Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	Datingverse
Website:	https://datingverse.org
Platform:	Ethereum
Language:	Solidity
Date:	May 7th, 2022

Table of contents

Email: audit@EtherAuthority.io

`

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

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by the Datingverse team to perform the Security audit of the Datingverse NFT token 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 7th, 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

The Datingverse is an ERC721A compliant NFT token smart contract. This Contract has functions like diamondPreMint, diamondMint, whiteMint, publicMint, setBaseURI, etc. The Datingverse contract inherits the Ownable, ReentrancyGuard, Strings, SafeMath, MerkleProof standard smart contracts from the OpenZeppelin library. These OpenZeppelin contracts are considered community-audited and time-tested, and hence are not part of the audit scope.

Audit scope

Name	**Code Review and Security Analysis Report for Datingverse Token Smart Contract**
Platform	**Ethereum / Solidity**
File	Datingverse.sol
File MD5 Hash	4FD9359F7E8174E18C2FB9714A2F367D
Audit Date	May 7th, 2022
Revision Code MD5 Hash	AC40AFDB8A22D141D30610AB8B630D3F
Revision Date	May 11th, 2022

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: ● Name: Datingverse ● Symbol: Datingverse ● Maximum Total Diamond Supply: 500 ● Maximum Total Supply: 5000 ● Maximum Mint: 2 ● Price: 0.1 ether ● Refund Guarantee Active Function ● Refunded NFTs will be burned ● The owner can withdraw ether from the contract after the Refund Guarantee period (which is 10 days) is over.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity based smart contracts are **“Secured”**. This token contract does contain owner control, which does not make it fully decentralized.

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

We found 0 critical, 0 high, 0 medium and 1 low and some very low level issues.
These issues are fixed / acknowledged in the revised 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.

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	Moderated
	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	Moderated
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 Datingverse 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 Datingverse Token.

The Datingverse 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** commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a Datingverse 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. But the contract is straightforward so it's easy to understand its programming logic.

Another source of information was its official website https://datingverse.org 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.

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	nonReentrant	modifier	Passed	No Issue
8	reserveDatingverses	external	access only Owner	No Issue
9	callerIsUser	modifier	Passed	No Issue
10	mintQuantityVerify	modifier	Passed	No Issue
11	mintPriceVerify	modifier	Passed	No Issue
12	setStatus	external	access only Owner	No Issue
13	setMerkleRoot	external	access only Owner	No Issue
14	setDiamondPreMerkleRoot	external	access only Owner	No Issue
15	setDiamondMerkleRoot	external	access only Owner	No Issue
16	getPrice	read	Passed	No Issue
17	diamondPreMint	external	Passed	No Issue
18	diamondMint	external	Passed	No Issue
19	whiteMint	external	Passed	No Issue
20	publicMint	external	Passed	No Issue
21	_baseURI	internal	Passed	No Issue
22	setBaseURI	external	access only Owner	No Issue
23	tokenURI	read	Passed	No Issue
24	togglePlaceholder	external	access only Owner	No Issue
25	withdraw	external	access only Owner	No Issue
26	refund	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

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Missing events in many functions:

It is good practice to emit appropriate events in the contract when significant state change is being done. This helps UI / clients to properly coordinate with the blockchain. Following functions can have an event:

- setStatus
- setMerkleRoot
- setDiamondPreMerkleRoot
- setDiamondMerkleRoot
- setBaseURI
- togglePlaceholder
- withdraw
- refund

Status: This issue is acknowledged

Very Low / Informational / Best practices:

(1) SafeMath Library:

SafeMath Library is used in this contract code, but the compiler version is greater than or equal to 0.8.0, Then it will be not required to use, solidity automatically handles overflow/underflow.

Resolution: Remove the SafeMath library and use normal math operators, It will improve code size, and less gas consumption.

Status: This issue is acknowledged

(2) These loops can be merged

```
function refund(uint256[] memory tokenIds) external callerIsUser {
    require(status == Status.Refund, "refund has not started yet");
    require(tokenIds.length > 0, "no tokenId support");
    require(tokenIds.length <= maxTotalSupply, "Invalid token");
    for (uint i = 0; i < tokenIds.length; i++) {
        require(ownerOf(tokenIds[i]) == msg.sender, "Invalid owner");
    }
    for (uint256 i = 0; i < tokenIds.length; i++) {
        _burn(tokenIds[i]);
    }
    uint256 totalCost = PRICE * tokenIds.length;
    Address.sendValue(payable(msg.sender), totalCost);
```

The “for loops” in this function can be merged as their iterations are the same. Although this does not raise any security issues, to make the code clean and to increase the readability, we suggest combining these both loops.

Status: This issue is fixed in the revised 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:

- reserveDatingverses: Owner can reserve some Datingverses.
- setStatus: Owner can set status.
- setMerkleRoot: Owner can set merkle root value.
- setDiamondPreMerkleRoot: Owner can set diamond pre merkle root.
- setDiamondMerkleRoot: Owner can set diamond merkle root.
- setBaseURI: Owner can set baseURI.
- togglePlaceholder: Owner can toggle placeholder.
- withdraw: Owner can withdraw contract balance to creator.

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 any major issue. 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.

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".**

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).

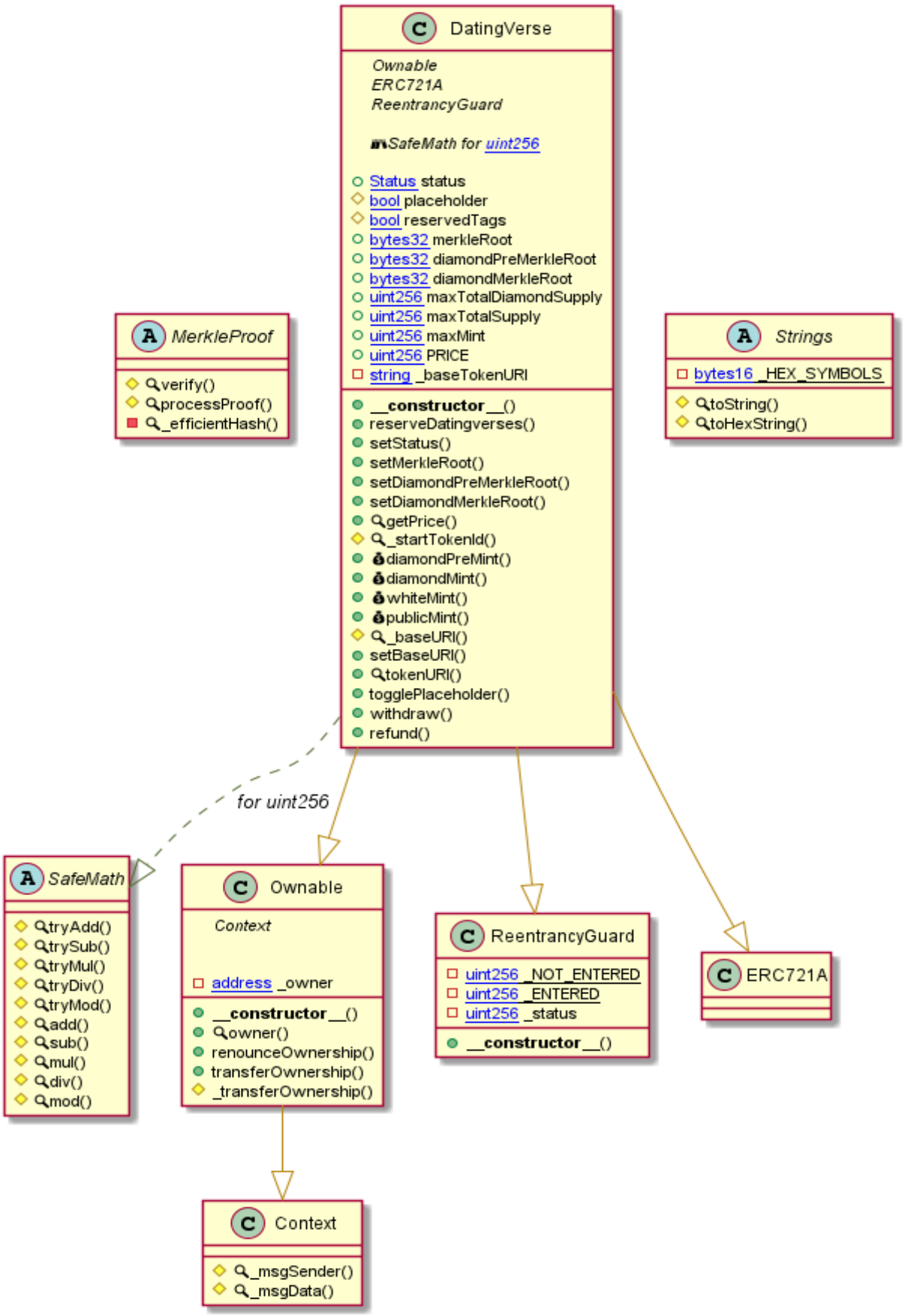
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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



Slither Results Log

Slither Log >> Datingverse.sol

```
INFO:Detectors:
DatingVerse.setStatus(DatingVerse.Status)._status (Datingverse.sol#1300) shadows:
        - ReentrancyGuard._status (Datingverse.sol#1230) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Variable 'ERC721A._checkOnERC721Received(address,address,uint256,bytes).retval (Datingverse.sol#1102)' in ERC721A._checkOnERC7
ceived(address,address,uint256,bytes) (Datingverse.sol#1095-1116) potentially used before declaration: retval == IERC721Receiv
nERC721Received.selector (Datingverse.sol#1103)
Variable 'ERC721A._checkOnERC721Received(address,address,uint256,bytes).reason (Datingverse.sol#1104)' in ERC721A._checkOnERC7
ceived(address,address,uint256,bytes) (Datingverse.sol#1095-1116) potentially used before declaration: reason.length == 0 (Dat
erse.sol#1105)
Variable 'ERC721A._checkOnERC721Received(address,address,uint256,bytes).reason (Datingverse.sol#1104)' in ERC721A._checkOnERC7
ceived(address,address,uint256,bytes) (Datingverse.sol#1095-1116) potentially used before declaration: revert(uint256,uint256)
+ reason,mload(uint256)(reason)) (Datingverse.sol#1109)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables
INFO:Detectors:
MerkleProof._efficientHash(bytes32,bytes32) (Datingverse.sol#193-199) uses assembly
        - INLINE ASM (Datingverse.sol#194-198)
Address.verifyCallResult(bool,bytes,string) (Datingverse.sol#393-413) uses assembly
        - INLINE ASM (Datingverse.sol#405-408)
ERC721A._checkOnERC721Received(address,address,uint256,bytes) (Datingverse.sol#1095-1116) uses assembly
        - INLINE ASM (Datingverse.sol#1108-1110)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Address.functionCall(address,bytes) (Datingverse.sol#277-279) is never used and should be removed
Address.functionCall(address,bytes,string) (Datingverse.sol#287-293) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (Datingverse.sol#306-312) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (Datingverse.sol#320-331) is never used and should be removed
Address.functionDelegateCall(address,bytes) (Datingverse.sol#366-368) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (Datingverse.sol#376-385) is never used and should be removed
Address.functionStaticCall(address,bytes) (Datingverse.sol#339-341) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (Datingverse.sol#349-358) is never used and should be removed
Address.verifyCallResult(bool,bytes,string) (Datingverse.sol#393-413) is never used and should be removed
Context._msgData() (Datingverse.sol#691-693) is never used and should be removed
DatingVerse._baseURI() (Datingverse.sol#1344-1346) is never used and should be removed
ERC721A._baseURI() (Datingverse.sol#816-818) is never used and should be removed
```

```
SafeMath.add(uint256,uint256) (Datingverse.sol#492-494) is never used and should be removed
SafeMath.div(uint256,uint256) (Datingverse.sol#534-536) is never used and should be removed
SafeMath.div(uint256,uint256,string) (Datingverse.sol#590-599) is never used and should be removed
SafeMath.mod(uint256,uint256) (Datingverse.sol#550-552) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (Datingverse.sol#616-625) is never used and should be removed
SafeMath.mul(uint256,uint256) (Datingverse.sol#520-522) is never used and should be removed
SafeMath.sub(uint256,uint256) (Datingverse.sol#506-508) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (Datingverse.sol#567-576) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (Datingverse.sol#421-427) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (Datingverse.sol#463-468) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (Datingverse.sol#475-480) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (Datingverse.sol#446-456) is never used and should be removed
SafeMath.trySub(uint256,uint256) (Datingverse.sol#434-439) is never used and should be removed
Strings.toHexString(uint256) (Datingverse.sol#658-669) is never used and should be removed
Strings.toHexString(uint256,uint256) (Datingverse.sol#674-684) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Pragma version^0.8.4 (Datingverse.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (Datingverse.sol#252-257):
        - (success) = recipient.call{value: amount}() (Datingverse.sol#255)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (Datingverse.sol#320-331):
        - (success,returndata) = target.call{value: value}(data) (Datingverse.sol#329)
Low level call in Address.functionStaticCall(address,bytes,string) (Datingverse.sol#349-358):
        - (success,returndata) = target.staticcall(data) (Datingverse.sol#356)
Low level call in Address.functionDelegateCall(address,bytes,string) (Datingverse.sol#376-385):
        - (success,returndata) = target.delegatecall(data) (Datingverse.sol#383)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter DatingVerse.setStatus(DatingVerse.Status)._status (Datingverse.sol#1300) is not in mixedCase
Variable DatingVerse.PRICE (Datingverse.sol#1272) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

```
INFO:Detectors:
Parameter DatingVerse.setStatus(DatingVerse.Status)._status (Datingverse.sol#1300) is not in mixedCase
Variable DatingVerse.PRICE (Datingverse.sol#1272) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
balanceOf(address) should be declared external:
        - ERC721A.balanceOf(address) (Datingverse.sol#773-776)
name() should be declared external:
        - ERC721A.name() (Datingverse.sol#790-792)
symbol() should be declared external:
        - ERC721A.symbol() (Datingverse.sol#797-799)
tokenURI(uint256) should be declared external:
        - DatingVerse.tokenURI(uint256) (Datingverse.sol#1350-1357)
        - ERC721A.tokenURI(uint256) (Datingverse.sol#804-809)
approve(address,uint256) should be declared external:
        - ERC721A.approve(address,uint256) (Datingverse.sol#823-829)
getApproved(uint256) should be declared external:
        - ERC721A.getApproved(uint256) (Datingverse.sol#834-838)
setApprovalForAll(address,bool) should be declared external:
        - ERC721A.setApprovalForAll(address,bool) (Datingverse.sol#843-845)
isApprovedForAll(address,address) should be declared external:
        - ERC721A.isApprovedForAll(address,address) (Datingverse.sol#850-852)
transferFrom(address,address,uint256) should be declared external:
        - ERC721A.transferFrom(address,address,uint256) (Datingverse.sol#857-866)
safeTransferFrom(address,address,uint256) should be declared external:
        - ERC721A.safeTransferFrom(address,address,uint256) (Datingverse.sol#871-877)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (Datingverse.sol#1190-1192)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (Datingverse.sol#1198-1201)
getPrice() should be declared external:
        - DatingVerse.getPrice() (Datingverse.sol#1312-1314)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:Datingverse.sol analyzed (14 contracts with 75 detectors), 57 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

Datingverse.sol

Security

Transaction origin:

Use of tx.origin: "tx.origin" is useful only in very exceptional cases. If you use it for authentication, you usually want to replace it by "msg.sender", because otherwise any contract you call can act on your behalf.
more
Pos: 47:16:

Gas & Economy

Gas costs:

Gas requirement of function DatingVerse.maxTotalDiamondSupply 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: 28:4:

Gas costs:

Gas requirement of function DatingVerse.withdraw 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: 123:4:

Gas costs:

Gas requirement of function DatingVerse.refund 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: 129:4:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.
more
Pos: 136:8:

Miscellaneous

Constant/View/Pure functions:

DatingVerse.tokenURI(uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 111:4:

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: 47:8:

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: 51:8:

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: 52:8:

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: 132:8:

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: 134:12:

Solhint Linter

Datingverse.sol

```
Datingverse.sol:8:1: Error: Compiler version ^0.8.7 does not satisfy
the r semver requirement
Datingverse.sol:23:5: Error: Explicitly mark visibility of state
Datingverse.sol:24:5: Error: Explicitly mark visibility of state
Datingverse.sol:31:30: Error: Variable name must be in mixedCase
Datingverse.sol:32:5: Error: Explicitly mark visibility in function
(Set ignoreConstructors to true if using solidity >=0.7.0)
Datingverse.sol:47:17: Error: Avoid to use tx.origin
Datingverse.sol:56:48: Error: Use double quotes for string literals
```

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