Marketplace & Bridge

Smart Contract Audit Report
Prepared for Astronize

Date Issued:	Apr 24, 2024
Project ID:	AUDIT2024001
Version:	v2.0
Confidentiality Level:	Public

Report Information

Project ID	AUDIT2024001
Version	v2.0
Client	Astronize
Project	Marketplace & Bridge
Auditor(s)	Phitchakorn Apiratisakul Ronnachai Chaipha
Author(s)	Phitchakorn Apiratisakul
Reviewer	Natsasit Jirathammanuwat
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
2.0	Apr 24, 2024	Update reassessment commit	Phitchakorn Apiratisakul
1.0	Feb 28, 2024	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

As requested by Astronize, Inspex team conducted an audit to verify the security posture of the Marketplace & Bridge smart contracts between Jan 24, 2024 and Feb 2, 2024. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of Marketplace & Bridge 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 critical, 3 high, 5 medium, 3 low, 1 very low, 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 Marketplace & Bridge 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

Astronize is the Next-Generation Hybrid GameFi Platform that allows users to bridge NFTs and tokens between on-chain and in-game assets. Users can utilize the bridged assets to sell on the marketplace and auction. Additionally, users are enabled to make offers to buy NFTs from other users. The platform has gashapon features, allowing users to purchase NFTs from various collections.

Scope Information:

Project Name	Marketplace & Bridge
Website	https://astronize.com/
Smart Contract Type	Ethereum Smart Contract
Chain	Bitkub chain
Programming Language	Solidity
Category	NFT, Marketplace

Audit Information:

Audit Method	Whitebox
Audit Date	Jan 24, 2024 - Feb 2, 2024
Reassessment Date	Feb 20, 2024

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: 4d114e9989d31ab1fecded5966c7daf624edc887)

Contract	Location (URL)
ASTTokenKAP20	https://github.com/astronize/astronize-smart-contract/blob/4d114e9989/contracts/astronize/v2/ast_kap20.sol
AstToken	https://github.com/astronize/astronize-smart-contract/blob/4d114e9989/contracts/astrobeam/AstTokenForEth.sol
AstronizeCouponNFTKAP721	https://github.com/astronize/astronize-smart-contract/blob/4d114e9989/contracts/astronize/v1/ast_nft_kap721.sol
AstrobeamBridge	https://github.com/astronize/astronize-smart-contract/blob/4d114e9989/contracts/astrobeam/AstrobeamBridge.sol
AstronizeNFTBridge	https://github.com/astronize/astronize-smart-contract/blob/4d114e9989/contracts/astronize/v2/astronize_nft_bridge.sol
AstronizeAuction	https://github.com/astronize/astronize-smart-contract/blob/4d114e9989/contracts/astrobeam/AstronizeAuction.sol
AstronizeGashapon	https://github.com/astronize/astronize-smart-contract/blob/4d114e9989/contracts/astrobeam/AstronizeGashapon.sol
AstronizeMarketplace	https://github.com/astronize/astronize-smart-contract/blob/4d114e9989/contracts/astronize/v1/astronize_mkp.sol
AstronizeOffer	https://github.com/astronize/astronize-smart-contract/blob/4d114e9989/contracts/astrobeam/AstronizeOffer.sol
TSXToken	https://github.com/astronize/astronize-smart-contract/blob/4d114e9989/contracts/astrobeam/TSXToken.sol
AstronizeVesting	https://github.com/astronize/astronize-smart-contract/blob/4d114e9989/contracts/astrobeam/AstronizeVesting.sol

Reassessment: (Commit: fc96e48bd7d51941bebbd0d718b35c729dae28af, ed475cc82b9ce7aa90c1394c841db7e318d6d39a, 805c7e3c7b919d44c6240b215a1c0e17b3f4006c)

Contract	Location (URL)
ASTTokenKAP20	https://github.com/astronize/astronize-smart-contract/blob/fc96e48bd7/contracts/astronize/ast_kap20.sol

AstToken	https://github.com/astronize/astronize-smart-contract/blob/fc96e48bd7/contracts/astrobeam/AstTokenForEth.sol
AstronizeCouponNFTKAP721	https://github.com/astronize/astronize-smart-contract/blob/fc96e48bd7/contracts/astronize/ast_nft_kap721.sol
AstrobeamBridge	https://github.com/astronize/astronize-smart-contract/blob/fc96e48bd7/contracts/astrobeam/AstrobeamBridge.sol
AstronizeNFTBridge	https://github.com/astronize/astronize-smart-contract/blob/fc96e48bd7/contracts/astronize/astronize_nft_bridge.sol
AstronizeAuction	https://github.com/astronize/astronize-smart-contract/blob/fc96e48bd7/contracts/astrobeam/AstronizeAuction.sol
AstronizeGashapon	https://github.com/astronize/astronize-smart-contract/blob/fc96e48bd7/contracts/astrobeam/AstronizeGashapon.sol
AstronizeMarketplace	https://github.com/astronize/astronize-smart-contract/blob/fc96e48bd7/contracts/astronize/astronize_mkp.sol
AstronizeOffer	https://github.com/astronize/astronize-smart-contract/blob/fc96e48bd7/contracts/astrobeam/AstronizeOffer.sol
TSXToken	https://github.com/astronize/astronize-smart-contract/blob/ed475cc82b/contracts/astrobeam/TSXToken.sol
AstronizeVesting	https://github.com/astronize/astronize-smart-contract/blob/805c7e3c7b/contracts/astrobeam/AstronizeVesting.sol

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

2.3. Security Model

2.3.1 Trust Modules

The Marketplace & Bridge has certain privileged roles with the authority to mutate the critical state variables of the contract. Changes to these state variables significantly impact the contract's functionality. The privileged roles and their corresponding privileged functions are enumerated as follows:

- The `onlyOwner` address can perform the following actions:
  - Pause or unpause the contract functions.
  - Set a new `ownerAccessControlRouter` address; it can be a malicious contract that returns a controlled state.
  - Configure the NFT parameter.
  - Configure the fee percentage and treasury address.
  - Configure the auction parameters.
  - Configure the Token bridge parameters.
- The `onlyMinter` and `MINTER_ROLE` address can unlimitedly mint the token; if there is an arbitrary token minted, it can be freely dumped on the market.
- The `onlyBurner` address can burn the `AstronizeCouponNFTKAP721` NFT and `ASTTokenKAP20` token; it is possible to burn tokens from any wallet address.
- The `DEFAULT_ADMIN_ROLE` address can perform the following actions:
  - Configure the NFT bridge parameters.
  - Configure the marketplace parameters.
  - Configure the vesting wallet address.
- The `PAUSER_ROLE` address can pause the function in contracts; if there is an arbitrary token burned, it prevents users from performing actions such as bridge assets between in-game and on-chain, buy and sell items.
- The `onlyModerator` address can perform the following actions:
  - Create/Edit/Add/Remove NFT gashapon collections.
  - Edit/Cancel Item in Auction.
- The `onlyRoot` address can perform the following actions:
  - Set a new `ownerAccessControlRouter` address; it can be a malicious contract that returns a controlled state.
  - Set `callhelper` address.
- The `committee` address can transfer a token via `adminTransfer()` function; this privileged operation could be used to drain all funds in the contract.
- The `callhelper` address can call the functions that interact with `BitkubNext`; if this is a malicious address, it can use `BitkubNext` function behalf of another user.

The Marketplace & Bridge several functionalities have relied on the external components, which may significantly impact the contract if they malfunction. The external components are listed as follows:

- Astronize's off-chain components:
    - Gashapon Transaction Submitter: an operator who sends the transaction to fulfill the Gashapon request from the user.
    - AstrobeamBridge Transaction Submitter: an operator who performs mint and burn transactions for token bridge.
    - Astrobeam Exchange Rate Updater is an operator who updates the token exchange rate periodically.
    - AstronizeNftBridge Transaction Submitter: an operator who performs mint and burn transactions for NFT bridge.
    - Transaction Monitoring: an operator who monitors on-chain platform transactions.

2.3.2 Trust Assumptions

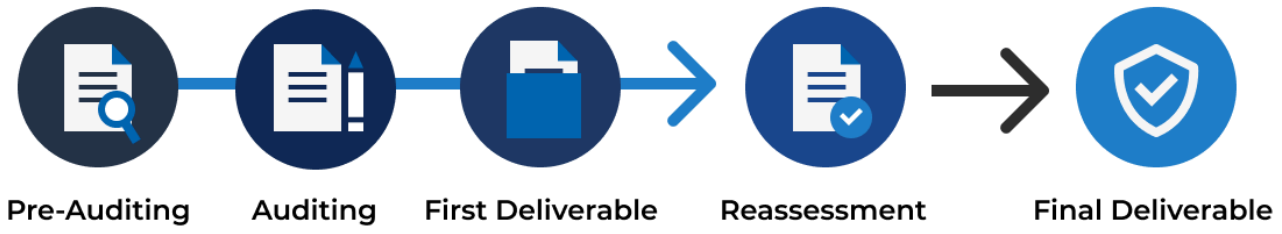
In the Marketplace & Bridge, the protocol's privileged roles, have the ability to change the critical state variables of the contract, also external components were assumed to be trusted. Acknowledging these trust assumptions is important, as it introduces substantial risks to the platform. Trust assumptions include, but are not limited to:

- The off-chain components are assumed to work securely and correctly all the time.
- The `committee` is trusted to use `adminTransfer()` only in emergency cases.
- The `callhelper` is trusted to work properly with `BitkubNext`.

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://docs.inspex.co/smart-contract-security-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**.

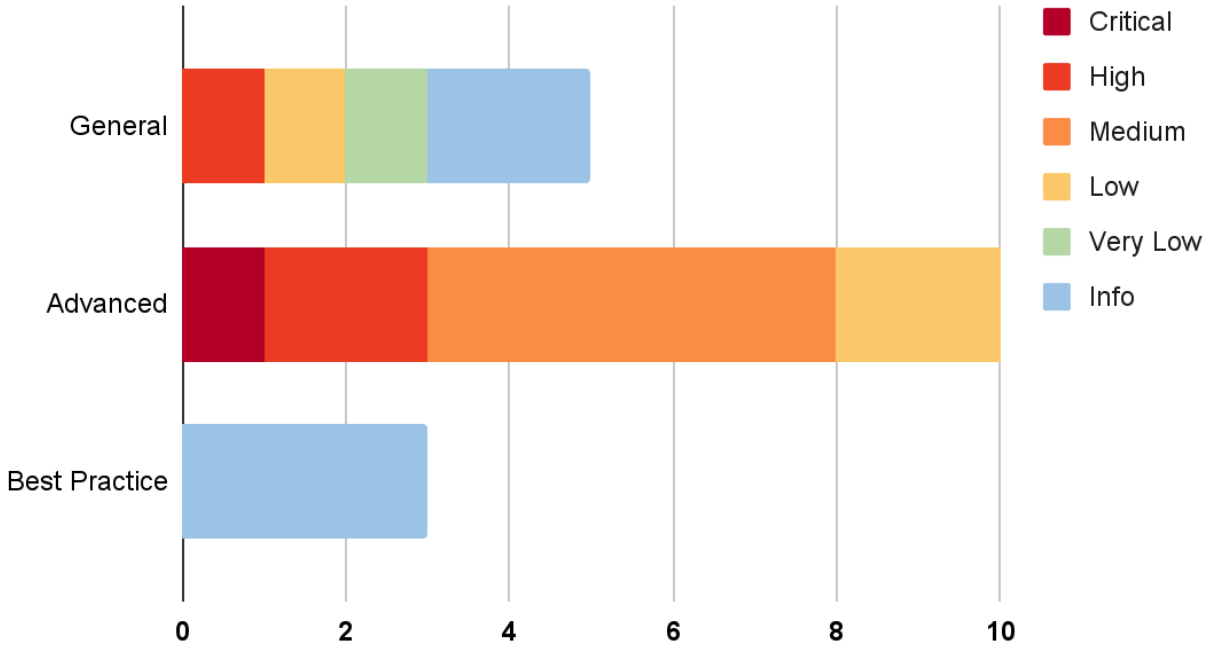
Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would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

Impact \ Likelihood	Low	Medium	High
Low	Very Low	Low	Medium
Medium	Low	Medium	High
High	Medium	High	Critical

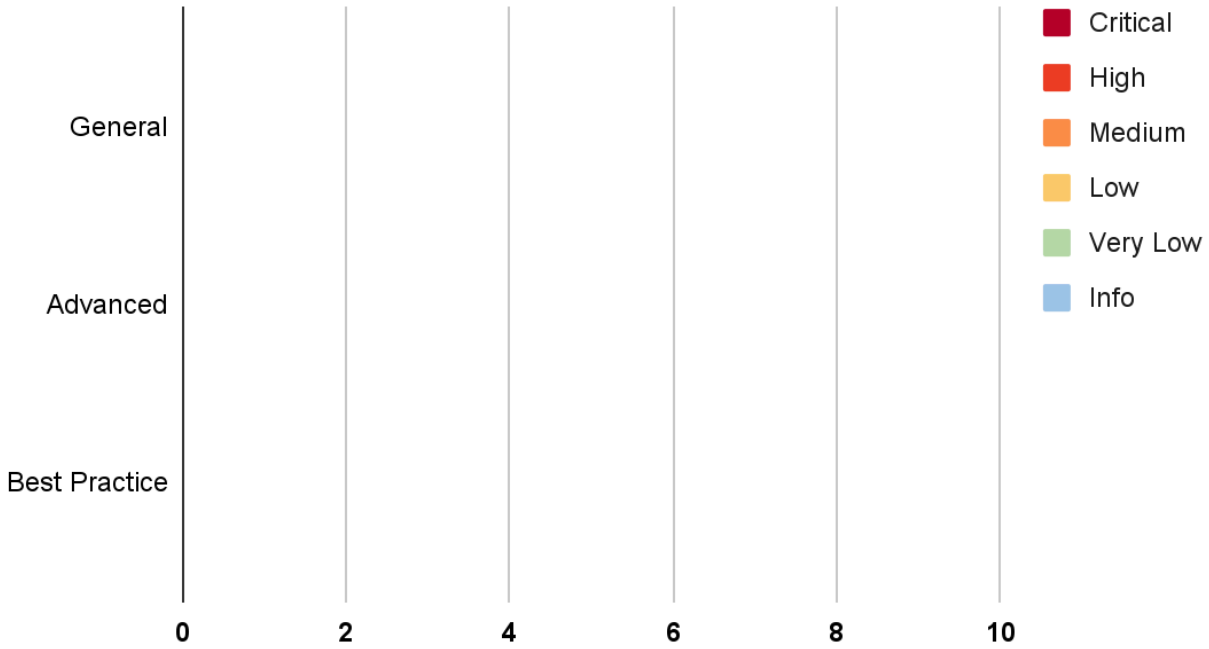
4. Summary of Findings

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	Lack of NFT Ownership Validation in redeem() Function	Advanced	Critical	Resolved
IDX-002	Centralized Authority Control	General	High	Resolved *
IDX-003	Weak Randomness	Advanced	High	Resolved
IDX-004	Insufficient Offer Token Validation	Advanced	High	Resolved
IDX-005	Invalid Removal Item in removeNfts() Function	Advanced	Medium	Resolved
IDX-006	Self Bidding in NFT Auction	Advanced	Medium	Resolved
IDX-007	Minimum Required Price Mismatched with Business Requirements	Advanced	Medium	Resolved
IDX-008	NFT Resaleable	Advanced	Medium	Resolved
IDX-009	Denial of Services in claim() Function	Advanced	Medium	Resolved
IDX-010	Unrestricted Gashapon Purchase Limit	Advanced	Low	Resolved
IDX-011	Loop Over Unbounded Data Structure	General	Low	Resolved
IDX-012	Lack of whenNotPaused Modifier for the BitkubNext Users	Advanced	Low	Resolved
IDX-013	Outdated Compiler Version	General	Very Low	Resolved
IDX-014	Improper Function Visibility	Best Practice	Info	Resolved
IDX-015	Insufficient Logging for Privileged Functions	General	Info	Resolved

IDX-016	Unchecked Return Value ERC20 Transfer	General	Info	Resolved
IDX-017	Misleading Error Message	Best Practice	Info	Resolved
IDX-018	Duplicated require() Check	Best Practice	Info	Resolved

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

5. Detailed Findings Information

5.1. Lack of NFT Ownership Validation in redeem() Function

ID	IDX-001
Target	AstronizeNFTBridge
Category	Advanced Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity: Critical** **Impact: High** The `redeem()` function can be used to burn another user's NFT. An attacker can front-run the `redeem()` transaction and use it before the victim transaction by sending a transaction with higher gas. Also, an attacker could use the valid signature to redeem another user that approved their NFT to the `AstronizeNFTBridge` contract. **Likelihood: High** An attacker can monitor the mempool to front-run the redeem transaction. In the other case, the victim needs to use the same NFT that was owned by an attacker and approve it for the `AstronizeNFTBridge`.
Status	Resolved The Astronize team has resolved this issue by adding ownership validation in commit `21b4dbb46c027cfc9269b322d0e27091d7b8895a`.

5.1.1. Description

In the `AstronizeNFTBridge` contract, the `redeem()` function is used to bridge an NFT from on-chain to in-game assets. Normally, the user uses the function with a signature that was signed from the backend to redeem the NFT.

astronize_nft_bridge.sol

```
284 function redeem(
285     uint256 tokenId,
286     uint256 nonce,
287     uint256 deadline,
288     address nftAddress,
289     bytes calldata signatures
290     ) external whenNotPaused {
291     require(deadline > block.timestamp, "deadline");
292     require(nftAddress != address(0), "token address cannot be zero");
293     require(
294             isWhitelistNFTToken(nftAddress),
```

```
295             "nft token address not whitelisted"
296         );
297
298     IKAP721 nft = IKAP721(nftAddress);
299     require( nft.getApproved(tokenId) == address(this), "nft address not
    approved");
300
301     //verify
302     bytes32 digest = _hashTypedDataV4(
303             keccak256(
304                 abi.encode(
305                     keccak256(
306                         "redeem(uint256 tokenId,uint256 nonce,uint256
    deadline,address nftAddress)"
307                     ),
308                     tokenId,
309                     nonce,
310                     deadline,
311                     nftAddress
312                 )
313             )
314         );
315
316     address signer = ECDSA.recover(digest, signatures);
317
318
319     require(trustedValidators[signer], "signer is not trusted validator");
320     require(!isRedeemConfirmed[signer][nonce], "signer already confirmed");
321     isRedeemConfirmed[signer][nonce] = true;
322
323     //redeem nft(burn)
324     nft.burn(tokenId);
325
326     emit Redeem(_msgSender(), nonce, tokenId, nftAddress);
327
328 }
```

However, the `redeem()` function only validates that the NFT was approved to the `AstronizeNFTBridge` contract and does not validate the owner of an NFT before burning with the tokenId.

This causes various issues as scenario below:

- The signature could be used within the deadline to burn another user's NFT.
  1. An attacker redeems an NFT; the back-end send a valid signature to attakcer.
  2. Sell the NFT to the victim. If the victim wants to bridge the NFT to an in-game asset, they need to approve the NFT to the `AstronizeNFTBridge` contract.
  3. Then, the attacker can use the `redeem()` function with the signature obtained in the first step to

burn the victim's asset.

- The `redeem()` function can be front-run, causing the `Redeem()` event to emit with the attacker's address.

5.1.2. Remediation

Inspex suggests adding NFT owner validation before burning the NFT as seen in line 301.

astronize_nft_bridge.sol

```
284 function redeem(
285     uint256 tokenId,
286     uint256 nonce,
287     uint256 deadline,
288     address nftAddress,
289     bytes calldata signatures
290     ) external whenNotPaused {
291     require(deadline > block.timestamp, "deadline");
292     require(nftAddress != address(0), "token address cannot be zero");
293     require(
294             isWhitelistNFTToken(nftAddress),
295             "nft token address not whitelisted"
296         );
297 
298     IKAP721 nft = IKAP721(nftAddress);
299     require( nft.getApproved(tokenId) == address(this), "nft address not
    approved");
300 
301     require( nft.ownerOf(tokenId) == _msgSender(), "owner is not match");
302 
303     //verify
304     bytes32 digest = _hashTypedDataV4(
305             keccak256(
306                 abi.encode(
307                     keccak256(
308                         "redeem(uint256 tokenId,uint256 nonce,uint256
    deadline,address nftAddress)"
309                     ),
310                     tokenId,
311                     nonce,
312                     deadline,
313                     nftAddress
314                 )
315             )
316         );
317 
318     address signer = ECDSA.recover(digest, signatures);
319 
```

```
320
321     require(trustedValidators[signer], "signer is not trusted validator");
322     require(!isRedeemConfirmed[signer][nonce], "signer already confirmed");
323     isRedeemConfirmed[signer][nonce] = true;
324
325     //redeem nft(burn)
326     nft.burn(tokenId);
327
328     emit Redeem(_msgSender(), nonce, tokenId, nftAddress);
329
330 }
```

5.2. Centralized Authority Control

ID	IDX-002
Target	ASTTokenKAP20 AstToken AstronizeCouponNFTKAP721 AstrobeamBridge AstronizeNFTBridge AstronizeAuction AstronizeGashapon AstronizeMarketplace AstronizeOffer TSXToken AstronizeVesting
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity: High** **Impact: High** The controlling authorities have the ability to manipulate critical state variables, altering the contract's behavior and potentially increasing their profits. This manipulation can lead to malfunctions and render the system unreliable, which is unfair to platform users. **Likelihood: Medium** The private key of the controlling authorities is unlikely to be compromised. Nevertheless, if it were to be compromised, there are no restrictions preventing unauthorized changes from being made. This action can also be done by the contract owner.
Status	Resolved * The Astronize team has confirmed that the contracts will be under the Timelock contract. Since the affected contracts are not yet deployed during the reassessment, the users should confirm that the contracts are under the Timelock contract before using them. Furthermore, the Astronize team has added multi signatures for the `mint()` and `burn()` functions, and added a pauser role to separate from `onlyOwner`.

5.2.1. 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 mechanism in place to prevent the authorities from altering these variables without notifying the users. In the event that the private key of the controlling authorities is compromised by an attacker, they can manipulate the contract's behavior and profit without any restrictions.

The controllable privileged state update functions are as follows:

Target	Function	Modifier
ast_kap20.sol (L:92)	mint()	onlyMinter
ast_kap20.sol (L:96)	burn()	onlyBurner
AstTokenForEth.sol.sol (L:18)	mint()	MINTER_ROLE
ast_nft_kap721.sol (L:70)	setOwnerAccessControlRouter()	onlyOwner
ast_nft_kap721.sol (L:70)	setTokenURI()	onlyOwner
ast_nft_kap721.sol (L:83)	setBaseURI()	onlyOwner
ast_nft_kap721.sol (L:95)	burn()	onlyBurner
ast_nft_kap721.sol (L:112)	mint()	onlyMinter
AstrobeamBridge.sol (L:136)	setTreasuryAddress()	onlyOwner
AstrobeamBridge.sol (L:154)	grantValidator()	onlyOwner
AstrobeamBridge.sol (L:380)	revokeValidator()	onlyOwner
AstrobeamBridge.sol (L:432)	recoverToken()	onlyOwner
astronize_nft_bridge.sol (L:112)	setASTTokenAddress()	DEFAULT_ADMIN_ROLE
astronize_nft_bridge.sol (L:116)	setAcceptedKycLevel()	DEFAULT_ADMIN_ROLE
astronize_nft_bridge.sol (L:120)	setKyc()	DEFAULT_ADMIN_ROLE
astronize_nft_bridge.sol (L:124)	setCallHelper()	DEFAULT_ADMIN_ROLE
astronize_nft_bridge.sol (L:128)	setNextTransferRouter()	DEFAULT_ADMIN_ROLE
astronize_nft_bridge.sol (L:136)	setMintFee()	DEFAULT_ADMIN_ROLE
astronize_nft_bridge.sol (L:140)	setWhitelistNFTToken()	DEFAULT_ADMIN_ROLE
astronize_nft_bridge.sol (L:374)	grantTrustedValidator()	MINTER_ROLE
astronize_nft_bridge.sol (L:384)	revokeTrustedValidator()	MINTER_ROLE
astronize_nft_bridge.sol (L:401)	setTreasuryAddress()	DEFAULT_ADMIN_ROLE
AstronizeAuction.sol (L:162)	setFeePercentage()	onlyOwner
AstronizeAuction.sol (L:171)	setExtendedDuration()	onlyOwner

AstronizeAuction.sol (L:179)	setMinDuration()	onlyOwner
AstronizeAuction.sol (L:187)	setTreasuryAddress()	onlyOwner
AstronizeAuction.sol (L:196)	setWhitelistedToken()	onlyOwner
AstronizeAuction.sol (L:212)	setWhitelistedNft()	onlyOwner
AstronizeAuction.sol (L:224)	setNextNFTTransferRouter()	onlyOwner
AstronizeAuction.sol (L:238)	editItemForAdmin()	onlyModerator
AstronizeAuction.sol (L:516)	cancelForAdmin()	onlyModerator
AstronizeGashapon.sol (L:124)	setTreasuryAddress()	onlyOwner
AstronizeGashapon.sol (L:133)	createCollection()	onlyModerator
AstronizeGashapon.sol (L:159)	editCollection()	onlyModerator
AstronizeGashapon.sol (L:185)	addNfts()	onlyModerator
AstronizeGashapon.sol (L:214)	removeNfts()	onlyModerator
astronize_mkp.sol (L:139)	setAcceptedKycLevel()	DEFAULT_ADMIN_ROLE
astronize_mkp.sol (L:143)	setKyc()	DEFAULT_ADMIN_ROLE
astronize_mkp.sol (L:147)	setCallHelper()	DEFAULT_ADMIN_ROLE
astronize_mkp.sol (L:151)	setNextTransferRouter()	DEFAULT_ADMIN_ROLE
astronize_mkp.sol (L:155)	setNextNFTTransferRouter()	DEFAULT_ADMIN_ROLE
astronize_mkp.sol (L:342)	setFee()	DEFAULT_ADMIN_ROLE
astronize_mkp.sol (L:350)	setTreasuryAddress()	DEFAULT_ADMIN_ROLE
astronize_mkp.sol (L:360)	setWhitelistNFTToken()	DEFAULT_ADMIN_ROLE
astronize_mkp.sol (L:374)	setWhitelistCurrencyToken()	DEFAULT_ADMIN_ROLE
AstronizeOffer.sol (L:94)	setTreasuryAddress()	onlyOwner
AstronizeOffer.sol (L:103)	setFeePercentage()	onlyOwner
TSXToken.sol (L:87)	mint()	onlyMinter
TSXToken.sol (L:92)	burn()	onlyBurner
TSXToken.sol (L:97)	burnFrom()	onlyBurner

AstronizeVesting.sol (L:140)	addWallets()	DEFAULT_ADMIN_ROLE
AstronizeVesting.sol (L:162)	removeWallets()	DEFAULT_ADMIN_ROLE

5.2.2. 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, Inspex 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, Inspex suggests mitigating the risk of this issue by using a timelock mechanism to delay the changes for a reasonable amount of time or using the multi-signature contract to reduce the risk of unauthorized or malicious alterations to the smart contract.

5.3. Weak Randomness

ID	IDX-003
Target	AstronizeGashapon
Category	Advanced Smart Contract Vulnerability
CWE	CWE-330: Use of Insufficiently Random Values
Risk	**Severity:** High **Impact:** Medium The attacker can obtain a specified high-value NFT, creating an unfair advantage over other users. This damages the fairness of the platform. **Likelihood:** High An attacker with basic knowledge of the mempool or techniques for manipulating on-chain randomness can easily exploit this vulnerability.
Status	Resolved The Astronize team has resolved this issue by implementing the commit and reveal mechanism with future blockhash in a random process in commit `ed9821314675ae5ebb8c0ab57f976d70464fa290`.

5.3.1. Description

The `AstronizeGashapon` contract enables users to purchase a random NFT by submitting a request through the `setRequest()` and `setRequestForBitkubNext()` functions. Subsequently, the platform operator will provide a random seed via the `purchase()` function to process the NFT random.

AstronizeGashapon.sol

```
306 function purchase(
307     uint256 _collectionId,
308     address _user,
309     bytes32[] calldata _seeds,
310     uint256 _deadline
311 ) external onlyOperator {
312     // check dead line
313     require(_deadline > block.timestamp, "deadline");
314 
315     // check collection status
316     Collection storage _collection = _collections[_collectionId];
317     require(_collection.tokenIds.length >= _seeds.length, "amount");
318     require(
319         _collection.tokenIds.length >= _seeds.length,
320         "remaining amount"
```

```
321     );
322
323     // check request
324     Request storage _request = _requests[_user][_collectionId];
325     require(_request.price == _collection.price, "price");
326     require(
327         _request.tokenAddress == _collection.tokenAddress,
328         "token address"
329     );
330     require(_request.amount >= _seeds.length, "amount");
331     _request.amount -= _seeds.length;
332
333     // random NFTs and send to user
334     uint256[] memory _tokenIds = new uint256[](_seeds.length);
335     for (uint256 i = 0; i < _seeds.length; i++) {
336         // random pick
337         nonce++;
338         uint256 _pickIndex = _random(
339             _collection.tokenIds.length,
340             nonce,
341             _seeds[i]
342         );
343         uint256 _tokenId = _collection.tokenIds[_pickIndex];
344         _removeItemByIndex(_collection.tokenIds, _pickIndex);
345
346         // transfer
347         IKAP721(_collection.nftAddress).safeTransferFrom(
348             address(this),
349             _user,
350             _tokenId
351         );
352
353         _tokenIds[i] = _tokenId;
354     }
355
356     // transfer token
357     _transferTokens(_collectionId, _user, _seeds.length);
358
359     emit Purchase(
360         _user,
361         _collectionId,
362         _collection.nftAddress,
363         _tokenIds,
364         _collection.tokenIds.length,
365         _request.tokenAddress,
366         _collection.price,
367         block.timestamp
```

```
368         );
369     }
```

However, an attacker can repeatedly reject the NFTs they receive until they obtain one that interests them. This is achieved by using a contract that modifies the `onKAP721Received()` function to revert while receiving low-value NFT.

Additionally, attackers can use the other methods to disrupt the NFT random process, such as front-running to resubmit an invalid request or revoking the token approval.

5.3.2. Remediation

Inspex suggests implementing the commit and reveal mechanism along with using the block hash of the future block and secret as a source of randomness with a commit-reveal scheme.

For example, as part of the request submission procedure, users must provide a hashed seed (generated from the server) along with the `setRequest()` transaction. In this step, the token must be transferred from the user and records the future block number, which will later be used to reveal in the `purchase()` function and determine if the request is expiring.

AstronizeGashapon.sol

```
69  struct Request {
70      address tokenAddress;
71      uint256 price;
72      uint256 amount;
73      uint256 revealBlock;
74      uint256 isPending;
75      bytes32 hashedSeed;
76  }
```

AstronizeGashapon.sol

```
258 function setRequest(
259     uint256 _collectionId,
260     address _tokenAddress,
261     uint256 _price,
262     uint256 _amount,
263     bytes32 _hashedSeed
264 ) external {
265     _setRequest(_collectionId, _tokenAddress, _price, _amount, _hashedSeed,
    msg.sender );
266 }
```

AstronizeGashapon.sol

```
267 function setRequestForBitkubNext(
268     uint256 _collectionId,
```

```
269     address _tokenAddress,
270     uint256 _price,
271     uint256 _amount,
272     bytes32 _hashedSeed,
273     address _bitkubNext
274 ) external onlyCallHelper onlyBitkubNextUser(_bitkubNext) {
275     _setRequest(_collectionId, _tokenAddress, _price, _amount, _hashedSeed,
    _bitkubNext);
276 }
```

AstronizeGashapon.sol

```
284 function _setRequest(
285     uint256 _collectionId,
286     address _tokenAddress,
287     uint256 _price,
288     uint256 _amount,
289     bytes32 _hashedSeed,
290     address _user,
291 ) internal {
292     Request storage _request = _requests[_user][_collectionId];
293     require(!_request.isPendding, "already request");
294 
295     Collection memory _collection = _collections[_collectionId];
296     require(_collection.tokenIds.length >= _amount, "nft not enough");
297 
298     _request.amount = _amount;
299     _request.price = _price;
300     _request.tokenAddress = _tokenAddress;
301     _request.revealBlock = block.number + 1;
302     _request.hashedSeed = _hashedSeed;
303     _request.isPending = true;
304 
305 
306         // transfer token
307     _transferTokens(_collectionId, _user, _amount);
308 
309     emit RequestSet(
310         _user,
311         _collectionId,
312         _tokenAddress,
313         _price,
314         _amount,
315         block.timestamp
316     );
317 }
```

For the `purchase()` function, block validation should be implemented, and the requested reveal block

number should be applied to the source of randomness. Additionally, if the NFT collection does not have sufficient items to fulfill the purchase, any remaining tokens should be refunded to the user.

AstronizeGashapon.sol

```
319 function purchase(
320     uint256 _collectionId,
321     address _user,
322     bytes32 _seeds,
323     uint256 _deadline
324 ) external onlyOperator {
325     // check dead line
326     require(_deadline > block.timestamp, "deadline");
327 
328     // check collection status
329     Collection storage _collection = _collections[_collectionId];
330 
331     // check request
332     Request storage _request = _requests[_user][_collectionId];
333     require(_request.price == _collection.price, "price");
334     require(
335         _request.tokenAddress == _collection.tokenAddress,
336         "token address"
337     );
338     require(block.number > _request.revealBlock, "reveal");
339     require(_request.isPending, "pending");
340     require(_request.hashedSeed == keccak256(abi.encodePacked(_seeds)),
    "hash");
341 
342 
343     uint256 returnAmount;
344     if (_collection.tokenIds.length < _request.amount){
345         returnAmount = _request.amount - _collection.tokenIds.length;
346         _request.amount = _collection.tokenIds.length;
347     }
348 
349     // random NFTs and send to user
350     uint256[] memory _tokenIds = new uint256[](_request.amount);
351     for (uint256 i = 0; i < _request.amount; i++) {
352         // random pick
353         nonce++;
354         uint256 _pickIndex = _random(
355             _collection.tokenIds.length,
356             nonce,
357             _seeds,
358             _request.revealBlock
359         );
360         uint256 _tokenId = _collection.tokenIds[_pickIndex];
```

```
361             _removeItemByIndex(_collection.tokenIds, _pickIndex);
362 
363             // transfer
364             IKAP721(_collection.nftAddress).transferFrom(
365                 address(this),
366                 _user,
367                 _tokenId
368             );
369 
370             _tokenIds[i] = _tokenId;
371         }
372 
373         //refund remain token to user
374         if (returnAmount > 0) {
375             _transferToken(treasuryAddress, _user, _collection.tokenAddress,
    _collection.price * returnAmount);
376         }
377 
378         _request.isPending = false;
379 
380         emit Purchase(
381             _user,
382             _collectionId,
383             _collection.nftAddress,
384             _tokenIds,
385             _collection.tokenIds.length,
386             _request.tokenAddress,
387             _collection.price,
388             block.timestamp
389         );
390     }
```

AstronizeGashapon.sol

```
396 function _random(
397     uint256 _length,
398     uint256 _nonce,
399     bytes32 _randSeed,
400     uint256 revealBlock
401 ) internal view returns (uint256) {
402     return
403         uint256(
404             keccak256(
405                 abi.encodePacked(
406                     blockhash(revealBlock),
407                     _nonce,
408                     _randSeed
409                 )
```

```
410                 )
411             ) % _length;
412     }
```

5.4. Insufficient Offer Token Validation

ID	IDX-004
Target	AstronizeOffer
Category	Advanced Smart Contract Vulnerability
CWE	CWE-20: Improper Input Validation
Risk	**Severity:** High **Impact:** High A malicious user could take advantage of this issue to trick sellers into selling NFTs on the marketplace through misleading offers. **Likelihood:** Medium Users with an understanding of smart contracts can easily create malicious offers to deceive others in the NFT marketplace.
Status	Resolved The Astronize team has resolved this issue by adding token whitelist validation in commit `eebc41d0e527666093bd751600656262f5363673`.

5.4.1. Description

The Astronize marketplace enables users to create offers for desired NFTs through its website, restricting the types of tokens that can be used for these offers.

However, the `makeOffer()` function within the `AstronizeOffer` contract lacks mechanisms to verify whether the token address is used in the incoming transaction. This allows for the processing of offers with any token address, bypassing the intended restrictions and potentially fraudulent activities.

AstronizeOffer.sol

```
137 function makeOffer(
138     uint256 _tokenId,
139     address _nftAddress,
140     address _tokenAddress,
141     uint256 _amount,
142     uint256 _endAt
143 ) external {
144     _makeOffer(
145         _tokenId,
146         _nftAddress,
147         _tokenAddress,
148         _amount,
149         _endAt,
```

```
150             msg.sender
151         );
152     }
```

AstronizeOffer.sol

```
178 function _makeOffer(
179     uint256 _tokenId,
180     address _nftAddress,
181     address _tokenAddress,
182     uint256 _amount,
183     uint256 _endAt,
184     address _offerorAddress
185 ) internal {
186     require(_endAt <= block.timestamp + (366 days), "offer: max duration");
187     require(
188         IKAP20(_tokenAddress).balanceOf(_offerorAddress) >= _amount,
189         "offer: balance"
190     );
191 
192     Offer storage _offer = _offers[_offerorAddress][_nftAddress][_tokenId];
193 
194     _offer.tokenAddress = _tokenAddress;
195     _offer.amount = _amount;
196     _offer.endAt = _endAt;
197     _offer.status = OfferStatus.Created;
198 
199     emit OfferCreated(
200         _tokenId,
201         _nftAddress,
202         _offerorAddress,
203         _offer,
204         block.timestamp
205     );
206 }
```

5.4.2. Remediation

Inspex suggests updating the `AstronizeOffer` contract by adding a validation mechanism to ensure that the token address utilized is on the approved list of tokens.

For instance, the constructor method could be modified to include the whitelisted token addresses shown in lines 63, 75 and 92-94.

AstronizeOffer.sol

```
63 mapping (address=>bool) whitelisted;
64 
```

```
65  constructor(
66      address _adminProjectRouter,
67      address _kyc,
68      address _committee,
69      uint256 _acceptedKycLevel,
70      address _ownerAccessControlRouter,
71      address _nextTransferRouter,
72      address _nextNFTTransferRouter,
73      address _treasuryAddress,
74      address _callHelper,
75      address[] memory _whitelisted
76  ) {
77      // Initialize BitkubChain
78      PROJECT = "astronize";
79      adminProjectRouter = IAdminProjectRouter(_adminProjectRouter);
80      kyc = IKYCBitkubChain(_kyc);
81      committee = _committee;
82      acceptedKycLevel = _acceptedKycLevel;
83      ownerAccessControlRouter = IOwnerAccessControlRouter(
84          _ownerAccessControlRouter
85      );
86      nextTransferRouter = INextTransferRouter(_nextTransferRouter);
87
88      // init
89      nextNFTTransferRouter = INextNFTTransferRouter(_nextNFTTransferRouter);
90      treasuryAddress = _treasuryAddress;
91      callHelper = _callHelper;
92      for (uint256 i = 0; i < _whitelisted.length; i++) {
93          whitelisted[_whitelisted[i]] = true;
94      }
95  }
96
97  function setWhitelistedToken(address _token, bool _isActive)
98      external
99      onlyOwner
100 {
101     emit WhitelistedTokenSet(_token, _isActive);
102     whitelisted[_token] = _isActive;
103 }
```

Additionally, enforce the restriction across all functions that accept a token address as a parameter. As shown in lines 191.

AstronizeOffer.sol

```
183 function _makeOffer(
184     uint256 _tokenId,
185     address _nftAddress,
```

```
186     address _tokenAddress,
187     uint256 _amount,
188     uint256 _endAt,
189     address _offerorAddress
190 ) internal {
191     require(whitelisted[_tokenAddress], "offer: not whitelisted");
192     require(_endAt <= block.timestamp + (366 days), "offer: max duration");
193     require(
194         IKAP20(_tokenAddress).balanceOf(_offerorAddress) >= _amount,
195         "offer: balance"
196     );
197 
198     Offer storage _offer = _offers[_offerorAddress][_nftAddress][_tokenId];
199 
200     _offer.tokenAddress = _tokenAddress;
201     _offer.amount = _amount;
202     _offer.endAt = _endAt;
203     _offer.status = OfferStatus.Created;
204 
205     emit OfferCreated(
206         _tokenId,
207         _nftAddress,
208         _offerorAddress,
209         _offer,
210         block.timestamp
211     );
212 }
```

5.5. Invalid Removal Item in removeNfts() Function

ID	IDX-005
Target	AstronizeGashapon
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Medium **Impact:** Medium The flaw in the `removeNfts()` function will be removing unintended items in the NFT collection. **Likelihood:** Medium This issue occurs when the platform owner attempts to remove and withdraw NFTs from the `AstronizeGashapon` contract.
Status	Resolved The Astronize team has resolved this issue by using `startIndex` and `endIndex` for removing NFT inside array in commit `ed9821314675ae5ebb8c0ab57f976d70464fa290`.

5.5.1. Description

The `removeNfts()` function is designed to delete multiple NFTs from a collection by specifying a range of NFT indexes (`_fromIndex`, `_toIndex`).

AstronizeGashapon.sol

```
214 function removeNfts(
215     uint256 _collectionId,
216     uint256 _fromIndex,
217     uint256 _toIndex
218 ) external onlyModerator {
219     require(_fromIndex <= _toIndex, "index");
220 
221     // get collection
222     Collection storage _collection = _collections[_collectionId];
223 
224     // remove NFTs
225     uint256 _amount = _toIndex - _fromIndex + 1;
226     uint256[] memory _tokenIds = new uint256[](_amount);
227     for (uint256 i = 0; i < _amount; i++) {
228         // pick
229         uint256 _pickIndex = _fromIndex;
230         uint256 _tokenId = _collection.tokenIds[_pickIndex];
```

```
231         _removeItemByIndex(_collection.tokenIds, _pickIndex);
232
233         // transfer
234         IKAP721(_collection.nftAddress).safeTransferFrom(
235             address(this),
236             msg.sender,
237             _tokenId
238         );
239
240         _tokenIds[i] = _tokenId;
241     }
242
243     emit NftRemoved(
244         msg.sender,
245         _collectionId,
246         _tokenIds,
247         _collection.tokenIds.length,
248         block.timestamp
249     );
250 }
251
```

In the source code above, the `_pickIndex` variable was incorrectly set to always equal `_fromIndex`, which meant it did not update to the next index within the loop.

AstronizeGashapon.sol

```
418 function _removeItemByIndex(
419     uint256[] storage _array,
420     uint256 _index
421 ) internal {
422     require(_index < _array.length);
423     if (_index != _array.length - 1) {
424         _array[_index] = _array[_array.length - 1];
425     }
426     delete _array[_array.length - 1];
427     _array.pop();
428 }
```

Lastly, the `_removeItemByIndex()` function's logic ensures that the item at the specified index is removed by swapping it with the last item in the array and then popping the last element off the array.

5.5.2. Remediation

Inspex suggests adjusting the `_startIndex`and `_endIndex` variable within the `removeNfts()` function and changing the loop condition to align with the `_startIndex`and `_endIndex` specified through the function parameters.

AstronizeGashapon.sol

```
257 function removeNfts(
258     uint256 _collectionId,
259     uint256 _startIndex,
260     uint256 _endIndex
261 ) external onlyModerator {
262     // get collection
263     Collection storage _collection = _collections[_collectionId];
264 
265     // remove NFTs
266     uint256[] memory _tokenIds = new uint256[](_amount);
267     for (uint256 i = _endIndex; i >= _startIndex; i--) {
268         // remove
269         uint256 _tokenId = _collection.tokenIds[i];
270         _removeItemByIndex(_collection.tokenIds, i);
271 
272 
273         // transfer
274         IKAP721(_collection.nftAddress).safeTransferFrom(
275             address(this),
276             msg.sender,
277             _tokenId
278         );
279 
280         _tokenIds[i] = _tokenId;
281         if(i==0) break;
282     }
283 
284     emit NftRemoved(
285         msg.sender,
286         _collectionId,
287         _tokenIds,
288         _collection.tokenIds.length,
289         block.timestamp
290     );
291 }
```

5.6. Self Bidding in NFT Auction

ID	IDX-006
Target	AstronizeAuction
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Medium **Impact:** Medium Sellers can bid their own NFT in order to prevent any other user from winning an auction with a low price. **Likelihood:** Medium Users with basic smart contract knowledge are likely to use this issue to gain an unfair advantage.
Status	Resolved The Astronize team has resolved this issue by adding the restriction from self-bidding in commit `eebc41d0e527666093bd751600656262f5363673`.

5.6.1. Description

The Astronize auction website prohibits sellers from bidding on their own NFT items. However, the `AstronizeAuction` contract lacks restrictions to prevent this action. As a result, the seller can bid their own NFT in order to prevent any other user from winning an auction.

The following `_bid()` function implementation does not contain any restrictions to prevent sellers from bidding their own NFT.

AstronizeAuction.sol

```
400 function _bid(
401     address _nftAddress,
402     uint256 _tokenId,
403     address _tokenAddress,
404     uint256 _price,
405     address _user
406 ) internal whenNotPaused {
407     // check item
408     Item storage _item = _items[_nftAddress][_tokenId];
409     require(!isExpired(_nftAddress, _tokenId), "expired");
410     require(block.timestamp >= _item.startAt, "startAt");
411     require(_item.price < _price, "require higher price");
412     require(_item.status == ItemStatus.Active, "status");
```

```
413     require(_tokenAddress == _item.tokenAddress, "tokenAddress");
414
415     // check step
416     WhitelistedToken storage _whitelistedToken = _whitelistedTokens[
417         _tokenAddress
418     ];
419     require(_price % _whitelistedToken.step == 0, "price step");
420
421     // transfer token to previous bidder
422     if (_item.bidderAddress != address(0)) {
423         IKAP20(_tokenAddress).transfer(_item.bidderAddress, _item.price);
424     }
425
426     // transfer token
427     _transferToken(_user, address(this), _tokenAddress, _price);
428
429     // get previous price and bidder
430     uint256 previousPrice = _item.price;
431     address previousBidderAddress = _item.bidderAddress;
432
433     // set new price and bidder
434     _item.price = _price;
435     _item.bidderAddress = _user;
436
437     // check endAt for extended time
438     if (_item.endAt < block.timestamp + extendedDuration) {
439         _item.endAt = block.timestamp + extendedDuration;
440     }
441
442     emit Bid(
443         _user,
444         _nftAddress,
445         _tokenId,
446         _item.tokenAddress,
447         _item.bidderAddress,
448         _item.price,
449         _item.endAt,
450         previousBidderAddress,
451         previousPrice,
452         block.timestamp
453     );
454 }
```

5.6.2. Remediation

Inspex suggests implementing a validation mechanism within the contract to prevent sellers from bidding on their own NFTs. For instance, adding checks to verify that the bidder is not the same entity as the seller before accepting a bid. As shown in line 414.

AstronizeAuction.sol

```
400 function _bid(
401     address _nftAddress,
402     uint256 _tokenId,
403     address _tokenAddress,
404     uint256 _price,
405     address _user
406 ) internal whenNotPaused {
407     // check item
408     Item storage _item = _items[_nftAddress][_tokenId];
409     require(!isExpired(_nftAddress, _tokenId), "expired");
410     require(block.timestamp >= _item.startAt, "startAt");
411     require(_item.price < _price, "require higher price");
412     require(_item.status == ItemStatus.Active, "status");
413     require(_tokenAddress == _item.tokenAddress, "tokenAddress");
414     require(_user != _item.sellerAddress, "seller");
415 
416     // check step
417     WhitelistedToken storage _whitelistedToken = _whitelistedTokens[
418         _tokenAddress
419     ];
420     require(_price % _whitelistedToken.step == 0, "price step");
421 
422     // transfer token to previous bidder
423     if (_item.bidderAddress != address(0)) {
424         IKAP20(_tokenAddress).transfer(_item.bidderAddress, _item.price);
425     }
426 
427     // transfer token
428     _transferToken(_user, address(this), _tokenAddress, _price);
429 
430     // get previous price and bidder
431     uint256 previousPrice = _item.price;
432     address previousBidderAddress = _item.bidderAddress;
433 
434     // set new price and bidder
435     _item.price = _price;
436     _item.bidderAddress = _user;
437 
438     // check endAt for extended time
439     if (_item.endAt < block.timestamp + extendedDuration) {
440         _item.endAt = block.timestamp + extendedDuration;
441     }
442 
443     emit Bid(
444         _user,
445         _nftAddress,
```

```
446             _tokenId,
447             _item.tokenAddress,
448             _item.bidderAddress,
449             _item.price,
450             _item.endAt,
451             previousBidderAddress,
452             previousPrice,
453             block.timestamp
454         );
455     }
```

5.7. Minimum Required Price Mismatched with Business Requirements

ID	IDX-007
Target	AstronizeMarketplace AstronizeOffer AstronizeAuction
Category	Advanced Smart Contract Vulnerability
CWE	CWE-20: Improper Input Validation
Risk	**Severity:** Medium **Impact:** Medium Anyone can perform an action directly within the contract, bypassing the restriction from the business requirements. **Likelihood:** Medium The function can be called by anyone without any restrictions at all.
Status	Resolved The Astronize team has resolved this issue by adding the minimum price validation in commit `a198d6349f4b6398887c8429e6d185b65b0e63a2`.

5.7.1. Description

The business requirements of the platform restrict certain actions from the front-end, but the contracts lack validation that aligns with these conditions.

On the Astronize Marketplace website, the seller must sell their item for at least 0.1 `$TSX`. However, in the `AstronizeMarketplace` contract, users can sell their NFTs using the `sellNFT()` function both in contract and `BitkubNext` without the price limitations that are restricted by the front-end business requirement.

astronize_mkp.sol

```
70  function sellNFT(
71      address nftTokenAddress,
72      uint256 tokenId,
73      address currencyTokenAddress,
74      uint256 price
75  ) external whenNotPaused{
76      require(price > 0, "price must be greater than 0");
77      require(
78          isWhitelistNFTToken(nftTokenAddress),
79          "nft token address not whitelisted"
80      );
81
```

```
82      require(
83          isWhitelistCurrencyToken(currencyTokenAddress),
84          "currency token address not whitelisted"
85      );
86
87      IERC721(nftTokenAddress).safeTransferFrom(
88          msg.sender,
89          address(this),
90          tokenId
91      );
92
93      _nftInMarketplace.push(
94          NFTSellInfo({
95              nftTokenAddress : nftTokenAddress,
96              tokenId: tokenId,
97              currencyTokenAddress : currencyTokenAddress,
98              price: price,
99              seller: msg.sender,
100             fee: fee
101         })
102     );
103
104     _indexOfTokenIds[nftTokenAddress][tokenId] = _nftInMarketplace.length - 1;
105     emit Sell(msg.sender, nftTokenAddress, tokenId, currencyTokenAddress,
    price, fee,  block.timestamp);
106
107 }
```

In another similar case in the `AstronizeOffer` contract, users can make offers by executing the `_makeOffer()` function within the contract without being restricted by the minimum price limitations set by the front-end business requirements.

AstronizeOffer.sol

```
178 function _makeOffer(
179     uint256 _tokenId,
180     address _nftAddress,
181     address _tokenAddress,
182     uint256 _amount,
183     uint256 _endAt,
184     address _offerorAddress
185 ) internal {
186     require(_endAt <= block.timestamp + (366 days), "offer: max duration");
187     require(
188         IKAP20(_tokenAddress).balanceOf(_offerorAddress) >= _amount,
189         "offer: balance"
190     );
```

```
191
192     Offer storage _offer = _offers[_offerorAddress][_nftAddress][_tokenId];
193
194     _offer.tokenAddress = _tokenAddress;
195     _offer.amount = _amount;
196     _offer.endAt = _endAt;
197     _offer.status = OfferStatus.Created;
198
199     emit OfferCreated(
200         _tokenId,
201         _nftAddress,
202         _offerorAddress,
203         _offer,
204         block.timestamp
205     );
206 }
```

In another case discovered in the `AstronizeAuction` contract, the `_sell()` function can be used to sell an NFT with any price, without being restricted by a minimum price, as indicated in the front-end business requirements.

AstronizeAuction.sol

```
325 function _sell(
326     address _nftAddress,
327     uint256 _tokenId,
328     address _tokenAddress,
329     uint256 _price,
330     uint256 _startAt,
331     uint256 _endAt,
332     address _user
333 ) internal whenNotPaused {
334     // check start at
335     if (_startAt < block.timestamp) {
336         _startAt = block.timestamp;
337     }
338
339     // check time
340     require(_startAt < _endAt, "startAt must be less than endAt");
341     require(_endAt >= block.timestamp + minDuration, "endAt");
342     require(
343         _endAt < block.timestamp + (90 days),
344         "endAt must be less than 60 days"
345     );
346     require((_endAt - _startAt) >= minDuration, "duration");
347
348     // check nft and token
```

```
349     require(_whitelistedNfts[_nftAddress], "whitelisted NFT");
350     WhitelistedToken storage _whitelistedToken = _whitelistedTokens[
351         _tokenAddress
352     ];
353     require(_price % _whitelistedToken.step == 0, "price step");
354     require(_whitelistedToken.isActive, "whitelisted token");
355
356     // store item
357     Item memory _item = Item({
358         sellerAddress: _user,
359         tokenAddress: _tokenAddress,
360         price: _price,
361         startAt: _startAt,
362         endAt: _endAt,
363         bidderAddress: address(0),
364         status: ItemStatus.Active
365     });
366     _items[_nftAddress][_tokenId] = _item;
367
368     // event
369     emit Sell(_user, _nftAddress, _tokenId, _item, block.timestamp);
370 }
```

5.7.2. Remediation

Inspex suggests adding the minimum price validation mechanism for each function in accordance with the business requirements.

5.8. NFT Resaleable

ID	IDX-008
Target	AstronizeAuction AstronizeOffer AstronizeMarketplace
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Medium **Impact:** Medium The NFT is being sold more than once, which does not comply with the business requirements. **Likelihood:** Medium Users with basic smart contract knowledge are likely to use this issue to gain an unfair advantage.
Status	Resolved The Astronize team has resolved this issue by implementing the `NFTResaleHandler` contract to handle the resale mechanism in commit `24a747cb44f38ffd7693fe4eb09dd426bec0bc9c`.

5.8.1. Description

According to the business requirements, an NFT should only be sold once, after which the seller is prohibited from selling that NFT again.

However, both the `AstronizeAuction` and `AstronizeOffer` contracts currently lack the necessary validation mechanisms to prevent an NFT from being sold more than once.

AstronizeOffer.sol

```
178 function _makeOffer(
179     uint256 _tokenId,
180     address _nftAddress,
181     address _tokenAddress,
182     uint256 _amount,
183     uint256 _endAt,
184     address _offerorAddress
185 ) internal {
186     require(_endAt <= block.timestamp + (366 days), "offer: max duration");
187     require(
188         IKAP20(_tokenAddress).balanceOf(_offerorAddress) >= _amount,
```

```
189             "offer: balance"
190         );
191
192         Offer storage _offer = _offers[_offerorAddress][_nftAddress][_tokenId];
193
194         _offer.tokenAddress = _tokenAddress;
195         _offer.amount = _amount;
196         _offer.endAt = _endAt;
197         _offer.status = OfferStatus.Created;
198
199         emit OfferCreated(
200             _tokenId,
201             _nftAddress,
202             _offerorAddress,
203             _offer,
204             block.timestamp
205         );
206     }
```

AstronizeAuction.sol

```
325 function _sell(
326     address _nftAddress,
327     uint256 _tokenId,
328     address _tokenAddress,
329     uint256 _price,
330     uint256 _startAt,
331     uint256 _endAt,
332     address _user
333 ) internal whenNotPaused {
334     // check start at
335     if (_startAt < block.timestamp) {
336         _startAt = block.timestamp;
337     }
338
339     // check time
340     require(_startAt < _endAt, "startAt must be less than endAt");
341     require(_endAt >= block.timestamp + minDuration, "endAt");
342     require(
343         _endAt < block.timestamp + (90 days),
344         "endAt must be less than 60 days"
345     );
346     require((_endAt - _startAt) >= minDuration, "duration");
347
348     // check nft and token
349     require(_whitelistedNfts[_nftAddress], "whitelisted NFT");
350     WhitelistedToken storage _whitelistedToken = _whitelistedTokens[
351         _tokenAddress
```

```
352     ];
353     require(_price % _whitelistedToken.step == 0, "price step");
354     require(_whitelistedToken.isActive, "whitelisted token");
355 
356     // store item
357     Item memory _item = Item({
358         sellerAddress: _user,
359         tokenAddress: _tokenAddress,
360         price: _price,
361         startAt: _startAt,
362         endAt: _endAt,
363         bidderAddress: address(0),
364         status: ItemStatus.Active
365     });
366     _items[_nftAddress][_tokenId] = _item;
367 
368     // event
369     emit Sell(_user, _nftAddress, _tokenId, _item, block.timestamp);
370 }
```

astronize_mkp.sol

```
159 function sellNFT(
160     address nftTokenAddress,
161     uint256 tokenId,
162     address currencyTokenAddress,
163     uint256 price,
164     address _bitkubnext
165 ) external onlyCallHelper onlyBitkubNextUser(_bitkubnext) {
166     require(price > 0, "price must be greater than 0");
167     require(
168         isWhitelistNFTToken(nftTokenAddress),
169         "nft token address not whitelisted"
170     );
171 
172     require(
173         isWhitelistCurrencyToken(currencyTokenAddress),
174         "currency token address not whitelisted"
175     );
176 
177     //bitkubnext transfer
178     nextNFTTransferRouterKap721.transferFromKAP721(PROJECT,
    address(nftTokenAddress), _bitkubnext, address(this), tokenId);
179 
180 
181     _nftInMarketplace.push(
182         NFTSellInfo({
183             nftTokenAddress : nftTokenAddress,
```

```
184             tokenId: tokenId,
185             currencyTokenAddress : currencyTokenAddress,
186             price: price,
187             seller: _bitkubnext,
188             fee: fee
189         })
190     );
191 
192     _indexOfTokenIds[nftTokenAddress][tokenId] = _nftInMarketplace.length - 1;
193     emit Sell(_bitkubnext, nftTokenAddress, tokenId, currencyTokenAddress,
    price, fee,  block.timestamp);
194 
195 }
196 
197 function sellNFT(
198     address nftTokenAddress,
199     uint256 tokenId,
200     address currencyTokenAddress,
201     uint256 price
202 ) external whenNotPaused{
203     require(price > 0, "price must be greater than 0");
204     require(
205         isWhitelistNFTToken(nftTokenAddress),
206         "nft token address not whitelisted"
207     );
208 
209     require(
210         isWhitelistCurrencyToken(currencyTokenAddress),
211         "currency token address not whitelisted"
212     );
213 
214     IERC721(nftTokenAddress).safeTransferFrom(
215         msg.sender,
216         address(this),
217         tokenId
218     );
219 
220     _nftInMarketplace.push(
221         NFTSellInfo({
222             nftTokenAddress : nftTokenAddress,
223             tokenId: tokenId,
224             currencyTokenAddress : currencyTokenAddress,
225             price: price,
226             seller: msg.sender,
227             fee: fee
228         })
229     );
```

```
230
231         _indexOfTokenIds[nftTokenAddress][tokenId] = _nftInMarketplace.length - 1;
232         emit Sell(msg.sender, nftTokenAddress, tokenId, currencyTokenAddress,
    price, fee,  block.timestamp);
233
234     }
```

5.8.2. Remediation

Inspex suggests adding a validation check to ensure an NFT cannot be sold more than once.

For instance, introduce a state marker to indicate whether the NFT has already been sold for the `AstronizeCouponNFTKAP721` contract.

ast_nft_kap721.sol

```
70  mapping(uint256 => bool) private isSold;
71
72  function canSell(uint256 _tokenId) external view returns (bool) {
73      return !isSold[_tokenId];
74  }
75
76  function setSold(uint256 _tokenId) external onlyMarketplace {
77      isSold[_tokenId] = true;
78
79  }
```

Furthermore, it is advised to implement this validation in both the `AstronizeAuction`, `AstronizeOffer` and `AstronizeMarketplace` contracts to prevent the resale of an NFT that has been sold previously.

AstronizeAuction.sol

```
325 function _sell(
326     address _nftAddress,
327     uint256 _tokenId,
328     address _tokenAddress,
329     uint256 _price,
330     uint256 _startAt,
331     uint256 _endAt,
332     address _user
333 ) internal whenNotPaused {
334     require(AstronizeCouponNFTKAP721(_nftAddress).canSell(_tokenId), "nft can
    not sell");
335     // check start at
336     if (_startAt < block.timestamp) {
337         _startAt = block.timestamp;
338     }
339
```

```
340     // check time
341     require(_startAt < _endAt, "startAt must be less than endAt");
342     require(_endAt >= block.timestamp + minDuration, "endAt");
343     require(
344         _endAt < block.timestamp + (90 days),
345         "endAt must be less than 60 days"
346     );
347     require((_endAt - _startAt) >= minDuration, "duration");
348 
349     // check nft and token
350     require(_whitelistedNfts[_nftAddress], "whitelisted NFT");
351     WhitelistedToken storage _whitelistedToken = _whitelistedTokens[
352         _tokenAddress
353     ];
354     require(_price % _whitelistedToken.step == 0, "price step");
355     require(_whitelistedToken.isActive, "whitelisted token");
356 
357     // store item
358     Item memory _item = Item({
359         sellerAddress: _user,
360         tokenAddress: _tokenAddress,
361         price: _price,
362         startAt: _startAt,
363         endAt: _endAt,
364         bidderAddress: address(0),
365         status: ItemStatus.Active
366     });
367     _items[_nftAddress][_tokenId] = _item;
368 
369     // event
370     emit Sell(_user, _nftAddress, _tokenId, _item, block.timestamp);
371 }
```

Additionally, mark the item as sold once the selling is completed.

AstronizeAuction.sol

```
580 function _claim(
581     address _nftAddress,
582     uint256 _tokenId,
583     address _user
584 ) internal whenNotPaused {
585     // check expire time
586     require(isExpired(_nftAddress, _tokenId), "time");
587 
588     // check item
589     Item storage _item = _items[_nftAddress][_tokenId];
590     require(_item.status == ItemStatus.Active, "status");
```

```
591         _item.status = ItemStatus.Claimed;
592
593         // transfer nft to bidder
594         IKAP721(_nftAddress).safeTransferFrom(
595             address(this),
596             _item.bidderAddress,
597             _tokenId
598         );
599
600         IKAP721(_nftAddress).setSold(_tokenId);
601
602         // transfer token to Seller
603         (uint256 _sellerPrice, uint256 _fee) = _splitPrice(
604             _item.price,
605             feePercentage
606         );
607         IKAP20(_item.tokenAddress).transfer(_item.sellerAddress, _sellerPrice);
608         if (_fee > 0) {
609             IKAP20(_item.tokenAddress).transfer(treasuryAddress, _fee);
610         }
611
612         emit Claim(
613             _user,
614             _nftAddress,
615             _tokenId,
616             _item.tokenAddress,
617             _item.sellerAddress,
618             _item.bidderAddress,
619             _item.price,
620             block.timestamp
621         );
622     }
```

The changes above should be implemented in the `_makeOffer()` and `_acceptOffer()` functions within the `AstronizeOffer` contract, as well as in the `sellNFT()` and `buyNFT()` functions of the `AstronizeMarketplace` contract.

5.9. Denial of Services in claim() Function

ID	IDX-009
Target	AstronizeAuction
Category	Advanced Smart Contract Vulnerability
CWE	CWE-755: Improper Handling of Exceptional Conditions
Risk	**Severity:** Medium **Impact:** High The seller's NFT will be stuck in the contract, and they will be unable to cancel to retrieve the NFT back. **Likelihood:** Low The attacker needs to win the bid, and there is a `cancelForAdmin()` function that can be used by the platform owner to forcibly cancel the bid.
Status	Resolved The Astronize team has resolved this issue by changing from `safeTransferFrom()` to `transferFrom()` for transfer NFT in commit `eebc41d0e527666093bd751600656262f5363673`.

5.9.1. Description

In the `AstronizeAuction` contract, the seller can use an NFT for auction, and buyers can bid on the NFT with a higher price. After the expiration of the time limit, the last bidder can claim the NFT.

However, the `_claim()` function uses `safeTransferFrom()`, which has a hook that calls the `onKAP721Received()` function if the target is a contract. A malicious bidder can implement the `onKAP721Received()` function with a revert inside. This could prevent the owner of the NFT from canceling and receiving the token price for the NFT.

AstronizeAuction.sol

```
580 function _claim(
581     address _nftAddress,
582     uint256 _tokenId,
583     address _user
584 ) internal whenNotPaused {
585     // check expire time
586     require(isExpired(_nftAddress, _tokenId), "time");
587 
588     // check item
589     Item storage _item = _items[_nftAddress][_tokenId];
590     require(_item.status == ItemStatus.Active, "status");
```

```
591         _item.status = ItemStatus.Claimed;
592
593         // transfer nft to bidder
594         IKAP721(_nftAddress).safeTransferFrom(
595             address(this),
596             _item.bidderAddress,
597             _tokenId
598         );
599
600         // transfer token to Seller
601         (uint256 _sellerPrice, uint256 _fee) = _splitPrice(
602             _item.price,
603             feePercentage
604         );
605         IKAP20(_item.tokenAddress).transfer(_item.sellerAddress, _sellerPrice);
606         if (_fee > 0) {
607             IKAP20(_item.tokenAddress).transfer(treasuryAddress, _fee);
608         }
609
610         emit Claim(
611             _user,
612             _nftAddress,
613             _tokenId,
614             _item.tokenAddress,
615             _item.sellerAddress,
616             _item.bidderAddress,
617             _item.price,
618             block.timestamp
619         );
620     }
```

5.9.2. Remediation

Inspex suggests implementing the function for buyers to claim NFT and the function for sellers to claim tokens separately after the auction ends.

Alternatively, Inspex suggests using the `transferFrom()` function instead of the `safeTransferFrom()` function to prevent the `onKAP721Received()` callback execution while transferring an NFT.

5.10. Unrestricted Gashapon Purchase Limit

ID	IDX-010
Target	AstronizeGashapon
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Low **Impact:** Low The `setRequest()` function can be used with an amount exceeding the limit specified by the business requirements. **Likelihood:** Medium There is no restriction on the amount parameter; anyone can purchase with any amount.
Status	Resolved The Astronize team has resolved this issue by adding max purchase limit in the `_setRequest()` function in commit `eebc41d0e527666093bd751600656262f5363673`.

5.10.1. Description

The `setRequest()` function enables users to submit requests with amounts that exceed the limit specified in the business requirements. This allows the caller of the `setRequest()` function to purchase NFT collections without any limits.

AstronizeGashapon.sol

```
255 function setRequest(
256     uint256 _collectionId,
257     address _tokenAddress,
258     uint256 _price,
259     uint256 _amount
260 ) external {
261     _setRequest(_collectionId, _tokenAddress, _price, _amount, msg.sender);
262 }
```

AstronizeGashapon.sol

```
280 function _setRequest(
281     uint256 _collectionId,
282     address _tokenAddress,
283     uint256 _price,
284     uint256 _amount,
285     address _user
286 ) internal {
```

```
287     Request storage _request = _requests[_user][_collectionId];
288
289     _request.amount = _amount;
290     _request.price = _price;
291     _request.tokenAddress = _tokenAddress;
292
293     emit RequestSet(
294         _user,
295         _collectionId,
296         _tokenAddress,
297         _price,
298         _amount,
299         block.timestamp
300     );
301 }
```

5.10.2. Remediation

Inspex suggests adding a restriction for the purchase amount according to business requirements.

5.11. Loop Over Unbounded Data Structure

ID	IDX-011
Target	AstronizeVesting
Category	General Smart Contract Vulnerability
CWE	CWE-400: Uncontrolled Resource Consumption
Risk	**Severity:** Low **Impact:** Medium The `_transferToken()` function will eventually be unusable due to excessive gas usage. **Likelihood:** Low It is very unlikely that the `_wallets` size will be added until the `_transferToken()` is eventually unusable.
Status	Resolved The Astronize team has resolved this issue by using a pull-over-push pattern for users to claim tokens in commit `eebc41d0e527666093bd751600656262f5363673`.

5.11.1. Description

The `claim()` function in the `AstronizeVesting` contract is used to transfer tokens to a list of wallets after certain conditions are met.

AstronizeVesting.sol

```
115 function claim(uint256 month, uint256 year) external {
116     require(desiredNumWallets == walletLength(), "invalid num wallets");
117
118     // check month
119     require(inArray(claimableMonths, month), "invalid month");
120
121     // check time
122     uint256 claimTime = getTimestampForMonthAndYear(month, year);
123     require(block.timestamp >= claimTime, "not yet time");
124     require(claimTime >= startAt, "start at");
125
126     // check claim
127     require(!_claims[month][year], "already claimed");
128     _claims[month][year] = true;
129
130     // transfer token
131     _transferToken();
132
133     // emit event
```

```
134         emit Claim(msg.sender, month, year, block.timestamp);
135     }
```

The `_transferToken()` function executes the `transfer()` function, transfers token to users for all added wallets, as shown below:

AstronizeVesting.sol

```
115 function _transferToken() internal {
116     uint256 _transferAmount;
117 
118     // check is first transfer
119     if (!isFirstTransfer) {
120         isFirstTransfer = true;
121         _transferAmount = firstTransferAmount;
122     }
123 
124     // check transfer amount
125     if (_transferAmount == 0) {
126         _transferAmount = transferAmount;
127     }
128 
129     // transfer
130     uint256 numUsers = _wallets.length();
131     for (uint256 i = 0; i < numUsers; i++) {
132         address user = _wallets.at(i);
133         token.transfer(user, _transferAmount);
134     }
135 }
```

However, in the current design, the `_transferToken()` function will execute within a single loop continuously. If an error or revert occurs, the other wallets will be affected and unable to receive a token. It could be more practical if the executor could decide the range of wallets to transfer, thus avoiding potential errors.

5.11.2. Remediation

Inspex suggests considering implementing the `claim()` function with a pull-over-push pattern instead.

5.12. Lack of whenNotPaused Modifier for the BitkubNext Users

ID	IDX-012
Target	AstronizeMarketplace
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** Low **Impact:** Medium Users utilizing the `BitkubNext` wallet retain the capability to execute all actions within the marketplace, including buying and selling. **Likelihood:** Low This issue may occur when the platform decides to temporarily pause operations on the marketplace, but users of `BitkubNext` wallet will still be able to carry out all actions.
Status	Resolved The Astronize team has resolved this issue by adding `whenNotPaused` Modifier for the listed function in commit `a30967a22fc85c55ec08c86a2066f975d439a408`

5.12.1. Description

The `sellNFT()` and `buyNFT()` functions in the `AstronizeMarketplace` contract lack of the modifiers to limit an execution by the `BitkubNext` users.

astronize_mkp.sol

```
159 function sellNFT(
160     address nftTokenAddress,
161     uint256 tokenId,
162     address currencyTokenAddress,
163     uint256 price,
164     address _bitkubnext
165 ) external onlyCallHelper onlyBitkubNextUser(_bitkubnext) {
166     require(price > 0, "price must be greater than 0");
167     require(
168         isWhitelistNFTToken(nftTokenAddress),
169         "nft token address not whitelisted"
170     );
171
172     require(
173         isWhitelistCurrencyToken(currencyTokenAddress),
174         "currency token address not whitelisted"
175     );
176
```

```
177     //bitkubnext transfer
178     nextNFTTransferRouterKap721.transferFromKAP721(PROJECT,
    address(nftTokenAddress), _bitkubnext, address(this), tokenId);
179
180
181     _nftInMarketplace.push(
182         NFTSellInfo({
183             nftTokenAddress : nftTokenAddress,
184             tokenId: tokenId,
185             currencyTokenAddress : currencyTokenAddress,
186             price: price,
187             seller: _bitkubnext,
188             fee: fee
189         })
190     );
191
192     _indexOfTokenIds[nftTokenAddress][tokenId] = _nftInMarketplace.length - 1;
193     emit Sell(_bitkubnext, nftTokenAddress, tokenId, currencyTokenAddress,
    price, fee,  block.timestamp);
194
195 }
```

astronize_mkp.sol

```
271 function buyNFT(address nftTokenAddress, uint256 tokenId, address
    currencyTokenAddress, uint256 price, address _bitkubnext) external
    onlyCallHelper onlyBitkubNextUser(_bitkubnext) {
272     NFTSellInfo memory item = itemByTokenId(nftTokenAddress, tokenId);
273
274     require(item.price == price, "price must match");
275     require(item.currencyTokenAddress == currencyTokenAddress, "currency token
    address must match");
276
277     //cal fee
278     uint256 toTreasury = _calculateFee(item.price, item.fee);
279
280     //fee (bitkubnext transfer)
281     nextTransferRouterKap20.transferFrom(PROJECT,
    address(item.currencyTokenAddress), _bitkubnext, treasuryAddress, toTreasury);
282     nextTransferRouterKap20.transferFrom(PROJECT,
    address(item.currencyTokenAddress), _bitkubnext, item.seller, item.price -
    toTreasury);
283
284     _removeItemByIndex(indexByTokenId(nftTokenAddress, tokenId));
285
286     IERC721(nftTokenAddress).safeTransferFrom(
287         address(this),
288         _bitkubnext,
```

```
289             tokenId
290         );
291
292         emit Buy(
293             _bitkubnext,
294             nftTokenAddress,
295             tokenId,
296             currencyTokenAddress,
297             item.seller,
298             item.price,
299             item.fee,
300             toTreasury,
301             block.timestamp
302         );
303
304     }
```

Consequently, `BitkubNext` wallet users could gain an unfair advantage, allowing them to buy and sell NFTs without the limitations. This could disrupt the balance and fairness of the marketplace.

5.12.2. Remediation

Inspex suggests adding the `whenNotPaused` modifier into the `sellNFT()` and `buyNFT()` functions. This addition will prevent `BitkubNext` users from executing operations while the marketplace is in a paused state.

astronize_mkp.sol

```
159 function sellNFT(
160     address nftTokenAddress,
161     uint256 tokenId,
162     address currencyTokenAddress,
163     uint256 price,
164     address _bitkubnext
165 ) external onlyCallHelper onlyBitkubNextUser(_bitkubnext) whenNotPaused {
166     require(price > 0, "price must be greater than 0");
167     require(
168         isWhitelistNFTToken(nftTokenAddress),
169         "nft token address not whitelisted"
170     );
171
172     require(
173         isWhitelistCurrencyToken(currencyTokenAddress),
174         "currency token address not whitelisted"
175     );
176
177     //bitkubnext transfer
178     nextNFTTransferRouterKap721.transferFromKAP721(PROJECT,
```

```
    address(nftTokenAddress), _bitkubnext, address(this), tokenId);
179
180
181         _nftInMarketplace.push(
182             NFTSellInfo({
183                 nftTokenAddress : nftTokenAddress,
184                 tokenId: tokenId,
185                 currencyTokenAddress : currencyTokenAddress,
186                 price: price,
187                 seller: _bitkubnext,
188                 fee: fee
189             })
190         );
191
192         _indexOfTokenIds[nftTokenAddress][tokenId] = _nftInMarketplace.length - 1;
193         emit Sell(_bitkubnext, nftTokenAddress, tokenId, currencyTokenAddress,
    price, fee,  block.timestamp);
194
195 }
```

astronize_mkp.sol

```
271 function buyNFT(address nftTokenAddress, uint256 tokenId, address
    currencyTokenAddress, uint256 price, address _bitkubnext) external
    onlyCallHelper onlyBitkubNextUser(_bitkubnext) whenNotPaused {
272     NFTSellInfo memory item = itemByTokenId(nftTokenAddress, tokenId);
273
274     require(item.price == price, "price must match");
275     require(item.currencyTokenAddress == currencyTokenAddress, "currency token
    address must match");
276
277     //cal fee
278     uint256 toTreasury = _calculateFee(item.price, item.fee);
279
280     //fee (bitkubnext transfer)
281     nextTransferRouterKap20.transferFrom(PROJECT,
    address(item.currencyTokenAddress), _bitkubnext, treasuryAddress, toTreasury);
282     nextTransferRouterKap20.transferFrom(PROJECT,
    address(item.currencyTokenAddress), _bitkubnext, item.seller, item.price -
    toTreasury);
283
284     _removeItemByIndex(indexByTokenId(nftTokenAddress, tokenId));
285
286     IERC721(nftTokenAddress).safeTransferFrom(
287         address(this),
288         _bitkubnext,
289         tokenId
290     );
```

```
291
292     emit Buy(
293         _bitkubnext,
294         nftTokenAddress,
295         tokenId,
296         currencyTokenAddress,
297         item.seller,
298         item.price,
299         item.fee,
300         toTreasury,
301         block.timestamp
302     );
303
304 }
```

5.13. Outdated Compiler Version

ID	IDX-013
Target	ASTTokenKAP20 AstToken AstronizeCouponNFTKAP721 AstrobeamBridge AstronizeNFTBridge AstronizeAuction AstronizeGashapon AstronizeMarketplace AstronizeOffer TSXToken AstronizeVesting
Category	General Smart Contract Vulnerability
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Very Low **Impact:** Low From the list of known Solidity bugs, direct impact cannot be caused from those bugs themselves. **Likelihood:** Low From the list of known Solidity bugs, it is very unlikely that those bugs would affect these smart contracts.
Status	Resolved The Astronize team has resolved this issue by changing solidity compiler version to 0.8.19 in `6416da05140bccae536c14534e7c534efef29c61`

5.13.1. Description

The Solidity compiler versions specified in the smart contracts were outdated (https://docs.soliditylang.org/en/latest/bugs.html). As the compilers are regularly updated with bug fixes and new features, the latest stable compiler version should be used to compile the smart contracts for best practice.

astronize_nft_bridge.sol

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity 0.8.10;
```

The table below represents the contracts that apply the outdated Solidity compiler version.

File	Version
ast_kap20.sol (L:4)	0.8.10
AstTokenForEth.sol (L:2)	0.8.19
ast_nft_kap721.sol (L:3)	0.8.10
AstrobeamBridge.sol (L:2)	0.8.19
astronize_nft_bridge.sol (L:2)	0.8.10
AstronizeAuction.sol (L:2)	0.8.19
AstronizeGashapon.sol (L:2)	0.8.19
astronize_mkp.sol (L:2)	0.8.10
AstronizeOffer.sol (L:2)	0.8.19
TSXToken.sol (L:2)	0.8.19
AstronizeVesting.sol (L:2)	0.8.19

5.13.2. 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 version of the Solidity compiler in major 0.8 is 0.8.24.

For chains that may not be compatible with Solidity compiler version 0.8.24, Inspex suggests using Solidity compiler version 0.8.19 instead, as Solidity compiler version 0.8.20 or later introduces the PUSH0 (0x5f) opcode, which some chains have not yet included.
(https://github.com/ethereum/solidity/releases/tag/v0.8.20)

astronize_nft_bridge.sol

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity 0.8.24;
```

5.14. Improper Function Visibility

ID	IDX-014
Target	AstronizeCouponNFTKAP721 AstrobeamBridge AstronizeNFTBridge AstronizeAuction AstronizeGashapon AstronizeMarketplace AstronizeOffer AstronizeVesting
Category	Smart Contract Best Practice
CWE	CWE-710: Improper Adherence to Coding
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The Astronize team has resolved this issue by changing function visibility to external in commit `9802a538d1a019496c2110d5887e21c74b9ef9c0`.

5.14.1. 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 function visibility is set to public and it is never called from any internal function.

ast_nft_kap721.sol

```
101 function tokensOfOwner(address owner) public view returns (uint256[] memory) {
102
103     uint256 balance = balanceOf(owner);
104
105     uint256[] memory itemList = new uint256[](balance);
106     for (uint256 i=0; i < balance; i++){
107         itemList[i] = tokenOfOwnerByIndex(owner, i);
108     }
109     return itemList;
110 }
```

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

File	Contract	Function
ast_nft_kap721.sol (L: 101)	AstronizeCouponNFTKAP721	tokensOfOwner()
AstrobeamBridge.sol (L: 144)	AstrobeamBridge	nonceOf()
AstrobeamBridge.sol (L: 169)	AstrobeamBridge	getValidators()
AstrobeamBridge.sol (L: 191)	AstrobeamBridge	nonceValidatorOf()
AstrobeamBridge.sol (L: 213)	AstrobeamBridge	depositSharePercentageOf()
astronize_nft_bridge.sol (L: 112)	AstronizeNFTBridge	setASTTokenAddress()
astronize_nft_bridge.sol (L: 116)	AstronizeNFTBridge	setAcceptedKycLevel()
astronize_nft_bridge.sol (L: 120)	AstronizeNFTBridge	setKyc()
astronize_nft_bridge.sol (L: 124)	AstronizeNFTBridge	setCallHelper()
astronize_nft_bridge.sol (L: 128)	AstronizeNFTBridge	setNextTransferRouter()
astronize_nft_bridge.sol (L: 132)	AstronizeNFTBridge	getMintFee()
astronize_nft_bridge.sol (L: 136)	AstronizeNFTBridge	setMintFee()
astronize_nft_bridge.sol (L: 140)	AstronizeNFTBridge	setWhitelistNFTToken()
astronize_nft_bridge.sol (L: 374)	AstronizeNFTBridge	grantTrustedValidator()
astronize_nft_bridge.sol (L: 387)	AstronizeNFTBridge	revokeTrustedValidator()
AstronizeAuction.sol (L: 136)	AstronizeAuction	whitelistedNftOf()
AstronizeAuction.sol (L: 143)	AstronizeAuction	whitelistedTokenOf()
AstronizeAuction.sol (L: 152)	AstronizeAuction	itemOf()
AstronizeGashapon.sol (L: 105)	AstronizeGashapon	collectionOf()
AstronizeGashapon.sol (L: 114)	AstronizeGashapon	requestOf()
astronize_mkp.sol (L: 139)	AstronizeMarketplace	setAcceptedKycLevel()
astronize_mkp.sol (L: 143)	AstronizeMarketplace	setKyc()
astronize_mkp.sol (L: 147)	AstronizeMarketplace	setCallHelper()
astronize_mkp.sol (L: 151)	AstronizeMarketplace	setNextTransferRouter()

astronize_mkp.sol (L: 155)	AstronizeMarketplace	setNextNFTTransferRouter()
astronize_mkp.sol (L: 360)	AstronizeMarketplace	setWhitelistNFTToken()
astronize_mkp.sol (L: 415)	AstronizeMarketplace	itemInMarketplace()
astronize_mkp.sol (L: 431)	AstronizeMarketplace	itemCount()
astronize_mkp.sol (L: 451)	AstronizeMarketplace	onKAP721Received()
AstronizeOffer.sol (L: 112)	AstronizeOffer	offerOf()
AstronizeVesting.sol (L: 176)	AstronizeVesting	wallets()

5.14.2. Remediation

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

ast_nft_kap721.sol

```
101 function tokensOfOwner(address owner) external view returns (uint256[] memory)
    {
102
103     uint256 balance = balanceOf(owner);
104
105     uint256[] memory itemList = new uint256[](balance);
106     for (uint256 i=0; i < balance; i++){
107         itemList[i] = tokenOfOwnerByIndex(owner, i);
108     }
109     return itemList;
110 }
```

5.15. Insufficient Logging for Privileged Functions

ID	IDX-015
Target	ASTTokenKAP20 AstronizeCouponNFTKAP721 AstronizeNFTBridge AstronizeMarketplace
Category	General Smart Contract Vulnerability
CWE	CWE-778: Insufficient Logging
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The Astronize team has resolved this issue by adding event and emit in the functions in commit `481f350145595a608cc2929363e2a6332566ace1`.

5.15.1. 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 owner can withdraw the ether in the contract by executing the `setOwnerAccessControlRouter()` function in the `ASTTokenKAP20` contract, and no events are emitted.

ast_kap20.sol

```
78 function setOwnerAccessControlRouter(address _ownerAccessControlRouter)
   external onlyOwner {
79     ownerAccessControlRouter =
   IOwnerAccessControlRouter(_ownerAccessControlRouter);
80 }
```

The following table contains all events that emit an incorrect event.

File	Contract	Event
ast_kap20.sol (L: 78)	ASTTokenKAP20	setOwnerAccessControlRouter()
ast_nft_kap721.sol (L: 70)	AstronizeCouponNFTKAP721	setTokenURI()
ast_nft_kap721.sol (L: 83)	AstronizeCouponNFTKAP721	setBaseURI()

astronize_nft_bridge.sol (L: 112)	AstronizeNFTBridge	setASTTokenAddress()
astronize_nft_bridge.sol (L: 116)	AstronizeNFTBridge	setAcceptedKycLevel()
astronize_nft_bridge.sol (L: 120)	AstronizeNFTBridge	setKyc()
astronize_nft_bridge.sol (L: 124)	AstronizeNFTBridge	setCallHelper()
astronize_nft_bridge.sol (L: 128)	AstronizeNFTBridge	setNextTransferRouter()
astronize_nft_bridge.sol (L: 136)	AstronizeNFTBridge	setMintFee()
astronize_nft_bridge.sol (L: 374)	AstronizeNFTBridge	grantTrustedValidator()
astronize_nft_bridge.sol (L: 387)	AstronizeNFTBridge	revokeTrustedValidator()
astronize_mkp.sol (L: 139)	AstronizeMarketplace	setAcceptedKycLevel()
astronize_mkp.sol (L: 139)	AstronizeMarketplace	setAcceptedKycLevel()
astronize_mkp.sol (L: 143)	AstronizeMarketplace	setKyc()
astronize_mkp.sol (L: 147)	AstronizeMarketplace	setCallHelper()
astronize_mkp.sol (L: 151)	AstronizeMarketplace	setNextTransferRouter()
astronize_mkp.sol (L: 155)	AstronizeMarketplace	setNextNFTTransferRouter()

5.15.2. Remediation

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

ast_kap20.sol

```
78 event OwnerAccessControlRouterSet(address indexed operator, address indexed
   oldAddress, address indexed newAddress);
79 function setOwnerAccessControlRouter(address _ownerAccessControlRouter)
   external onlyOwner {
80     emit OwnerAccessControlRouterSet(msg.sender,
   address(ownerAccessControlRouter), _ownerAccessControlRouter);
81     ownerAccessControlRouter =
   IOwnerAccessControlRouter(_ownerAccessControlRouter);
82 }
```

5.16. Unchecked Return Value ERC20 Transfer

ID	IDX-016
Target	AstronizeVesting AstronizeAuction AstrobeamBridge AstronizeBitkubBase
Category	General Smart Contract Vulnerability
CWE	CWE-710: Improper Adherence to Coding Standard
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The Astronize team has resolved this issue by using `safeTransfer()` instead in commit `eebc41d0e527666093bd751600656262f5363673`.

5.16.1. Description

ERC20 tokens can be implemented in multiple ways, allowing the execution of failed `transfer()` and `transferFrom()` functions by returning `false` instead of reverting when the invalid transfer amount occurs.

In the `AstronizeVesting` contract, the `_transferToken()` function can be called from `claim()` and used to transfer tokens in the contract to the users.

AstronizeVesting.sol

```
115 function _transferToken() internal {
116     uint256 _transferAmount;
117 
118     // check is first transfer
119     if (!isFirstTransfer) {
120         isFirstTransfer = true;
121         _transferAmount = firstTransferAmount;
122     }
123 
124     // check transfer amount
125     if (_transferAmount == 0) {
126         _transferAmount = transferAmount;
127     }
128 
129     // transfer
130     uint256 numUsers = _wallets.length();
```

```
131     for (uint256 i = 0; i < numUsers; i++) {
132         address user = _wallets.at(i);
133         token.transfer(user, _transferAmount);
134     }
135 }
```

The return value of the `transfer()` function is not checked, so the transfer transactions of tokens that return false on failure will not be reverted.

The following table contains all functions that unchecked transfer.

File	Contract	Function
AstronizeBitkubBase.sol (L: 125)	AstronizeBitkubBase	_transferToken()
AstrobeamBridge.sol (L: 263)	AstrobeamBridge	withdraw()
AstrobeamBridge.sol (L: 437)	AstrobeamBridge	recoverToken()
AstronizeAuction.sol (L: 400)	AstronizeAuction	_bid()
AstronizeAuction.sol (L: 545)	AstronizeAuction	_cancel()
AstronizeAuction.sol (L: 605, 607)	AstronizeAuction	_claim()
AstronizeVesting.sol (L: 133)	AstronizeVesting	_transferToken()

5.16.2. Remediation

Inspex suggests replacing the `transfer()` function with `safeTransfer()` function from OpenZeppelin's `SafeERC20` library, for example:

AstronizeVesting.sol

```
115 function _transferToken() internal {
116     uint256 _transferAmount;
117 
118     // check is first transfer
119     if (!isFirstTransfer) {
120         isFirstTransfer = true;
121         _transferAmount = firstTransferAmount;
122     }
123 
124     // check transfer amount
125     if (_transferAmount == 0) {
126         _transferAmount = transferAmount;
127     }
128 
129     // transfer
```

```
130         uint256 numUsers = _wallets.length();
131         for (uint256 i = 0; i < numUsers; i++) {
132             address user = _wallets.at(i);
133             token.safeTransfer(user, _transferAmount);
134         }
135     }
```

5.17. Misleading Error Message

ID	IDX-017
Target	AstronizeAuction
Category	Smart Contract Best Practice
CWE	CWE-1116: Inaccurate Comments
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The Astronize team has resolved this issue by removing the misleading comment in commit `eebc41d0e527666093bd751600656262f5363673`.

5.17.1. Description

The message when the required condition fails does not match the code, as you can see in lines 343-344.

AstronizeAuction.sol

```
325 function _sell(
326     address _nftAddress,
327     uint256 _tokenId,
328     address _tokenAddress,
329     uint256 _price,
330     uint256 _startAt,
331     uint256 _endAt,
332     address _user
333 ) internal whenNotPaused {
334     // check start at
335     if (_startAt < block.timestamp) {
336         _startAt = block.timestamp;
337     }
338 
339     // check time
340     require(_startAt < _endAt, "startAt must be less than endAt");
341     require(_endAt >= block.timestamp + minDuration, "endAt");
342     require(
343         _endAt < block.timestamp + (90 days),
344         "endAt must be less than 60 days"
345     );
346     require((_endAt - _startAt) >= minDuration, "duration");
347 
```

```
348     // check nft and token
349     require(_whitelistedNfts[_nftAddress], "whitelisted NFT");
350     WhitelistedToken storage _whitelistedToken = _whitelistedTokens[
351         _tokenAddress
352     ];
353     require(_price % _whitelistedToken.step == 0, "price step");
354     require(_whitelistedToken.isActive, "whitelisted token");
355
356     // store item
357     Item memory _item = Item({
358         sellerAddress: _user,
359         tokenAddress: _tokenAddress,
360         price: _price,
361         startAt: _startAt,
362         endAt: _endAt,
363         bidderAddress: address(0),
364         status: ItemStatus.Active
365     });
366     _items[_nftAddress][_tokenId] = _item;
367
368     // event
369     emit Sell(_user, _nftAddress, _tokenId, _item, block.timestamp);
370 }
```

5.17.2. Remediation

Inspex suggests changing the condition or error message to satisfy the business requirements.

5.18. Duplicated require() Check

ID	IDX-018
Target	AstronizeGashapon
Category	Smart Contract Best Practice
CWE	CWE-1041: Use of Redundant Code
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The Astronize team has resolved this issue by changing require condition in commit `eebc41d0e527666093bd751600656262f5363673`

5.18.1. Description

In the `AstronizeGashapon` contract, the `purchase()` function has duplicated `require` checks for the same condition, as you can see in lines 317-321.

AstronizeGashapon.sol

```
306 function purchase(
307     uint256 _collectionId,
308     address _user,
309     bytes32[] calldata _seeds,
310     uint256 _deadline
311 ) external onlyOperator {
312     // check dead line
313     require(_deadline > block.timestamp, "deadline");
314 
315     // check collection status
316     Collection storage _collection = _collections[_collectionId];
317     require(_collection.tokenIds.length >= _seeds.length, "amount");
318     require(
319         _collection.tokenIds.length >= _seeds.length,
320         "remaining amount"
321     );
322 
323     // check request
324     Request storage _request = _requests[_user][_collectionId];
325     require(_request.price == _collection.price, "price");
326     require(
327         _request.tokenAddress == _collection.tokenAddress,
```

```
328             "token address"
329         );
330         require(_request.amount >= _seeds.length, "amount");
331         _request.amount -= _seeds.length;
332 
333         // random NFTs and send to user
334         uint256[] memory _tokenIds = new uint256[](_seeds.length);
335         for (uint256 i = 0; i < _seeds.length; i++) {
336             // random pick
337             nonce++;
338             uint256 _pickIndex = _random(
339                 _collection.tokenIds.length,
340                 nonce,
341                 _seeds[i]
342             );
343             uint256 _tokenId = _collection.tokenIds[_pickIndex];
344             _removeItemByIndex(_collection.tokenIds, _pickIndex);
345 
346             // transfer
347             IKAP721(_collection.nftAddress).safeTransferFrom(
348                 address(this),
349                 _user,
350                 _tokenId
351             );
352 
353             _tokenIds[i] = _tokenId;
354         }
355 
356         // transfer token
357         _transferTokens(_collectionId, _user, _seeds.length);
358 
359         emit Purchase(
360             _user,
361             _collectionId,
362             _collection.nftAddress,
363             _tokenIds,
364             _collection.tokenIds.length,
365             _request.tokenAddress,
366             _collection.price,
367             block.timestamp
368         );
369     }
```

5.18.2. Remediation

Inspex suggests removing the unused code or reviewing if there are more conditions needed for business requirements.

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

inspeX

CYBERSECURITY PROFESSIONAL SERVICE