Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Customer:	Musée
Website:	http://www.musee.art/
Platform:	Ethereum
Language:	Solidity
Date:	June 14th, 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 10
Audit Findings 10
Conclusion 13
Our Methodology 14
Disclaimers 16
Appendix
- Code Flow Diagram 17
- Slither Report Log 18

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

We were contracted by the Musée team to perform the Security audit of the Musée 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 June 14th, 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

Musée's platform is built on top of a limited quantity of plots each powered by their own unique NFTs. Using the Ethereum blockchain ERC-721 tokens, one can buy, own, combine and sell the NFTs that have full ownership of the plots. Musée allows users to buy an NFT on the Grid. It will allow users to buy more than one NFT at a time.

Audit scope

Name	**Code Review and Security Analysis Report for Musée Token (NFT) Smart Contract**
Platform	**Ethereum / Solidity**
File	Musée.sol
File MD5 Hash	4753A880BC9112F9C9E64797F34119A4
File SHA256 Hash	7D55487A7B4FDB116FAF14497A6C519B3854CAB3AFD0E46863D14D1CDE635D00
Audit Date	June 14th, 2021

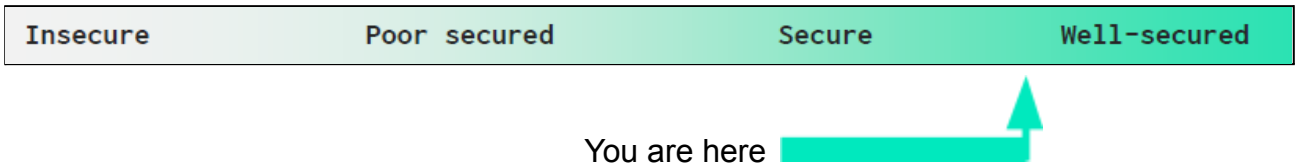
PS: There are 3 external imports from open zeppelin. These files are not included in the audit scope and thus they are not audited.

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Name: Musée Token	**YES, This is valid.**
Symbol: MUSEE	**YES, This is valid.**
MaxIDs: 10000	**YES, This is valid.**
PresalePrice1: 0.5 ether	**YES, This is valid.**
PresalePrice2: 1 ether	**YES, This is valid.**
PresalePrice3: 1.5 ether	**YES, This is valid.**
PresalePrice4: 2 ether	**YES, This is valid.**
Price: 2.5 ether	**YES, This is valid.**
Owner can set URI, price, and presale prices. Owner also can withdraw funds.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity smart contract is **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 1 low and some very low level issues.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Moderated
	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	Passed
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	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
	Other code specification issues	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Moderated
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Code Quality

This audit scope has 1 smart contract. This smart contract also contains Libraries, Smart contracts inherits and Interfaces. This is a compact and well written contract.

The libraries in the Musée 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 Musée Token.

The Musée 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 Musée Token smart contract code in the form of a hash code MD5 and SHA256 format. 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 http://www.musee.art/ 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

(1) Interface

(a) IERC721

(2) Inherited contracts

(a) ERC721

(b) Ownable

(3) Usages

(a) using SafeMath for uint256;

(4) Events

(a) event BuyItem(address buyer, uint256 id);

(b) event NewPrice(uint256 oldPrice, uint256 newPrice);

(c) event NewPresalePrice1(uint256 oldPrice, uint256 newPrice);

(d) event NewPresalePrice2(uint256 oldPrice, uint256 newPrice);

(e) event NewPresalePrice3(uint256 oldPrice, uint256 newPrice);

(f) event NewPresalePrice4(uint256 oldPrice, uint256 newPrice);

(g) event NewURI(string oldURI, string newURI);

(h) event WithdrawBalance(uint256 balance, address to);

(5) Functions

Sl.	Functions	Type	Observation	Conclusion
1	buy	internal	Passed	No Issue
2	buyMultiple	external	High Gas Loop	No Issue
3	setUri	external	access only Owner	No Issue
4	setPrice	external	access only Owner	No Issue
5	setPresalePrice1	external	access only Owner	No Issue
6	setPresalePrice2	external	access only Owner	No Issue
7	setPresalePrice3	external	access only Owner	No Issue
8	setPresalePrice4	external	access only Owner	No Issue
9	withdrawFunds	external	access only Owner	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 functions
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) Possibility of high gas consumption.

```
109         for (uint256 i = 0; i < _ids.length; i++) {
110             buy(_ids[i]);
111         }
```

If so many NFTs are purchased, then this logic will fail, as it might hit the block's gas limit..

Resolution: Practically, this does not create an issue in most cases. Because the chance of someone buying 1000+ NFT in one go is very rare. So, as long as limited purchases are done, then this is completely ok.

Very Low / Discussion / Best practices:

(1) Use latest solidity version:

```
pragma solidity ^0.7.3;
```

Using the latest solidity will prevent any compiler level bugs.

Resolution: Please use 0.8.5 which is the latest version at a time of this audit.

(2) While purchasing tokens, the user has to provide a list of token IDs. Since people can buy any token ID, the user may not know available token IDs and he has to first find the token IDs he wants to buy. In general user experience scenarios, this may be troublesome.

Resolution: Ideally, there should be an internal token ID counter. Counter gets incremented as new tokens are minted. So, users do not have to specifically provide a list of tokens IDs. On another hand, If current logic is part of the plan, then completely ok.

Centralization

This smart contract has 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:

- withdrawFunds: allows the owner to withdraw funds.
- setUri: Allows the owner to change token URI.
- setPrice: Allows the owner to set the price.
- setPresalePrice1: Allows the owner to set the Presale price1.
- setPresalePrice2: Allows the owner to set the Presale price2.
- setPresalePrice3: Allows the owner to set the Presale price3.
- setPresalePrice4: Allows the owner to set the Presale price4.

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 are fixed/acknowledged in the smart contracts. **So it is 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 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).

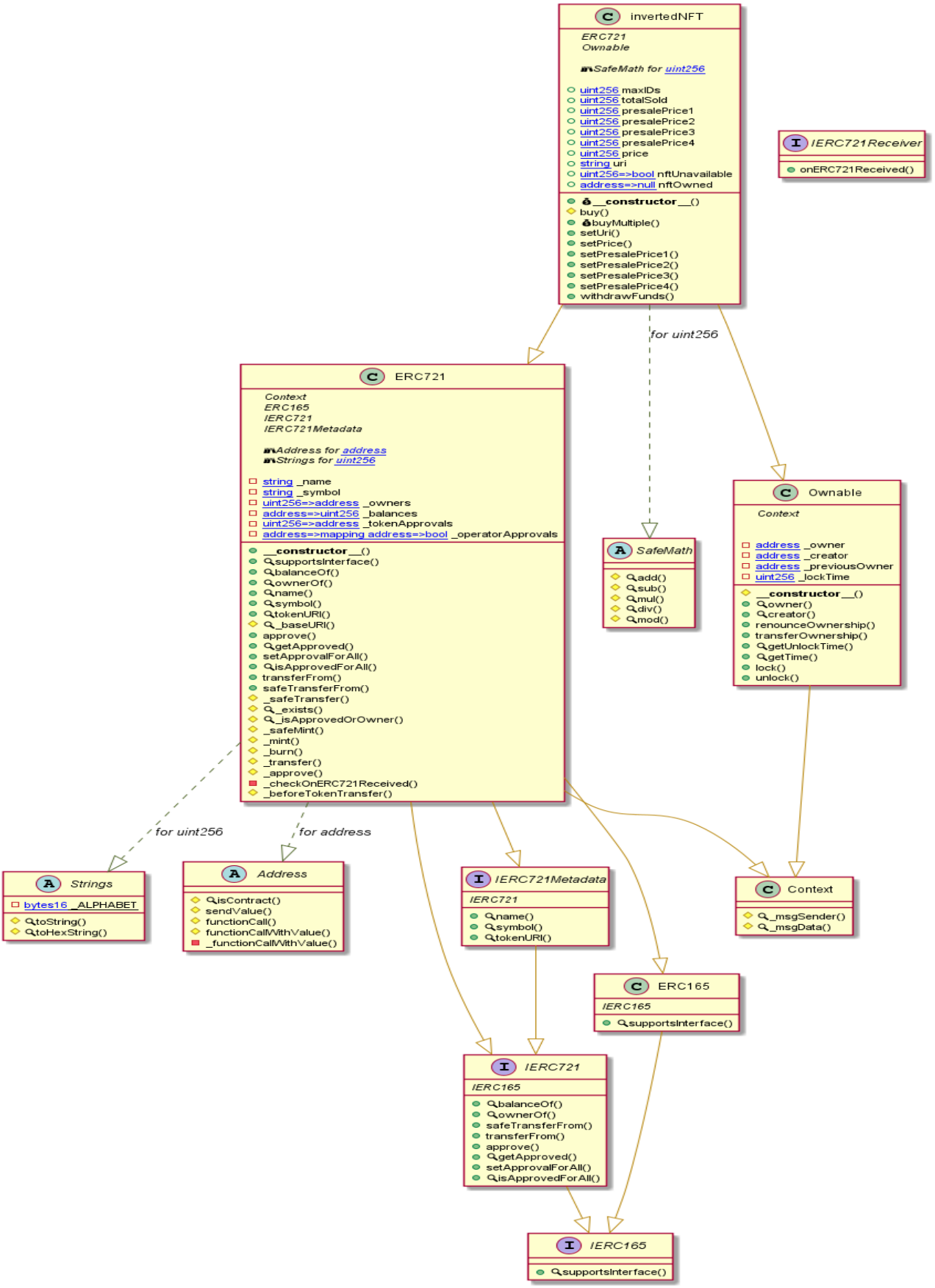
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we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



Email: audit@EtherAuthority.io

Slither Results Log

Slither log >> Musée.sol

```
INFO:Detectors:
Musée.withdrawFunds() (Musée.sol#1078-1084) sends eth to arbitrary user
	Dangerous calls:
	- address(owner()).transfer(address(this).balance) (Musée.sol#1083)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinatio
ns
INFO:Detectors:
Reentrancy in Musée.buy(uint256) (Musée.sol#984-1004):
	External calls:
	- _safeMint(recipient,_id) (Musée.sol#993)
		- IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,_data)
(Musée.sol#774-784)
	State variables written after the call(s):
	- nftUnavailable[_id] = true (Musée.sol#996)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1
INFO:Detectors:
ERC721._checkOnERC721Received(address,address,uint256,bytes) (Musée.sol#767-788) ignores return
value by IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,_data) (Musée.sol#774-784)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).retval (Musée.sol#774)' in
ERC721._checkOnERC721Received(address,address,uint256,bytes) (Musée.sol#767-788) potentially used
before declaration: retval == IERC721Receiver(to).onERC721Received.selector (Musée.sol#775)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (Musée.sol#776)' in
ERC721._checkOnERC721Received(address,address,uint256,bytes) (Musée.sol#767-788) potentially used
before declaration: reason.length == 0 (Musée.sol#777)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (Musée.sol#776)' in
ERC721._checkOnERC721Received(address,address,uint256,bytes) (Musée.sol#767-788) potentially used
before declaration: revert(uint256,uint256)(32 + reason,mload(uint256)(reason)) (Musée.sol#781)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables
INFO:Detectors:
Reentrancy in Musée.buy(uint256) (Musée.sol#984-1004):
	External calls:
	- _safeMint(recipient,_id) (Musée.sol#993)
		- IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,_data)
(Musée.sol#774-784)
	State variables written after the call(s):
	- nftOwned[recipient].push(_id) (Musée.sol#999)
	- totalSold = totalSold.add(1) (Musée.sol#997)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in Musée.buy(uint256) (Musée.sol#984-1004):
	External calls:
	- _safeMint(recipient,_id) (Musée.sol#993)
		- IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,_data)
(Musée.sol#774-784)
	Event emitted after the call(s):
	- BuyItem(msg.sender,_id) (Musée.sol#1001)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
INFO:Detectors:
Ownable.unlock() (Musée.sol#925-930) uses timestamp for comparisons
	Dangerous comparisons:
	- require(bool,string)(now > _lockTime,Contract is locked until 7 days) (Musée.sol#927)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
```

```
INFO:Detectors:
Address.isContract(address) (Musée.sol#87-98) uses assembly
	- INLINE ASM (Musée.sol#94-96)
Address._functionCallWithValue(address,bytes,uint256,string) (Musée.sol#195-221) uses assembly
	- INLINE ASM (Musée.sol#213-216)
ERC721._checkOnERC721Received(address,address,uint256,bytes) (Musée.sol#767-788) uses assembly
	- INLINE ASM (Musée.sol#780-782)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Musée.buy(uint256) (Musée.sol#984-1004) compares to a boolean constant:
	-require(bool,string)(nftUnavailable[_id] == false,buy::NFT is already purchased.) (Musée.sol#988)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
Address._functionCallWithValue(address,bytes,uint256,string) (Musée.sol#195-221) is never used and
should be removed
Address.functionCall(address,bytes) (Musée.sol#142-144) is never used and should be removed
Address.functionCall(address,bytes,string) (Musée.sol#152-158) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (Musée.sol#171-177) is never used and should be
removed
Address.functionCallWithValue(address,bytes,uint256,string) (Musée.sol#185-193) is never used and should
be removed
Address.sendValue(address,uint256) (Musée.sol#116-122) is never used and should be removed
Context._msgData() (Musée.sol#410-413) is never used and should be removed
ERC721._burn(uint256) (Musée.sol#702-714) is never used and should be removed
SafeMath.div(uint256,uint256) (Musée.sol#843-845) is never used and should be removed
SafeMath.div(uint256,uint256,string) (Musée.sol#847-853) is never used and should be removed
SafeMath.mod(uint256,uint256) (Musée.sol#855-857) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (Musée.sol#859-862) is never used and should be removed
SafeMath.sub(uint256,uint256) (Musée.sol#820-822) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (Musée.sol#824-829) is never used and should be removed
Strings.toHexString(uint256) (Musée.sol#41-52) is never used and should be removed
Strings.toHexString(uint256,uint256) (Musée.sol#57-67) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (Musée.sol#116-122):
	- (success) = recipient.call{value: amount}() (Musée.sol#120)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (Musée.sol#195-221):
	- (success,returndata) = target.call{value: weiValue}(data) (Musée.sol#204)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter ERC721.safeTransferFrom(address,address,uint256,bytes)._data (Musée.sol#579) is not in
mixedCase
Contract Musée (Musée.sol#933-1085) is not in CapWords
Parameter Musée.buy(uint256)._id (Musée.sol#984) is not in mixedCase
Parameter Musée.buyMultiple(uint256[])._ids (Musée.sol#1010) is not in mixedCase
Parameter Musée.setUri(string)._uri (Musée.sol#1042) is not in mixedCase
Parameter Musée.setPrice(uint256)._price (Musée.sol#1048) is not in mixedCase
Parameter Musée.setPresalePrice1(uint256)._presalePrice1 (Musée.sol#1054) is not in mixedCase
Parameter Musée.setPresalePrice2(uint256)._presalePrice2 (Musée.sol#1060) is not in mixedCase
Parameter Musée.setPresalePrice3(uint256)._presalePrice3 (Musée.sol#1066) is not in mixedCase
Parameter Musée.setPresalePrice4(uint256)._presalePrice4 (Musée.sol#1072) is not in mixedCase
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (Musée.sol#411)" inContext (Musée.sol#405-414)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
INFO:Detectors:
Variable Musée.setPresalePrice1(uint256)._presalePrice1 (Musée.sol#1054) is too similar to
Musée.setPresalePrice2(uint256)._presalePrice2 (Musée.sol#1060)
Variable Musée.setPresalePrice1(uint256)._presalePrice1 (Musée.sol#1054) is too similar to
Musée.setPresalePrice3(uint256)._presalePrice3 (Musée.sol#1066)
Variable Musée.setPresalePrice1(uint256)._presalePrice1 (Musée.sol#1054) is too similar to
Musée.setPresalePrice4(uint256)._presalePrice4 (Musée.sol#1072)
Variable Musée.setPresalePrice2(uint256)._presalePrice2 (Musée.sol#1060) is too similar to
Musée.setPresalePrice3(uint256)._presalePrice3 (Musée.sol#1066)
```

Variable Musée.setPresalePrice2(uint256)._presalePrice2 (Musée.sol#1060) is too similar to Musée.setPresalePrice4(uint256)._presalePrice4 (Musée.sol#1072)
Variable Musée.setPresalePrice3(uint256)._presalePrice3 (Musée.sol#1066) is too similar to Musée.setPresalePrice4(uint256)._presalePrice4 (Musée.sol#1072)
Variable Musée.presalePrice1 (Musée.sol#963) is too similar to Musée.presalePrice2 (Musée.sol#964)
Variable Musée.presalePrice1 (Musée.sol#963) is too similar to Musée.presalePrice3 (Musée.sol#965)
Variable Musée.presalePrice1 (Musée.sol#963) is too similar to Musée.presalePrice4 (Musée.sol#966)
Variable Musée.presalePrice2 (Musée.sol#964) is too similar to Musée.presalePrice3 (Musée.sol#965)
Variable Musée.presalePrice2 (Musée.sol#964) is too similar to Musée.presalePrice4 (Musée.sol#966)
Variable Musée.presalePrice3 (Musée.sol#965) is too similar to Musée.presalePrice4 (Musée.sol#966)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
Musée.maxIDs (Musée.sol#960) should be constant
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
balanceOf(address) should be declared external:
- ERC721.balanceOf(address) (Musée.sol#459-462)
name() should be declared external:
- ERC721.name() (Musée.sol#476-478)
symbol() should be declared external:
- ERC721.symbol() (Musée.sol#483-485)
tokenURI(uint256) should be declared external:
- ERC721.tokenURI(uint256) (Musée.sol#490-495)
approve(address,uint256) should be declared external:
- ERC721.approve(address,uint256) (Musée.sol#509-519)
setApprovalForAll(address,bool) should be declared external:
- ERC721.setApprovalForAll(address,bool) (Musée.sol#533-538)
transferFrom(address,address,uint256) should be declared external:
- ERC721.transferFrom(address,address,uint256) (Musée.sol#550-559)
safeTransferFrom(address,address,uint256) should be declared external:
- ERC721.safeTransferFrom(address,address,uint256) (Musée.sol#564-570)
creator() should be declared external:
- Ownable.creator() (Musée.sol#885-887)
renounceOwnership() should be declared external:
- Ownable.renounceOwnership() (Musée.sol#899-902)
transferOwnership(address) should be declared external:
- Ownable.transferOwnership(address) (Musée.sol#904-908)
Reference:
https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:Musée.sol analyzed (12 contracts with 75 detectors), 70 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration

Email: audit@EtherAuthority.io