Security Audit Report for Arken Swap Protocol

Date: May 15, 2023

Version: 1.0

Contact: contact@blocksec.com

BlockSec
Audit Team

Digitally signed by BlockSec Audit Team
Date: 2023.05.15 13:58:32 +08'00'

Contents

Report Manifest

Item	Description
Client	Arken Lab
Target	Arken Swap Protocol

Version History

Version	Date	Description
1.0	May 15, 2023	First Release

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 5 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The target of this audit is the smart contracts of Arken Swap Protocol [1] of Arken Lab. Arken Swap Protocol aggregates multiple decentralized exchanges (i.e., DEX) to provide an all-in-one trading experience for DEX traders. Note that, not all modules in the repository are within the audit scope. Specifically, only the following three smart contracts are covered in this report:

- /contracts/swapV4/ArkenDexTrader.sol
- /contracts/swapV4/ArkenDexV4.sol
- /contracts/swapV4/ArkenERC1967Proxy.sol

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (`Version 1`), as well as new code (in the following versions) to fix issues in the audit report.

Project	Version	Commit Hash
Arken DEX	`Version 1`	`b524132b67fbc2922f4d6ccda3483e8f10ad60e7`
	`Version 2`	`6373b204f36f9695104c2025d3f2282bfbf9a199`

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

[1] https://github.com/arken-lab/arken-swap-protocol

1.3 Auditing Approaches

This section provides an overview of the auditing approaches we adopted and the corresponding checkpoints we focused on. Specifically, we conduct the audit by engaging the following approaches:

- **Static analysis**. We scan smart contracts with both open-source and in-house tools, and then manually verify (reject or confirm) the issues reported by them.
- **Fuzzing testing**. We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an in-house fuzzing tool (developed and customized by our security research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Manual code review**. We manually review the design and the corresponding implementation of the whole project in a comprehensive manner, and check the known attack surface accordingly.

Generally, the checkpoints fall into four categories, i.e., **software security**, **DeFi security**, **NFT security** and **additional recommendation**. The main concrete checkpoints are summarized in the following.

1.3.1 Software Security

* Reentrancy
* DoS
* Access control
* Data handling and data flow
* Exception handling
* Untrusted external call and control flow
* Initialization consistency
* Events operation
* Error-prone randomness
* Improper use of the proxy system

1.3.2 DeFi Security

* Semantic consistency
* Functionality consistency
* Permission management
* Business logic
* Token operation
* Emergency mechanism
* Oracle security
* Whitelist and blacklist
* Economic impact
* Batch transfer

1.3.3 NFT Security

* Duplicated item
* Verification of the token receiver

* Off-chain metadata security

1.3.4 Additional Recommendation

* Gas optimization
* Code quality and style

Note *The above checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Table 1.1: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Low*
High	High	Medium
Low	Medium	Low

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Chapter 2 Findings

In total, we find **two** potential issues. Besides, we also have **one** recommendation.

- Low Risk: 1
- High Risk: 1
- Recommendation: 1

ID	Severity	Description	Category	Status
1	Low	Painless harvest of the assets accidentally received by the contract	DeFi Security	Acknowledged
2	High	Trading without any cost	DeFi Security	Fixed
3	-	Check the validity of `msg.sender`'s source token amount	Recommendation	Confirmed

The details are provided in the following sections.

2.1 Software Security

2.2 DeFi Security

2.2.1 Painless harvest of the assets accidentally received by the contract

Severity Low

Status Acknowledged

Introduced by `Version 1`

Description The `ArkenDexTrader.sol` contract will never reserve any asset after helping users trade their tokens. However, if this contract has some assets (e.g., some users accidentally transfer assets to the contract), any user can harvest these assets by invoking the 'trade' function with or without executing any swap. Specifically, there is no record of the state of the contract balance in the code before receiving assets from the user, and the `_tradeRoute` function of the `ArkenDexTrader.sol` contract will reset `amountIn` based on the total balance of the contract.

```
463  function _trade(
464        ArkenDexTrader.TradeDescription calldata desc,
465        ArkenDexTrader.TradeData memory data
466    ) internal returns (uint256 returnAmount) {
467        if (desc.isRouterSource && ArkenDexTrader._ETH_ != desc.srcToken) {
468            data.amountIn = ArkenDexTrader._transferFromSender(
469                desc.srcToken,
470                address(this),
471                data.amountIn,
472                desc.srcToken,
473                data
474            );
475        }
476        if (ArkenDexTrader._ETH_ == desc.srcToken) {
477            ArkenDexTrader._wrapEther(_WETH_, address(this).balance);
```

```
478         }
479
480         for (uint256 i = 0; i < desc.routes.length; i++) {
481             data = ArkenDexTrader._tradeRoute(
482                 desc.routes[i],
483                 desc,
484                 data,
485                 _WETH_DFYN_,
486                 _DODO_APPROVE_ADDR_,
487                 _WOOFI_QUOTE_TOKEN_
488             );
489         }
490
491         if (ArkenDexTrader._ETH_ == desc.dstToken) {
492             returnAmount = IERC20(_WETH_).balanceOf(address(this));
493             ArkenDexTrader._unwrapEther(_WETH_, returnAmount);
494         } else {
495             returnAmount = IERC20(desc.dstToken).balanceOf(address(this));
496         }
497     }
```

Listing 2.1: ArkenDexV4.sol

```
95  function _tradeRoute(
96          TradeRoute calldata route,
97          TradeDescription calldata desc,
98          TradeData memory data,
99          address wethDfyn,
100         address dodoApproveAddress,
101         address woofiQuoteToken
102     ) public returns (TradeData memory) {
103         require(
104             route.part <= 100000000,
105             'Route percentage can not exceed 100000000'
106         );
107         require(
108             route.fromToken != _ETH_ && route.toToken != _ETH_,
109             'TradeRoute from/to token cannot be Ether'
110         );
111         if (route.from == address(1)) {
112             require(
113                 route.fromToken == desc.srcToken,
114                 'Cannot transfer token from msg.sender'
115             );
116         }
117         if (
118             !desc.isSourceFee &&
119             (route.toToken == desc.dstToken ||
120                 (_ETH_ == desc.dstToken && data.weth == route.toToken))
121         ) {
122             require(
123                 route.to == address(0),
124                 'Destination swap have to be ArkenDex'
```

```
125             );
126         }
127         uint256 amountIn;
128         if (route.from == address(0)) {
129             amountIn =
130                 (IERC20(
131                     route.fromToken == wethDfyn ? data.weth : route.fromToken
132                 ).balanceOf(address(this)) * route.part) /
133                 100000000;
134         } else if (route.from == address(1)) {
135             amountIn = (data.amountIn * route.part) / 100000000;
136         }
137         ...
```

Listing 2.2: ArkenDexTrader.sol

As a result, there are three scenarios of stealing contract assets that are accidentally received.

- **Scenario 1: Receiving Ether**
  1. User1 accidentally transfers 1 Ether into the contract `ArkenERC1967Proxy.sol`.
  2. User2 invokes the `trade` function by setting *desc.srcToken = ArkenDexTrader._ETH_* and providing a route with *route.from = address*(0).
  3. The contract will first convert all Ether to WETH (line 477 of the listing 2.1), then reset the `amountIn` based on all WETH held by the contract (lines 130-133 of the listing 2.2).
  4. After that, the accidentally transferred 1 Ether by User1 will be swapped by the contract and transferred to User2.
- **Scenario 2: Receiving WETH**
  1. User1 accidentally transfers 1 WETH into the contract `ArkenERC1967Proxy.sol`.
  2. User2 invokes the `trade` function by setting *desc.srcToken = ERC20_Token* and *desc.dstToken = ArkenDexTrader._ETH_* and providing no `routes`.
  3. As shown in listing 2.1, the code snippet from line 492 to line 493 will be triggered to unwrap the extra WETH laid in the contract.
  4. Finally, WETH accidentally transferred by User1 will be added into the variable `returnAmount` (line 492 of listing 2.1) and the asset will be looted by User2 without executing any swap.
- **Scenario 3: Receiving ERC20 tokens**
  1. User1 accidentally transfer 100 A (i.e., an ERC20 token) into the contract `ArkenERC1967Proxy.sol`.
  2. User2 invokes the `trade` function by setting *desc.dstToken = A* and providing no `routes`.
  3. Similar to **Scenario2**, the accidentally transferred ERC20 token A will be transferred to User2.

Impact Accidentally transferred user assets can be harvested by others with or without executing trades.

Suggestion Record the contract balance state to trade the received assets accordingly.

2.2.2 Trading without any cost

Severity High

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The `tradeStopLimit` function in the `ArkenDexV4.sol` contract allows users to trade without fee payment. Specifically, as shown in the below code snippet, the `_collectStopLimitFee` function can only be reached when both `stopLImitFee` and `minimumStopLimitFee` are greater than zero. As such, when `stopLimitFee = 0` and `minimumStopLimitFee = 0` are set, the user could conduct the trading without any cost.

```
506         ...
507         if (desc.isSourceFee && stopLimitFee > 0 && minimumStopLimitFee > 0) {
508             if (ArkenDexTrader._ETH_ == desc.srcToken) {
509                 data.amountIn = _collectStopLimitFee(
510                     data,
511                     false,
512                     desc.amountIn,
513                     desc.srcToken,
514                     stopLimitFee,
515                     minimumStopLimitFee
516                 );
517             } else {
518                 data.amountIn = _collectStopLimitFee(
519                     data,
520                     true,
521                     desc.amountIn,
522                     desc.srcToken,
523                     stopLimitFee,
524                     minimumStopLimitFee
525                 );
526             }
527         }
528         uint256 returnAmount = _trade(desc, data);
529
530         if (!desc.isSourceFee && stopLimitFee > 0 && minimumStopLimitFee > 0) {
531             require(
532                 returnAmount >= desc.amountOutMin && returnAmount > 0,
533                 'Return amount is not enough'
534             );
535             returnAmount = _collectStopLimitFee(
536                 data,
537                 false,
538                 returnAmount,
539                 desc.dstToken,
540                 stopLimitFee,
541                 minimumStopLimitFee
542             );
543         }
544         ...
```

Listing 2.3: ArkenDexV4.sol::tradeStopLimit

Impact Traders can trade by invoking `tradeStopLimit` without paying fees.

Suggestion Verify both `stopLImitFee` and `minimumStopLimitFee` (to be greater than zero) at the beginning of the function `tradeStopLimit`.

2.3 Additional Recommendation

2.3.1 Check the validity of `msg.sender`'s source token amount

Status Confirmed

Introduced by `Version 1`

Description The `trade` function (and the `tradeOutside` function) does not verify the value of `desc.amountIn` when source token is not `_ETH_`. The lack of validation might result in a transaction revert if `msg.sender` does not have enough source token as claimed in the variable `desc.amountIn`.

```
130    function trade(
131        ArkenDexTrader.TradeDescription calldata desc
132    ) external payable {
133        require(desc.amountIn > 0, 'Amount-in needs to be more than zero');
134        require(
135            desc.amountOutMin > 0,
136            'Amount-out minimum needs to be more than zero'
137        );
138        if (ArkenDexTrader._ETH_ == desc.srcToken) {
139            require(
140                desc.amountIn == msg.value,
141                'Ether value not match amount-in'
142            );
143            require(
144                desc.isRouterSource,
145                'Source token Ether requires isRouterSource=true'
146            );
147        }
148
149        uint256 beforeSrcAmt = ArkenDexTrader._getBalance(
150            desc.srcToken,
151            msg.sender
152        );
153        uint256 beforeDstAmt = ArkenDexTrader._getBalance(
154            desc.dstToken,
155            desc.to
156        );
157
158        ArkenDexTrader.TradeData memory data = ArkenDexTrader.TradeData({
159            amountIn: desc.amountIn,
160            weth: _WETH_
161        });
162        ...
```

Listing 2.4: ArkenDexV4.sol::trade

Impact N/A

Suggestion Add corresponding sanity checks.

Feedback from the Project After careful consideration, our team has concluded that this step is not necessary for the function to execute correctly.