SMART CONTRACT AUDIT REPORT

For

RHINO

Prepared By: SFI Team

Prepared for: RidiculousRhinos

Prepared on: 18/10/2021

Table of Content

• 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) 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.

- # Overview of the audit

The project has 1 file. It contains approx 348 lines of Solidity code. Most of the functions and state variables are well commented on using the Nat spec documentation, but that does not create any vulnerability.

- # Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices automatically.

1. Unit tests passing.

2. Compilator warnings;

3. Race Conditions. Reentrancy. Cross-function Race Conditions. Pitfalls in Race Condition solutions;

4. Possible delays in data delivery;

5. Transaction-Ordering Dependence (front running);

6. Timestamp Dependence;

7. Integer Overflow and Underflow;

8. DoS with (unexpected) Revert;

9. DoS with Block Gas Limit;

10. Call Depth Attack. Not relevant in modern ethereum network

11. Methods execution permissions;

12. Oracles calls;

13. Economy model. It’s important to forecast scenarios when a user is provided with additional economic motivation or faced with limitations. If application logic is based on incorrect economy model, the application will not function correctly and participants will incur financial losses. This type of issue is most often found in bonus rewards systems.
14. The impact of the exchange rate on the logic;
15. Private user data leaks.

- # Good things in smart contract

Compiler version is static: -
=> In this file, you have put “pragma solidity 0.8.0;” which is a good way to define the compiler version.

```
pragma solidity 0.8.0;
```

- **Openzeppelin library: -**
  - RHINO is using openzeppelin library it is a good thing. This protects RHINO from underflow and overflow attacks.

```
import "@openzeppelin/contracts/token/ERC721/ERC721.sol";

import
"@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol";

import
"@openzeppelin/contracts/token/ERC721/extensions/ERC721Burnable.sol";

import "@openzeppelin/contracts/access/Ownable.sol";

import "@openzeppelin/contracts/utils/Address.sol";

import "@openzeppelin/contracts/utils/Counters.sol";
```

- **Good required condition in functions: -**
  - Here you are Setting mint price.

```
function setMintPrice(uint256 newPrice) public
   onlyOwner {
        mintPrice = newPrice;
       }
```

  - Here you are Adding address to presale list.

```
function addToAllowList(address[] calldata
   addresses) external onlyOwner {
        for (uint256 i = 0; i < addresses.length;
   i++) {
             require(addresses[i] != address(0),
   "Can't add the null address");

             _allowList[addresses[i]] = true;
             /**
              * @dev We don't want to reset
   _allowListClaimed count
              * if we try to add someone more than
   once.
              */
```

```
                _allowListClaimed[addresses[i]] > 0
                    ? _allowListClaimed[addresses[i]]
                    : 0;
            }
        }
```

- Here you are Checking if the address has been added to the presale list

```
function onAllowList(address addr) external view
returns (bool) {
        return _allowList[addr];
    }
```

- Here you are checking Mint to giveaway winners address.

```
function giveawayMint(uint256 reservedAmount, address mintAddress)
        public
        onlyOwner
    {
        uint256 supply = _tokenIdCounter.current();
        for (uint256 i = 1; i <= reservedAmount; i++) {
            _safeMint(mintAddress, supply + i);
            _tokenIdCounter.increment();
        }
    }
```

- Here you are checking View amount claimed by address.

```
function allowListClaimedBy(address owner)
        external
        view
        returns (uint256)
    {
        require(owner != address(0), "Zero address not on
 Allow List");


        return _allowListClaimed[owner];
    }
```

• Critical vulnerabilities found in the contract

There not Critical severity vulnerabilities found

• High vulnerabilities found in the contract

There not High severity vulnerabilities found

• Medium vulnerabilities found in the contract

There not Medium severity vulnerabilities found

• Low severity vulnerabilities found

#Check-effects-interaction:

```
function functionCallWithValue(address target,
        bytes memory data,
        uint256 value,
        string memory errorMessage
    ) internal returns (bytes memory) {
        require(address(this).balance >= value, "Address: insufficient bala
        require(isContract(target), "Address: call to non-contract");

        (bool success, bytes memory returndata) = target.call{value: value}
        return verifyCallResult(success, returndata, errorMessage);
    }
```

In detail (@openzeppelin/contracts/utils/Address.sol)file

Potential violation of Checks-Effects-Interaction pattern in Address.functionCallWithValue(address,bytes,uint256,string): Could potentially lead to re-entrancy vulnerability.
For more reading:
https://docs.soliditylang.org/en/v0.8.0/security-considerations.html#re-entrancy

#Inline assembly

```
assembly {
            size := extcodesize(account)
        }
and
assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
```

In detail (@openzeppelin/contracts/utils/Address.sol)file

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.
For more reading:
https://docs.soliditylang.org/en/v0.8.0/assembly.html

- **Notes**

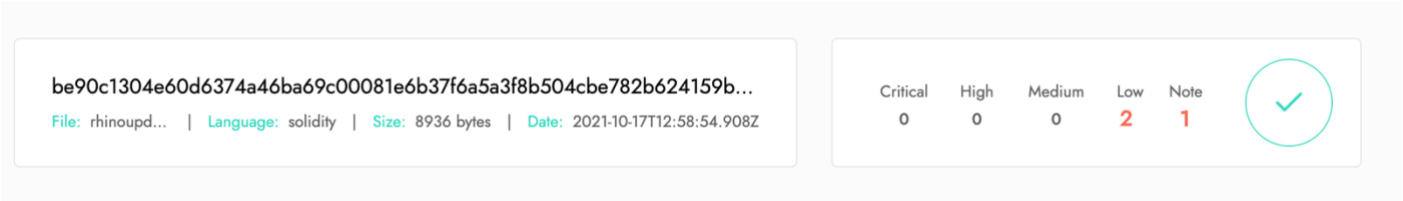
#Call

```
   (bool success, ) = recipient.call{value: amount}("");
(bool success, bytes memory returndata) = target.call{value:
value}(data);
        (bool success, bytes memory returndata) =
target.delegatecall(data);
```

In detail (@openzeppelin/contracts/utils/Address.sol)file
Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.
For more reading:
https://docs.soliditylang.org/en/v0.8.0/control-structures.html#external-function-calls

Testing proves:

1- Check for security



2- SOLIDITY STATIC ANALYSIS

SOLIDITY STATIC ANALYSIS

Select all | Autorun | Run

Security

Select Security

- **Transaction origin:** 'tx.origin' used
- **Check-effects-interaction:** Potential reentrancy bugs
- **Inline assembly:** Inline assembly used
- **Block timestamp:** Can be influenced by miners
- **Low level calls:** Should only be used by experienced devs
- **Block hash:** Can be influenced by miners
- **Selfdestruct:** Contracts using destructed contract can be broken

Gas & Economy

Select Gas & Economy

- **Gas costs:** Too high gas requirement of functions
- **This on local calls:** Invocation of local functions via 'this'
- **Delete dynamic array:** Use require/assert to ensure complete deletion
- **For loop over dynamic array:** Iterations depend on dynamic array's size
- **Ether transfer in loop:** Transferring Ether in a for/while/do-while loop

SOLIDITY STATIC ANALYSIS

ERC

Select ERC

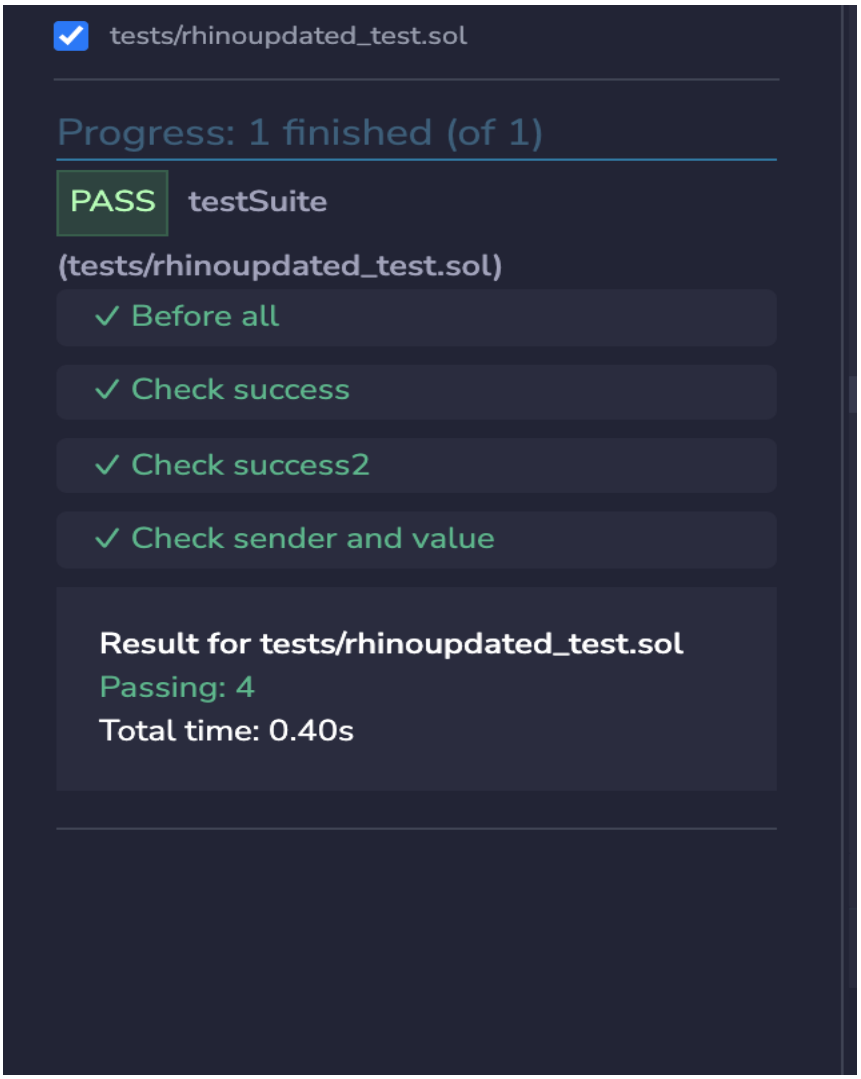
- **ERC20:** 'decimals' should be 'uint8'

Miscellaneous

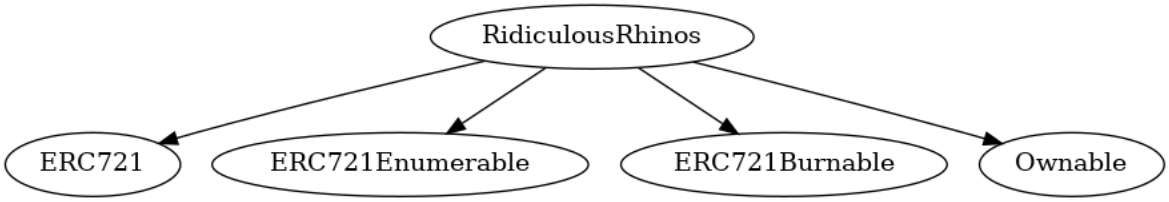
Select Miscellaneous

- **Constant/View/Pure functions:** Potentially constant/view/pure functions
- **Similar variable names:** Variable names are too similar
- **No return:** Function with 'returns' not returning
- **Guard conditions:** Ensure appropriate use of require/assert
- **Result not used:** The result of an operation not used
- **String length:** Bytes length != String length
- **Delete from dynamic array:** 'delete' leaves a gap in array
- **Data truncated:** Division on int/uint values truncates the result

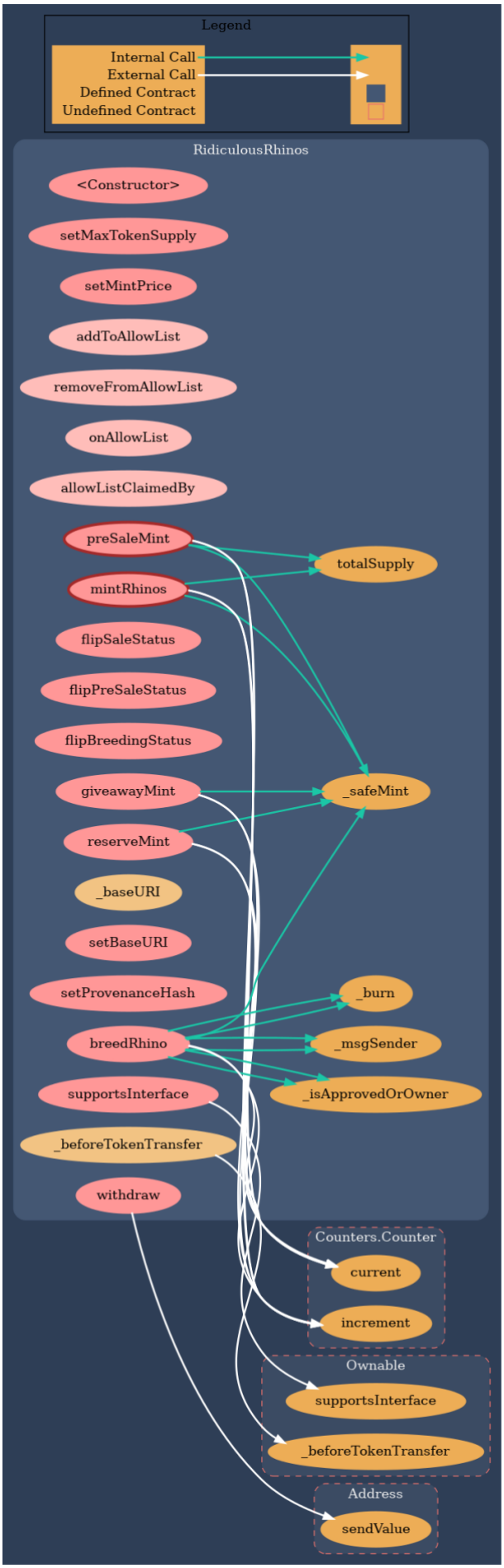
3- SOLIDITY UNIT TESTING



4- Inheritance graph



5- Call graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
RidiculousRhinos
<Constructor>
setMaxTokenSupply
setMintPrice
addToAllowList
removeFromAllowList
onAllowList
allowListClaimedBy
preSaleMint
totalSupply
mintRhinos
flipSaleStatus
flipPreSaleStatus
flipBreedingStatus
giveawayMint
_safeMint
reserveMint
_baseURI
setBaseURI
setProvenanceHash
_burn
breedRhino
_msgSender
supportsInterface
_isApprovedOrOwner
_beforeTokenTransfer
withdraw
Counters.Counter
current
increment
Ownable
supportsInterface
_beforeTokenTransfer
Address
sendValue

- # Automatic general report

Files Description Table

File Name	SHA-1 Hash
/Users/macbook/Desktop/smart contracts/rhinoupdated.sol	e8cc8c396ca9f423258b2f094344d1d50e2006ca

Contracts Description Table

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
RidiculousRhinos	Implementation	ERC721, ERC721Enumerable, ERC721Burnable, Ownable		
└	<Constructor>	Public ❗	🛑	ERC721
└	setMaxTokenSupply	Public ❗	🛑	onlyOwner
└	setMintPrice	Public ❗	🛑	onlyOwner
└	addToAllowList	External ❗	🛑	onlyOwner
└	removeFromAllowList	External ❗	🛑	onlyOwner
└	onAllowList	External ❗	NO❗	
└	allowListClaimedBy	External ❗	NO❗	
└	reserveMint	Public ❗	🛑	onlyOwner
└	giveawayMint	Public ❗	🛑	onlyOwner
└	flipSaleStatus	Public ❗	🛑	onlyOwner
└	flipPreSaleStatus	Public ❗	🛑	onlyOwner
└	flipBreedingStatus	Public ❗	🛑	onlyOwner
└	mintRhinos	Public ❗	💵	NO❗
└	preSaleMint	Public ❗	💵	NO❗
└	_baseURI	Internal 🔒		
└	setBaseURI	Public ❗	🛑	onlyOwner
└	setProvenanceHash	Public ❗	🛑	onlyOwner
└	_beforeTokenTransfer	Internal 🔒	🛑	
└	supportsInterface	Public ❗	NO❗	
└	breedRhino	Public ❗	🛑	NO❗
└	withdraw	Public ❗	🛑	onlyOwner

Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

- # Summary of the Audit

According to automatically test, the customer`s solidity smart contract is **Very Secured**.

The general overview is presented in the Project Information section and all issues found are located in the audit overview section.

The test found 0 critical, 0 high, 0 medium, 2 low issues, and 1 notes.