ArkenDexV3 & ArkenApprove

Smart Contract Audit Report
Prepared for Arken Finance

Date Issued:	Feb 8, 2022
Project ID:	AUDIT2022006
Version:	v1.0
Confidentiality Level:	Public

inspeX
CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2022006
Version	v1.0
Client	Arken Finance
Project	ArkenDexV3 & ArkenApprove
Auditor(s)	Peeraphut Punsuwan Natsasit Jirathammanuwat
Author(s)	Natsasit Jirathammanuwat
Reviewer	Patipon Suwanbol
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Feb 8, 2022	Full report	Natsasit Jirathammanuwat

Contact Information

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

Table of Contents

1. Executive Summary

As requested by Arken Finance, Inspex team conducted an audit to verify the security posture of the ArkenDexV3 & ArkenApprove smart contracts on Jan 28, 2022. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of ArkenDexV3 & ArkenApprove 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 2 high and 5 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 ArkenDexV3 & ArkenApprove 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 offers the all-in-one trading tool for DEX traders to monitor tokens and synthetic assets, forecast the market movement, and trade at the BEST RATE. Arken aggregates multiple DEXs into a single platform and tackles the principal vulnerability of today's DEXs.

ArkenDexV3 & ArkenApprove contracts are used for handling the swapping on multiple DEXs and provide the functionality to split trade into multiple routes at the best rate.

Scope Information:

Project Name	ArkenDexV3 & ArkenApprove
Website	https://arken.finance/
Smart Contract Type	Ethereum Smart Contract
Chain	Ethereum, Binance Smart Chain, Polygon, Arbitrum One, Avalanche
Programming Language	Solidity

Audit Information:

Audit Method	Whitebox
Audit Date	Jan 28, 2022
Reassessment Date	Feb 7, 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: 4144548e447c98a69c768e41c92b69d5bb2cb320)

Contract	Location (URL)
ArkenDexV3	https://github.com/arken-lab/arken-swap-protocol/blob/4144548e44/contracts/ArkenDexV3.sol
ArkenApprove	https://github.com/arken-lab/arken-swap-protocol/blob/4144548e44/contracts/ArkenApprove.sol

Reassessment: (Commit: b34ccb587d7578ef15870803bf1324eedc731b99)

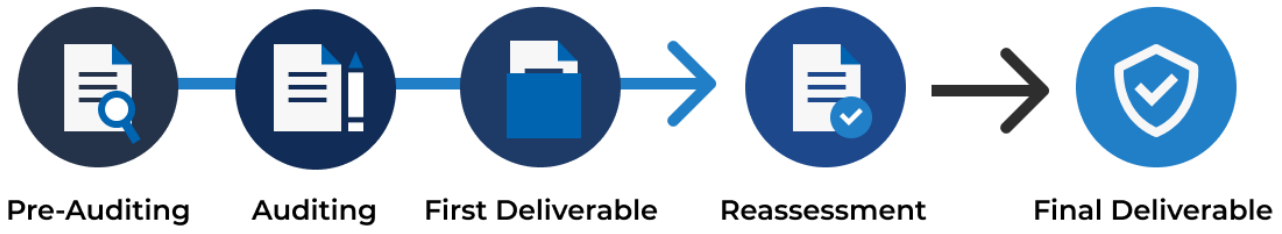
Contract	Location (URL)
ArkenDexV3	https://github.com/arken-lab/arken-swap-protocol/blob/b34ccb587d/contracts/ArkenDexV3.sol
ArkenApprove	https://github.com/arken-lab/arken-swap-protocol/blob/b34ccb587d/contracts/ArkenApprove.sol

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 following audit items were checked during the auditing activity.

General
Reentrancy Attack
Integer Overflows and Underflows
Unchecked Return Values for Low-Level Calls
Bad Randomness
Transaction Ordering Dependence
Time Manipulation
Short Address Attack
Outdated Compiler Version
Use of Known Vulnerable Component
Deprecated Solidity Features
Use of Deprecated Component
Loop with High Gas Consumption
Unauthorized Self-destruct
Redundant Fallback Function
Insufficient Logging for Privileged Functions
Invoking of Unreliable Smart Contract
Use of Upgradable Contract Design
Centralized Control of State Variable
Advanced
Business Logic Flaw
Ownership Takeover
Broken Access Control
Broken Authentication

Improper Kill-Switch Mechanism
Improper Front-end Integration
Insecure Smart Contract Initiation
Denial of Service
Improper Oracle Usage
Memory Corruption
Best Practice
Use of Variadic Byte Array
Implicit Compiler Version
Implicit Visibility Level
Implicit Type Inference
Function Declaration Inconsistency
Token API Violation
Best Practices Violation

3.3. Risk Rating

OWASP Risk Rating Methodology[1] 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

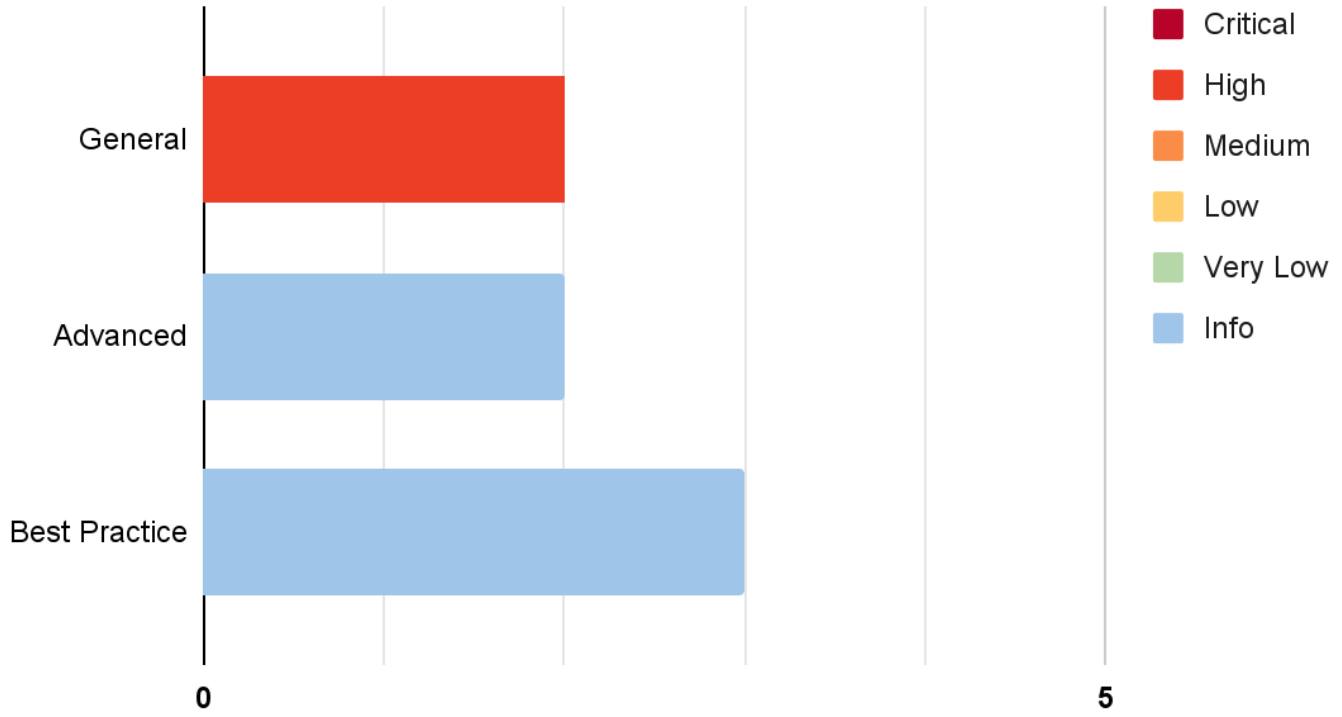
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

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

From the assessments, Inspex has found 7 issues in three categories. The following chart shows the number of the issues categorized into three categories: **General**, **Advanced**, and **Best Practice**.



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	Use of Upgradable Contract Design	General	**High**	Resolved *
IDX-002	Centralized Control of State Variable	General	**High**	Resolved *
IDX-003	Swapping Without Fees	Advance	**Info**	Resolved
IDX-004	Missing Caller Address Verification in uniswapV3SwapCallback()	Advance	**Info**	Resolved
IDX-005	Inexplicit Solidity Compiler Version	Best Practice	**Info**	Resolved
IDX-006	Incorrect Logging Parameter for Privileged Functions	Best Practice	**Info**	Resolved
IDX-007	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. Use of Upgradable Contract Design

ID	IDX-001
Target	ArkenApprove
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** High **Impact:** High The logic of affected contracts can be arbitrarily changed. This allows the proxy owner to perform malicious actions e.g., stealing the user funds anytime they want. **Likelihood:** Medium This action can be performed by the proxy owner without any restriction.
Status	Resolved * The Arken Finance team has confirmed that a Timelock contract with 24 hours delay will be deployed and used for the affected roles.

5.1.1. Description

Smart contracts are designed to be used as agreements that cannot be changed forever. When a smart contract is upgraded, the agreement can be changed from what was previously agreed upon.

As the `ArkenApprove` smart contract is upgradable, the contract logic can be modified by the owner anytime, making the smart contract untrustworthy.

ArkenApprove.sol

```
5  import '@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol';
6  import '@openzeppelin/contracts/proxy/utils/Initializable.sol';
7  // import 'hardhat/console.sol';
8  import './lib/OwnableUpgradeable.sol';
9
10 contract ArkenApprove is Initializable, OwnableUpgradeable {
11     ...
12 }
```

5.1.2. Remediation

Inspex suggests deploying the contract without the proxy pattern or any solution that can make smart contract upgradable.

However, if the upgradability is needed, Inspex suggests mitigating this issue by implementing a timelock mechanism with a sufficient length of time to delay the changes. This allows the platform users to monitor the timelock and be notified of the potential changes being done on the smart contract.

5.2. Centralized Control of State Variable

ID	IDX-002
Target	ArkenDexV3 ArkenApprove
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity: High** **Impact: High** The controlling authorities can change the critical state variables to gain additional profit. Thus, it is unfair to the other users. **Likelihood: Medium** There is nothing to restrict the changes from being done; however, this action can only be done by the contract owner.
Status	Resolved * The Arken Finance team has confirmed that a Timelock contract with 24 hours delay will be deployed and used for the affected roles.

5.2.1. Description

Critical state variables can be updated 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 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
ArkenDexV3.sol (L:94)	ArkenDexV3	updateFeeWallet()	isOwner
ArkenDexV3.sol (L:100)	ArkenDexV3	updateWETH()	isOwner
ArkenDexV3.sol (L:106)	ArkenDexV3	updateWETHDfyn()	isOwner
ArkenDexV3.sol (L:112)	ArkenDexV3	updateDODOApproveAddress()	isOwner
ArkenDexV3.sol (L:121)	ArkenDexV3	updateArkenApprove()	isOwner
ArkenApprove.sol (L:57)	ArkenApprove	updateCallableAddress()	isOwner

5.2.2. Remediation

In the ideal case, the critical state variables should not be modifiable to keep the integrity of the smart contract, Inspex suggests removing these functions to prevent malicious usage.

However, if modifications are needed, Inspex suggests limiting the use of these functions via the following options:

- Implementing a community-run governance to control the use of these functions
- Using a `Timelock` contract to delay the changes for a sufficient amount of time, e.g., 24 hours

5.3. Swapping Without Fees

ID	IDX-003
Target	ArkenDexV3
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The Arken Finance team has resolved this issue in commit `b34ccb587d7578ef15870803bf1324eedc731b99` by checking the destination address have to be `ArkenDexV3` contract when `desc.isSourceFee` value is `false`.

5.3.1. Description

In the `_trade()` function, after all swap transactions are done and all of the destination tokens are already transferred to the `ArkenDexV3` contract as shown in line 251. If the `desc.isSourceFee` parameter is set to `false`, the fee will be collected as shown in line 265.

ArkenDexV3.sol

```
180 function trade(TradeDescription memory desc) external payable {
181     require(desc.amountIn > 0, 'Amount-in needs to be more than zero');
182     if (_ETH_ == desc.srcToken) {
183         require(
184             desc.amountIn == msg.value,
185             'Ether value not match amount-in'
186         );
187         require(
188             desc.isRouterSource,
189             'Source token Ether requires isRouterSource=true'
190         );
191     }
192
193     uint256 beforeDstAmt = _getBalance(desc.dstToken, desc.to);
194
195     uint256 returnAmount = _trade(desc);
196
197     if (returnAmount > 0) {
198         if (_ETH_ == desc.dstToken) {
199             (bool sent, ) = desc.to.call{value: returnAmount}('');
```

```
200             require(sent, 'Failed to send Ether');
201         } else {
202             IERC20(desc.dstToken).safeTransfer(desc.to, returnAmount);
203         }
204     }
205
206     uint256 receivedAmt = _getBalance(desc.dstToken, desc.to).sub(
207         beforeDstAmt
208     );
209     require(
210         receivedAmt >= desc.amountOutMin,
211         'Received token is not enough'
212     );
213
214     emit Swapped(desc.srcToken, desc.dstToken, desc.amountIn, receivedAmt);
215 }
216
217 function _trade(TradeDescription memory desc)
218     internal
219     returns (uint256 returnAmount)
220 {
221     TradeData memory data = TradeData({amountIn: desc.amountIn});
222     if (desc.isSourceFee) {
223         if (_ETH_ == desc.srcToken) {
224             data.amountIn = _collectFee(desc.amountIn, desc.srcToken);
225         } else {
226             uint256 fee = _calculateFee(desc.amountIn);
227             require(fee < desc.amountIn, 'Fee exceeds amount');
228             _transferFromSender(
229                 desc.srcToken,
230                 _FEE_WALLET_ADDR_,
231                 fee,
232                 desc.srcToken,
233                 data
234             );
235         }
236     }
237     if (desc.isRouterSource && _ETH_ != desc.srcToken) {
238         _transferFromSender(
239             desc.srcToken,
240             address(this),
241             data.amountIn,
242             desc.srcToken,
243             data
244         );
245     }
246     if (_ETH_ == desc.srcToken) {
```

```
247         _wrapEther(_WETH_, address(this).balance);
248     }
249
250     for (uint256 i = 0; i < desc.routes.length; i++) {
251         _tradeRoute(desc.routes[i], desc, data);
252     }
253
254     if (_ETH_ == desc.dstToken) {
255         returnAmount = IERC20(_WETH_).balanceOf(address(this));
256         _unwrapEther(_WETH_, returnAmount);
257     } else {
258         returnAmount = IERC20(desc.dstToken).balanceOf(address(this));
259     }
260     if (!desc.isSourceFee) {
261         require(
262             returnAmount >= desc.amountOutMin,
263             'Return amount is not enough'
264         );
265         returnAmount = _collectFee(returnAmount, desc.dstToken);
266     }
267 }
```

However, in some router types the fee is calculated from the `returnAmount` which can be zero by setting the `route.to` address to the destination address (`msg.sender` of the `trade()` function) or `address(1)`. For example in the `_tradeUniswapV2()` function, the `to` address is assigned in lines 360 to 362.

ArkenDexV3.sol

```
328 function _tradeUniswapV2(
329     TradeRoute memory route,
330     uint256 amountIn,
331     TradeDescription memory desc,
332     TradeData memory data
333 ) internal {
334     if (route.from == address(0)) {
335         IERC20(route.fromToken).safeTransfer(route.lpAddress, amountIn);
336     } else if (route.from == address(1)) {
337         _transferFromSender(
338             route.fromToken,
339             route.lpAddress,
340             amountIn,
341             desc.srcToken,
342             data
343         );
344     }
345     IUniswapV2Pair pair = IUniswapV2Pair(route.lpAddress);
346     (uint256 reserve0, uint256 reserve1, ) = pair.getReserves();
```

```
347     (uint256 reserveFrom, uint256 reserveTo) = route.fromToken ==
348         pair.token0()
349         ? (reserve0, reserve1)
350         : (reserve1, reserve0);
351     amountIn = IERC20(route.fromToken).balanceOf(route.lpAddress).sub(
352         reserveFrom
353     );
354     uint256 amountOut = UniswapV2Library.getAmountOut(
355         amountIn,
356         reserveFrom,
357         reserveTo,
358         route.amountAfterFee
359     );
360     address to = route.to;
361     if (to == address(0)) to = address(this);
362     if (to == address(1)) to = desc.to;
363     if (route.toToken == pair.token0()) {
364         pair.swap(amountOut, 0, to, '');
365     } else {
366         pair.swap(0, amountOut, to, '');
367     }
368 }
```

This means if the `desc.isSourceFee` is set to `false` and the `route.to` address is set to the destination address or `address(1)`, the `returnAmount` value will be zero, resulting in the platform not receiving swap fees in this case. Furthermore, this issue will happen when the `returnAmount` value is greater than the `desc.amountOutMin` value in order to pass the checking in lines 261 to 264.

ArkenDexV3.sol

```
254     if (_ETH_ == desc.dstToken) {
255         returnAmount = IERC20(_WETH_).balanceOf(address(this));
256         _unwrapEther(_WETH_, returnAmount);
257     } else {
258         returnAmount = IERC20(desc.dstToken).balanceOf(address(this));
259     }
260     if (!desc.isSourceFee) {
261         require(
262             returnAmount >= desc.amountOutMin,
263             'Return amount is not enough'
264         );
265         returnAmount = _collectFee(returnAmount, desc.dstToken);
266     }
```

Then the `returnAmount` value will be passed to the `_collectFee()` function which collects no fee since the `_calculateFee()` function will return zero.

ArkenDexV3.sol

```
580 function _collectFee(uint256 amount, address token)
581     internal
582     returns (uint256 remainingAmount)
583 {
584     uint256 fee = _calculateFee(amount);
585     require(fee < amount, 'Fee exceeds amount');
586     remainingAmount = amount.sub(fee);
587     if (_ETH_ == token) {
588         (bool sent, ) = _FEE_WALLET_ADDR_.call{value: fee}('');
589         require(sent, 'Failed to send Ether too fee');
590     } else {
591         IERC20(token).safeTransfer(_FEE_WALLET_ADDR_, fee);
592     }
593 }
594
595 function _calculateFee(uint256 amount) internal pure returns (uint256 fee) {
596     return amount.div(1000);
597 }
```

5.3.2. Remediation

Inspex suggests checking the total swapped parts in the `ArkenDexV3` contract is equal to `100000000` (100%) while `desc.isSourceFee` is set to `false` as shown in lines 250, 252 and 263:

ArkenDexV3.sol

```
217 function _trade(TradeDescription memory desc)
218     internal
219     returns (uint256 returnAmount)
220 {
221     TradeData memory data = TradeData({amountIn: desc.amountIn});
222     if (desc.isSourceFee) {
223         if (_ETH_ == desc.srcToken) {
224             data.amountIn = _collectFee(desc.amountIn, desc.srcToken);
225         } else {
226             uint256 fee = _calculateFee(desc.amountIn);
227             require(fee < desc.amountIn, 'Fee exceeds amount');
228             _transferFromSender(
229                 desc.srcToken,
230                 _FEE_WALLET_ADDR_,
231                 fee,
232                 desc.srcToken,
233                 data
234             );
235         }
236     }
237     if (desc.isRouterSource && _ETH_ != desc.srcToken) {
```

```
238         _transferFromSender(
239             desc.srcToken,
240             address(this),
241             data.amountIn,
242             desc.srcToken,
243             data
244         );
245     }
246     if (_ETH_ == desc.srcToken) {
247         _wrapEther(_WETH_, address(this).balance);
248     }
249 
250     uint32 totalSwappedPart = 0;
251     for (uint256 i = 0; i < desc.routes.length; i++) {
252         if (!desc.isSourceFee && desc.routes[i].to == address(0))
    totalSwappedPart.add(desc.routes[i].part);
253         _tradeRoute(desc.routes[i], desc, data);
254     }
255 
256     if (_ETH_ == desc.dstToken) {
257         returnAmount = IERC20(_WETH_).balanceOf(address(this));
258         _unwrapEther(_WETH_, returnAmount);
259     } else {
260         returnAmount = IERC20(desc.dstToken).balanceOf(address(this));
261     }
262     if (!desc.isSourceFee) {
263         require(totalSwappedPart == 100000000, "totalSwappedPart is not
    enough");
264         require(
265             returnAmount >= desc.amountOutMin,
266             'Return amount is not enough'
267         );
268         returnAmount = _collectFee(returnAmount, desc.dstToken);
269     }
270 }
```

5.4. Missing Caller Address Verification in uniswapV3SwapCallback()

ID	IDX-004
Target	ArkenDexV3
Category	Advanced Smart Contract Vulnerability
CWE	CWE-20: Improper Input Validation
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The Arken Finance team has resolved this issue in commit `13f29325a83d07546ad81573a45a9dba57f3414f` by checking whether the caller of the method is a legitimate `UniswapV3Pool` contract address by calling the `verifyCallback` function.

5.4.1. Description

In the `uniswapV3SwapCallback()` function the caller address is not checked properly. The `uniswapV3SwapCallback()` function can be called by any address to withdraw the `ERC20` token from the `ArkenDexV3` contract.

ArkenDexV3.sol

```
557 function uniswapV3SwapCallback(
558     int256 amount0Delta,
559     int256 amount1Delta,
560     bytes calldata _data
561 ) external {
562     UniswapV3CallbackData memory data = abi.decode(
563         _data,
564         (UniswapV3CallbackData)
565     );
566     if (amount0Delta > 0) {
567         IERC20(data.token0).safeTransfer(msg.sender, uint256(amount0Delta));
568     } else if (amount1Delta > 0) {
569         IERC20(data.token1).safeTransfer(msg.sender, uint256(amount1Delta));
570     }
571 }
```

5.4.2. Remediation

Inspex suggests checking the caller of this method is a legitimate `UniswapV3Pool` contract address by calling the `verifyCallback()` function in the `CallbackValidation` library[2] as shown in the following example:

Import `CallbackValidation` library in line 9.

ArkenDexV3.sol

```
3  pragma solidity ^0.8.0;
4
5  import '@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol';
6  import '@openzeppelin/contracts/proxy/utils/Initializable.sol';
7  // import 'hardhat/console.sol';
8  import './lib/OwnableUpgradeable.sol';
9  import './libraries/CallbackValidation.sol';
```

Create `_UNISWAPV3_FACTORY_` variable in line 44.

ArkenDexV3.sol

```
39 address payable public _FEE_WALLET_ADDR_;
40 address public _DODO_APPROVE_ADDR_;
41 address public _WETH_;
42 address public _WETH_DFYN_;
43 address public _ARKEN_APPROVE_;
44 address public _UNISWAPV3_FACTORY_;
```

Assign `_UNISWAPV3_FACTORY_` value in the constructor.

ArkenDexV3.sol

```
74 constructor(
75     address _ownerAddress,
76     address payable _feeWalletAddress,
77     address _wrappedEther,
78     address _wrappedEtherDfyn,
79     address _dodoApproveAddress,
80     address _arkenApprove,
81     address _uniswapv3Factory
82 ) {
83     transferOwnership(_ownerAddress);
84     _FEE_WALLET_ADDR_ = _feeWalletAddress;
85     _DODO_APPROVE_ADDR_ = _dodoApproveAddress;
86     _WETH_ = _wrappedEther;
87     _WETH_DFYN_ = _wrappedEtherDfyn;
88     _ARKEN_APPROVE_ = _arkenApprove;
89     _UNISWAPV3_FACTORY_ = _uniswapv3Factory;
90 }
```

Check the caller address by calling the `verifyCallback()` function as shown in lines 566 to 567

ArkenDexV3.sol

```
557 function uniswapV3SwapCallback(
558     int256 amount0Delta,
559     int256 amount1Delta,
560     bytes calldata _data
561 ) external {
562     UniswapV3CallbackData memory data = abi.decode(
563         _data,
564         (UniswapV3CallbackData)
565     );
566     (address tokenIn, address tokenOut, uint24 fee) =
    data.path.decodeFirstPool();
567     CallbackValidation.verifyCallback(_UNISWAPV3_FACTORY_, tokenIn, tokenOut,
    fee);
568     if (amount0Delta > 0) {
569         IERC20(data.token0).safeTransfer(msg.sender, uint256(amount0Delta));
570     } else if (amount1Delta > 0) {
571         IERC20(data.token1).safeTransfer(msg.sender, uint256(amount1Delta));
572     }
573 }
```

5.5. Inexplicit Solidity Compiler Version

ID	IDX-005
Target	ArkenDexV3 ArkenApprove
Category	Smart Contract Best Practice
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The Arken Finance team has resolved this issue in commit `13f29325a83d07546ad81573a45a9dba57f3414f` by fixing the solidity compiler to version =0.8.11.

5.5.1. Description

The Solidity compiler versions declared in the smart contracts were not explicit. Each compilation may be done using different compiler versions, which may potentially result in compatibility issues, for example:

ArkenDexV3.sol

```
1 // SPDX-License-Identifier: UNLICENSED
2
3 pragma solidity ^0.8.0;
```

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

Contract	Version
ArkenDexV3	^0.8.0
ArkenApprove	^0.8.0

5.5.2. Remediation

Inspex suggests fixing the solidity compiler to the latest stable version. At the time of the audit, the latest stable version of Solidity compiler in major 0.8 is v0.8.11[3].

ArkenDexV3.sol

```
3 pragma solidity 0.8.11;
```

5.6. Incorrect Logging Parameter for Privileged Functions

ID	IDX-006
Target	ArkenApprove
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 in commit `13f29325a83d07546ad81573a45a9dba57f3414f` by logging the old `_CALLABLE_ADDRESS_` value as suggested.

5.6.1. Description

The `SetCallableAddress` event is designed to emit the old and the new addresses of the callable address.

ArkenApprove.sol

```
31 event SetCallableAddress(
32     address indexed oldAddress,
33     address indexed newAddress
34 );
```

But, the 1st parameter that is emitted in the `SetCallableAddress` event is not the old address.

ArkenApprove.sol

```
57 event SetCallableAddress(
58     address indexed oldAddress,
59     address indexed newAddress
60 );
61
62 function updateCallableAddress(address _callableAddress)
63     external
64     onlyOwner
65 {
66     emit SetCallableAddress(address(0), _callableAddress);
67     _CALLABLE_ADDRESS_ = _callableAddress;
68 }
```

When the first parameter is set to `address(0)` the event always emits the incorrect old address value.

5.6.2. Remediation

Inspex suggests changing the `address(0)` parameter value to the `_CALLABLE_ADDRESS_` which is the old address value of the callable address as shown in line 61:

ArkenApprove.sol

```
57 function updateCallableAddress(address _callableAddress)
58     external
59     onlyOwner
60 {
61     emit SetCallableAddress(_CALLABLE_ADDRESS_, _callableAddress);
62     _CALLABLE_ADDRESS_ = _callableAddress;
63 }
```

5.7. Improper Function Visibility

ID	IDX-007
Target	ArkenApprove
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 in commit `13f29325a83d07546ad81573a45a9dba57f3414f` by changing the function visibility to external.

5.7.1. Description

Functions with public visibility copy calldata to memory when being executed, while external functions can read directly from calldata. Memory allocation uses more resources (gas) than reading directly from calldata.

The following source code shows that the `initialize()` function of the `ArkenApprove` contracts is set to public and it is never called from any internal function.

ArkenApprove.sol

```
39 function initialize(address _ownerAddress, address _callableAddress)
40     public
41     initializer
42 {
43     ownableUpgradeableInitialize();
44     transferOwnership(_ownerAddress);
45     _CALLABLE_ADDRESS_ = _callableAddress;
46 }
```

5.7.2. Remediation

Inspex suggests changing the function visibility to external if it is not called from any internal function as shown in the following example:

ArkenApprove.sol

```
39 function initialize(address _ownerAddress, address _callableAddress)
40     external
41     initializer
```

```
42 {
43     ownableUpgradeableInitialize();
44     transferOwnership(_ownerAddress);
45     _CALLABLE_ADDRESS_ = _callableAddress;
46 }
```

6. Appendix

6.1. About Inspex

Inspex 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

6.2. References

[1] “OWASP Risk Rating Methodology.” [Online]. Available: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology. [Accessed: 08-May-2021]

[2] “CallbackValidation | Uniswap” [Online]. Available: https://docs.uniswap.org/protocol/reference/periphery/libraries/CallbackValidation . [Accessed: 28-Jan-2022]

[3] “Ethereum, Releases · ethereum/solidity.” [Online]. Available: https://github.com/ethereum/solidity/releases. [Accessed: 28-Jan-2022]

inspeX

CYBERSECURITY PROFESSIONAL SERVICE