BLOCKSEC

# Security Audit Report for ApxExchange Contract

**Date:** June 25, 2025 **Version:** 1.0

**Contact:** contact@blocksec.com

# Contents

**Chapter 1 Introduction** 1

1.1 About Target Contracts . . . 1

1.2 Disclaimer . . . 1

1.3 Procedure of Auditing . . . 2

1.3.1 Security Issues . . . 2

1.3.2 Additional Recommendation . . . 2

1.4 Security Model . . . 3

**Chapter 2 Findings** 4

2.1 Recommendation . . . 4

2.1.1 Lack of invoking function `_disableInitializers()` . . . 4

2.1.2 Add non-zero address checks . . . 5

2.1.3 Unify the type for exchange rates . . . 5

2.1.4 Add non-zero value checks for the variable `exchangeAmount` . . . 6

2.1.5 Follow the CEI pattern when transferring tokens . . . 6

2.2 Note . . . 7

2.2.1 Users can claim `AST` tokens when the project is paused . . . 7

2.2.2 The decimals of the tokens `APX` and `AST` . . . 7

2.2.3 Proper configuration of the exchange rates . . . 8

2.2.4 Proper management of the vault . . . 8

2.2.5 Proper upgrade of the contract `ApxExchange` . . . 9

2.2.6 Potential centralization risks . . . 9

## Report Manifest

| Item | Description |
|---|---|
| Client | asterdex |
| Target | ApxExchange Contract |

## Version History

| Version | Date | Description |
|---|---|---|
| 1.0 | June 25, 2025 | First release |

## Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2025-06-26 06:53:42+0000

**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 14 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 code repository [1] of ApxExchange Contract of asterdex.

The ApxExchange Contract is designed for users to exchange their `APX` tokens for the `AST` tokens at preset exchange rates.

Note this audit only focuses on the smart contracts in the following directories/files:

- contracts/ApxExchange.sol

Other files are not within the scope of the audit. Additionally, all dependencies of the smart contracts within the audit scope are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

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 |
|---|---|---|
| ApxExchange | Version 1 | f5993b7a90283f5dab7eed3f036a593c3dc08e3c |
| | Version 2 | ec2def1cee3454ed9883480ae515095b74ff719f |

## 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.

[1] https://github.com/asterdex/apx-exchange-contract/tree/AP-2631

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.

## 1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

### 1.3.1 Security Issues

- Access control
- Permission management
- Whitelist and blacklist mechanisms
- Initialization consistency
- Improper use of the proxy system
- Reentrancy
- Denial of Service (DoS)
- Untrusted external call and control flow
- Exception handling
- Data handling and flow
- Events operation
- Error-prone randomness
- Oracle security
- Business logic correctness
- Semantic and functional consistency
- Emergency mechanism
- Economic and incentive impact

### 1.3.2 Additional Recommendation

- Gas optimization
- Code quality and style

**Note** *The previous 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** |  |  |
|---|---|---|
| *High* | High | Medium |
| *Low* | Medium | Low |
|  | *High* | *Low* |
|  | **Likelihood** |  |

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 five 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.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **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 have **five** recommendations and **six** notes.

- Recommendation: 5
- Note: 6

| ID | Severity | Description | Category | Status |
|---|---|---|---|---|
| 1 | - | Lack of invoking function `_disableInitializers()` | Recommendation | Fixed |
| 2 | - | Add non-zero address checks | Recommendation | Confirmed |
| 3 | - | Unify the type for exchange rates | Recommendation | Fixed |
| 4 | - | Add non-zero value checks for the variable `exchangeAmount` | Recommendation | Confirmed |
| 5 | - | Follow the CEI pattern when transferring tokens | Recommendation | Confirmed |
| 6 | - | Users can claim `AST` tokens when the project is paused | Note | - |
| 7 | - | The decimals of the tokens `APX` and `AST` | Note | - |
| 8 | - | Proper configuration of the exchange rates | Note | - |
| 9 | - | Proper management of the vault | Note | - |
| 10 | - | Proper upgrade of the contract `ApxExchange` | Note | - |
| 11 | - | Potential centralization risks | Note | - |

The details are provided in the following sections.

## 2.1 Recommendation

### 2.1.1 Lack of invoking function `_disableInitializers()`

**Status** Fixed in `Version 2`

**Introduced by** `Version 1`

**Description** In the contract `ApxExchange`, the function `_disableInitializers()` is not invoked in the constructor. Invoking this function prevents the contract itself from being initialized, thereby avoiding unexpected behaviors.

```
constructor(address _timelock, IERC20 _apx, IERC20 _ast, IVotingEscrow _votingEscrow, uint256
     _startTime) {
    if (_timelock == address(0)) revert ZeroAddress();
    TIME_LOCK = _timelock;
    if (address(_apx) == address(0)) revert ZeroAddress();
    APX = _apx;
    if (address(_ast) == address(0)) revert ZeroAddress();
    AST = _ast;
    if (address(_votingEscrow) == address(0)) revert ZeroAddress();
    VOTING_ESCROW = _votingEscrow;
    if (_startTime < block.timestamp) revert IllegalTime();
    START_TIME = _startTime;
}
```

Listing 2.1: contracts/ApxExchange.sol

**Suggestion** Invoke the function `_disableInitializers()` in the constructor.

### 2.1.2 Add non-zero address checks

**Status** Confirmed

**Introduced by** `Version 1`

**Description** In function `initialize()`, several address variables (i.e., `_vaultAddress` and `_admin`) are not checked to ensure they are not zero. It is recommended to add such checks to prevent potential mis-operations.

```
function initialize(address _vaultAddress, uint56 _exchangeRate, ExchangeInfo[] calldata _exchangeInfos, address _admin) external initializer {
    __AccessControlEnumerable_init();
    __Pausable_init();
    __ReentrancyGuard_init();

    vaultAddress = _vaultAddress;
    exchangeRate = _exchangeRate;
    for (uint i = 0; i < _exchangeInfos.length; i ++) {
        exchangeInfos.push(_exchangeInfos[i]);
    }

    _grantRole(DEFAULT_ADMIN_ROLE, TIME_LOCK);
    _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
    _grantRole(ADMIN_ROLE, _admin);
}
```

Listing 2.2: contracts/ApxExchange.sol

**Suggestion** Add non-zero address checks accordingly.

### 2.1.3 Unify the type for exchange rates

**Status** Fixed in `Version 2`

**Introduced by** `Version 1`

**Description** In the contract `ApxExchange`, the types (i.e., `uint56` and `uint256`) of the variables `exchangeRate` and `ExchangeInfo.exchangeRate` are inconsistent. It is recommended to unify the type for exchange rates.

```
uint56 public exchangeRate;
```

Listing 2.3: contracts/ApxExchange.sol

```
struct ExchangeInfo {
    uint256 lockPeriod;
    uint256 exchangeRate;
    uint256 minLockAmount;
}
```

Listing 2.4: contracts/ApxExchange.sol

**Suggestion** Unify the type for exchange rates.

### 2.1.4 Add non-zero value checks for the variable `exchangeAmount`

**Status** Confirmed

**Introduced by** `Version 1`

**Description** In the contract `ApxExchange`, the variable `exchangeAmount` in the functions `exchange()` and `lockFor()` may become zero due to the potential precision loss issues. It is recommended to add non-zero checks for the variable `exchangeAmount`.

```
function exchange(uint256 amount) external whenNotPaused nonReentrant {
    if (amount == 0) revert ZeroAmount();
    if (block.timestamp < START_TIME) revert NotStarted();

    uint256 exchangeAmount = amount * exchangeRate / EXCHANGE_RATE_PRECISION;
```

Listing 2.5: contracts/ApxExchange.sol

```
    uint256 exchangeAmount = lockAmount * exchangeInfo.exchangeRate / EXCHANGE_RATE_PRECISION;
```

Listing 2.6: contracts/ApxExchange.sol

**Suggestion** Add non-zero value checks for the variable `exchangeAmount`.

### 2.1.5 Follow the CEI pattern when transferring tokens

**Status** Confirmed

**Introduced by** `Version 1`

**Description** In the contract `ApxExchange`, some functions (e.g., `exchange()` and `claim()`) transfer tokens before the state mutation. It is recommended to follow the CEI (i.e., Checks-Effects-Interactions) pattern when transferring tokens.

```
function exchange(uint256 amount) external whenNotPaused nonReentrant {
    if (amount == 0) revert ZeroAmount();
    if (block.timestamp < START_TIME) revert NotStarted();

    uint256 exchangeAmount = amount * exchangeRate / EXCHANGE_RATE_PRECISION;

    APX.safeTransferFrom(msg.sender, DEAD_ADDRESS, amount);
    AST.safeTransferFrom(vaultAddress, msg.sender, exchangeAmount);

    totalExchangedAmount += exchangeAmount;

    emit Exchange(msg.sender, amount, exchangeAmount, block.timestamp);
}
```

Listing 2.7: contracts/ApxExchange.sol

```
function claim(uint256[] calldata lockOrderIds) external nonReentrant {
    for (uint i = 0; i < lockOrderIds.length; i ++) {
        uint256 lockOrderId = lockOrderIds[i];
        LockOrder memory lockOrder = lockOrders[lockOrderId];
        if (lockOrder.user != msg.sender) revert LockOrderNoPermission(lockOrderId, msg.sender, lockOrder.user);
        if (lockOrder.unlockTime > block.timestamp) revert NotUnlock(lockOrderId, block.timestamp, lockOrder.unlockTime);

        AST.safeTransfer(lockOrder.user, lockOrder.exchangeAmount);

        delete lockOrders[lockOrderId];
        totalClaimedAmount += lockOrder.exchangeAmount;

        emit Claimed(lockOrder.user, lockOrderId, lockOrder.exchangeAmount, block.timestamp);
    }
}
```

**Listing 2.8:** contracts/ApxExchange.sol

**Suggestion** Follow the CEI pattern when transferring tokens.

## 2.2 Note

### 2.2.1 Users can claim AST tokens when the project is paused

**Introduced by** Version 1

**Description** According to the design, the project allows users to claim AST tokens when the project is paused.

### 2.2.2 The decimals of the tokens APX and AST

**Introduced by** Version 1

**Description** In the contract ApxExchange, the functions exchange() and lockFor() calculate the exchange amount (i.e., exchangeAmount) based on the burnt APX tokens and the exchange rate (i.e., exchangeRate). The project must ensure that the decimals of the tokens APX and AST are the same.

```
function exchange(uint256 amount) external whenNotPaused nonReentrant {
    if (amount == 0) revert ZeroAmount();
    if (block.timestamp < START_TIME) revert NotStarted();

    uint256 exchangeAmount = amount * exchangeRate / EXCHANGE_RATE_PRECISION;

    APX.safeTransferFrom(msg.sender, DEAD_ADDRESS, amount);
    AST.safeTransferFrom(vaultAddress, msg.sender, exchangeAmount);

    totalExchangedAmount += exchangeAmount;

    emit Exchange(msg.sender, amount, exchangeAmount, block.timestamp);
}
```

Listing 2.9: contracts/ApxExchange.sol

```
function lockFor(uint256 exchangeInfoId, uint256 lockAmount, address user) private {
    if (lockAmount == 0) revert ZeroAmount();
    if (block.timestamp < START_TIME) revert NotStarted();
    ExchangeInfo memory exchangeInfo = exchangeInfos[exchangeInfoId];
    if (exchangeInfo.lockPeriod == 0) revert ExchangeInfoNotExist(exchangeInfoId);
    if (lockAmount < exchangeInfo.minLockAmount) revert LockAmountTooLow(lockAmount,
        exchangeInfo.minLockAmount);
    uint256 exchangeAmount = lockAmount * exchangeInfo.exchangeRate / EXCHANGE_RATE_PRECISION;
```

Listing 2.10: contracts/ApxExchange.sol

### 2.2.3 Proper configuration of the exchange rates

**Introduced by** Version 1

**Description** In the contract ApxExchange, the exchange rates (i.e., the variables exchangeRate and exchangeInfos) are only set in the function initialize(). The project must ensure that the exchange rates are properly set during initialization process.

```
function initialize(address _vaultAddress, uint56 _exchangeRate, ExchangeInfo[] calldata
     _exchangeInfos, address _admin) external initializer {
    __AccessControlEnumerable_init();
    __Pausable_init();
    __ReentrancyGuard_init();

    vaultAddress = _vaultAddress;
    exchangeRate = _exchangeRate;
    for (uint i = 0; i < _exchangeInfos.length; i ++) {
        exchangeInfos.push(_exchangeInfos[i]);
    }

    _grantRole(DEFAULT_ADMIN_ROLE, TIME_LOCK);
    _grantRole(DEFAULT_ADMIN_ROLE, msg.sender);
    _grantRole(ADMIN_ROLE, _admin);
}
```

Listing 2.11: contracts/ApxExchange.sol

### 2.2.4 Proper management of the vault

**Introduced by** Version 1

**Description** In the project, the vault (i.e., the variable vaultAddress) is responsible to provide AST tokens (i.e., via the function safeTransferFrom()) for exchanges. The project must ensure that the vault holds sufficient AST tokens and grants sufficient approvals to the contract ApxExchange to provide fluent services.

### 2.2.5 Proper upgrade of the contract ApxExchange

**Introduced by** Version 1

**Description** In the project, the contract ApxExchange follows the UUPS proxy upgrade pattern, which is implemented using an ERC1967Proxy. The project must ensure that the function _authorizeUpgrade() is overridden with a proper access control mechanism before upgrading the contract ApxExchange.

### 2.2.6 Potential centralization risks

**Introduced by** Version 1

**Description** In the project, several privileged roles (e.g., the roles ADMIN_ROLE and PAUSE_ROLE) can conduct sensitive operations, which introduces potential centralization risks. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS