.</p> <p>Likelihood: Medium There is nothing to restrict the changes from being done; however, this action can only be done by the privileged roles.</p> |
| Status | <p>Resolved *</p> <p>The Arken Finance team has confirmed to mitigate this issue by using a time lock mechanism with a minimum of 24 hours for the affected roles.</p> |

5.1.1. Description

Critical state variables can be updated at any time by the controlling authorities. Changes in these variables can cause impacts to the users, so the users should accept or be notified before these changes are effective.

However, there is currently no constraint to prevent the authorities from modifying these variables without notifying the users.

The controllable privileged state update functions are as follows:

| File | Contract | Function | Modifier/role |
|-----------------------------|-----------------|--------------------------|--------------------|
| ArkenSmithy.sol (L: 72) | ArkenSmithy | add() | onlyOwner |
| ArkenSmithy.sol (L: 103) | ArkenSmithy | set() | onlyOwner |
| ArkenSmithy.sol (L: 124) | ArkenSmithy | setPoolAdmin() | poolAdmins, _owner |
| ArkenSmithy.sol (L: 137) | ArkenSmithy | massSet() | onlyOwner |
| ArkenSmithyPool.sol (L: 56) | ArkenSmithyPool | emergencyAdminWithdraw() | onlyOwner |

| | | | |
|--------------------------|-------------|-----------|-----------|
| ArkenStaker.sol (L: 104) | ArkenStaker | add() | onlyOwner |
| ArkenStaker.sol (L: 131) | ArkenStaker | massSet() | onlyOwner |

5.1.2. Remediation

In the ideal case, the critical state variables should not be modifiable to keep the integrity of the smart contract. However, if modifications are needed, Inspect suggests implementing a community-run smart contract governance to control the use of these functions.

If removing the functions or implementing the smart contract governance is not possible, Inspect suggests mitigating the risk of this issue by using a time lock mechanism to delay the changes for a reasonable amount of time e.g., 24 hours.

5.2. Division Before Multiplication

| | |
|----------|--|
| ID | IDX-002 |
| Target | ArkenStaker |
| Category | General Smart Contract Vulnerability |
| CWE | CWE-682: Incorrect Calculation |
| Risk | <p>Severity: Medium</p> <p>Impact: Low</p> <p>The rounding error can cause the rewards to be slightly miscalculated.</p> <p>Likelihood: High</p> <p>The result of multiplication divided by the denominator is likely to result in a decimal value.</p> |
| Status | <p>Resolved</p> <p>The Arken Finance team has resolved this issue as suggested by modifying the affected lines of code in commit 54ee7192e9f9c71b4a84eade1fd2480c48ada215.</p> |

5.2.1. Description

Solidity supports only integer values but not floating point values. The division of integers can result in a value with decimal points, which will be rounded off. This rounding error can cause the calculation to be different from what it should be, especially when that value is later multiplied with another value.

For example, in the `updatePool()` function, the `pool.accPerShare` value in line 184 is calculated by multiplying `arkenPerBlock()` by `pool.allocPoint` and `multiplier`, dividing the result by `totalAllocPoint`, multiplying it by `ACC_ARKEN_PRECISION`, and then dividing it by `lpSupply`. The rounding error caused by the division is amplified in the multiplication and can increase the amount of miscalculation.

ArkenStaker.sol

```

175 function updatePool(uint256 _pid) public returns (PoolInfo memory pool) {
176     pool = poolInfos[_pid];
177     if (block.number > pool.lastRewardBlock) {
178         uint256 lpSupply = pool.totalShare;
179         if (lpSupply > 0 && totalAllocPoint > 0) {
180             uint256 multiplier = block.number - pool.lastRewardBlock;
181             uint256 arkenReward = (multiplier *
182                 arkenPerBlock() *
183                 pool.allocPoint) / totalAllocPoint;
184             pool.accPerShare =
185                 pool.accPerShare +
186                 ((arkenReward * ACC_ARKEN_PRECISION) / lpSupply);

```

```

187     }
188     pool.lastRewardBlock = block.number;
189     poolInfos[_pid] = pool;
190     emit UpdatePool(
191         _pid,
192         pool.lastRewardBlock,
193         lpSupply,
194         pool.accPerShare
195     );
196 }
197 }
```

The functions that use division before multiplication are as follows:

| File | Contract | Function |
|-------------------------|-------------|----------------|
| ArkenStaker.sol (L:175) | ArkenStaker | updatePool() |
| ArkenStaker.sol (L:320) | ArkenStaker | pendingArken() |

5.2.2. Remediation

Inspex suggests modifying the affected lines of code to perform multiplication before division, for example:

ArkenStaker.sol

```

175 function updatePool(uint256 _pid) public returns (PoolInfo memory pool) {
176     pool = poolInfos[_pid];
177     if (block.number > pool.lastRewardBlock) {
178         uint256 lpSupply = pool.totalShare;
179         if (lpSupply > 0 && totalAllocPoint > 0) {
180             uint256 multiplier = block.number - pool.lastRewardBlock;
181             uint256 arkenReward = (multiplier *
182                 arkenPerBlock() *
183                 pool.allocPoint) * ACC_ARKEN_PRECISION / totalAllocPoint;
184             pool.accPerShare =
185                 pool.accPerShare +
186                 (arkenReward / lpSupply);
187         }
188         pool.lastRewardBlock = block.number;
189         poolInfos[_pid] = pool;
190         emit UpdatePool(
191             _pid,
192             pool.lastRewardBlock,
193             lpSupply,
194             pool.accPerShare
195         );
196     }
```

197 }

5.3. Outdated Compiler Version

| | |
|----------|---|
| ID | IDX-003 |
| Target | ArkenPair ArkenPairFactory ArkenRouter ArkenSmithy ArkenSmithyPool ArkenStaker ArkenToken |
| Category | General Smart Contract Vulnerability |
| CWE | CWE-1104: Use of Unmaintained Third Party Components |
| Risk | <p>Severity: Very Low</p> <p>Impact: Low</p> <p>From the list of known Solidity bugs, direct impact cannot be caused from those bugs themselves.</p> <p>Likelihood: Low</p> <p>From the list of known Solidity bugs, it is very unlikely that those bugs would affect these smart contracts.</p> |
| Status | <p>Resolved</p> <p>The Arken Finance team has resolved this issue by changing the solidity compiler version to be the latest version in commit 54ee7192e9f9c71b4a84eade1fd2480c48ada215.</p> |

5.3.1. Description

The solidity compiler versions declared in the smart contracts were outdated. These versions have publicly known inherent bugs (<https://docs.soliditylang.org/en/latest/bugs.html>) that may potentially be used to cause damage to the smart contracts or the users of the smart contracts.

ArkenPair.sol

```

1 // SPDX-License-Identifier: GPL-3.0-or-later
2
3 pragma solidity =0.8.11;

```

The following table contains all targets which the outdated compiler version is declared.

| Contract | Version |
|------------------|---------|
| ArkenPair | 0.8.11 |
| ArkenPairFactory | 0.8.11 |
| ArkenRouter | 0.8.11 |
| ArkenSmithy | 0.8.11 |
| ArkenSmithyPool | 0.8.11 |
| ArkenStaker | 0.8.11 |
| ArkenToken | 0.8.11 |

5.3.2. Remediation

Inspex suggests upgrading the solidity compiler to the latest stable version (<https://github.com/ethereum/solidity/releases>). At the time of audit, the latest stable versions of Solidity compiler in major 0.8 is 0.8.16.

5.4. Insufficient Logging for Privileged Functions

| | |
|----------|--|
| ID | IDX-004 |
| Target | ArkenPairFactory |
| Category | General Smart Contract Vulnerability |
| CWE | CWE-778: Insufficient Logging |
| Risk | Severity: Info Impact: None Likelihood: None |
| Status | Resolved The Arken Finance team has resolved this issue as suggested by emitting events for the execution of privileged functions in commit 54ee7192e9f9c71b4a84eade1fd2480c48ada215. |

5.4.1. Description

Privileged functions that are executable by the controlling parties are not logged properly by emitting events. Without events, it is not easy for the public to monitor the execution of those privileged functions, allowing the controlling parties to perform actions that cause big impacts on the platform.

For example, the current `feeToSetter` role can update the `feeToSetter` address by executing `setFeeToSetter()` function in `ArkenPairFactory` contract, and no events are emitted, resulting in the user did not notice that the `feeToSetter` state has been changed.

ArkenPairFactory.sol

```

55  function setFeeToSetter(address _feeToSetter) external override {
56    require(msg.sender == feeToSetter, 'ArkenPairFactory: FORBIDDEN');
57    feeToSetter = _feeToSetter;
58 }
```

The privileged functions without sufficient logging are as follows:

| File | Contract | Function |
|-----------------------------|------------------|------------------|
| ArkenPairFactory.sol (L:50) | ArkenPairFactory | setFeeTo() |
| ArkenPairFactory.sol (L:55) | ArkenPairFactory | setFeeToSetter() |

5.4.2. Remediation

Inspex suggests emitting events for the execution of privileged functions, for example:

ArkenPairFactory.sol

```
55 event SetFeeToSetter(address _feeToSetter);
56 function setFeeToSetter(address _feeToSetter) external override {
57     require(msg.sender == feeToSetter, 'ArkenPairFactory: FORBIDDEN');
58     feeToSetter = _feeToSetter;
59     emit SetFeeToSetter(address _feeToSetter);
60 }
```

5.5. Improper Function Visibility

| | |
|----------|--|
| ID | IDX-005 |
| Target | ArkenRouter ArkenStaker |
| Category | Smart Contract Best Practice |
| CWE | CWE-710: Improper Adherence to Coding Standards |
| Risk | Severity: Info Impact: None Likelihood: None |
| Status | Resolved The Arken Finance team has resolved this issue as suggested by changing all functions' visibility to external if they are not called from any internal function in commit 54ee7192e9f9c71b4a84eade1fd2480c48ada215. |

5.5.1. Description

Public functions that are never called internally by the contract itself should have external visibility. This improves the readability of the contract, allowing clear distinction between functions that are externally used and functions that are also called internally.

The following source code shows that the `approve()` function visibility in the `ArkenStaker` contact is set to public and it is never called from any internal functions.

ArkenStaker.sol

```

375 function approve(
376     uint256 _pid,
377     address _spender,
378     uint256 _amount
379 ) public {
380     _approve(_pid, msg.sender, _spender, _amount);
381 }
```

The following table contains all functions that have public visibility and are never called from any internal functions.

| Target | Contract | Function |
|-------------------------|-------------|----------------------|
| ArkenRouter.sol (L:530) | ArkenRouter | removeLiquidity() |
| ArkenRouter.sol (L:555) | ArkenRouter | removeLiquidityETH() |
| ArkenStaker.sol (L:104) | ArkenStaker | add() |
| ArkenStaker.sol (L:131) | ArkenStaker | massSet() |
| ArkenStaker.sol (L:375) | ArkenStaker | approve() |

5.5.2. Remediation

Inspex suggests changing all functions' visibility to external if they are not called from any internal function, as shown in the following example:

ArkenStaker.sol

```

375 function approve(
376     uint256 _pid,
377     address _spender,
378     uint256 _amount
379 ) external {
380     _approve(_pid, msg.sender, _spender, _amount);
381 }
```

6. Appendix

6.1. About Inspect



CYBERSECURITY PROFESSIONAL SERVICE

Inspect is formed by a team of cybersecurity experts highly experienced in various fields of cybersecurity. We provide blockchain and smart contract professional services at the highest quality to enhance the security of our clients and the overall blockchain ecosystem.

Follow Us On:

| | |
|----------|---|
| Website | https://inspex.co |
| Twitter | @InspexCo |
| Facebook | https://www.facebook.com/InspexCo |
| Telegram | @inspex_announcement |



inspex

CYBERSECURITY PROFESSIONAL SERVICE