Smart Contract Security Audit V1

Marketplace Assets

17/12/2021

https://saferico.com/

business@saferico.com
https://t.me/SFI_ANN

–

Table of Contents

Background

The purpose of the audit was to achieve the following:

- Ensure that the smart contract functions as intended.
- Identify potential security issues with the smart contract.

The information in this report should be used to understand the risk exposure of the smart contract, and as a guide to improve the security posture of the smart contract by remediating the issues that were identified.

Project Information

- **Website**:
- **Telegram group:**
- **Twitter**:
- **GitHub**:
- **WhitePaper**:
- **Platform**:
- **Contract Address**:

Executive Summary

According to our assessment, the customer`s solidity smart contract is **Well Secured**.

Well Secured	✔
Secured	
Poor Secured	
Insecure	

Automated checks are with remix IDE. All issues were performed by the team, which included the analysis of code functionality, manual audit found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the audit overview section. The general overview is presented in the Project Information section and all issues found are located in the audit overview section.

Team found 0 critical, 0 high, 0 medium, 0 low, 0 very low-level issues and 0 notes in all solidity files of the contract

The files:

MarketplaceAssets.sol
IMarketplaceAssets.sol

File and Function Level Report

File in Scope:

Contract Name	SHA 256 hash	Contract Address
MarketplaceAssets.sol	**0cda5c30eae251015a5b014e420070ca70ae9d4f092ad5a954b7af2184b5cdf2**	0x77fb6c9b4626db6b697c71ef92caa6a74ac6e50e

- Contract: MarketplaceAssets
- Inherit: IMarketplaceAssets,Ownable,ERC721Holder
- Observation: All passed including security check
- Test Report: passed
- Score: passed
- Conclusion: passed

Function	Test Result	Type / Return Type	Score
getExistsAssetId	✔	Read / public	**Passed**
getAssetName	✔	Read / public	**Passed**
getExistsName	✔	Read / public	**Passed**
_MarketTokenIds	✔	Read / private	**Passed**
getFullAssetIdAndTokenId	✔	Read / public	**Passed**
getFullDataByOwner	✔	Read / public	**Passed**
getFullDataByTokenId	✔	Read / public	**Passed**
getFullDataByType	✔	Read / public	**Passed**
getNFTContract	✔	Read / public	**Passed**
getPrice	✔	Read / public	**Passed**
getSupplyByType	✔	Read / public	**Passed**

owner	✔	Read / public	**Passed**
_transferMoney	✔	Write / private/payable	**Passed**
addAssetToMarket	✔	Write / public	**Passed**
addBatchAssets	✔	Write / public	**Passed**
addNewAssets	✔	Write / public	**Passed**
buyAsset	✔	Write / payable	**Passed**
onERC721Received	✔	Write / public	**Passed**
Deposit	✔	Write / payable	**Passed**
removeAssetFromMarket	✔	Write / public	**Passed**
setAssetName	✔	Write / public	**Passed**
setTokenName	✔	Write / public	**Passed**
setTokenURI	✔	Write / public	**Passed**
TransferFrom	✔	Write / public	**Passed**
renounceOwnership	✔	Write / public	**Passed**
setNFTContract	✔	Write / public	**Passed**
setTokenAssetType	✔	Write / public	**Passed**
setTokenOwner	✔	Write / public	**Passed**
transferToken	✔	Write / payable	**Passed**
transferTokenByOwner	✔	Write / public	**Passed**

Issues Checking Status

No.	Issue Description	Checking Status
1	Compiler warnings.	**Passed**
2	Race conditions and Reentrancy. Cross-function race conditions.	**Passed**
3	Possible delays in data delivery.	**Passed**
4	Oracle calls.	**Passed**
5	Front running.	**Passed**
6	Timestamp dependence.	**Passed**
7	Integer Overflow and Underflow.	**Passed**
8	DoS with Revert.	**Passed**
9	DoS with block gas limit.	**Passed**
10	Methods execution permissions.	**Passed**
11	Economy model. If application logic is based on an incorrect economic model, the application would not function correctly and participants would incur financial losses. This type of issue is most often found in bonus rewards systems, Staking and Farming contracts, Vault and Vesting contracts, etc.	**Passed**
12	The impact of the exchange rate on the logic.	**Passed**
13	Private user data leaks.	**Passed**
14	Malicious Event log.	**Passed**
15	Scoping and Declarations.	**Passed**
16	Uninitialized storage pointers.	**Passed**
17	Arithmetic accuracy.	**Passed**
18	Design Logic.	**Passed**

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens 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
Note	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:

No Low severity vulnerabilities were found.

Very Low:

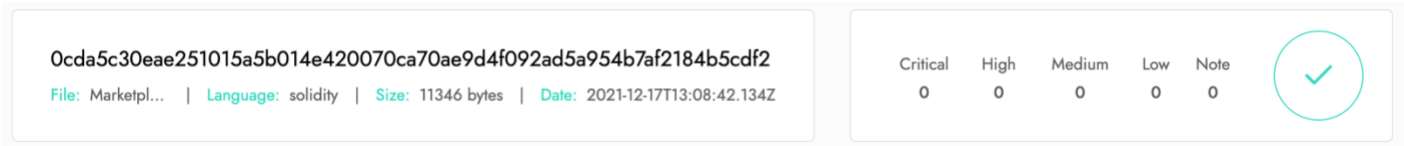
No Very Low severity vulnerabilities were found.

Notes:

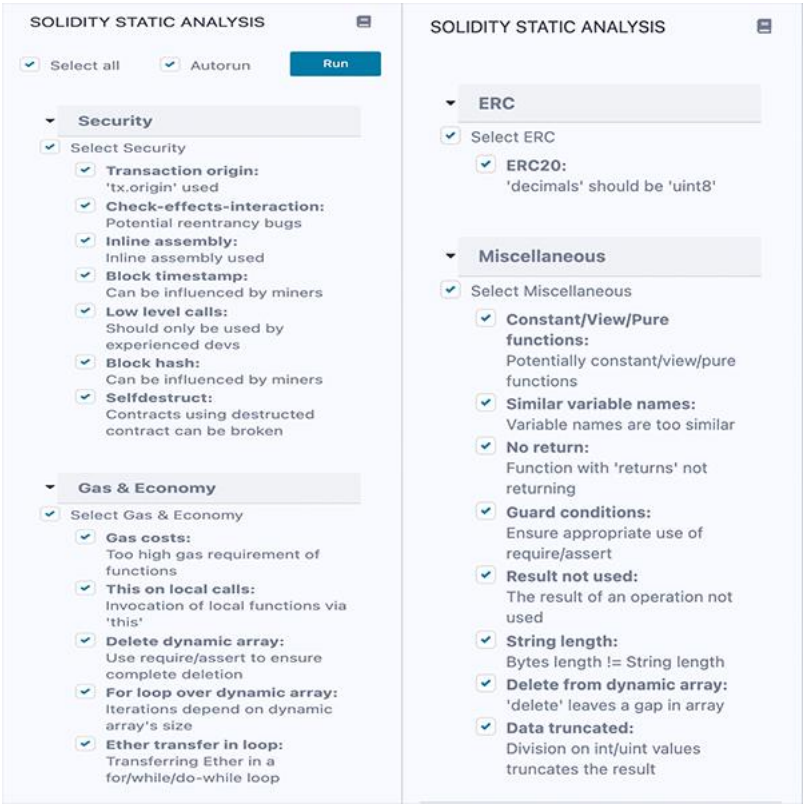
No Notes were found.

Automatic Testing

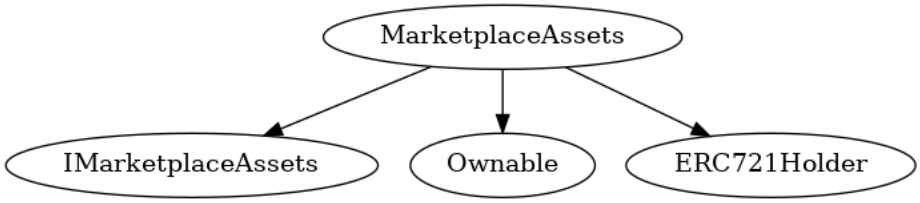
1- Check for security



2- SOLIDITY STATIC ANALYSIS

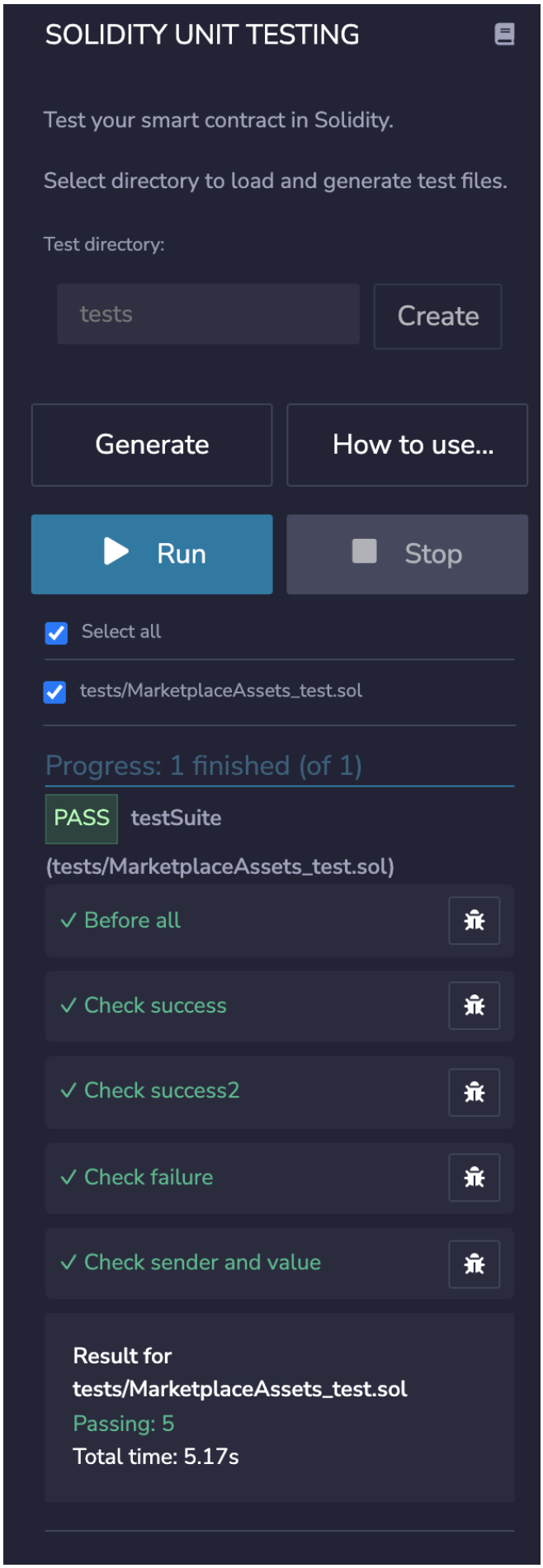


3- Inheritance graph

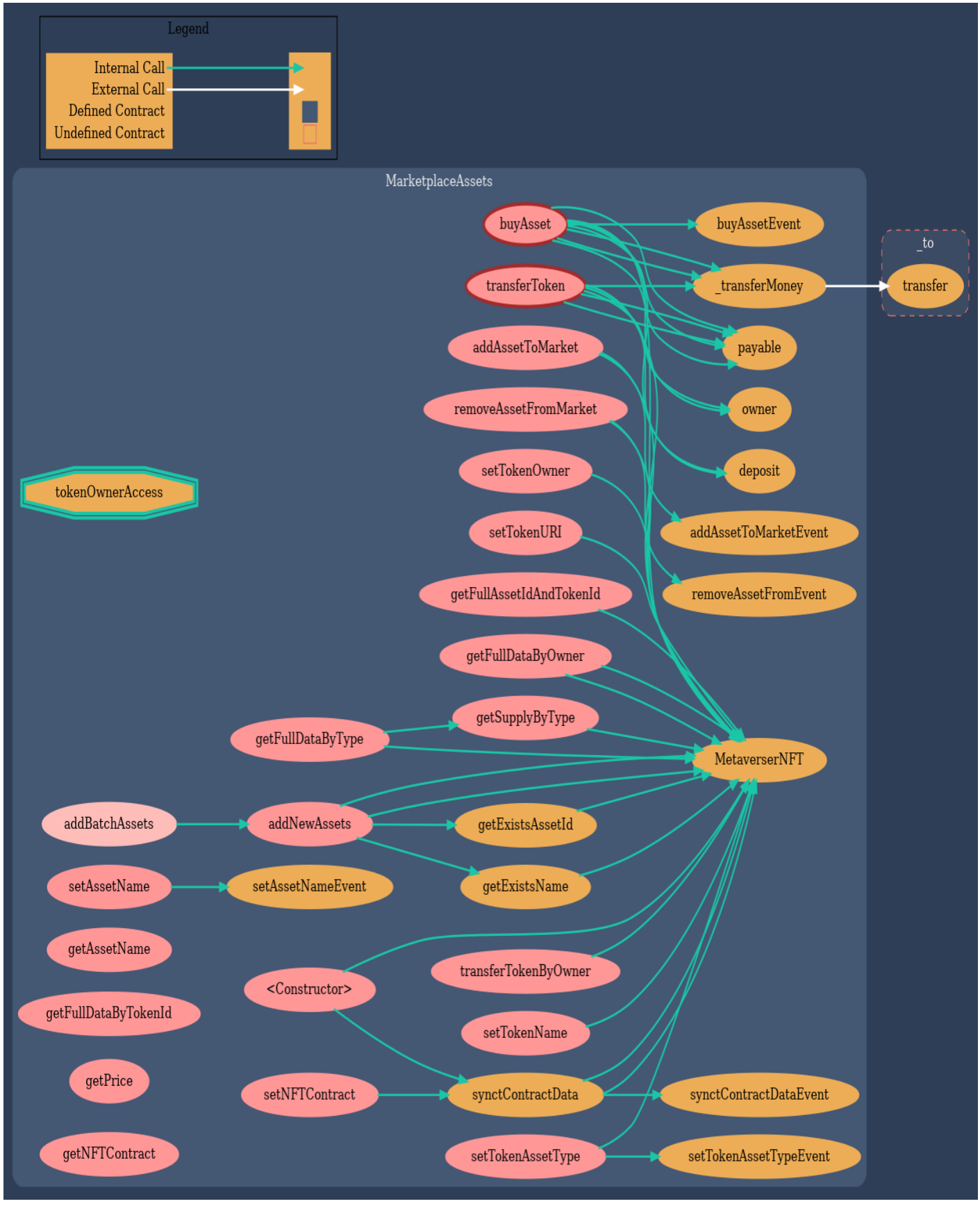


4- SOLIDITY UNIT TESTING

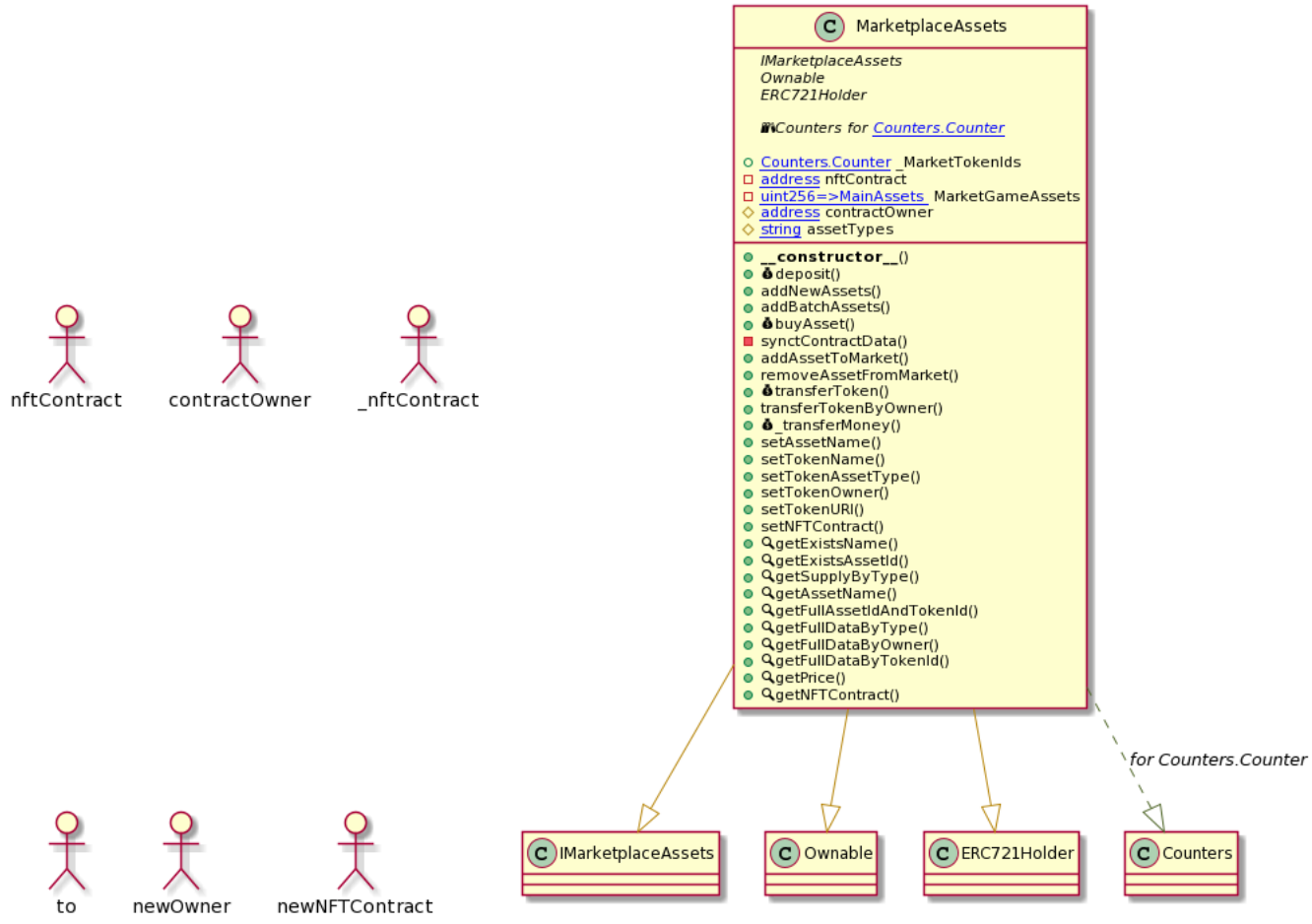
```
// SPDX-License-Identifier: GPL-3.0

pragma solidity >=0.4.22 <0.9.0;

// This import is automatically injected by Remix
import "remix_tests.sol";

// This import is required to use custom transaction context
// Although it may fail compilation in 'Solidity Compiler' plugin
// But it will work fine in 'Solidity Unit Testing' plugin
import "remix_accounts.sol";
import "../MarketplaceAssets.sol";

// File name has to end with '_test.sol', this file can contain more than one
testSuite contracts
contract testSuite {

    /// 'beforeAll' runs before all other tests
    /// More special functions are: 'beforeEach', 'beforeAll', 'afterEach' &
'afterAll'
    function beforeAll() public {
        // <instantiate contract>
        Assert.equal(uint(1), uint(1), "1 should be equal to 1");
    }

    function checkSuccess() public {
        // Use 'Assert' methods: https://remix-
ide.readthedocs.io/en/latest/assert_library.html
        Assert.ok(2 == 2, 'should be true');
        Assert.greaterThan(uint(2), uint(1), "2 should be greater than to 1");
        Assert.lesserThan(uint(2), uint(3), "2 should be lesser than to 3");
    }

    function checkSuccess2() public pure returns (bool) {
        // Use the return value (true or false) to test the contract
        return true;
    }

    function checkFailure() public {
        Assert.notEqual(uint(1), uint(2), "1 should not be equal to 1");
    }

    /// Custom Transaction Context: https://remix-
ide.readthedocs.io/en/latest/unittesting.html#customization
    /// #sender: account-1
    /// #value: 100
    function checkSenderAndValue() public payable {
        // account index varies 0-9, value is in wei
        Assert.equal(msg.sender, TestsAccounts.getAccount(1), "Invalid sender");
        Assert.equal(msg.value, 100, "Invalid value");
    }
}
```


SOLIDITY UNIT TESTING
Test your smart contract in Solidity.
Select directory to load and generate test files.
Test directory:
tests
Create
Generate
How to use...
Run
Stop
Select all
tests/MarketplaceAssets_test.sol
Progress: 1 finished (of 1)
PASS testSuite
(tests/MarketplaceAssets_test.sol)
✓ Before all
✓ Check success
✓ Check success2
✓ Check failure
✓ Check sender and value
Result for
tests/MarketplaceAssets_test.sol
Passing: 5
Total time: 5.17s

5- Call graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
MarketplaceAssets
buyAsset
buyAssetEvent
_to
transferToken
_transferMoney
transfer
addAssetToMarket
payable
removeAssetFromMarket
owner
deposit
tokenOwnerAccess
setTokenOwner
setTokenURI
addAssetToMarketEvent
getFullAssetIdAndTokenId
removeAssetFromEvent
getFullDataByOwner
getSupplyByType
getFullDataByType
MetaverserNFT
addBatchAssets
addNewAssets
getExistsAssetId
setAssetName
setAssetNameEvent
getExistsName
getAssetName
transferTokenByOwner
<Constructor>
getFullDataByTokenId
setTokenName
getPrice
setNFTContract
synctContractData
synctContractDataEvent
getNFTContract
setTokenAssetType
setTokenAssetTypeEvent

Unified Modeling Language (UML)



Functions signature

```
d0e30db0  =>  deposit()
e8b6848d  =>  addNewAssets(string,string,uint256,string)
2ce41b17  =>  addBatchAssets(MainAssetsInput[])
110d35b8  =>  buyAsset(uint256)
e706365d  =>  synctContractData()
04edb753  =>  addAssetToMarket(uint256,uint256)
6e6446e4  =>  removeAssetFromMarket(uint256)
bad42590  =>  transferToken(uint256,address)
1cc5baa4  =>  transferTokenByOwner(uint256,address)
c170f1a8  =>  _transferMoney(address,uint256)
21c87c2f  =>  setAssetName(uint256,string)
cdb0e89e  =>  setTokenName(uint256,string)
dacc6e3e  =>  setTokenAssetType(uint256,uint256)
23eac4bd  =>  setTokenOwner(uint256,address)
162094c4  =>  setTokenURI(uint256,string)
a7ccabdf  =>  setNFTContract(address)
353ed9a8  =>  getExistsName(string)
b3cde590  =>  getExistsAssetId(string)
1b1b905f  =>  getSupplyByType(uint256)
24fc099b  =>  getAssetName(uint256)
2eb71f1e  =>  getFullAssetIdAndTokenId()
2129904b  =>  getFullDataByType(uint256)
1782c356  =>  getFullDataByOwner(address)
e022f02b  =>  getFullDataByTokenId(uint256)
e7572230  =>  getPrice(uint256)
2b1f4ad6  =>  getNFTContract()
```

Automatic general report

Files Description Table

File Name	SHA-1 Hash
/Users/macbook/Desktop/smart contracts/MarketplaceAssets.sol	74ac554ee1be75dca53eec2388b9e15a3a998f82
/Users/macbook/Desktop/smart contracts/MetaverserNFT.sol	80df2982f170ac08a12306ae62f7c3b22d0b32bd

Contracts Description Table

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
MarketplaceAssets	Implementation	IMarketplaceAssets, Ownable, ERC721Holder		
└	<Constructor>	Public ❗	🛑	NO❗
└	deposit	Public ❗	💵	NO❗
└	addNewAssets	Public ❗	🛑	onlyOwner
└	addBatchAssets	External ❗	🛑	onlyOwner
└	buyAsset	Public ❗	💵	NO❗
└	synctContractData	Private 🔐	🛑	
└	addAssetToMarket	Public ❗	🛑	tokenOwnerAccess
└	removeAssetFromMarket	Public ❗	🛑	tokenOwnerAccess
└	transferToken	Public ❗	💵	tokenOwnerAccess
└	transferTokenByOwner	Public ❗	🛑	onlyOwner tokenOwnerAccess
└	_transferMoney	Public ❗	💵	NO❗
└	setAssetName	Public ❗	🛑	onlyOwner
└	setTokenName	Public ❗	🛑	NO❗
└	setTokenAssetType	Public ❗	🛑	onlyOwner
└	setTokenOwner	Public ❗	🛑	onlyOwner
└	setTokenURI	Public ❗	🛑	onlyOwner
└	setNFTContract	Public ❗	🛑	onlyOwner
└	getExistsName	Public ❗		NO❗
└	getExistsAssetId	Public ❗		NO❗
└	getSupplyByType	Public ❗		NO❗
└	getAssetName	Public ❗		NO❗
└	getFullAssetIdAndTokenId	Public ❗		NO❗
└	getFullDataByType	Public ❗		NO❗
└	getFullDataByOwner	Public ❗		NO❗
└	getFullDataByTokenId	Public ❗		NO❗
└	getPrice	Public ❗		NO❗
└	getNFTContract	Public ❗		NO❗
IMarketplaceAssets	Interface			
MetaverserNFT	Implementation	IMarketplaceAssets, ERC721URIStorage, ERC721Enumerable, Ownable, ERC721Holder		
└	<Constructor>	Public ❗	🛑	ERC721
└	_beforeTokenTransfer	Internal 🔒	🛑	
└	safeTransferFrom	Public ❗	🛑	onlyAccessable

└	safeTransferFrom	Public ❗	🛑	onlyAccessable
└	transferFrom	Public ❗	🛑	onlyAccessable
└	createToken	Public ❗	🛑	NO❗
└	_burn	Internal 🔒	🛑	
└	supportsInterface	Public ❗		NO❗
└	tokenURI	Public ❗		NO❗
└	getGameAssetsByTokenId	Public ❗		NO❗
└	isContract	Private 🔐		
└	setaccessListAddress	Public ❗	🛑	onlyOwner
└	setTokenName	Public ❗	🛑	onlyAccessable
└	setTokenAssetType	Public ❗	🛑	onlyAccessable
└	setTokenURI	Public ❗	🛑	onlyAccessable

Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

Conclusion

The contracts are written systematically. Team found no critical issues. So, it is good to go for production.

Since possible test cases can be unlimited and developer level documentation (code flow diagram with function level description) not provided, for such an extensive smart contract 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.

Security state of the reviewed contract is “Well secured”.

- ✔ No volatile code.
- ✔ Not many high severity issues were found.

Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice as of the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against the team on the basis of what it says or doesn't say, or how team produced it, and it is important for you to conduct your own independent investigations before making any decisions. team go into more detail on this in the below disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis, and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Saferico and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (Saferico s) owe no duty of care towards you or any other person, nor does Saferico make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and Saferico hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Saferico hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Saferico, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.