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: | June 4th, 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 NFT 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 June 4th, 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.
- The AM Chronicles enumerates  vision to be a decentralized shonen jump, focused on community-led initiatives and social interactions that enable the creation of the next generation IPs.
- Its vision is for Anime Metaverse to be a decentralized shonen jump. It is focused on community-led initiatives and social interactions that enable the creation of next generational Intellectual Properties (IPs).

# Audit scope

| **Name** | **Code Review and Security Analysis Report for Anime Metaverse Token Smart Contract** |
|---|---|
| **Platform** | **Ethereum / Solidity** |
| **File** | AmvNftStaker.sol |
| **File MD5 Hash** | BB0C624A4425A22E22AB61ABC257B1BE |
| **Online Code Link** | 0xA6C215215F65e4cBa632D3aeB26384CE69C0ac26 |
| **Audit Date** | June 4th, 2022 |


**Email: audit@EtherAuthority.io**

# Claimed Smart Contract Features

| Claimed Feature Detail | Our Observation |
| --- | --- |
| **Tokenomics:**<br>• Staking NFT Token: AMS<br>• Allowed Maximum Input Size: 100<br>• Maximum Input Size: 10<br>• This is a smart contract which provides staking and unstaking facilities with time-lock only for AnimeMetaverse NftTokens. | **YES, This is valid.** |
| **Ownership Control:**<br>• Owners of Anime Metaverse: Soulmates (AMS) Tokens can call `stake` function to stake their Nft Tokens and `unstake` function to unstake their Nft Tokens.<br>• Owner of the Nft Tokens call `stake` function he provides a list of tokenIds and a time-lock type. | **YES, This is valid.** |

# Audit Summary

According to the standard audit assessment, Customer`s solidity based smart contracts are **“Secured”**. This token contract does contain owner control, which does not make it 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 | Passed |
| 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 Anime Metaverse Token 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 is recommended.

## Documentation

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

As mentioned above, code parts are well commented. so it's easy to understand its programming logic.

Another source of information was its official website https://www.animemetaverse.ai/ which provided rich information about the project architecture.

## 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 | constructor | write | Passed | No Issue |
| 2 | owner | read | Passed | No Issue |
| 3 | onlyOwner | modifier | Passed | No Issue |
| 4 | renounceOwnership | write | access only Owner | No Issue |
| 5 | transferOwnership | write | access only Owner | No Issue |
| 6 | transferOwnership | internal | Passed | No Issue |
| 7 | setMaxInputSize | write | access only Owner | No Issue |
| 8 | setIsOpenForStaking | write | access only Owner | No Issue |
| 9 | setIsTimeLockActive | write | access only Owner | No Issue |
| 10 | stake | external | Passed | No Issue |
| 11 | addNewNftToVault | write | Passed | No Issue |
| 12 | addNewTokenIdToList | write | Passed | No Issue |
| 13 | unstake | external | Passed | No Issue |
| 14 | timeLockCheck | read | Passed | No Issue |
| 15 | removeNftFromVault | write | Passed | No Issue |
| 16 | tokensOfOwner | read | Infinite loops possibility | Refer audit findings |
| 17 | onERC721Received | external | 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. |

# 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 loops possibility:

```
/**
 * @dev Public function to get a list of NFTs which are staked in our smart contract.
 * Checks every stake stored in this `vault` against this `account`
 * If the owner of any stake matches with this `account`, then collects them in a list and are returned.
 * @param account address The address that owns the NFTs.
 * @return ownrTokens A list of tokens owned by `account` from `vault`
 */
function tokensOfOwner(address account)
    public
    view
    returns (Stake[] memory ownrTokens)
{
    uint256 supply = nftTokenIds.length;
    Stake[] memory tmp = new Stake[](supply);

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

As array elements will increase, then it will cost more and more gas. And eventually, it will stop all the functionality. After several hundreds of transactions, all those functions depending on it will stop. We suggest avoiding loops. For example, use mapping to store the array index. And query that data directly, instead of looping through all the elements to find an element.

**Resolution**: Adjust logic to replace loops with mapping or other code structure.

- tokensOfOwner() - nftTokenIds.length, nftCount

## Very Low / Informational / Best practices:

No Informational severity vulnerabilities were found.

# 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:

- setIsTimeLockActive: Owner can set time lock active status.
- setIsOpenForStaking: Owner can set open for staking status.
- setMaxInputSize: Owner can set maximum input size value.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

# Conclusion

We were given a contract code  in the form of an Etherscan weblink. And we have used all possible tests based on given objects as files. We have not observed 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 **"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).

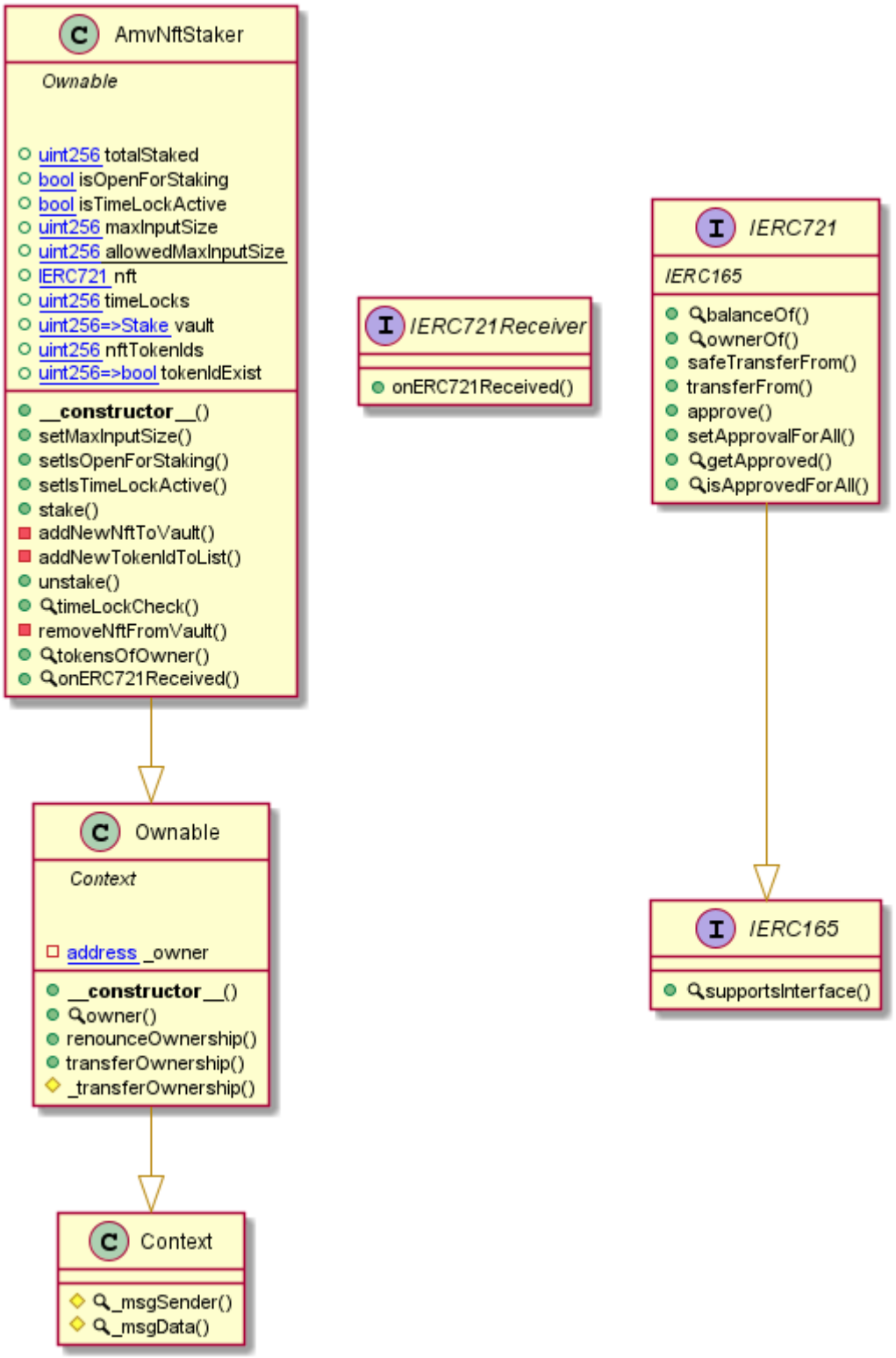
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we 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 Token



# Slither Results Log

## Slither Log >> AmvNftStaker.sol

```
INFO:Detectors:
AmvNftStaker.stake(uint256[],uint8) (AmvNftStaker.sol#407-460) has external calls inside a loop: nft.transferFrom(msg.sender,ad
dress(this),tokenId) (AmvNftStaker.sol#453)
AmvNftStaker.unstake(uint256[]) (AmvNftStaker.sol#494-536) has external calls inside a loop: nft.transferFrom(address(this),msg
.sender,tokenId) (AmvNftStaker.sol#532)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
Reentrancy in AmvNftStaker.stake(uint256[],uint8) (AmvNftStaker.sol#407-460):
        External calls:
        - nft.transferFrom(msg.sender,address(this),tokenId) (AmvNftStaker.sol#453)
        State variables written after the call(s):
        - addNewTokenIdToList(tokenId) (AmvNftStaker.sol#456)
                - nftTokenIds.push(tokenId) (AmvNftStaker.sol#485)
        - addNewTokenIdToList(tokenId) (AmvNftStaker.sol#456)
                - tokenIdExist[tokenId] = true (AmvNftStaker.sol#484)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
AmvNftStaker.stake(uint256[],uint8) (AmvNftStaker.sol#407-460) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(vault[tokenId].tokenId == 0,Token is already staked) (AmvNftStaker.sol#447)
AmvNftStaker.timeLockCheck(uint256,uint256) (AmvNftStaker.sol#543-551) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)((block.timestamp - stakedAt) > timeLock || ! isTimeLockActive,Tokens cannot be unstaked before i
ts chosen minimum time lock period) (AmvNftStaker.sol#547-550)
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
Ownable._transferOwnership(address) (AmvNftStaker.sol#93-97) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version0.8.0 (AmvNftStaker.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
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#299-610) should inherit from IERC721Receiver (AmvNftStaker.sol#100-116)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-inheritance
INFO:Detectors:
Parameter AmvNftStaker.setMaxInputSize(uint256)._maxInputSize (AmvNftStaker.sol#374) is not in mixedCase
Parameter AmvNftStaker.setIsOpenForStaking(bool)._isOpenForStaking (AmvNftStaker.sol#390) is not in mixedCase
Parameter AmvNftStaker.setIsTimeLockActive(bool)._isTimeLockActive (AmvNftStaker.sol#398) is not in mixedCase
Constant AmvNftStaker.allowedMaxInputSize (AmvNftStaker.sol#313) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
AmvNftStaker (AmvNftStaker.sol#299-610) does not implement functions:
        - Context._msgData() (AmvNftStaker.sol#19-21)
        - Context._msgSender() (AmvNftStaker.sol#15-17)
        - Ownable._transferOwnership(address) (AmvNftStaker.sol#93-97)
        - AmvNftStaker.onERC721Received(address,address,uint256,bytes) (AmvNftStaker.sol#600-608)
        - Ownable.owner() (AmvNftStaker.sol#54-56)
        - Ownable.renounceOwnership() (AmvNftStaker.sol#73-75)
        - Ownable.transferOwnership(address) (AmvNftStaker.sol#81-87)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unimplemented-functions
INFO:Detectors:
Ownable._owner (AmvNftStaker.sol#37) is never used in AmvNftStaker (AmvNftStaker.sol#299-610)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variables
```

```
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)
setMaxInputSize(uint256) should be declared external:
        - AmvNftStaker.setMaxInputSize(uint256) (AmvNftStaker.sol#374-384)
setIsOpenForStaking(bool) should be declared external:
        - AmvNftStaker.setIsOpenForStaking(bool) (AmvNftStaker.sol#390-392)
setIsTimeLockActive(bool) should be declared external:
        - AmvNftStaker.setIsTimeLockActive(bool) (AmvNftStaker.sol#398-400)
timeLockCheck(uint256,uint256) should be declared external:
        - AmvNftStaker.timeLockCheck(uint256,uint256) (AmvNftStaker.sol#543-551)
tokensOfOwner(address) should be declared external:
        - AmvNftStaker.tokensOfOwner(address) (AmvNftStaker.sol#568-589)
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), 25 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

# Solidity Static Analysis

**AmvNftStaker.sol**

## Security

### Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in AmvNftStaker.stake(uint256[],uint8): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 407:4:

### 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: 463:45:

## Gas & Economy

### Gas costs:

Gas requirement of function AmvNftStaker.stake 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: 407:4:

### 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: 610:4:



**Email: audit@EtherAuthority.io**

### 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: 435:8:

## Miscellaneous

### Constant/View/Pure functions:

AmvNftStaker.tokensOfOwner(address) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 578:4:

### Similar variable names:

AmvNftStaker.stake(uint256[],uint8) : Variables have very similar names "tokenId" and "tokenIds". Note: Modifiers are currently not considered by this static analysis.
Pos: 427:16:

### Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 616:8:

### 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: 568:8:

# Solhint Linter

**AmvNftStaker.sol**

```
AmvNftStaker.sol:2:1: Error: Compiler version 0.8.7 does not satisfy
the r semver requirement
AmvNftStaker.sol:47:5: Error: Explicitly mark visibility in function
(Set ignoreConstructors to true if using solidity >=0.7.0)
AmvNftStaker.sol:313:29: Error: Constant name must be in capitalized
SNAKE_CASE
AmvNftStaker.sol:366:5: Error: Explicitly mark visibility in function
(Set ignoreConstructors to true if using solidity >=0.7.0)
AmvNftStaker.sol:463:46: Error: Avoid to make time-based decisions in
your business logic
AmvNftStaker.sol:475:31: Error: Avoid to make time-based decisions in
your business logic
AmvNftStaker.sol:545:48: Error: Avoid to make time-based decisions in
your business logic
AmvNftStaker.sol:558:14: 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**