Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: Anime Metaverse
Website: https://animemetaverse.ai
Platform: Ethereum
Language: Solidity
Date: May 29th, 2022

Table of contents

Email: audit@EtherAuthority.io

`

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by the Anime Metaverse team to perform the Security audit of the Anime Metaverse Staking smart contract code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on May 29th, 2022.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

- This project consists of the staking smart contract of the Anime Metaverse platform.
- The Anime Metaverse is the intersection of Web3 Ownership, Anime, and Fashion.
- In this smart contract, users can stake the AMS tokens and unstake them at any time. Users do not get any monetary rewards by staking the AMS tokens.

Audit scope

Name	**Code Review and Security Analysis Report for Anime Metaverse Staking Smart Contract**
Platform	**Ethereum / Solidity**
File	AmvNftStaker.sol
File MD5 Hash	C3F05675E29C0A3163AC8D2BF001592A
Online Code Link	0x609A6FB483fa95496Ceee2163189Af36B782aE95
Audit Date	May 29th, 2022

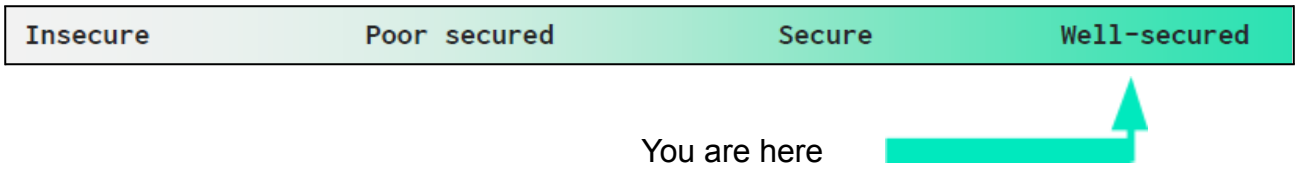
Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
AmvNftStaker: • Staking NFT Token: AMS • Users can stake it • Users can unstake it anytime • Max tokens can be staked at a time: 50 • Max tokens can be unstaked at a time: 50 • Users do not get monetary rewards by staking AMS tokens.	**YES, This is valid.**
Ownership Control: • Owner can transfer the contract ownership • Owner can renounce the contract ownership	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity based smart contracts are **"Well Secured"**. This staking smart contract does contain ownership smart contract, but it does not have any functionality for owner, hence this stakung contract is fully **decentralized**.



We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium and 1 low and some very low level issues.

Investors Advice: Technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Passed
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Moderated
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Moderated
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Email: audit@EtherAuthority.io

Code Quality

This audit scope has 1 smart contract. Smart contract contains Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in the Anime Metaverse Staking contract are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the Anime Metaverse Token.

The Anime Metaverse Token team has **not** provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are **well** commented on in the smart contracts. Ethereum's NatSpec commenting style was used, which is a good thing.

Documentation

We were given an Anime Metaverse Staking smart contract code in the form of an Etherscan web link. The hash of that code is mentioned above in the table.

As mentioned above, code parts are **well** commented. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure that are based on well known industry standard open source projects.

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	owner	Read	Passed	No Issue
2	renounceOwnership	Write	Passed	No Issue
3	transferOwnership	Write	Passed	No Issue
4	_transferOwnership	Internal	Passed	No Issue
5	constructor	Write	Passed	No Issue
6	stake	Write	Passed	No Issue
7	addNewNftToVault	Private	Passed	No Issue
8	addNewTokenIdToList	Private	Passed	No Issue
9	unstake	Write	Passed	No Issue
10	removeNftFromVault	Private	Passed	No Issue
11	tokensOfOwner	Read	Infinite loop possibility	Refer audit findings
12	onERC721Received	Read	Passed	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Infinite loop possibility

```
function tokensOfOwner(address account)
    public
    view
    returns (uint256[] memory ownrTokenIds)
{
    uint256 supply = nftTokenIds.length;
    uint256[] memory tmp = new uint256[](supply);

    uint256 nftCount = 0;
    for (uint256 i = 0; i < supply; i++) {
        Stake memory staked = vault[nftTokenIds[i]];
        if (staked.owner == account) {
            tmp[nftCount] = nftTokenIds[i];
            nftCount += 1;
        }
    }
    uint256[] memory ownerTokenIds = new uint256[](nftCount);
    for (uint256 i = 0; i < nftCount; i++) {
        ownerTokenIds[i] = tmp[i];
    }
    return ownerTokenIds;
}
```

The tokensOfOwner function has two loops which can hit a block's gas limit if there is a high number of staked NFTs. So, this function will stop working after many NFTs are staked.

Resolution: We suggest changing the code logic. Just define a mapping which maps the wallet to an array. This way all the user's staked NFTs can be tracked. On another hand, user's staked NFTs can be tracked through events as well.

Status: Open

Very Low / Informational / Best practices:

(1) Unnecessary Ownership contract

```
abstract contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(
        address indexed previousOwner,
        address indexed newOwner
    );
```

This Ownable (Ownership) contract is not used anywhere. There are no owner functions in the main staking contract. Thus this contract is redundant (useless). If this is not needed, then it is better to remove it to make the code clean and it will save some gas as well during contract deployment.

Resolution: Remove this Ownable contract if not needed.

Status: Open

(2) Declare variable constant

```
// Stores AMV smart contract details.
IERC721 public nft;
```

The variable nft is not being changed anywhere. So, it is recommended to declare it as a constant. This saves some gas.

Resolution: Just put the ‘constant’ keyword in the variable declaration.

Status: Open

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet private key would be compromised, then it would create trouble. Following are Admin functions:

- renounceOwnership: Owner can renounce (give up) the ownership.
- transferOwnership: Owner can transfer the ownership to another wallet.

On the other hand, the core stacking contract does not have any owner functions. So, whoever is the owner, does not make any difference in the functionality of the core staking smart contract.

Therefore, this staking smart contract does not have any human influence and thus it is **fully decentralized**.

Conclusion

We were given a contract code form of an Etherscan web link. And we have used all possible tests based on given objects as files. We have not observed any major issues. So, **it's good to go for the production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security state of the reviewed smart contract, based on standard audit procedure scope, is **"Well Secured".**

Email: audit@EtherAuthority.io

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

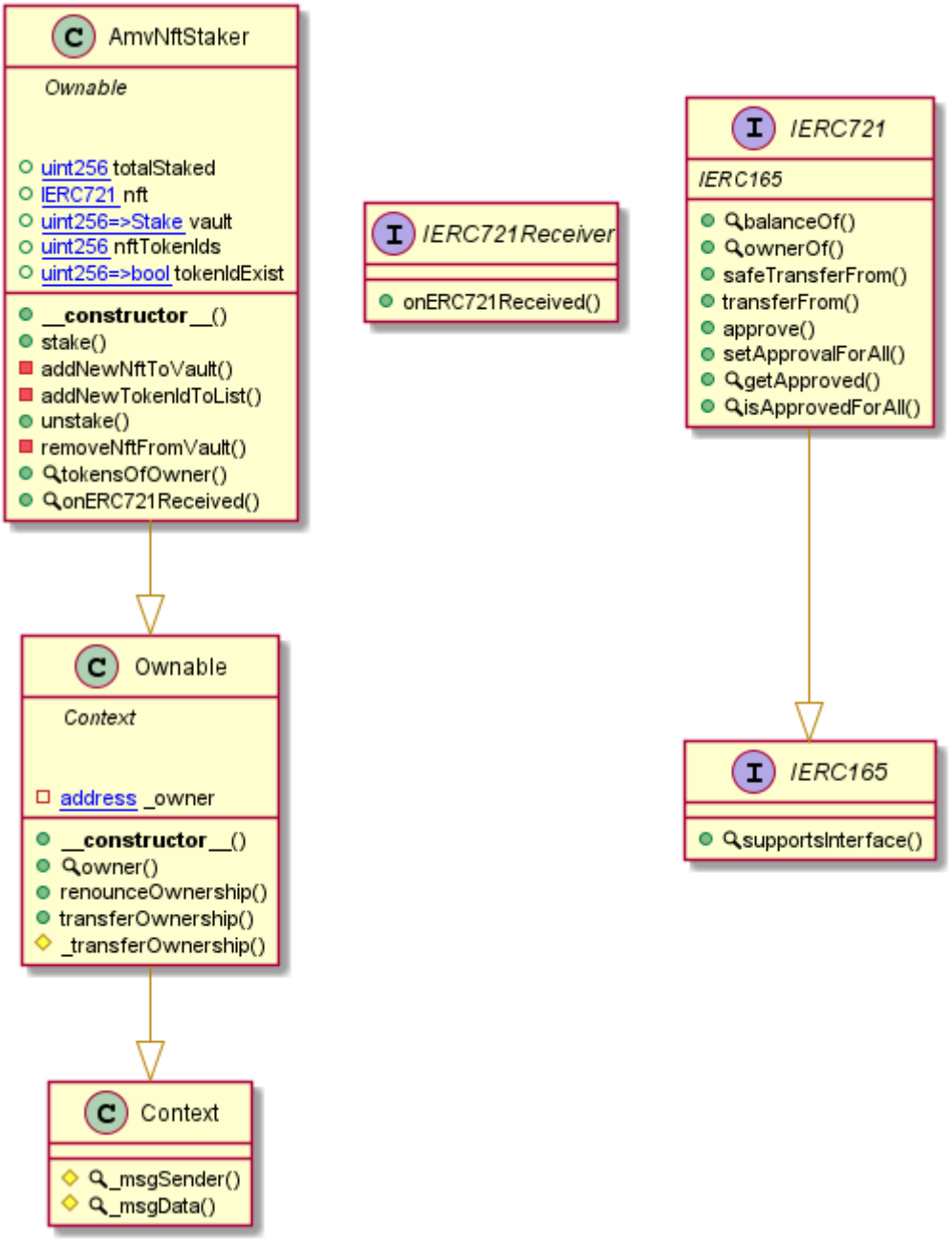
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - Anime Metaverse Staking



Slither Results Log

Slither Log >> AmvNftStaker.sol

```
INFO:Detectors:
AmvNftStaker.stake(uint256[]) (AmvNftStaker.sol#337-378) uses a dangerous strict equality:
        - require(bool,string)(vault[tokenId].tokenId == 0,token is already staked) (AmvNftStaker.sol#365)
AmvNftStaker.tokensOfOwner(address) (AmvNftStaker.sol#464-485) uses a dangerous strict equality:
        - staked.owner == account (AmvNftStaker.sol#475)
AmvNftStaker.unstake(uint256[]) (AmvNftStaker.sol#411-447) uses a dangerous strict equality:
        - require(bool,string)(staked.owner == msg.sender,sender is not the owner of the token) (AmvNftStaker.sol#433-436)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in AmvNftStaker.stake(uint256[]) (AmvNftStaker.sol#337-378):
        External calls:
        - nft.transferFrom(msg.sender,address(this),tokenId) (AmvNftStaker.sol#371)
        State variables written after the call(s):
        - addNewNftToVault(tokenId) (AmvNftStaker.sol#373)
                - vault[tokenId] = Stake(msg.sender,uint24(tokenId),uint48(block.timestamp)) (AmvNftStaker.sol#386-390)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
AmvNftStaker.stake(uint256[]) (AmvNftStaker.sol#337-378) has external calls inside a loop: nftOwnerAddress = nft.ownerOf(token
 (AmvNftStaker.sol#352)
AmvNftStaker.stake(uint256[]) (AmvNftStaker.sol#337-378) has external calls inside a loop: nft.transferFrom(msg.sender,address
is),tokenId) (AmvNftStaker.sol#371)
AmvNftStaker.unstake(uint256[]) (AmvNftStaker.sol#411-447) has external calls inside a loop: nft.transferFrom(address(this),ms
ender,tokenId) (AmvNftStaker.sol#444)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
Reentrancy in AmvNftStaker.stake(uint256[]) (AmvNftStaker.sol#337-378):
        External calls:
        - nft.transferFrom(msg.sender,address(this),tokenId) (AmvNftStaker.sol#371)
        State variables written after the call(s):
        - addNewTokenIdToList(tokenId) (AmvNftStaker.sol#374)
                - nftTokenIds.push(tokenId) (AmvNftStaker.sol#402)
        - addNewTokenIdToList(tokenId) (AmvNftStaker.sol#374)
                - tokenIdExist[tokenId] = true (AmvNftStaker.sol#401)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
```

```
INFO:Detectors:
AmvNftStaker.stake(uint256[]) (AmvNftStaker.sol#337-378) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(vault[tokenId].tokenId == 0,token is already staked) (AmvNftStaker.sol#365)
AmvNftStaker.unstake(uint256[]) (AmvNftStaker.sol#411-447) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(staked.owner == msg.sender,sender is not the owner of the token) (AmvNftStaker.sol#433-436)
AmvNftStaker.tokensOfOwner(address) (AmvNftStaker.sol#464-485) uses timestamp for comparisons
        Dangerous comparisons:
        - staked.owner == account (AmvNftStaker.sol#475)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Context._msgData() (AmvNftStaker.sol#19-21) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.0 (AmvNftStaker.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
AmvNftStaker (AmvNftStaker.sol#285-506) should inherit from IERC721Receiver (AmvNftStaker.sol#100-116)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-inheritance
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (AmvNftStaker.sol#73-75)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (AmvNftStaker.sol#81-87)
tokensOfOwner(address) should be declared external:
        - AmvNftStaker.tokensOfOwner(address) (AmvNftStaker.sol#464-485)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:AmvNftStaker.sol analyzed (6 contracts with 75 detectors), 18 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

AmvNftStaker.sol

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in AmvNftStaker.unstake(uint256[]): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 411:4:

Block timestamp:

Use of "block.timestamp": "block.timestamp" can be influenced by miners to a certain degree. That means that a miner can "choose" the block.timestamp, to a certain degree, to change the outcome of a transaction in the mined block.

more

Pos: 377:45:

Gas costs:

Gas requirement of function AmvNftStaker.onERC721Received is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 496:4:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

more

Pos: 350:8:

Similar variable names:

AmvNftStaker.tokensOfOwner(address) : Variables have very similar names "ownerTokenIds" and "ownrTokenIds". Note: Modifiers are currently not considered by this static analysis.

Pos: 484:15:

No return:

IERC721Receiver.onERC721Received(address,address,uint256,bytes): Defines a return type but never explicitly returns a value.

Pos: 110:4:

Delete from dynamic array:

Using "delete" on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the "length" property.

more

Pos: 454:8:

Email: audit@EtherAuthority.io

Solhint Linter

AmvNftStaker.sol

```
contracts/2_Owner.sol:2:1: Error: Compiler version ^0.8.0 does not
satisfy the r semver requirement
contracts/2_Owner.sol:47:5: Error: Explicitly mark visibility in function
(Set ignoreConstructors to true if using solidity >=0.7.0)
contracts/2_Owner.sol:328:5: Error: Explicitly mark visibility in
function (Set ignoreConstructors to true if using solidity >=0.7.0)
contracts/2_Owner.sol:377:46: Error: Avoid to make time-based decisions
in your business logic
contracts/2_Owner.sol:389:31: Error: Avoid to make time-based decisions
in your business logic
contracts/2_Owner.sol:446:48: Error: Avoid to make time-based decisions
in your business logic
```

Software analysis result:

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

Email: audit@EtherAuthority.io