www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Customer:	NFTMeme
Website:	https://catecoin.club
Platform:	Binance Smart Chain
Language:	Solidity
Date:	July 17th, 2021

Table of contents

Introduction 4
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 13
Audit Findings 13
Conclusion 21
Our Methodology 22
Disclaimers 24
Appendix
- Code Flow Diagram 25
- Slither Report Log 26

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 PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by the CateCoin team to perform the Security audit of the NFTMeme 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 July 17th, 2021.

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

NFTMeme Platform where meme creators will submit memes and get paid by users in CateCoin, Whole system will be decentralised.

Audit scope

Name	Code Review and Security Analysis Report for NFTMeme token Smart Contract
Platform	**BSC / Solidity**
File	NFTMeme.sol
Smart Contract Online Code	https://bscscan.com/address/0xc30b6f7ADc735FcD7B38BaC2B6F05b50482481eB#code
File MD5 Hash	88AA8B0B7697A17D156876E1961A9E1A
Audit Date	July 17th, 2021
Updated Smart Contract Online Code	https://bscscan.com/address/0x7C331FFD3EB1FC89a7562258597225cC5cC48f7E#code
Updated File MD5 Hash	124F3774022259646386989 4E591FC5E
Revised Audit Date	July 19th, 2021
Updated Smart Contract Online Code	https://bscscan.com/address/0x2F9FbB154e6C3810f8B2D786cB863F8893E43354#code
Updated File MD5 Hash	FA25B7323424A89A5C65F9AB4266BFE1
Revised Audit Date	July 22nd, 2021

Email: audit@EtherAuthority.io

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Name: NFTMeme	**YES, This is valid.**
Penalty: 20%	**YES, This is valid. The smart contract owner can change this anytime.**
Minimum Stake Time: 31 days	**YES, This is valid. The smart contract owner can change this anytime.**
Minimum Likes To Buy Meme: 500	**YES, This is valid. The smart contract owner can change this anytime.** Removed
APY:10%	**YES, This is valid. The smart contract owner can change this anytime.**
• The Owner can access functions like changeEnabled, changeMinimumtokensForPost, changeMinimumtokensForLike, changeMinimumtokensForComment , changeMinimumLikesToBuyMeme (Removed), etc.	**YES, This is valid.The smart contract owner controls these functions, so the owner must handle the private key of the owner's wallet very securely. Because if the private key is compromised, then it will create problems.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contracts are **"Secured"**. These contracts also have owner functions (described in the centralization section below), which does not make everything 100% decentralized. Thus, the owner must execute those smart contract functions as per the business plan.

We used various tools like MythX, Slither 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 3 low and some very low level issues.
These issues can be fixed/acknowledged in the revised smart contract code.

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	Not Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Resolved
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Resolved
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	Passed
	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	Resolved
	Other code specification issues	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Resolved
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Resolved
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. This smart contract also contains Libraries, Smart contracts inherits and Interfaces. These are compact and well written contracts.

The libraries in the NFTMeme 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 NFTMeme token.

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

Some code parts are **not well** commented on smart contracts.

Documentation

We were given NFTMeme token smart contracts code in the form of a BscScan web link. The hashes of that code are mentioned above in the table.

As mentioned above, some code parts are **not well** commented. So it is difficult 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://catecoin.club/ which provided rich information about the project architecture and tokenomics.

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. And their core code blocks are written well.

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

AS-IS overview

NFTMeme.sol

(1) Interface

(a) IERC165

(2) Inherited contracts

(a) IRC20

(b) ERC20

(c) IERC721

(d) IERC721Receiver

(e) ERC165

(f) ERC721

(g) IERC721Enumerable

(h) ERC721Enumerable

(i) ERC721Full

(j) ReentrancyGuard

(k) ERC721NFTMarket

(l) StakeContract

(3) Struct

(a) Meme

(b) Reward

(c) Stake

(4) Usages

(a) using SafeMath for uint256;

(5) Events

(a) event BuyMeme(uint256 indexed post_id, address indexed from, address indexed to, uint256 value);

(b) event Claim(address indexed user, bool like, bool comment, bool post, uint256[] posts);

(c) event Like(address indexed user, uint256 indexed post_id);

(d) event Comment(address indexed user, uint256 indexed post_id);

(e) event CreateMeme(address indexed user, uint256 indexed post_id, uint256 nft_id);

(6) Functions

Sl.	Functions	Type	Observation	Conclusion
1	user_pending_posts_likes_length	read	Passed	No Issue
2	user_pending_posts_comment_length	read	Passed	No Issue
3	owner_pending_posts_likes_length	read	Passed	No Issue
4	owner_pending_posts_comments_length	read	Passed	No Issue
5	owner_pending_posts_length	read	Passed	No Issue
6	setRewards	write	Critical operation lacks event log	Refer Audit Findings
7	createMeme	write	Passed	No Issue
8	changeIsNFT	write	Critical operation lacks event log	Refer Audit Findings
9	changeEnabled	write	Critical operation lacks event log	Refer Audit Findings
10	changeMinimumtokensForPost	write	Critical operation lacks event log	Refer Audit Findings
11	changeMinimumtokensForLike	write	Functions to set values for unused variables	Refer Audit Findings
12	changeMinimumtokensForComment	write	Functions to set values for unused variables	Refer Audit Findings
13	changeMinimumLikesToBuyMeme	write	Critical operation lacks event log	Removed

14	like	write	Infinite loops possibility	Removed
15	comment	write	Infinite loops possibility	Removed
16	ClaimMyRewards	write	Require statements should be preferred instead of IF Statement	Refer Audit Findings
17	buyMeme	write	Passed	No Issue
18	stakesOfOwnerLength	read	Use meaningful words rather than alphabets	Refer Audit Findings
19	modifyLimit	external	access only Owner	No Issue
20	modifyMinimum	external	access only Owner	No Issue
21	modifyAnnualInterestRatePercentage	external	access only Owner	No Issue
22	modifyMinimumStakeTime	external	access only Owner	No Issue
23	modifyPenalty	external	access only Owner	No Issue
24	queryOwnersAccounts	external	access only Owner	No Issue
25	calculateInterest	read	Passed	No Issue
26	checkAvailableLimit	read	Passed	No Issue
27	hasActiveStakes	read	Passed	No Issue
28	createStake	external	Critical operation lacks event log	Refer Audit Findings
29	withdrawStake	external	Critical operation lacks event log	Refer Audit Findings
30	queryCollectedPenalty	external	access only Owner	Refer Audit Findings
31	withdrawPenalty	external	Critical operation lacks event log	Refer Audit Findings
32	setToken	write	access only Owner	No Issue
33	canSell	read	Passed	No Issue
34	sell	write	Critical operation lacks event log	Refer Audit Findings
35	canBuy	read	Passed	No Issue
36	buy	write	Passed	No Issue
37	callOptionalReturn	write	Passed	No Issue

38	updateAdmin	write	Critical operation lacks event log	Refer Audit Findings
39	onlyOwner	modifier	Passed	No Issue
40	nonReentrant	modifier	Passed	No Issue
41	exists	read	Passed	No Issue
42	tokensOfOwner	read	Passed	No Issue
43	_autoMint	internal	Passed	No Issue
44	transfer	write	Passed	No Issue
45	tokenOfOwnerByIndex	read	Passed	No Issue
46	totalSupply	read	Passed	No Issue
47	tokenByIndex	read	Passed	No Issue
48	_transferFrom	internal	Passed	No Issue
49	_mint	internal	Passed	No Issue
50	_tokensOfOwner	internal	Passed	No Issue
51	_addTokenToOwnerEnumeration	write	Passed	No Issue
52	_addTokenToAllTokensEnumeration	write	Passed	No Issue
53	_removeTokenFromOwnerEnumeration	write	Passed	No Issue
54	totalSupply	read	Passed	No Issue
55	tokenOfOwnerByIndex	read	Passed	No Issue
56	tokenByIndex	read	Passed	No Issue
57	balanceOf	read	Passed	No Issue
58	ownerOf	read	Passed	No Issue
59	approve	write	Passed	No Issue
60	getApproved	read	Passed	No Issue
61	transferFrom	write	Passed	No Issue
62	safeTransferFrom	write	Passed	No Issue
63	_exists	read	Passed	No Issue
64	_isApprovedOrOwner	internal	Passed	No Issue
65	_checkOnERC721Received	internal	Passed	No Issue
66	_clearApproval	write	Passed	No Issue
67	supportsInterface	external	Passed	No Issue
68	_registerInterface	internal	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

No Critical severity vulnerabilities were found.

High

No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Infinite loops possibility at multiple places:

```
// add the unassigned reward to pending for owners and users
// add amount to pending rewards for owners and users
for(uint256 i=0; i<user_unassigned_posts_likes[_id].length; i++) {
    // users
    user_pending_posts_likes[user_unassigned_posts_likes[_id][i]].push(_id);
    total_pending_rewards[user_unassigned_posts_likes[_id][i]] += likes.amount_sender;
    // owner
    owner_pending_posts_likes[memes[_id].owner].push(_id);
    total_pending_rewards[memes[_id].owner] += likes.amount_owner;
}
```

There are many functions in the smart contracts, where the Like() function in user_unassigned_posts_likes[_id].length Comment() function in user_unassigned_posts_comment[_id].length Variable is used directly in the loops. It is recommended to put some kind of limits.

.

Resolution: So it does not go wild and create any scenario where it can hit the block gas limit.

Status: Fixed

(2) Require statements should be preferred instead of IF Statement:

```
function ClaimMyRewards() public nonReentrant {
    if (total_pending_rewards[msg.sender] > 0) {
        // transfer all the rewards to user
        token_to_pay.transferFrom(contract_owner, msg.sender, total_pending_rewards[msg.sender]);

        // reset total_pending_rewards
        total_pending_rewards[msg.sender] = 0;

        // if the user had likes as user emit claim event and reset
        if (user_pending_posts_likes[msg.sender].length>0) {
            emit Claim(msg.sender, true, false, false, user_pending_posts_likes[msg.sender]);
            user_pending_posts_likes[msg.sender].length = 0;
        }

        // if the user had comments as user emit claim event and reset
        if (user_pending_posts_comment[msg.sender].length>0) {
            emit Claim(msg.sender, false, true, false, user_pending_posts_comment[msg.sender]);
            user_pending_posts_comment[msg.sender].length = 0;
```

As per the current design of the adjustInterest function, it is a strict requirement that total_pending_rewards of the user must be more than 0, in order to execute this function successfully.

.

Resolution: Therefore, it is considered a better practice to use require statements for such strict validations in a function.

Status: Fixed.

(3) State variables written after the call:

```
function ClaimMyRewards() public nonReentrant {

    if (total_pending_rewards[msg.sender] > 0) {

        // transfer all the rewards to user
        token_to_pay.transferFrom(contract_owner, msg.sender, total_pending_rewards[msg.sender]);

        // reset total_pending_rewards
        total_pending_rewards[msg.sender] = 0;

        // if the user had likes as user emit claim event and reset
        if (user_pending_posts_likes[msg.sender].length>0) {
            emit Claim(msg.sender, true, false, false, user_pending_posts_likes[msg.sender]);
            user_pending_posts_likes[msg.sender].length = 0;
        }
```

ClaimMyRewards function has transfer and then sets pending rewards to 0. Same issue in the withdrawPenalty function.

Resolution: Set pending rewards to 0 before transfer call.

Status: Fixed.

Very Low / Discussion / Best practices:

(1) Use the latest solidity version:

```
pragma solidity ^0.5.7;
```

Using the latest solidity will prevent any compiler level bugs.

Resolution: Please use 0.8.6 which is the latest version.

Status: Acknowledged.

(2) Critical operation lacks event log:

```
// change if the Meme can be sold as NFT or no
function changeIsNFT(uint256 _id, bool _is_nft) public {

    // validations, only contract owner
    require(memes[_id].nft_id > 0, "this meme is not registered");
    require(msg.sender == contract_owner || msg.sender == memes[_id].owner, "you do not have permission");

    memes[_id].is_nft = _is_nft;
}
```

```
// set rewards parameters
function setRewards (
    uint256 _rewards_like_owner,        // Earnings of the owner for liking
    uint256 _rewards_like_user,         // Earnings of the user for liking
    uint256 _limit_likes,               // What is the minimum number of likes to get the rewards
    uint256 _rewards_comments_owner,    // Earnings of the owner for commenting
    uint256 _rewards_comments_user,     // Earnings of the user for commenting
    uint256 _rewards_posting            // Earnings of the owner for postings
) public onlyOwner {

    likes.amount_owner = _rewards_like_owner;
    likes.amount_sender = _rewards_like_user;
    likes.limit = _limit_likes;

    comments.amount_owner = _rewards_comments_owner;
    comments.amount_sender = _rewards_comments_user;

    posting.amount_owner = _rewards_posting;
}
```

```
// the admin can ban an NFT for selling or not
function changeEnabled(uint256 _id, bool _enabled) public onlyOwner{
    require(memes[_id].nft_id > 0, "this meme is not registered");
    memes[_id].enabled = _enabled;
}
```

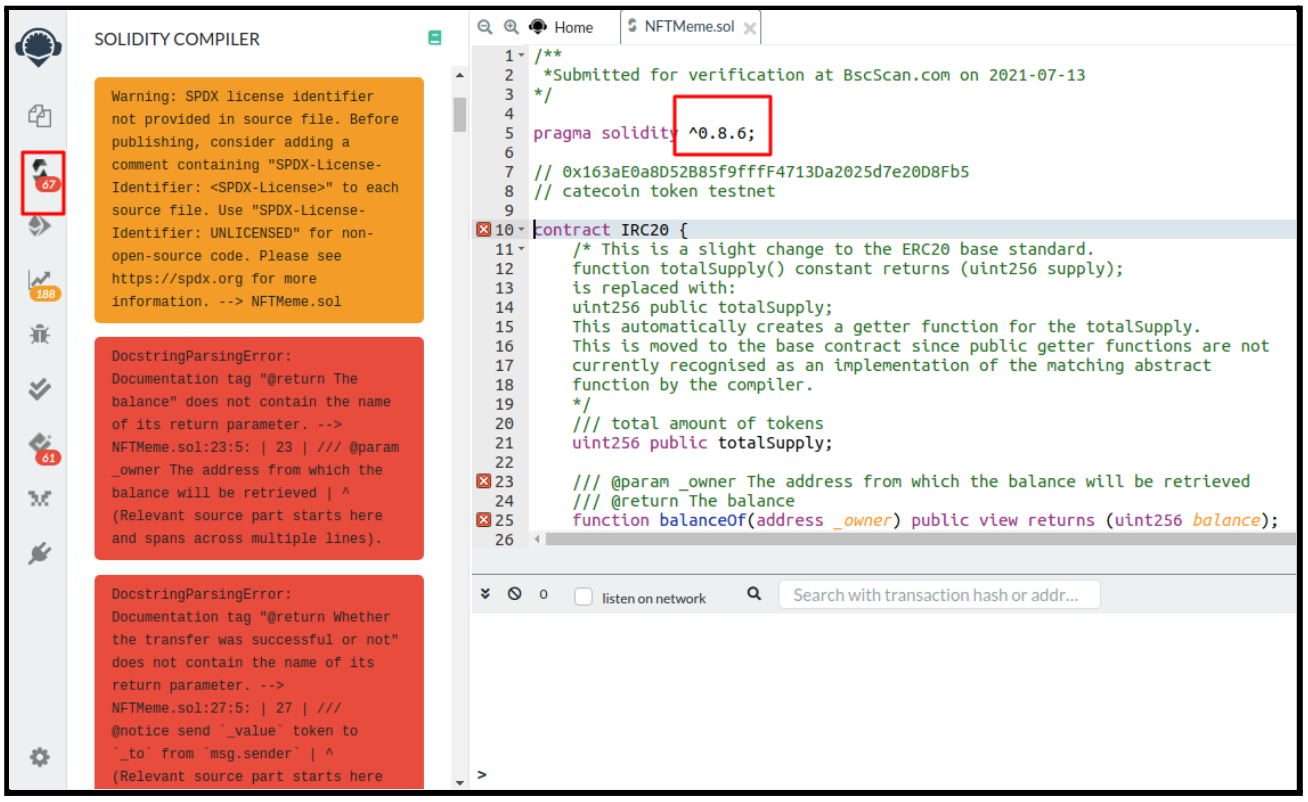
There are several places in the smart contracts, where not added critical functions call event logs.

Resolution: Functions are:

- setRewards()
- changeIsNFT()
- changeEnabled()
- withdrawPenalty()
- withdrawStake()
- createStake()
- updateAdmin()
- sell()

.**Status:** Fixed.

(3) Use keywords/functions to be deprecated:



Compiled with the latest solidity version on remix.

Resolution: Please use 0.8.6 which is the latest version and fix that syntax errors.

Status: Acknowledged.

(4) Unwanted comments:

```
// 0x163aE0a8D52B85f9fffF4713Da2025d7e20D8Fb5
// catecoin token testnet
```

Unwanted comments found in code.

Resolution: We suggest removing unwanted comments like this.

Status: Fixed.

(5) Spelling mistake:

```
// transfer the interes from owner account, it has to have enough funds approved
token_to_pay.transferFrom(contract_owner, msg.sender, interest);
```

Spelling mistake of Interest.

Resolution: Please correct the spelling from interes to interest.

Status: Fixed.

(6) Unused Library:

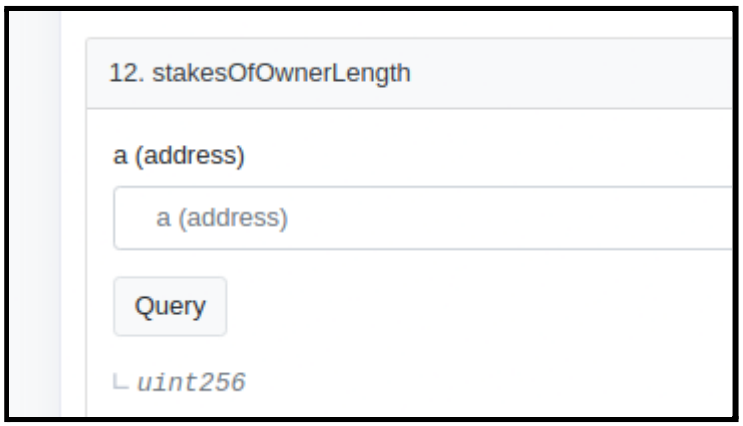
```
/**
 * @title Roles
 * @dev Library for managing addresses assigned to a Role.
 */
library Roles {
    struct Role {
        mapping(address => bool) bearer;
    }
```

Roles library is not used anywhere in code.

Resolution: We suggest removing the unwanted library.

Status: Fixed.

(7) Use meaningful words rather than alphabets for parameters:



```
function stakesOfOwnerLength(address a) public view returns (uint256) {
    return stakesOfOwner[a].length;
}
```

In some functions there is an address parameter .

Resolution: Change the letter a to some meaningful word so in bsc scan users can read it correctly.

Status: Fixed.

(8) Unused Variable:

```
// Mapping from owner to operator approvals
mapping(address => mapping(address => bool)) private _operatorApprovals;
```

_operatorApprovals is not used in the contract.

Resolution: Remove _operatorApprovals if it's not used.

Status: Fixed.

(9) Missing zero address validation:

Detects missing zero address validation. Constructor updateAdmin has no check for an address.

Resolution: Check that the address is not zero. Suggest to check the address in the constructor and updateAdmin.

Status: Fixed in updateAdmin function.

(10) Unused Variable:

```
uint256 public minimumtokensForLike;
uint256 public minimumtokensForComment;
```

Unused Variable

Resolution: Remove unused variables.

Status: Open.

(11) Functions to set values for unused variables:

```
function changeMinimumtokensForLike(uint256 _newVal) public onlyOwner{
    minimumtokensForLike = _newVal;
}
function changeMinimumtokensForComment(uint256 _newVal) public onlyOwner{
    minimumtokensForComment = _newVal;
}
```

Functions to set values for unused variables.

Resolution: Remove these functions.

Status: Open.

Centralization

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

- changeEnabled: The admin can ban an NFT for selling or not.
- changeMinimumtokensForPost: The Owner can change minimum tokens for post and register reward.
- changeMinimumtokensForLike: The owner can change minimum tokens for like and register reward.
- changeMinimumtokensForComment: The owner can change minimum tokens for comment and register reward.
- changeMinimumLikesToBuyMeme: The owner can change minimum likes to buy memes.

Conclusion

We were given a contract code. And we have used all possible tests based on given objects as files. We observed some issues in the smart contracts and those issues are fixed in revised code. So, **it's good to go to 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 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 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).

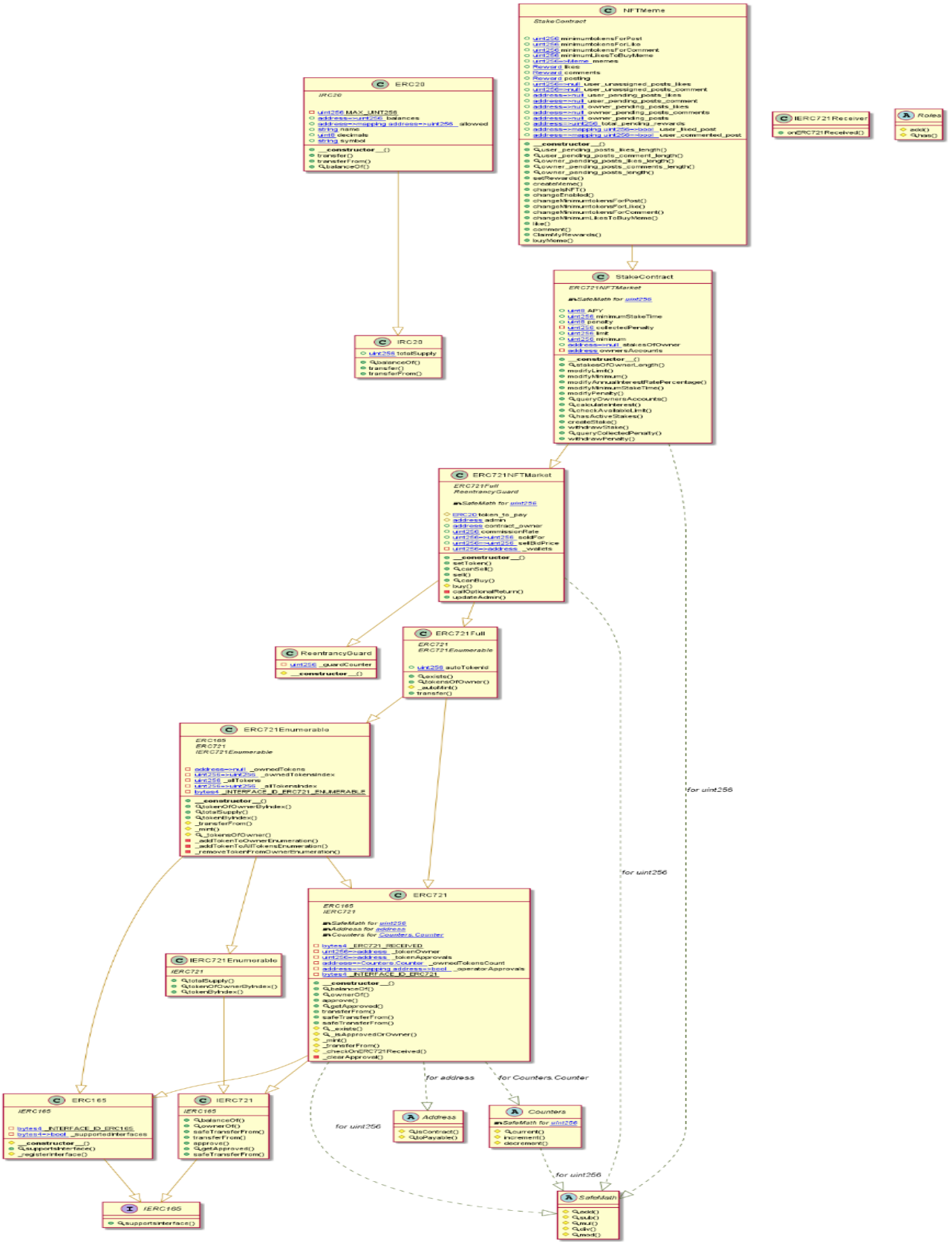
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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



Slither Results Log

Slither log >> NFTMeme.sol

```
INFO:Detectors:
StakeContract.createStake(uint256) (NFTMeme.sol#1308-1326) ignores return value by
token_to_pay.transferFrom(msg.sender,address(this),amount) (NFTMeme.sol#1316)
StakeContract.withdrawStake(uint256) (NFTMeme.sol#1330-1374) ignores return value by
token_to_pay.transfer(msg.sender,amountToWithdraw) (NFTMeme.sol#1346)
StakeContract.withdrawStake(uint256) (NFTMeme.sol#1330-1374) ignores return value by
token_to_pay.transferFrom(contract_owner,msg.sender,interest) (NFTMeme.sol#1363)
StakeContract.withdrawStake(uint256) (NFTMeme.sol#1330-1374) ignores return value by
token_to_pay.transfer(msg.sender,stakesOfOwner[msg.sender][arrayIndex].amount) (NFTMeme.sol#1366)
StakeContract.withdrawPenalty() (NFTMeme.sol#1380-1383) ignores return value by
token_to_pay.transfer(msg.sender,collectedPenalty) (NFTMeme.sol#1381)
NFTMeme.ClaimMyRewards() (NFTMeme.sol#1693-1740) ignores return value by
token_to_pay.transferFrom(contract_owner,msg.sender,total_pending_rewards[msg.sender])
(NFTMeme.sol#1698)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer
INFO:Detectors:
StakeContract.calculateInterest(address,uint256) (NFTMeme.sol#1265-1280) performs a multiplication on
the result of a division:
        -interest_per_year = stakesOfOwner[_ownerAccount][i].amount.mul(APY).div(100)
(NFTMeme.sol#1268)
        -num_seconds.mul(interest_per_year).div(31536000) (NFTMeme.sol#1278)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
Reentrancy in NFTMeme.ClaimMyRewards() (NFTMeme.sol#1693-1740):
        External calls:
        - token_to_pay.transferFrom(contract_owner,msg.sender,total_pending_rewards[msg.sender])
(NFTMeme.sol#1698)
        State variables written after the call(s):
        - total_pending_rewards[msg.sender] = 0 (NFTMeme.sol#1701)
Reentrancy in ERC721NFTMarket.buy(uint256) (NFTMeme.sol#1094-1129):
        External calls:
        - callOptionalReturn(this,abi.encodeWithSelector(this.transferFrom.selector,owner,msg.sender,tokenId))
(NFTMeme.sol#1105)
                - (success,returndata) = address(token).call(data) (NFTMeme.sol#1149)
        - success = token_to_pay.transferFrom(msg.sender,_wallets[tokenId],amount4owner)
(NFTMeme.sol#1112)
        - success2 = token_to_pay.transferFrom(msg.sender,admin,amount4admin) (NFTMeme.sol#1116)
        State variables written after the call(s):
        - _wallets[tokenId] = address(0) (NFTMeme.sol#1123)
        - sellBidPrice[tokenId] = 0 (NFTMeme.sol#1122)
Reentrancy in NFTMeme.buyMeme(uint256) (NFTMeme.sol#1742-1749):
        External calls:
        - buy(memes[post_id].nft_id) (NFTMeme.sol#1746)
                - (success,returndata) = address(token).call(data) (NFTMeme.sol#1149)
                - success = token_to_pay.transferFrom(msg.sender,_wallets[tokenId],amount4owner)
(NFTMeme.sol#1112)
                - success2 = token_to_pay.transferFrom(msg.sender,admin,amount4admin) (NFTMeme.sol#1116)
        State variables written after the call(s):
        - memes[post_id].owner = msg.sender (NFTMeme.sol#1748)
Reentrancy in StakeContract.createStake(uint256) (NFTMeme.sol#1308-1326):
        External calls:
        - token_to_pay.transferFrom(msg.sender,address(this),amount) (NFTMeme.sol#1316)
        State variables written after the call(s):
        - stakesOfOwner[msg.sender].push(Stake(block.timestamp,amount,0,0,0,false)) (NFTMeme.sol#1324)
Reentrancy in StakeContract.withdrawPenalty() (NFTMeme.sol#1380-1383):
        External calls:
```

- token_to_pay.transfer(msg.sender,collectedPenalty) (NFTMeme.sol#1381)
State variables written after the call(s):
- collectedPenalty = 0 (NFTMeme.sol#1382)
Reentrancy in StakeContract.withdrawStake(uint256) (NFTMeme.sol#1330-1374):
External calls:
- token_to_pay.transfer(msg.sender,amountToWithdraw) (NFTMeme.sol#1346)
State variables written after the call(s):
- stakesOfOwner[msg.sender][arrayIndex].penalty = the_penalty (NFTMeme.sol#1352)
- stakesOfOwner[msg.sender][arrayIndex].finishedDate = block.timestamp (NFTMeme.sol#1353)
- stakesOfOwner[msg.sender][arrayIndex].closed = true (NFTMeme.sol#1354)
Reentrancy in StakeContract.withdrawStake(uint256) (NFTMeme.sol#1330-1374):
External calls:
- token_to_pay.transferFrom(contract_owner,msg.sender,interest) (NFTMeme.sol#1363)
- token_to_pay.transfer(msg.sender,stakesOfOwner[msg.sender][arrayIndex].amount)
(NFTMeme.sol#1366)
State variables written after the call(s):
- stakesOfOwner[msg.sender][arrayIndex].interest = interest (NFTMeme.sol#1369)
- stakesOfOwner[msg.sender][arrayIndex].finishedDate = block.timestamp (NFTMeme.sol#1370)
- stakesOfOwner[msg.sender][arrayIndex].closed = true (NFTMeme.sol#1371)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
ERC721NFTMarket.updateAdmin(address,address,uint256) (NFTMeme.sol#1159-1164) should emit an
event for:
- contract_owner = _owner (NFTMeme.sol#1162)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-access-control
INFO:Detectors:
ERC721NFTMarket.constructor(address,address,uint256,ERC20)._admin (NFTMeme.sol#1046) lacks a
zero-check on :
- admin = _admin (NFTMeme.sol#1048)
ERC721NFTMarket.constructor(address,address,uint256,ERC20)._owner (NFTMeme.sol#1046) lacks a
zero-check on :
- contract_owner = _owner (NFTMeme.sol#1049)
ERC721NFTMarket.updateAdmin(address,address,uint256)._admin (NFTMeme.sol#1159) lacks a
zero-check on :
- admin = _admin (NFTMeme.sol#1161)
ERC721NFTMarket.updateAdmin(address,address,uint256)._owner (NFTMeme.sol#1159) lacks a
zero-check on :
- contract_owner = _owner (NFTMeme.sol#1162)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in NFTMeme.ClaimMyRewards() (NFTMeme.sol#1693-1740):
External calls:
- token_to_pay.transferFrom(contract_owner,msg.sender,total_pending_rewards[msg.sender])
(NFTMeme.sol#1698)
State variables written after the call(s):
- delete owner_pending_posts[msg.sender] (NFTMeme.sol#1735)
- delete owner_pending_posts_comments[msg.sender] (NFTMeme.sol#1728)
- delete owner_pending_posts_likes[msg.sender] (NFTMeme.sol#1721)
- delete user_pending_posts_comment[msg.sender] (NFTMeme.sol#1714)
- delete user_pending_posts_likes[msg.sender] (NFTMeme.sol#1707)
Reentrancy in ERC721NFTMarket.buy(uint256) (NFTMeme.sol#1094-1129):
External calls:
- callOptionalReturn(this,abi.encodeWithSelector(this.transferFrom.selector,owner,msg.sender,tokenId))
(NFTMeme.sol#1105)
- (success,returndata) = address(token).call(data) (NFTMeme.sol#1149)
- success = token_to_pay.transferFrom(msg.sender,_wallets[tokenId],amount4owner)
(NFTMeme.sol#1112)
- success2 = token_to_pay.transferFrom(msg.sender,admin,amount4admin) (NFTMeme.sol#1116)
State variables written after the call(s):
- soldFor[tokenId] = sellBidPrice[tokenId] (NFTMeme.sol#1119)
Reentrancy in StakeContract.createStake(uint256) (NFTMeme.sol#1308-1326):
External calls:
- token_to_pay.transferFrom(msg.sender,address(this),amount) (NFTMeme.sol#1316)
State variables written after the call(s):
- ownersAccounts.push(msg.sender) (NFTMeme.sol#1320)

Reentrancy in StakeContract.withdrawStake(uint256) (NFTMeme.sol#1330-1374):
External calls:
- token_to_pay.transfer(msg.sender,amountToWithdraw) (NFTMeme.sol#1346)
State variables written after the call(s):
- collectedPenalty = collectedPenalty.add(the_penalty) (NFTMeme.sol#1349)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in NFTMeme.ClaimMyRewards() (NFTMeme.sol#1693-1740):
External calls:
- token_to_pay.transferFrom(contract_owner,msg.sender,total_pending_rewards[msg.sender]) (NFTMeme.sol#1698)
Event emitted after the call(s):
- Claim(msg.sender,true,false,false,user_pending_posts_likes[msg.sender]) (NFTMeme.sol#1705)
- Claim(msg.sender,false,true,false,user_pending_posts_comment[msg.sender]) (NFTMeme.sol#1712)
- Claim(msg.sender,true,false,false,owner_pending_posts_likes[msg.sender]) (NFTMeme.sol#1719)
- Claim(msg.sender,false,true,false,owner_pending_posts_comments[msg.sender]) (NFTMeme.sol#1726)
- Claim(msg.sender,false,false,true,owner_pending_posts[msg.sender]) (NFTMeme.sol#1733)
Reentrancy in ERC721NFTMarket.buy(uint256) (NFTMeme.sol#1094-1129):
External calls:
- callOptionalReturn(this,abi.encodeWithSelector(this.transferFrom.selector,owner,msg.sender,tokenId)) (NFTMeme.sol#1105)
- (success,returndata) = address(token).call(data) (NFTMeme.sol#1149)
- success = token_to_pay.transferFrom(msg.sender,_wallets[tokenId],amount4owner) (NFTMeme.sol#1112)
- success2 = token_to_pay.transferFrom(msg.sender,admin,amount4admin) (NFTMeme.sol#1116)
Event emitted after the call(s):
- Commission(tokenId,owner,sellBidPrice[tokenId],commissionRate,amount4admin) (NFTMeme.sol#1127)
- Sale(tokenId,owner,msg.sender,sellBidPrice[tokenId]) (NFTMeme.sol#1126)
Reentrancy in NFTMeme.buyMeme(uint256) (NFTMeme.sol#1742-1749):
External calls:
- buy(memes[post_id].nft_id) (NFTMeme.sol#1746)
- (success,returndata) = address(token).call(data) (NFTMeme.sol#1149)
- success = token_to_pay.transferFrom(msg.sender,_wallets[tokenId],amount4owner) (NFTMeme.sol#1112)
- success2 = token_to_pay.transferFrom(msg.sender,admin,amount4admin) (NFTMeme.sol#1116)
Event emitted after the call(s):
- BuyMeme(post_id,memes[post_id].owner,msg.sender,sellBidPrice[memes[post_id].nft_id]) (NFTMeme.sol#1747)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
StakeContract.withdrawStake(uint256) (NFTMeme.sol#1330-1374) uses timestamp for comparisons
Dangerous comparisons:
- block.timestamp.sub(stakesOfOwner[msg.sender][arrayIndex].startDate) < minimumStakeTime (NFTMeme.sol#1337)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Address.isContract(address) (NFTMeme.sol#344-360) uses assembly
- INLINE ASM (NFTMeme.sol#356-358)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
StakeContract.withdrawStake(uint256) (NFTMeme.sol#1330-1374) compares to a boolean constant:
-require(bool,string)(stakesOfOwner[msg.sender][arrayIndex].closed == false,This stake is closed) (NFTMeme.sol#1334)
NFTMeme.like(uint256) (NFTMeme.sol#1548-1622) compares to a boolean constant:
-require(bool,string)(user_liked_post[msg.sender][_id] == false,You already liked this meme) (NFTMeme.sol#1554)
NFTMeme.comment(uint256) (NFTMeme.sol#1625-1691) compares to a boolean constant:
-require(bool,string)(user_commented_post[msg.sender][_id] == false,You already commented this meme) (NFTMeme.sol#1631)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Address.toPayable(address) (NFTMeme.sol#366-372) is never used and should be removed

ERC721Enumerable._addTokenToAllTokensEnumeration(uint256) (NFTMeme.sol#895-898) is never used and should be removed
ERC721Enumerable._addTokenToOwnerEnumeration(address,uint256) (NFTMeme.sol#886-889) is never used and should be removed
ERC721Enumerable._removeTokenFromOwnerEnumeration(address,uint256) (NFTMeme.sol#908-929) is never used and should be removed
Roles.add(Roles.Role,address) (NFTMeme.sol#768-771) is never used and should be removed
Roles.has(Roles.Role,address) (NFTMeme.sol#777-784) is never used and should be removed
SafeMath.mod(uint256,uint256) (NFTMeme.sol#323-326) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Low level call in ERC721NFTMarket.callOptionalReturn(IERC721,bytes) (NFTMeme.sol#1137-1156):
- (success,returndata) = address(token).call(data) (NFTMeme.sol#1149)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter ERC20.transfer(address,uint256)._to (NFTMeme.sol#71) is not in mixedCase
Parameter ERC20.transfer(address,uint256)._value (NFTMeme.sol#71) is not in mixedCase
Parameter ERC20.transferFrom(address,address,uint256)._from (NFTMeme.sol#79) is not in mixedCase
Parameter ERC20.transferFrom(address,address,uint256)._to (NFTMeme.sol#79) is not in mixedCase
Parameter ERC20.transferFrom(address,address,uint256)._value (NFTMeme.sol#79) is not in mixedCase
Parameter ERC20.balanceOf(address)._owner (NFTMeme.sol#91) is not in mixedCase
Parameter ERC721.safeTransferFrom(address,address,uint256,bytes)._data (NFTMeme.sol#630) is not in mixedCase
Parameter ERC721NFTMarket.setToken(ERC20)._token (NFTMeme.sol#1055) is not in mixedCase
Parameter ERC721NFTMarket.updateAdmin(address,address,uint256)._owner (NFTMeme.sol#1159) is not in mixedCase
Parameter ERC721NFTMarket.updateAdmin(address,address,uint256)._admin (NFTMeme.sol#1159) is not in mixedCase
Parameter ERC721NFTMarket.updateAdmin(address,address,uint256)._commissionRate (NFTMeme.sol#1159) is not in mixedCase
Variable ERC721NFTMarket.token_to_pay (NFTMeme.sol#1024) is not in mixedCase
Variable ERC721NFTMarket.contract_owner (NFTMeme.sol#1029) is not in mixedCase
Parameter StakeContract.modifyLimit(uint256)._newVal (NFTMeme.sol#1240) is not in mixedCase
Parameter StakeContract.modifyMinimum(uint256)._newVal (NFTMeme.sol#1244) is not in mixedCase
Parameter StakeContract.modifyAnnualInterestRatePercentage(uint8)._newVal (NFTMeme.sol#1250) is not in mixedCase
Parameter StakeContract.modifyMinimumStakeTime(uint256)._newVal (NFTMeme.sol#1253) is not in mixedCase
Parameter StakeContract.modifyPenalty(uint8)._newVal (NFTMeme.sol#1256) is not in mixedCase
Parameter StakeContract.calculateInterest(address,uint256)._ownerAccount (NFTMeme.sol#1265) is not in mixedCase
Parameter StakeContract.checkAvailableLimit(address)._account (NFTMeme.sol#1282) is not in mixedCase
Parameter StakeContract.hasActiveStakes(address)._account (NFTMeme.sol#1298) is not in mixedCase
Variable StakeContract.APY (NFTMeme.sol#1184) is not in mixedCase
Function NFTMeme.user_pending_posts_likes_length(address) (NFTMeme.sol#1457-1459) is not in mixedCase
Function NFTMeme.user_pending_posts_comment_length(address) (NFTMeme.sol#1461-1463) is not in mixedCase
Function NFTMeme.owner_pending_posts_likes_length(address) (NFTMeme.sol#1465-1467) is not in mixedCase
Function NFTMeme.owner_pending_posts_comments_length(address) (NFTMeme.sol#1469-1471) is not in mixedCase
Function NFTMeme.owner_pending_posts_length(address) (NFTMeme.sol#1473-1475) is not in mixedCase
Parameter NFTMeme.setRewards(uint256,uint256,uint256,uint256,uint256,uint256)._rewards_like_owner (NFTMeme.sol#1479) is not in mixedCase
Parameter NFTMeme.setRewards(uint256,uint256,uint256,uint256,uint256,uint256)._rewards_like_user (NFTMeme.sol#1480) is not in mixedCase
Parameter NFTMeme.setRewards(uint256,uint256,uint256,uint256,uint256,uint256)._limit_likes (NFTMeme.sol#1481) is not in mixedCase
Parameter NFTMeme.setRewards(uint256,uint256,uint256,uint256,uint256,uint256)._rewards_comments_owner (NFTMeme.sol#1482) is not in mixedCase
Parameter NFTMeme.setRewards(uint256,uint256,uint256,uint256,uint256,uint256)._rewards_comments_user (NFTMeme.sol#1483) is not in mixedCase

Parameter NFTMeme.setRewards(uint256,uint256,uint256,uint256,uint256,uint256)._rewards_posting (NFTMeme.sol#1484) is not in mixedCase
Parameter NFTMeme.createMeme(uint256,bool,uint256)._id (NFTMeme.sol#1498) is not in mixedCase
Parameter NFTMeme.createMeme(uint256,bool,uint256)._is_nft (NFTMeme.sol#1498) is not in mixedCase
Parameter NFTMeme.createMeme(uint256,bool,uint256)._sale_price (NFTMeme.sol#1498) is not in mixedCase
Parameter NFTMeme.changeIsNFT(uint256,bool)._id (NFTMeme.sol#1519) is not in mixedCase
Parameter NFTMeme.changeIsNFT(uint256,bool)._is_nft (NFTMeme.sol#1519) is not in mixedCase
Parameter NFTMeme.changeEnabled(uint256,bool)._id (NFTMeme.sol#1529) is not in mixedCase
Parameter NFTMeme.changeEnabled(uint256,bool)._enabled (NFTMeme.sol#1529) is not in mixedCase
Parameter NFTMeme.changeMinimumtokensForPost(uint256)._newVal (NFTMeme.sol#1534) is not in mixedCase
Parameter NFTMeme.changeMinimumtokensForLike(uint256)._newVal (NFTMeme.sol#1537) is not in mixedCase
Parameter NFTMeme.changeMinimumtokensForComment(uint256)._newVal (NFTMeme.sol#1540) is not in mixedCase
Parameter NFTMeme.changeMinimumLikesToBuyMeme(uint256)._newVal (NFTMeme.sol#1543) is not in mixedCase
Parameter NFTMeme.like(uint256)._id (NFTMeme.sol#1548) is not in mixedCase
Parameter NFTMeme.comment(uint256)._id (NFTMeme.sol#1625) is not in mixedCase
Function NFTMeme.ClaimMyRewards() (NFTMeme.sol#1693-1740) is not in mixedCase
Parameter NFTMeme.buyMeme(uint256).post_id (NFTMeme.sol#1742) is not in mixedCase
Variable NFTMeme.user_unassigned_posts_likes (NFTMeme.sol#1422) is not in mixedCase
Variable NFTMeme.user_unassigned_posts_comment (NFTMeme.sol#1423) is not in mixedCase
Variable NFTMeme.user_pending_posts_likes (NFTMeme.sol#1426) is not in mixedCase
Variable NFTMeme.user_pending_posts_comment (NFTMeme.sol#1427) is not in mixedCase
Variable NFTMeme.owner_pending_posts_likes (NFTMeme.sol#1430) is not in mixedCase
Variable NFTMeme.owner_pending_posts_comments (NFTMeme.sol#1431) is not in mixedCase
Variable NFTMeme.owner_pending_posts (NFTMeme.sol#1432) is not in mixedCase
Variable NFTMeme.total_pending_rewards (NFTMeme.sol#1435) is not in mixedCase
Variable NFTMeme.user_liked_post (NFTMeme.sol#1438) is not in mixedCase
Variable NFTMeme.user_commented_post (NFTMeme.sol#1439) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
ERC721._operatorApprovals (NFTMeme.sol#485) is never used in NFTMeme (NFTMeme.sol#1390-1751)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variables
INFO:Detectors:
balanceOf(address) should be declared external:
- ERC20.balanceOf(address) (NFTMeme.sol#91-93)
- IRC20.balanceOf(address) (NFTMeme.sol#27)
transfer(address,uint256) should be declared external:
- ERC20.transfer(address,uint256) (NFTMeme.sol#71-77)
- IRC20.transfer(address,uint256) (NFTMeme.sol#33)
transferFrom(address,address,uint256) should be declared external:
- ERC20.transferFrom(address,address,uint256) (NFTMeme.sol#79-89)
- IRC20.transferFrom(address,address,uint256) (NFTMeme.sol#40)
safeTransferFrom(address,address,uint256) should be declared external:
- ERC721.safeTransferFrom(address,address,uint256) (NFTMeme.sol#606-612)
- IERC721.safeTransferFrom(address,address,uint256) (NFTMeme.sol#160-164)
onERC721Received(address,address,uint256,bytes) should be declared external:
- IERC721Receiver.onERC721Received(address,address,uint256,bytes) (NFTMeme.sol#215-220)
tokenOfOwnerByIndex(address,uint256) should be declared external:
- ERC721Enumerable.tokenOfOwnerByIndex(address,uint256) (NFTMeme.sol#838-841)
- IERC721Enumerable.tokenOfOwnerByIndex(address,uint256) (NFTMeme.sol#793)
tokenByIndex(uint256) should be declared external:
- ERC721Enumerable.tokenByIndex(uint256) (NFTMeme.sol#857-860)
- IERC721Enumerable.tokenByIndex(uint256) (NFTMeme.sol#795)
exists(uint256) should be declared external:
- ERC721Full.exists(uint256) (NFTMeme.sol#942-944)
tokensOfOwner(address) should be declared external:
- ERC721Full.tokensOfOwner(address) (NFTMeme.sol#946-948)
transfer(address,uint256) should be declared external:
- ERC721Full.transfer(address,uint256) (NFTMeme.sol#969-974)
setToken(ERC20) should be declared external:

- ERC721NFTMarket.setToken(ERC20) (NFTMeme.sol#1055-1057)
canSell(uint256) should be declared external:
- ERC721NFTMarket.canSell(uint256) (NFTMeme.sol#1059-1061)
canBuy(uint256) should be declared external:
- ERC721NFTMarket.canBuy(uint256) (NFTMeme.sol#1085-1091)
updateAdmin(address,address,uint256) should be declared external:
- ERC721NFTMarket.updateAdmin(address,address,uint256) (NFTMeme.sol#1159-1164)
stakesOfOwnerLength(address) should be declared external:
- StakeContract.stakesOfOwnerLength(address) (NFTMeme.sol#1236-1238)
user_pending_posts_likes_length(address) should be declared external:
- NFTMeme.user_pending_posts_likes_length(address) (NFTMeme.sol#1457-1459)
user_pending_posts_comment_length(address) should be declared external:
- NFTMeme.user_pending_posts_comment_length(address) (NFTMeme.sol#1461-1463)
owner_pending_posts_likes_length(address) should be declared external:
- NFTMeme.owner_pending_posts_likes_length(address) (NFTMeme.sol#1465-1467)
owner_pending_posts_comments_length(address) should be declared external:
- NFTMeme.owner_pending_posts_comments_length(address) (NFTMeme.sol#1469-1471)
owner_pending_posts_length(address) should be declared external:
- NFTMeme.owner_pending_posts_length(address) (NFTMeme.sol#1473-1475)
setRewards(uint256,uint256,uint256,uint256,uint256,uint256) should be declared external:
- NFTMeme.setRewards(uint256,uint256,uint256,uint256,uint256,uint256) (NFTMeme.sol#1478-1495)
createMeme(uint256,bool,uint256) should be declared external:
- NFTMeme.createMeme(uint256,bool,uint256) (NFTMeme.sol#1498-1516)
changeIsNFT(uint256,bool) should be declared external:
- NFTMeme.changeIsNFT(uint256,bool) (NFTMeme.sol#1519-1526)
changeEnabled(uint256,bool) should be declared external:
- NFTMeme.changeEnabled(uint256,bool) (NFTMeme.sol#1529-1532)
changeMinimumtokensForPost(uint256) should be declared external:
- NFTMeme.changeMinimumtokensForPost(uint256) (NFTMeme.sol#1534-1536)
changeMinimumtokensForLike(uint256) should be declared external:
- NFTMeme.changeMinimumtokensForLike(uint256) (NFTMeme.sol#1537-1539)
changeMinimumtokensForComment(uint256) should be declared external:
- NFTMeme.changeMinimumtokensForComment(uint256) (NFTMeme.sol#1540-1542)
changeMinimumLikesToBuyMeme(uint256) should be declared external:
- NFTMeme.changeMinimumLikesToBuyMeme(uint256) (NFTMeme.sol#1543-1545)
like(uint256) should be declared external:
- NFTMeme.like(uint256) (NFTMeme.sol#1548-1622)
comment(uint256) should be declared external:
- NFTMeme.comment(uint256) (NFTMeme.sol#1625-1691)
ClaimMyRewards() should be declared external:
- NFTMeme.ClaimMyRewards() (NFTMeme.sol#1693-1740)
buyMeme(uint256) should be declared external:
- NFTMeme.buyMeme(uint256) (NFTMeme.sol#1742-1749)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:NFTMeme.sol analyzed (18 contracts with 75 detectors), 130 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration

Email: audit@EtherAuthority.io