Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Pig Fitness Club
Website:	www.pigfitnessclub.io
Platform:	Ethereum
Language:	Solidity
Date:	May 19th, 2022

Table of contents

`

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 Pig Fitness Club team to perform the Security audit of the Pig Fitness Club smart contract code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on May 19th, 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

- Pig Fitness Club is a collection of 10,000 unique 3D Pig NFTs living on the Ethereum blockchain. We are here to start a revolution, rewriting how fitness works and shaping the future of fitness. We strive to integrate Web3 and the metaverse with the fitness industry in order to bring it to another level.
- The Pig Fitness Club is a NFT token contract which allows minting, airdrop, burn.

Audit scope

Name	**Code Review and Security Analysis Report for Pig Fitness Club Token Smart Contract**
Platform	**Ethereum / Solidity**
File	PigFitnessClub.sol
File MD5 Hash	7378BEE3BC4C29C47EDE643A38939ADC
Updated File MD5 Hash	393BC5FA31D99D0D7B416D74357CFEE9
Audit Date	May 19th, 2022
Revise Audit Date	May 21st, 2022

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Pig Fitness Club • Symbol: PFC • WhiteList Cost: 0.18 ether • Maximum Supply: 10000 • Cost: 0.25 ether • Maximum WL Mint Amount Per Wallet: 1 • Maximum Mint Amount Per Wallet: 10 • Maximum Airdrop Mint Amount Per Wallet: 1	**YES, This is valid.**
Ownership Control: • Owner can set the cost rate value. • Owner can set a whitelist cost value.	**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 3 low and some very low level issues.

All the issues have been fixed / acknowledged in the revised 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	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**

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 Pig Fitness Club 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 Pig Fitness Club Token.

The Pig Fitness Club 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 **not well** commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a Pig Fitness Club Token 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 **not well** commented. So it is not 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.pigfitnessclub.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 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	tokensOfOwner	external	Passed	No Issue
8	setSecondContract	external	access only Owner	No Issue
9	updateFitnessLevel	write	Passed	No Issue
10	updateStamina	write	Passed	No Issue
11	updateAge	write	Passed	No Issue
12	updateGender	write	Passed	No Issue
13	updateWeight	write	Passed	No Issue
14	updateHeight	write	Passed	No Issue
15	updateBodyFat	write	Passed	No Issue
16	flipProxyState	write	access only Owner	No Issue
17	isApprovedForAll	read	Passed	No Issue
18	_beforeTokenTransfers	internal	Passed	No Issue
19	mint	external	Passed	No Issue
20	Reserve	external	access only Owner	No Issue
21	Airdrop	external	access only Owner	No Issue
22	burn	external	Owner can burn anyone's token, Burn token does not refund the payable amount	Refer Audit Findings
23	setMaxSupply	external	access only Owner	No Issue
24	tokenURI	read	Passed	No Issue
25	setWLPaused	external	access only Owner	No Issue
26	setAirdropPaused	external	access only Owner	No Issue
27	setWLCost	external	access only Owner	No Issue
28	setMaxTxPerAirdropAddress	external	access only Owner	No Issue
29	setMaxTxPerWlAddress	external	access only Owner	No Issue
30	setWhitelistMerkleRoot	external	access only Owner	No Issue
31	getLeafNode	internal	Passed	No Issue
32	_verify	internal	Passed	No Issue
33	whitelistMint	external	Passed	No Issue
34	ClaimAirdrop	external	The payable not required in function	Refer Audit Findings
35	setUriPrefix	external	access only Owner	No Issue
36	setHiddenUri	external	access only Owner	No Issue
37	setPaused	external	access only Owner	No Issue
38	setCost	external	access only Owner	No Issue

39	setRevealed	external	access only Owner	No Issue
40	setMaxMintAmountPerWallet	external	access only Owner	No Issue
41	withdraw	external	access only Owner	No Issue
42	_baseURI	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.

Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Exceeds max supply:

```
function Airdrop(uint16 _mintAmount, address [] memory _receiver) external onlyOwner {
   uint16 totalSupply = uint16(totalSupply());
   require(totalSupply + _mintAmount <= maxSupply, "Excedes max supply.");
   for(uint i =0 ; i< _receiver.length ; i++)

   { _safeMint(_receiver[i] , _mintAmount);}
   delete _mintAmount;
   delete _receiver;
   delete totalSupply;
}
```

The owner can Airdrop token and there is a condition for check exceeding max supply only for the one time.

ClaimAirdrop and whitelistMint functions don’t validate for max supply.

Resolution: We suggest checking the max supply should not exceed from all the receivers' mintamount.

Status: Fixed

(2) Owner can burn anyone's token:

Owner can burn any users' tokens.

Resolution: We suggest changing the code so only token holders can burn their own tokens and not anyone else. Not even a contract creator.

Status: Acknowledged.

(3) Burn token does not refund the payable amount:

When the contract owner burns users' tokens, it does not refund the paid amount to the owner of that token.

Resolution: We suggest changing the code so only token holders can burn their own tokens and not anyone else. Not even a contract creator. and token owners can get a refund when burning their own token.

Status: Acknowledged.

Very Low / Informational / Best practices:

(1) Multiple pragma:

There are multiple pragmas with different compiler versions.

Resolution: We suggest using only one pragma and removing the other.

(2) Make variables constant:

```
bytes32 public airdropMerkleRoot = 0x0cc622db75575674b2bc8dfe29c3ac04df24f1567b769fa31f6d4ef384ea66d2;
```

These variables will be unchanged. So, please make it constant. It will save some gas.

Resolution: Declare those variables as constant. Just put a constant keyword. Deployer has to confirm before deploying the contract to production.

(3) The payable not required in function:

There is a function ClaimAirdrop() and it's payable in smart contract code, but there is no requirement to pay the amount when user ClaimAirdrop. so there is no requirement to make this function as payable.

Resolution: We suggest removing the payable keyword from ClaimAirdrop() function.

(4) Spelling mistake:

Spelling mistake in function name. Function is: ClaimAirdop..

Resolution: Correct the spelling, “Airdop” word should be “Airdrop”.

Status: Fixed

(5) Unused local variable:

```
function whitelistMint(uint8 _mintAmount, bytes32[] calldata merkleProof) external payable {

       bytes32  leafnode = getLeafNode(msg.sender);
        uint8 _txPerAddress = NFTPerWLAddress[msg.sender];
       require(_verify(leafnode , whitelistMerkleRoot,  merkleProof   ),  "Invalid merkle proof");
       require (_txPerAddress + _mintAmount <= maxWLMintAmountPerWallet, "Exceeds max tx per address");


    require(!WLpaused, "Whitelist minting is over!");
    require(msg.value >= whiteListCost * _mintAmount, "Insufficient funds!");

       uint16 totalSupply = uint16(totalSupply());
     _safeMint(msg.sender , _mintAmount);
      NFTPerWLAddress[msg.sender] =_txPerAddress + _mintAmount;
      delete totalSupply;
      delete _mintAmount;
       delete _txPerAddress;

    }
    function ClaimAirdop(uint8 _mintAmount, bytes32[] calldata merkleProof) external payable {

       bytes32  leafnode = getLeafNode(msg.sender);
        uint8 _txPerAddress = NFTPerAirdropAddress[msg.sender];
       require(_verify(leafnode ,  airdropMerkleRoot ,  merkleProof   ),  "Invalid merkle proof");
       require (_txPerAddress + _mintAmount <= maxAirdropMintAmountPerWallet, "Exceeds max tx per address");

    require(!Airdroppaused, "Claiming free airdrops is over!");

       uint16 totalSupply = uint16(totalSupply());
     _safeMint(msg.sender , _mintAmount);
      NFTPerAirdropAddress[msg.sender] =_txPerAddress + _mintAmount;
      delete totalSupply;
      delete _mintAmount;
       delete _txPerAddress;

    }
```

In ClaimAirdrop and whitelistMint functions, totalSupply variable has been set but not used.

Resolution: We suggest removing unused variables.

Status: Fixed

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:

- setSecondContract: Owner can set second contract address.
- flipProxyState: Owner can integrate address with future staking proxy address.
- Reserve: Owner can reserve a mint amount with the address.
- Airdrop: Owner can airdrop mint amount with the receiver address.
- burn: Owner can burn the amount .
- setMaxSupply: Owner can set maximum supply value.
- setWLPaused: Owner can set whitelist paused status.
- setAirdropPaused: Owner can set airdrop paused status.
- setWLCost: Owner can set whitelist cost value.
- setMaxTxPerAirdropAddress: Owner can set airdrop address maximum transaction value.
- setMaxTxPerWlAddress: Owner can set maximum transaction whitelist address.
- setWhitelistMerkleRoot: Owner can set whitelist merkle root value.
- setUriPrefix: Owner can set URI prefix value.
- setHiddenUri: Owner can set hidden URI value.
- setPaused: Owner can set paused status true.
- setCost: Owner can set cost value.
- setRevealed: Owner can set revealed value.
- setMaxMintAmountPerWallet: Owner can set maximum mint amount per wallet value.
- withdraw: Owner can withdraw token balance.

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. 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).

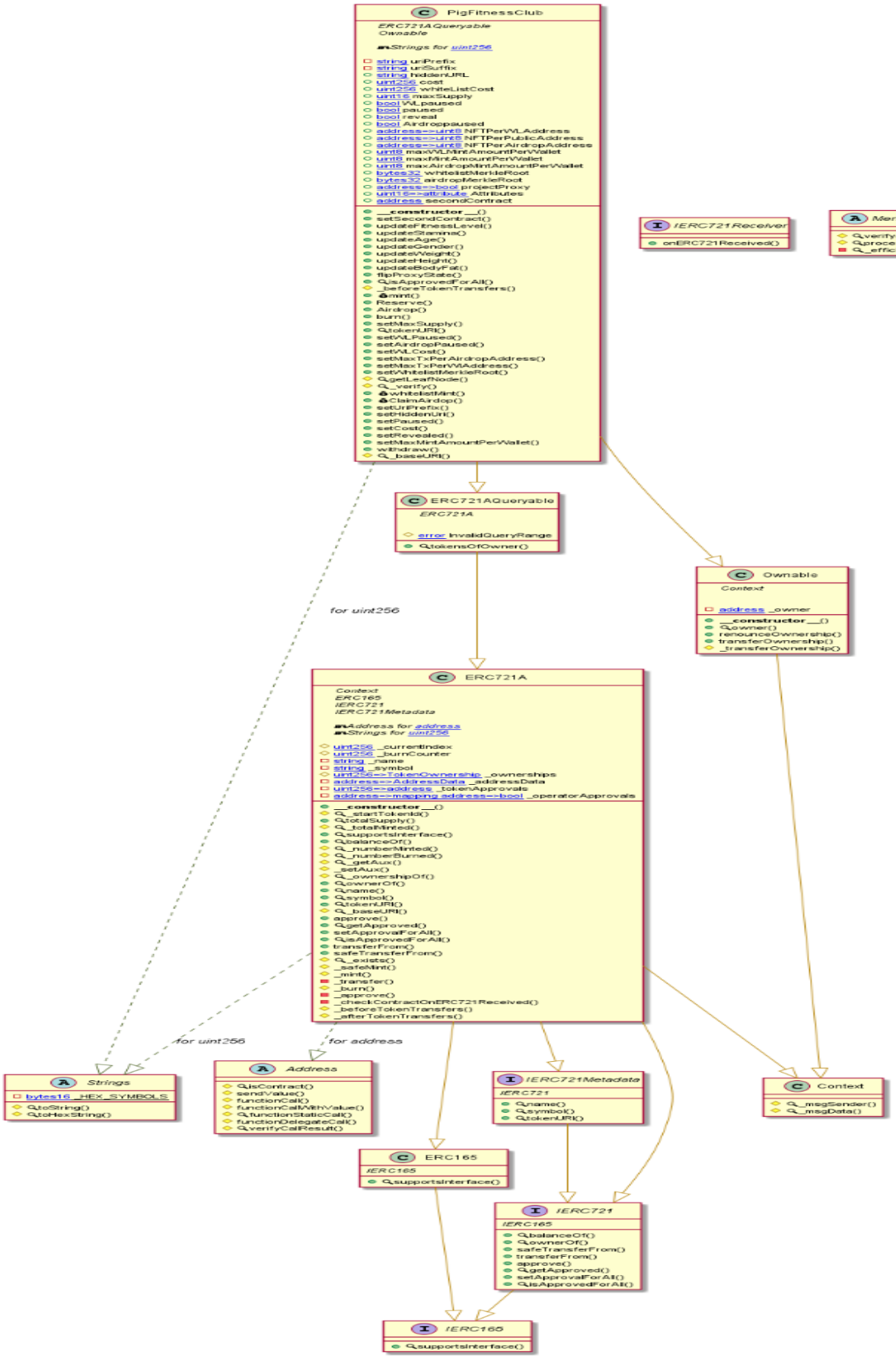
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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



Slither Results Log

Slither Log >> PigFitnessClub.sol

```
INFO:Detectors:
PigFitnessClub.isApprovedForAll(address,address)._owner (PigFitnessClub.sol#769) shadows:
        - Ownable._owner (PigFitnessClub.sol#620) (state variable)
PigFitnessClub.mint(uint8).totalSupply (PigFitnessClub.sol#785) shadows:
        - ERC721A.totalSupply() (PigFitnessClub.sol#263-267) (function)
PigFitnessClub.Reserve(uint16,address).totalSupply (PigFitnessClub.sol#800) shadows:
        - ERC721A.totalSupply() (PigFitnessClub.sol#263-267) (function)
PigFitnessClub.Airdrop(uint16,address[]).totalSupply (PigFitnessClub.sol#808) shadows:
        - ERC721A.totalSupply() (PigFitnessClub.sol#263-267) (function)
PigFitnessClub.whitelistMint(uint8,bytes32[]).totalSupply (PigFitnessClub.sol#902) shadows:
        - ERC721A.totalSupply() (PigFitnessClub.sol#263-267) (function)
PigFitnessClub.ClaimAirdop(uint8,bytes32[]).totalSupply (PigFitnessClub.sol#919) shadows:
        - ERC721A.totalSupply() (PigFitnessClub.sol#263-267) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
PigFitnessClub.setSecondContract(address)._address (PigFitnessClub.sol#733) lacks a zero-check on :
                - secondContract = _address (PigFitnessClub.sol#734)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Variable 'ERC721A._checkContractOnERC721Received(address,address,uint256,bytes).retval (PigFitnessClub.sol#563)' in ERC721A._ch
kContractOnERC721Received(address,address,uint256,bytes) (PigFitnessClub.sol#557-574) potentially used before declaration: retv
 == IERC721Receiver(to).onERC721Received.selector (PigFitnessClub.sol#564)
Variable 'ERC721A._checkContractOnERC721Received(address,address,uint256,bytes).reason (PigFitnessClub.sol#565)' in ERC721A._ch
kContractOnERC721Received(address,address,uint256,bytes) (PigFitnessClub.sol#557-574) potentially used before declaration: reas
.length == 0 (PigFitnessClub.sol#566)
Variable 'ERC721A._checkContractOnERC721Received(address,address,uint256,bytes).reason (PigFitnessClub.sol#565)' in ERC721A._ch
kContractOnERC721Received(address,address,uint256,bytes) (PigFitnessClub.sol#557-574) potentially used before declaration: reve
(uint256,uint256)(32 + reason,mload(uint256)(reason)) (PigFitnessClub.sol#570)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables
INFO:Detectors:
Address.verifyCallResult(bool,bytes,string) (PigFitnessClub.sol#137-155) uses assembly
        - INLINE ASM (PigFitnessClub.sol#147-150)
ERC721A._checkContractOnERC721Received(address,address,uint256,bytes) (PigFitnessClub.sol#557-574) uses assembly
        - INLINE ASM (PigFitnessClub.sol#569-571)
MerkleProof._efficientHash(bytes32,bytes32) (PigFitnessClub.sol#611-617) uses assembly
```

```
MerkleProof._efficientHash(bytes32,bytes32) (PigFitnessClub.sol#611-617) uses assembly
        - INLINE ASM (PigFitnessClub.sol#612-616)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
PigFitnessClub.tokenURI(uint256) (PigFitnessClub.sol#829-852) compares to a boolean constant:
        -reveal == false (PigFitnessClub.sol#842)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Address.functionCall(address,bytes) (PigFitnessClub.sol#74-76) is never used and should be removed
Address.functionCall(address,bytes,string) (PigFitnessClub.sol#78-84) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (PigFitnessClub.sol#86-92) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (PigFitnessClub.sol#94-105) is never used and should be removed
Address.functionDelegateCall(address,bytes) (PigFitnessClub.sol#122-124) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (PigFitnessClub.sol#126-135) is never used and should be removed
Address.functionStaticCall(address,bytes) (PigFitnessClub.sol#107-109) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (PigFitnessClub.sol#111-120) is never used and should be removed
Address.sendValue(address,uint256) (PigFitnessClub.sol#67-72) is never used and should be removed
Address.verifyCallResult(bool,bytes,string) (PigFitnessClub.sol#137-155) is never used and should be removed
Context._msgData() (PigFitnessClub.sol#57-59) is never used and should be removed
ERC721A._baseURI() (PigFitnessClub.sol#345-347) is never used and should be removed
ERC721A._beforeTokenTransfers(uint256) (PigFitnessClub.sol#576-580) is never used and should be removed
ERC721A._getAux(address) (PigFitnessClub.sol#295-297) is never used and should be removed
ERC721A._numberBurned(address) (PigFitnessClub.sol#291-293) is never used and should be removed
ERC721A._numberMinted(address) (PigFitnessClub.sol#287-289) is never used and should be removed
ERC721A._setAux(address,uint64) (PigFitnessClub.sol#299-301) is never used and should be removed
ERC721A._totalMinted() (PigFitnessClub.sol#269-273) is never used and should be removed
Strings.toHexString(uint256) (PigFitnessClub.sol#27-38) is never used and should be removed
Strings.toHexString(uint256,uint256) (PigFitnessClub.sol#40-50) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.0 (PigFitnessClub.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7
solc-0.8.0 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (PigFitnessClub.sol#67-72):
        - (success) = recipient.call{value: amount}() (PigFitnessClub.sol#70)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (PigFitnessClub.sol#94-105):
        - (success,returndata) = target.call{value: value}(data) (PigFitnessClub.sol#103)
Low level call in Address.functionStaticCall(address,bytes,string) (PigFitnessClub.sol#111-120):
        - (success,returndata) = target.staticcall(data) (PigFitnessClub.sol#118)
Low level call in Address.functionDelegateCall(address,bytes,string) (PigFitnessClub.sol#126-135):
        - (success,returndata) = target.delegatecall(data) (PigFitnessClub.sol#133)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter ERC721A.safeTransferFrom(address,address,uint256,bytes)._data (PigFitnessClub.sol#397) is not in mixedCase
```

```
Function PigFitnessClub.ClaimAirdop(uint8,bytes32[]) (PigFitnessClub.sol#910-926) is not in mixedCase
Parameter PigFitnessClub.ClaimAirdop(uint8,bytes32[])._mintAmount (PigFitnessClub.sol#910) is not in mixedCase
Parameter PigFitnessClub.setUriPrefix(string)._uriPrefix (PigFitnessClub.sol#928) is not in mixedCase
Parameter PigFitnessClub.setHiddenUri(string)._uriPrefix (PigFitnessClub.sol#931) is not in mixedCase
Parameter PigFitnessClub.setCost(uint256)._cost (PigFitnessClub.sol#941) is not in mixedCase
Parameter PigFitnessClub.setMaxMintAmountPerWallet(uint8)._maxtx (PigFitnessClub.sol#950) is not in mixedCase
Variable PigFitnessClub.WLpaused (PigFitnessClub.sol#699) is not in mixedCase
Variable PigFitnessClub.Airdroppaused (PigFitnessClub.sol#702) is not in mixedCase
Variable PigFitnessClub.NFTPerWLAddress (PigFitnessClub.sol#704) is not in mixedCase
Variable PigFitnessClub.NFTPerPublicAddress (PigFitnessClub.sol#705) is not in mixedCase
Variable PigFitnessClub.NFTPerAirdropAddress (PigFitnessClub.sol#706) is not in mixedCase
Variable PigFitnessClub.Attributes (PigFitnessClub.sol#726) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
PigFitnessClub.airdropMerkleRoot (PigFitnessClub.sol#713) should be constant
PigFitnessClub.uriSuffix (PigFitnessClub.sol#687) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
name() should be declared external:
        - ERC721A.name() (PigFitnessClub.sol#330-332)
symbol() should be declared external:
        - ERC721A.symbol() (PigFitnessClub.sol#334-336)
tokenURI(uint256) should be declared external:
        - ERC721A.tokenURI(uint256) (PigFitnessClub.sol#338-343)
        - PigFitnessClub.tokenURI(uint256) (PigFitnessClub.sol#829-852)
approve(address,uint256) should be declared external:
        - ERC721A.approve(address,uint256) (PigFitnessClub.sol#349-358)
setApprovalForAll(address,bool) should be declared external:
        - ERC721A.setApprovalForAll(address,bool) (PigFitnessClub.sol#366-371)
```

```
setApprovalForAll(address,bool) should be declared external:
        - ERC721A.setApprovalForAll(address,bool) (PigFitnessClub.sol#366-371)
transferFrom(address,address,uint256) should be declared external:
        - ERC721A.transferFrom(address,address,uint256) (PigFitnessClub.sol#377-383)
safeTransferFrom(address,address,uint256) should be declared external:
        - ERC721A.safeTransferFrom(address,address,uint256) (PigFitnessClub.sol#385-391)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (PigFitnessClub.sol#637-639)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (PigFitnessClub.sol#641-644)
updateFitnessLevel(uint16,uint16) should be declared external:
        - PigFitnessClub.updateFitnessLevel(uint16,uint16) (PigFitnessClub.sol#736-739)
updateStamina(uint16,uint16) should be declared external:
        - PigFitnessClub.updateStamina(uint16,uint16) (PigFitnessClub.sol#740-743)
updateAge(uint16,uint8) should be declared external:
        - PigFitnessClub.updateAge(uint16,uint8) (PigFitnessClub.sol#744-747)
updateGender(uint16,uint8) should be declared external:
        - PigFitnessClub.updateGender(uint16,uint8) (PigFitnessClub.sol#748-751)
updateWeight(uint16,uint16) should be declared external:
        - PigFitnessClub.updateWeight(uint16,uint16) (PigFitnessClub.sol#752-755)
updateHeight(uint16,uint16) should be declared external:
        - PigFitnessClub.updateHeight(uint16,uint16) (PigFitnessClub.sol#756-759)
updateBodyFat(uint16,uint16) should be declared external:
        - PigFitnessClub.updateBodyFat(uint16,uint16) (PigFitnessClub.sol#760-763)
flipProxyState(address) should be declared external:
        - PigFitnessClub.flipProxyState(address) (PigFitnessClub.sol#764-766)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:PigFitnessClub.sol analyzed (13 contracts with 75 detectors), 109 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

PigFitnessClub.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Address.functionCallWithValue(address,bytes,uint256,string): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 192:4:

Inline assembly:

The Contract uses inline assembly, this is only advised in rare cases. Additionally static analysis modules do not parse inline Assembly, this can lead to wrong analysis results.
more
Pos: 1116:8:

Gas & Economy

Gas costs:

Gas requirement of function PigFitnessClub.updateHeight 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: 1295:0:

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: 1350:4:

Miscellaneous

Constant/View/Pure functions:

PigFitnessClub._verify(bytes32,bytes32,bytes32[]) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 1427:4:

Similar variable names:

PigFitnessClub._verify(bytes32,bytes32,bytes32[]) : Variables have very similar names "root" and "proof". Note: Modifiers are currently not considered by this static analysis.
Pos: 1428:41:

Similar variable names:

PigFitnessClub.whitelistMint(uint8,bytes32[]) : Variables have very similar names "WLpaused" and "paused". Note: Modifiers are currently not considered by this static analysis.
Pos: 1440:13:

No return:

MerkleProof._efficientHash(bytes32,bytes32): Defines a return type but never explicitly returns a value.
Pos: 1115:4:

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: 1296:4:

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: 1300:4:

Delete from dynamic array:

Using "delete" on an array leaves a gap. The length of the array remains the same. If you want to remove the empty position you need to shift items manually and update the "length" property.
more
Pos: 1321:8:

Solhint Linter

PigFitnessClub.sol

```
PigFitnessClub.sol:475:6: Error: Parse error: missing 'constant' at
'ApprovalCallerNotOwnerNorApproved'
PigFitnessClub.sol:475:39: Error: Parse error: missing '=' at '('
PigFitnessClub.sol:476:6: Error: Parse error: missing 'constant' at
'ApprovalQueryForNonexistentToken'
PigFitnessClub.sol:476:38: Error: Parse error: missing '=' at '('
PigFitnessClub.sol:477:6: Error: Parse error: missing 'constant' at
'ApproveToCaller'
PigFitnessClub.sol:477:21: Error: Parse error: missing '=' at '('
PigFitnessClub.sol:478:6: Error: Parse error: missing 'constant' at
'ApprovalToCurrentOwner'
PigFitnessClub.sol:478:28: Error: Parse error: missing '=' at '('
PigFitnessClub.sol:479:6: Error: Parse error: missing 'constant' at
'BalanceQueryForZeroAddress'
PigFitnessClub.sol:479:32: Error: Parse error: missing '=' at '('
PigFitnessClub.sol:480:6: Error: Parse error: missing 'constant' at
'MintToZeroAddress'
PigFitnessClub.sol:480:23: Error: Parse error: missing '=' at '('
PigFitnessClub.sol:481:6: Error: Parse error: missing 'constant' at
'MintZeroQuantity'
PigFitnessClub.sol:481:22: Error: Parse error: missing '=' at '('
PigFitnessClub.sol:482:6: Error: Parse error: missing 'constant' at
'OwnerQueryForNonexistentToken'
PigFitnessClub.sol:482:35: Error: Parse error: missing '=' at '('
PigFitnessClub.sol:483:6: Error: Parse error: missing 'constant' at
'TransferCallerNotOwnerNorApproved'
PigFitnessClub.sol:483:39: Error: Parse error: missing '=' at '('
PigFitnessClub.sol:484:6: Error: Parse error: missing 'constant' at
'TransferFromIncorrectOwner'
PigFitnessClub.sol:484:32: Error: Parse error: missing '=' at '('
PigFitnessClub.sol:485:6: Error: Parse error: missing 'constant' at
'TransferToNonERC721ReceiverImplementer'
PigFitnessClub.sol:485:44: Error: Parse error: missing '=' at '('
PigFitnessClub.sol:486:6: Error: Parse error: missing 'constant' at
'TransferToZeroAddress'
PigFitnessClub.sol:486:27: Error: Parse error: missing '=' at '('
PigFitnessClub.sol:487:6: Error: Parse error: missing 'constant' at
'URIQueryForNonexistentToken'
PigFitnessClub.sol:487:33: Error: Parse error: missing '=' at '('
PigFitnessClub.sol:561:18: Error: Parse error: missing ';' at '{'
PigFitnessClub.sol:572:18: Error: Parse error: missing ';' at '{'
PigFitnessClub.sol:591:66: Error: Parse error: mismatched input '('
expecting {';', '='}
PigFitnessClub.sol:631:18: Error: Parse error: missing ';' at '{'
PigFitnessClub.sol:652:44: Error: Parse error: mismatched input '('
expecting {';', '='}
PigFitnessClub.sol:680:65: Error: Parse error: mismatched input '('
expecting {';', '='}
PigFitnessClub.sol:700:54: Error: Parse error: mismatched input '('
expecting {';', '='}
```

```
PigFitnessClub.sol:703:52: Error: Parse error: mismatched input '('
expecting {';', '='}
PigFitnessClub.sol:713:70: Error: Parse error: mismatched input '('
expecting {';', '='}
PigFitnessClub.sol:722:60: Error: Parse error: mismatched input '('
expecting {';', '='}
PigFitnessClub.sol:768:57: Error: Parse error: mismatched input '('
expecting {';', '='}
PigFitnessClub.sol:823:54: Error: Parse error: mismatched input '('
expecting {';', '='}
PigFitnessClub.sol:824:50: Error: Parse error: mismatched input '('
expecting {';', '='}
PigFitnessClub.sol:830:18: Error: Parse error: missing ';' at '{'
PigFitnessClub.sol:844:69: Error: Parse error: mismatched input '('
expecting {';', '='}
PigFitnessClub.sol:876:73: Error: Parse error: mismatched input '('
expecting {';', '='}
PigFitnessClub.sol:882:72: Error: Parse error: mismatched input '('
expecting {';', '='}
PigFitnessClub.sol:883:58: Error: Parse error: mismatched input '('
expecting {';', '='}
PigFitnessClub.sol:893:18: Error: Parse error: missing ';' at '{'
PigFitnessClub.sol:946:76: Error: Parse error: mismatched input '('
expecting {';', '='}
PigFitnessClub.sol:957:18: Error: Parse error: missing ';' at '{'
PigFitnessClub.sol:982:48: Error: Parse error: mismatched input ';'
expecting '('
PigFitnessClub.sol:986:18: Error: Parse error: missing ';' at '{'
PigFitnessClub.sol:1024:61: Error: Parse error: mismatched input '('
expecting {';', '='}
PigFitnessClub.sol:1182:29: Error: Parse error: mismatched input '('
expecting {';', '='}
PigFitnessClub.sol:1197:18: 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