Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	AKIMME VIRTUAL LAND SMART CONTRACT
Website:	https://www.akimme.io
Platform:	Ethereum
Language:	Solidity
Date:	March 3rd, 2023

Table of contents

Introduction 3

Project Background 4

Audit Scope 4

Claimed Smart Contract Features 5

Audit Summary 6

Technical Quick Stats 7

Code Quality 8

Documentation 8

Use of Dependencies 8

AS-IS overview 9

Severity Definitions 10

Audit Findings 11

Conclusion 16

Our Methodology 17

Disclaimers 19

Appendix

- Code Flow Diagram 20
- Slither Results Log 21
- Solidity static analysis 23
- Solhint Linter 26

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by the AKIMME Metaverse Team to perform the Security audit of the AKIMME Virtual Land NFT 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 March 3rd, 2023.

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

- AKIMME is a metaverse on the Ethereum blockchain that aims to create a new and exciting business frontier for progressive thinkers. The metaverse has matured into a prolific technological breakthrough that may power several commercial prospects with a strategic focus.
- AKIMME will provide lucrative investment opportunities in the form of virtual land. Both companies and individual investors will have the chance to purchase this virtual land and profit from it by either reselling, auctioning it, or developing a virtual office space for their own use.
- The smart contract is for a Metaverse Virtual Land (NFT) ERC721 on the Ethereum Blockchain.

Audit scope

Name	**Code Review and Security Analysis Report for AKIMME VIRTUAL LAND Smart Contract**
Platform	**Ethereum / Solidity**
File	AKIMME_Virtual_Land_NFT.sol
File MD5 Hash	1B3ECE6BA9DF61CB2AFC2FE8907718F3
Updated File MD5 Hash	b8982b153f1d820572abc1cc932e03a6
Audit Date	March 3rd, 2023
Revised Audit Date	March 9th, 2023

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Owner Specifications: ● Pause/unpause sale status can be set by the Owner. ● NFTs can be pre-minted by the owner. ● NFT Locking Period Control can be updated by the owner. ● The presale start time, end time can be set by the owner. ● In the admin panel, the developer has provided a lock up period with owner access. ● All funds should be transferred to the COLD WALLET ledger in the Admin ledger. ● Admin can change the price of the plot and input the number of NFTs each wallet is allowed to mint .	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity based smart contracts are **“Secured”**. This Virtual Land NFT smart 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 1 critical, 0 high, 0 medium and 0 low and some very low level issues.

All the issues have been resolved / acknowledged in the revised contract code.

Email: audit@EtherAuthority.io

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	Passed
	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

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 AKIMME VIRTUAL LAND NFT 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 AKIMME Virtual Land NFT smart contract.

The AKIMME Metaverse team has not provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way. However, all the tests have been performed in a manual way.

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

Documentation

We were given a AKIMME Virtual Land Smart Contract code in the form of a file. The hash of that code is mentioned above in the table.

As mentioned above, code parts are **well** commented and the logic is straightforward. 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.

Another source of information was its official website: https://www.akimme.io 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 not used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	tokenURI	read	Passed	No Issue
3	_setTokenURI	internal	Passed	No Issue
4	_burn	internal	Passed	No Issue
5	onlyOwner	modifier	Passed	No Issue
6	ispausecheck	modifier	Passed	No Issue
7	Checklocking	modifier	Passed	No Issue
8	CheckPreSale	modifier	Passed	No Issue
9	isUserOneByOne	read	Passed	No Issue
10	isUserTwoByTwo	read	Passed	No Issue
11	isUserThreeByThree	read	Passed	No Issue
12	PauseSale	write	access only Owner	No Issue
13	UnPauseSale	write	access only Owner	No Issue
14	Premint	write	access only Owner	No Issue
15	setApprovalForAll	write	Passed	No Issue
16	approve	write	Passed	No Issue
17	UpdatePrice	write	access only Owner	No Issue
18	UpdatePlotQtyMax	write	access only Owner	No Issue
19	updateLockinPeriod	write	access only Owner	No Issue
20	setPreSaleStartTime	write	access only Owner	No Issue
21	isWhitelisted	read	Passed	No Issue
22	updateRoot	write	Passed	No Issue
23	transferFrom	write	Passed	No Issue
24	Buy	external	Passed	No Issue
25	BuyPublically	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.

Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

(1) Logical vulnerability:

In the buy function, msg.value does not get validated with the price of a given plot type. The user can buy it for a very small amount and the contract transfers the actual plot type price to the deposit address.

Resolution: The msg.value needs to be validated for the respective plot type, as well as the minimum price for buy.

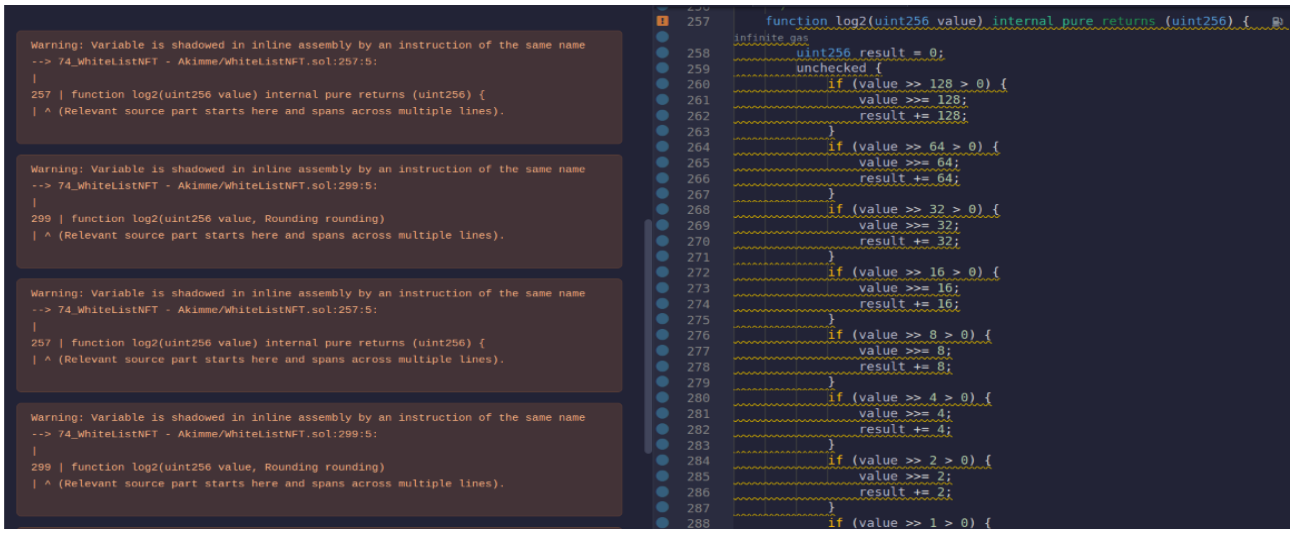
Status: This issue is fixed in revised contract code.

High Severity

No High severity vulnerabilities were found.

Medium

(1) Compilation error:



Resolution: We suggest correcting the code.

Status: This issue is acknowledged in revised contract code.

Low

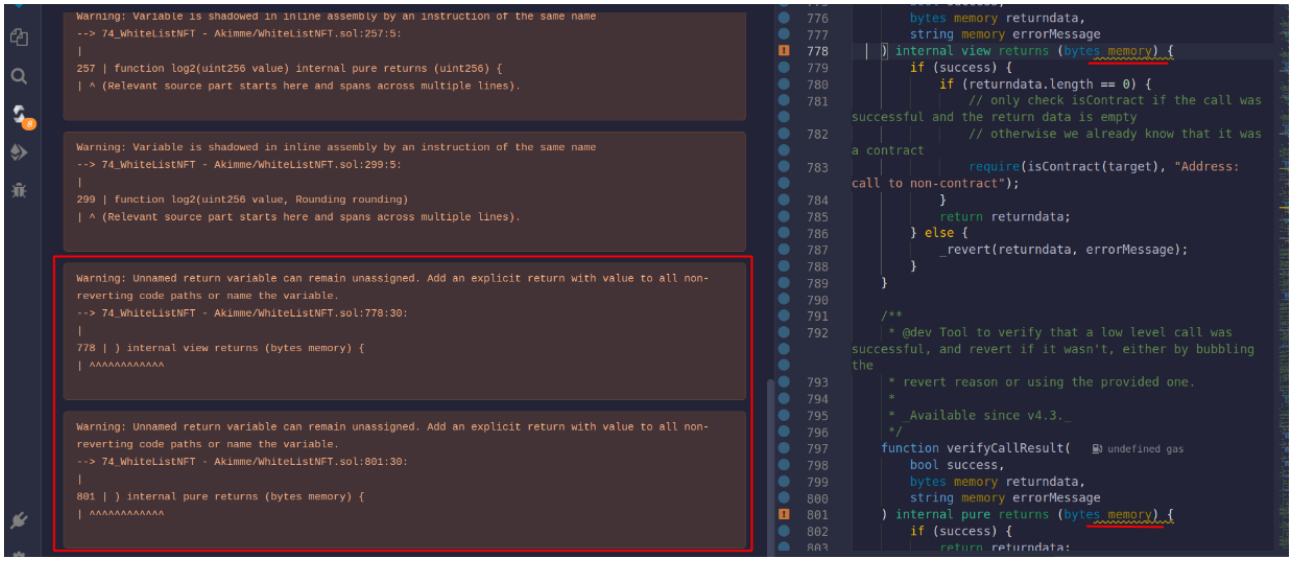
No low severity vulnerabilities were found.

Very Low / Informational / Best practices:

(1) Compile warning:

library Math:

Warning: Variable is shadowed in inline assembly by an instruction of the same name.



Warning: Unnamed return variable can remain unassigned. Add an explicit return with value to all non-reverting code paths or name the variable.

Resolution: Pass return data variable name to avoid this warning.

Status: This issue is acknowledged in revised contract code.

(2) Spelling mistake:

Spelling mistake in function comments.

1 - **“tranfer”** word should be **“transfer”**

```
// Update Lockin Period Control for tranfer Token
function updateLockinPeriod(uint256 _LockingPeriodForSale)
    public
    ispausecheck
    onlyOwner
{
    LockingPeriodForSale = _LockingPeriodForSale;
    emit LockinuUpdate(LockingPeriodForSale, "Locking Period updated");
}
```

2 - **“Validiation”** word should be **“Validation”**

```
// Transfer Token With Timelock Validiation
function transferFrom(
    address from,
    address to,
    uint256 tokenId
) public virtual override ispausecheck Checklocking {
    require(
        _isApprovedOrOwner(_msgSender(), tokenId),
        "ERC721: cal[DEPOSITE_ADDRESS]ler is not token owner or approved"
    );
```

3,4 - **“Fucntion”** word should be **“Function”**.

```
// Buy And Mint Fucntion
function Buy(
    address to,
    string[] memory _tokenURI,
    uint256 _plottype,
    bytes32 _root,
    bytes32[] memory proof
) external payable ispausecheck CheckPreSale {
    require(to != address(0), "to is the zero address");
```

```
// Buy And Mint After Pre-Sale Fucntion
function BuyPublically(
    address to,
    string[] memory _tokenURI,
    uint256 _plottype
) external payable ispausecheck {
    require(to != address(0), "to is the zero address");
```

Resolution: Correct the spelling.

Status: This issue is fixed in revised contract code.

(3) Unused event:

There are some events that are defined but not used in code.

Events are:

- UpdateMerkel
- Validplot

Resolution: We suggest removing unused events.

Status: This issue is fixed in revised contract code.

(4) Missing require error messages:

```
function mulDiv(
    uint256 x,
    uint256 y,
    uint256 denominator
) internal pure returns (uint256 result) {
    unchecked {
        // 512-bit multiply [prod1 prod0] = x * y. Compute the product mod 2^256 and mod 2^256 - 1, then use
        // use the Chinese Remainder Theorem to reconstruct the 512 bit result. The result is stored in two 256
        // variables such that product = prod1 * 2^256 + prod0.
        uint256 prod0; // Least significant 256 bits of the product
        uint256 prod1; // Most significant 256 bits of the product
        assembly {
            let mm := mulmod(x, y, not(0))
            prod0 := mul(x, y)
            prod1 := sub(sub(mm, prod0), lt(mm, prod0))
        }

        // Handle non-overflow cases, 256 by 256 division.
        if (prod1 == 0) {
            return prod0 / denominator;
        }

        // Make sure the result is less than 2^256. Also prevents denominator == 0.
        require(denominator > prod1);
```

There is one place in the Math library where require has been used for validation, but the error message has not been mentioned.

Resolution: We suggest adding an error message.It is helpful to get failure of the transaction.

Status: This issue is fixed in revised contract code.

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:

- PauseSale: Owner can set a pause sale status true.
- UnPauseSale: Owner can set a pause sale status false.
- Premint: Owner can pre mint tokens.
- UpdatePrice: Owner can update the price of the land.
- UpdatePlotQtyMax: Owner can update the qty of lands.
- updateLockinPeriod: Owner can update lockin period control for transfer NFT.
- setPreSaleStartTime: Owner can set the presale start time period.

The purpose of having these functions is firstly, this is a NFT contract. The Locking Period is there to prevent the floor price from dropping and protecting investors and the project as a whole. The functions such as the price and number of NFTs a wallet is able to mint should be decided by the owner.

Conclusion

We were given a contract code in the form of a file and we have used all possible tests based on given objects as files. We have observed 1 critical issue and some informational severity issues in the Virtual Land NFT smart contract. These issues have been resolved in the revised code. So, **it's good to go for the mainnet deployment**.

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.

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

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

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).

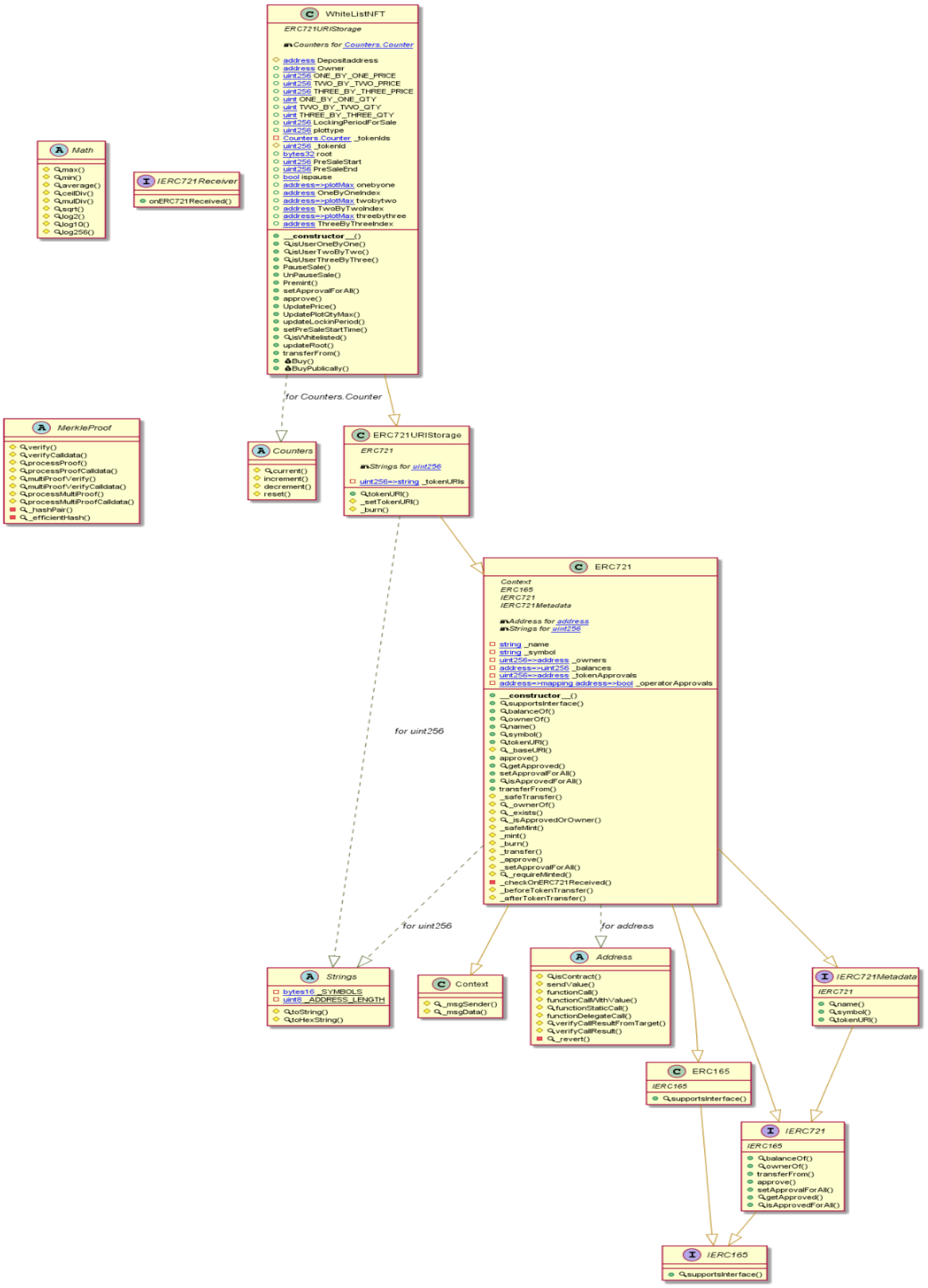
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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



Email: audit@EtherAuthority.io

Slither Results Log

Slither Log >> AKIMME_Virtual_Land_NFT.sol

```
ERC721._checkOnERC721Received(address,address,uint256,bytes) (WhiteListNFT.sol#1567-1598) has external calls inside a loop: IER
C721Receiver(to).onERC721Received(_msgSender(),from,tokenId,data) (WhiteListNFT.sol#1574-1594)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).retval (WhiteListNFT.sol#1581)' in ERC721._checkOnERC721
Received(address,address,uint256,bytes) (WhiteListNFT.sol#1567-1598) potentially used before declaration: retval == IERC721Rece
iver.onERC721Received.selector (WhiteListNFT.sol#1582)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (WhiteListNFT.sol#1583)' in ERC721._checkOnERC721
Received(address,address,uint256,bytes) (WhiteListNFT.sol#1567-1598) potentially used before declaration: reason.length == 0 (W
hiteListNFT.sol#1584)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (WhiteListNFT.sol#1583)' in ERC721._checkOnERC721
Received(address,address,uint256,bytes) (WhiteListNFT.sol#1567-1598) potentially used before declaration: revert(uint256,uint25
6)(32 + reason,mload(uint256)(reason)) (WhiteListNFT.sol#1591)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

Reentrancy in WhiteListNFT.Buy(address,string[],uint256,bytes32,bytes32[]) (WhiteListNFT.sol#2300-2384):
        External calls:
        - (sent1) = Depositaddress.call{value: ONE_BY_ONE_PRICE}() (WhiteListNFT.sol#2320)
        - (sent2) = Depositaddress.call{value: TWO_BY_TWO_PRICE}() (WhiteListNFT.sol#2332)
        - (sent3) = Depositaddress.call{value: THREE_BY_THREE_PRICE}() (WhiteListNFT.sol#2344-2346)
        - _safeMint(to,_tokenId) (WhiteListNFT.sol#2353)
                - IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,data) (WhiteListNFT.sol#1574-1594)
        External calls sending eth:
        - (sent1) = Depositaddress.call{value: ONE_BY_ONE_PRICE}() (WhiteListNFT.sol#2320)
        - (sent2) = Depositaddress.call{value: TWO_BY_TWO_PRICE}() (WhiteListNFT.sol#2332)
        - (sent3) = Depositaddress.call{value: THREE_BY_THREE_PRICE}() (WhiteListNFT.sol#2344-2346)
        State variables written after the call(s):
        - _setTokenURI(_tokenId,_tokenURI[i]) (WhiteListNFT.sol#2354)
                - _tokenURIs[tokenId] = _tokenURI (WhiteListNFT.sol#1709)
```

```
Reentrancy in WhiteListNFT.Premint(address,string) (WhiteListNFT.sol#2143-2155):
        External calls:
        - _safeMint(_userAddress,newItemId) (WhiteListNFT.sol#2150)
                - IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,data) (WhiteListNFT.sol#1574-1594)
        Event emitted after the call(s):
        - PremintComplete(_userAddress,newItemId) (WhiteListNFT.sol#2154)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

WhiteListNFT.setPreSaleStartTime(uint256,uint256) (WhiteListNFT.sol#2247-2268) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(_PreSaleStart > block.timestamp,Presale Start time should bhe greator then current time) (WhiteL
istNFT.sol#2255-2258)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Math.mulDiv(uint256,uint256,uint256) (WhiteListNFT.sol#101-181) uses assembly
        - INLINE ASM (WhiteListNFT.sol#112-116)
        - INLINE ASM (WhiteListNFT.sol#132-139)
        - INLINE ASM (WhiteListNFT.sol#146-155)
Strings.toString(uint256) (WhiteListNFT.sol#433-453) uses assembly
        - INLINE ASM (WhiteListNFT.sol#439-441)
        - INLINE ASM (WhiteListNFT.sol#445-447)
Address._revert(bytes,string) (WhiteListNFT.sol#809-824) uses assembly
        - INLINE ASM (WhiteListNFT.sol#817-820)
ERC721._checkOnERC721Received(address,address,uint256,bytes) (WhiteListNFT.sol#1567-1598) uses assembly
        - INLINE ASM (WhiteListNFT.sol#1590-1592)
MerkleProof._efficientHash(bytes32,bytes32) (WhiteListNFT.sol#1963-1974) uses assembly
        - INLINE ASM (WhiteListNFT.sol#1969-1973)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

WhiteListNFT.ispausecheck() (WhiteListNFT.sol#2046-2049) compares to a boolean constant:
        -require(bool,string)(ispause == false,Sale is Paused !) (WhiteListNFT.sol#2047)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
```

```
WhiteListNFT.Buy(address,string[],uint256,bytes32,bytes32[]) (WhiteListNFT.sol#2300-2384) has costly operations inside a loop:
        - _tokenId = _tokenIds.current() (WhiteListNFT.sol#2351)
WhiteListNFT.BuyPublically(address,string[],uint256) (WhiteListNFT.sol#2387-2449) has costly operations inside a loop:
        - _tokenId = _tokenIds.current() (WhiteListNFT.sol#2416)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
```

```
MerkleProof.processProofCalldata(bytes32[],bytes32) (WhiteListNFT.sol#1799-1809) is never used and should be removed
MerkleProof.verifyCalldata(bytes32[],bytes32,bytes32) (WhiteListNFT.sol#1766-1772) is never used and should be removed
Strings.toHexString(address) (WhiteListNFT.sol#486-488) is never used and should be removed
Strings.toHexString(uint256) (WhiteListNFT.sol#458-462) is never used and should be removed
Strings.toHexString(uint256,uint256) (WhiteListNFT.sol#467-481) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```

```
Pragma version^0.8.0 (WhiteListNFT.sol#5) allows old versions
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity

Low level call in Address.sendValue(address,uint256) (WhiteListNFT.sol#577-588):
        - (success) = recipient.call{value: amount}() (WhiteListNFT.sol#583)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (WhiteListNFT.sol#666-686):
        - (success,returndata) = target.call{value: value}(data) (WhiteListNFT.sol#676-678)
Low level call in Address.functionStaticCall(address,bytes,string) (WhiteListNFT.sol#713-726):
        - (success,returndata) = target.staticcall(data) (WhiteListNFT.sol#718)
Low level call in Address.functionDelegateCall(address,bytes,string) (WhiteListNFT.sol#752-765):
        - (success,returndata) = target.delegatecall(data) (WhiteListNFT.sol#757)
Low level call in WhiteListNFT.Buy(address,string[],uint256,bytes32,bytes32[]) (WhiteListNFT.sol#2300-2384):
        - (sent1) = Depositaddress.call{value: ONE_BY_ONE_PRICE}() (WhiteListNFT.sol#2320)
        - (sent2) = Depositaddress.call{value: TWO_BY_TWO_PRICE}() (WhiteListNFT.sol#2332)
        - (sent3) = Depositaddress.call{value: THREE_BY_THREE_PRICE}() (WhiteListNFT.sol#2344-2346)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
```

```
Struct WhiteListNFT.plotMax (WhiteListNFT.sol#2007-2010) is not in CapWords
Function WhiteListNFT.PauseSale() (WhiteListNFT.sol#2135-2137) is not in mixedCase
Function WhiteListNFT.UnPauseSale() (WhiteListNFT.sol#2139-2141) is not in mixedCase
Function WhiteListNFT.Premint(address,string) (WhiteListNFT.sol#2143-2155) is not in mixedCase
Parameter WhiteListNFT.Premint(address,string)._userAddress (WhiteListNFT.sol#2143) is not in mixedCase
Parameter WhiteListNFT.Premint(address,string).Token_uri (WhiteListNFT.sol#2143) is not in mixedCase
Function WhiteListNFT.UpdatePrice(uint256,uint256,uint256) (WhiteListNFT.sol#2184-2211) is not in mixedCase
Parameter WhiteListNFT.UpdatePrice(uint256,uint256,uint256)._ONE_BY_ONE_PRICE (WhiteListNFT.sol#2185) is not in mixedCase
Parameter WhiteListNFT.UpdatePrice(uint256,uint256,uint256)._TWO_BY_TWO_PRICE (WhiteListNFT.sol#2186) is not in mixedCase
Parameter WhiteListNFT.UpdatePrice(uint256,uint256,uint256)._THREE_BY_THREE_PRICE (WhiteListNFT.sol#2187) is not in mixedCase
Function WhiteListNFT.UpdatePlotQtyMax(uint256,uint256,uint256) (WhiteListNFT.sol#2214-2235) is not in mixedCase
Parameter WhiteListNFT.UpdatePlotQtyMax(uint256,uint256,uint256)._ONE_BY_ONE_QTY (WhiteListNFT.sol#2215) is not in mixedCase
Parameter WhiteListNFT.UpdatePlotQtyMax(uint256,uint256,uint256)._TWO_BY_TWO_QTY (WhiteListNFT.sol#2216) is not in mixedCase
Parameter WhiteListNFT.UpdatePlotQtyMax(uint256,uint256,uint256)._THREE_BY_THREE_QTY (WhiteListNFT.sol#2217) is not in mixedCas
e
Parameter WhiteListNFT.updateLockinPeriod(uint256)._LockingPeriodForSale (WhiteListNFT.sol#2238) is not in mixedCase
Parameter WhiteListNFT.setPreSaleStartTime(uint256,uint256)._PreSaleStart (WhiteListNFT.sol#2247) is not in mixedCase
Parameter WhiteListNFT.setPreSaleStartTime(uint256,uint256)._PreSaleEnd (WhiteListNFT.sol#2247) is not in mixedCase
Parameter WhiteListNFT.updateRoot(bytes32)._root (WhiteListNFT.sol#2280) is not in mixedCase
Function WhiteListNFT.Buy(address,string[],uint256,bytes32,bytes32[]) (WhiteListNFT.sol#2300-2384) is not in mixedCase
Parameter WhiteListNFT.Buy(address,string[],uint256,bytes32,bytes32[])._tokenURI (WhiteListNFT.sol#2302) is not in mixedCase
Parameter WhiteListNFT.Buy(address,string[],uint256,bytes32,bytes32[])._plottype (WhiteListNFT.sol#2303) is not in mixedCase
Parameter WhiteListNFT.Buy(address,string[],uint256,bytes32,bytes32[])._root (WhiteListNFT.sol#2304) is not in mixedCase
Function WhiteListNFT.BuyPublically(address,string[],uint256) (WhiteListNFT.sol#2387-2449) is not in mixedCase
Parameter WhiteListNFT.BuyPublically(address,string[],uint256)._tokenURI (WhiteListNFT.sol#2389) is not in mixedCase
Parameter WhiteListNFT.BuyPublically(address,string[],uint256)._plottype (WhiteListNFT.sol#2390) is not in mixedCase
Variable WhiteListNFT.Depositaddress (WhiteListNFT.sol#1984) is not in mixedCase
Variable WhiteListNFT.Owner (WhiteListNFT.sol#1985) is not in mixedCase
Variable WhiteListNFT.ONE_BY_ONE_PRICE (WhiteListNFT.sol#1986) is not in mixedCase
```

```
Variable WhiteListNFT.TWO_BY_TWO_PRICE (WhiteListNFT.sol#1987) is not in mixedCase
Variable WhiteListNFT.THREE_BY_THREE_PRICE (WhiteListNFT.sol#1988) is not in mixedCase
Variable WhiteListNFT.ONE_BY_ONE_QTY (WhiteListNFT.sol#1989) is not in mixedCase
Variable WhiteListNFT.TWO_BY_TWO_QTY (WhiteListNFT.sol#1990) is not in mixedCase
Variable WhiteListNFT.THREE_BY_THREE_QTY (WhiteListNFT.sol#1991) is not in mixedCase
Variable WhiteListNFT.LockingPeriodForSale (WhiteListNFT.sol#1992) is not in mixedCase
Variable WhiteListNFT._tokenId (WhiteListNFT.sol#1997) is not in mixedCase
Variable WhiteListNFT.PreSaleStart (WhiteListNFT.sol#2002) is not in mixedCase
Variable WhiteListNFT.PreSaleEnd (WhiteListNFT.sol#2003) is not in mixedCase
Variable WhiteListNFT.OneByOneIndex (WhiteListNFT.sol#2013) is not in mixedCase
Variable WhiteListNFT.TwoByTwoIndex (WhiteListNFT.sol#2016) is not in mixedCase
Variable WhiteListNFT.ThreeByThreeIndex (WhiteListNFT.sol#2019) is not in mixedCase
Modifier WhiteListNFT.Checklocking() (WhiteListNFT.sol#2051-2058) is not in mixedCase
Modifier WhiteListNFT.CheckPreSale() (WhiteListNFT.sol#2060-2064) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

Reentrancy in WhiteListNFT.BuyPublically(address,string[],uint256) (WhiteListNFT.sol#2387-2449):
        External calls:
        - Depositaddress.transfer(ONE_BY_ONE_PRICE) (WhiteListNFT.sol#2400)
        - Depositaddress.transfer(TWO_BY_TWO_PRICE) (WhiteListNFT.sol#2406)
        - Depositaddress.transfer(THREE_BY_THREE_PRICE) (WhiteListNFT.sol#2412)
        State variables written after the call(s):
        - OneByOneIndex.push(msg.sender) (WhiteListNFT.sol#2426)
        - ThreeByThreeIndex.push(msg.sender) (WhiteListNFT.sol#2442)
        - TwoByTwoIndex.push(msg.sender) (WhiteListNFT.sol#2434)
        - _safeMint(to,_tokenId) (WhiteListNFT.sol#2418)
                - _balances[from] -= batchSize (WhiteListNFT.sol#1622)
                - _balances[to] += 1 (WhiteListNFT.sol#1432)
                - _balances[to] += batchSize (WhiteListNFT.sol#1625)
        - _safeMint(to,_tokenId) (WhiteListNFT.sol#2418)
                - _owners[tokenId] = to (WhiteListNFT.sol#1435)
        - _tokenId = _tokenIds.current() (WhiteListNFT.sol#2416)
        - _setTokenURI(_tokenId,_tokenURI[i]) (WhiteListNFT.sol#2419)
                - _tokenURIs[tokenId] = _tokenURI (WhiteListNFT.sol#1709)
        - onebyone[msg.sender].count += 1 (WhiteListNFT.sol#2423)
        - onebyone[msg.sender].count = 1 (WhiteListNFT.sol#2425)
        - onebyone[msg.sender].index = OneByOneIndex.length - 1 (WhiteListNFT.sol#2427)
```

```
        - onebyone[msg.sender].index = OneByOneIndex.length - 1 (WhiteListNFT.sol#2427)
        - threebythree[msg.sender].count += 1 (WhiteListNFT.sol#2439)
        - threebythree[msg.sender].count = 1 (WhiteListNFT.sol#2441)
        - threebythree[msg.sender].index = ThreeByThreeIndex.length - 1 (WhiteListNFT.sol#2443)
        - twobytwo[msg.sender].count += 1 (WhiteListNFT.sol#2431)
        - twobytwo[msg.sender].count = 1 (WhiteListNFT.sol#2433)
        - twobytwo[msg.sender].index = TwoByTwoIndex.length - 1 (WhiteListNFT.sol#2435)
        Event emitted after the call(s):
        - Buying(to,_tokenId) (WhiteListNFT.sol#2448)
        - Transfer(address(0),to,tokenId) (WhiteListNFT.sol#1437)
                - _safeMint(to,_tokenId) (WhiteListNFT.sol#2418)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-4

WhiteListNFT.Depositaddress (WhiteListNFT.sol#1984) should be constant
WhiteListNFT.plottype (WhiteListNFT.sol#1993) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

WhiteListNFT.Owner (WhiteListNFT.sol#1985) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
WhiteListNFT.sol analyzed (14 contracts with 84 detectors), 124 result(s) found
```

Solidity Static Analysis

AKIMME_Virtual_Land_NFT.sol

Security

Check-effects-interaction:

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

more

Pos: 2387:8:

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: 2256:32:

Low level calls:

Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.

more

Pos: 2320:33:

Gas & Economy

Gas costs:

Gas requirement of function WhiteListNFT.BuyPublically 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: 2387:8:

Email: audit@EtherAuthority.io

Delete dynamic array:

The "delete" operation when applied to a dynamically sized array in Solidity generates code to delete each of the elements contained. If the array is large, this operation can surpass the block gas limit and raise an OOG exception. Also nested dynamically sized objects can produce the same results.

more

Pos: 1721:16:

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: 2415:12:

Miscellaneous

Constant/View/Pure functions:

WhiteListNFT.isWhitelisted(bytes32[],bytes32) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 2271:8:

Similar variable names:

WhiteListNFT.Buy(address,string[],uint256,bytes32,bytes32[]) : Variables have very similar names "root" and "proof". Note: Modifiers are currently not considered by this static analysis.

Pos: 2329:34:

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: 2396:16:

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: 1721:16:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.
Pos: 233:35:

Solhint Linter

AKIMME_Virtual_Land_NFT.sol

```
AKIMME_Virtual_Land_NFT.sol:28:18: Error: Parse error: missing ';' at
'{'
AKIMME_Virtual_Land_NFT.sol:36:18: Error: Parse error: missing ';' at
'{'
AKIMME_Virtual_Land_NFT.sol:106:18: Error: Parse error: missing ';'
at '{'
AKIMME_Virtual_Land_NFT.sol:225:18: Error: Parse error: missing ';'
at '{'
AKIMME_Virtual_Land_NFT.sol:245:18: Error: Parse error: missing ';'
at '{'
AKIMME_Virtual_Land_NFT.sol:259:18: Error: Parse error: missing ';'
at '{'
AKIMME_Virtual_Land_NFT.sol:304:18: Error: Parse error: missing ';'
at '{'
AKIMME_Virtual_Land_NFT.sol:318:18: Error: Parse error: missing ';'
at '{'
AKIMME_Virtual_Land_NFT.sol:359:18: Error: Parse error: missing ';'
at '{'
AKIMME_Virtual_Land_NFT.sol:375:18: Error: Parse error: missing ';'
at '{'
AKIMME_Virtual_Land_NFT.sol:408:18: Error: Parse error: missing ';'
at '{'
AKIMME_Virtual_Land_NFT.sol:434:18: Error: Parse error: missing ';'
at '{'
AKIMME_Virtual_Land_NFT.sol:459:18: Error: Parse error: missing ';'
at '{'
AKIMME_Virtual_Land_NFT.sol:1427:18: Error: Parse error: missing ';'
at '{'
AKIMME_Virtual_Land_NFT.sol:1464:18: Error: Parse error: missing ';'
at '{'
AKIMME_Virtual_Land_NFT.sol:1509:18: Error: Parse error: missing ';'
at '{'
```

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