Smart Contract
Security Audit
V1

Unusual Guests Smart Contract

7/4/2022

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

- **Platform**: Ethereum
- **Contract Address**: 0xb52f1B5B298680ca2BCAfD72DE17fCF223FC84ea
- **Code:**

https://rinkeby.etherscan.io/address/0xb52f1b5b298680ca2bcafd72de17fcf223fc84ea#code

NFT Information

- Name: UG
- Total Supply: 4567
- Holders:
- Total transactions:

Contracts address deployed to test net (ETH)

Unusual Guests Smart contract on ETH test net to test write functions by the auditor.

https://rinkeby.etherscan.io/address/0xcc1a754478bd5b95654b4acb91ae654dd5a6d587

Executive Summary

According to our assessment, the customer`s solidity smart contract is **Well-Secured**. Because the team fix all high and low issues.

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, 1 high, 0 medium, 3 low, 0 very low-level issues and 1 note in all solidity files of the contract

The files:

UnusualGuests.sol

File and Function Level Report

File in Scope:

Contract Name	SHA 256 hash	Contract Address
UnusualGuests.sol	**23e28949d46a469bfaf7b1a29b998bff37f581023f2713ae3199acd7a946c1be**	0xcc1a754478bd5B95654B4AcB91Ae654Dd5A6d587

- Contract: UnusualGuests
- Inherit: ERC721Enumerable, Ownable
- Observation: All passed including security check
- Test Report: passed
- Score: passed
- Conclusion: passed

Function	Test Result	Type / Return Type	Score
name	✓	Read / public	**Passed**
symbol	✓	Read / public	**Passed**
maximumMintSupply	✓	Read / public	**Passed**
supportsInterface	✓	Read / public	**Passed**
mintPrice	✓	Read / public	**Passed**
balanceOf	✓	Read / public	**Passed**
Owner	✓	Read / public	**Passed**
getContractOwner	✓	Read / public	**Passed**
isClosedMintForever	✓	Read / public	**Passed**
getApprovedForAll	✓	Read / public	**Passed**
ownerOf	✓	Read / public	**Passed**
getApproved	✓	Read / public	**Passed**

numAvailableToMint	✓	Read / public	**Passed**
tokenByIndex	✓	Read / public	**Passed**
tokenOfOwnerByIndex	✓	Read / public	**Passed**
getTotalSupply	✓	Read / public	**Passed**
walletOfOwner	✓	Read / public	**Passed**
isMintActive	✓	Read / public	**Passed**
maximumAllowedTokens PerPurchase	✓	Read / public	**Passed**
isPresaleMintActive	✓	Read / public	**Passed**
tokenURI	✓	Read / public	**Passed**
preSaleMintPrice	✓	Read / public	**Passed**
PROVENANCE	✓	Read / public	**Passed**
totalSupply	✓	Read / public	**Passed**
setMintActive	✓	Write / public	**Passed**
approve	✓	Write / public	**Passed**
safeTransferFrom	✓	Write / public	**Passed**
safeTransferFrom	✓	Write / public	**Passed**
setBaseURI	✓	Write / public	**Passed**
mint	✓	Write / payable	**Passed**
transferOwnership	✓	Write / public	**Passed**
setApprovalForAll	✓	Write / public	**Passed**
transferFrom	✓	Write / public	**Passed**
preSaleMint	✓	Write / payable	**Passed**
setIsClosedMintForever	✓	Write / public	**Passed**
setProvenanceHash	✓	Write / public	**Passed**
renounceOwnership	✓	Write / public	**Passed**
withdraw	✓	Write / payable	**Passed**

reserveToCustomWallet	✓	Write / public	**Passed**
setPreSaleActive	✓	Write / public	**Passed**
addToAllowList	✓	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	Design Logic.	**Passed**
6	Timestamp dependence.	**Passed with Notes**
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**

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:

Possibility of losing all funds

Description
Due to missing or insufficient access controls, malicious parties can withdraw some or all Ether from the contract account. This bug is sometimes caused by unintentionally exposing initialization functions. By wrongly naming a function intended to be a constructor, the constructor code ends up in the runtime byte code and can be called by anyone to re-initialize the contract.

```
function withdraw() external onlyAuthorized {
    uint balance = address(this).balance;
    payable(OtherAddress).transfer(balance * 10000 / 10000);
    payable(owner()).transfer(balance * 0 / 10000);
  }
```

Remediation
Remove this function and add another withdraw function to allow the owner to send the funds to any address he wants.

Status: Closed. Fixed in version 2.

Medium:

No Medium severity vulnerabilities were found

Low:

#Missing zero address validation

Description
When the owner wants to reserve for custom wallet it has to check for the zero address to make, he didn't mint for the burn address. Otherwise, the mint function will act like the burn function.

```
function reserveToCustomWallet(address _walletAddress, uint256 _count) public
onlyAuthorized {
    for (uint256 i = 0; i < _count; i++) {
      emit AssetMinted(totalSupply(), _walletAddress);
      _safeMint(_walletAddress, totalSupply());
    }
  }
```

Remediation

Use the require statement to check for zero addresses.

```
require(recipient!= address(0), "Not Mint for the zero address");
```

Status: Closed. Fixed in version2.

#Pragam version not fixed

Description

It is a good practice to lock the solidity version for a live deployment (use 0.8.2 instead of ^0.8.1). contracts should be deployed with the same compiler version and flags that they have been tested the most with. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may have higher risks of undiscovered bugs. Contracts may also be deployed by others and the pragma indicates the compiler version intended by the original authors.

Remediation

Remove the ^ sign to lock the pragma version.

Status: Closed. Fixed in version2

#Owner privileges (In the period when the owner isn't renounced)

Description

The owner can close mint forever.

```
function setIsClosedMintForever() external onlyAuthorized {
    isClosedMintForever = true;
  }
```

Remediation

Make these functions internal in next version or the team should announce the investors before close mint to give them time if they want to do anything.

P.S: This issue is common to the majority of NFT smart contracts.

Status: Acknowledged.

Very Low:

No Very Low severity vulnerabilities were found.

Notes:

#Missing SPDX-License-Identifier:

Description

Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see https://spdx.org for more information .

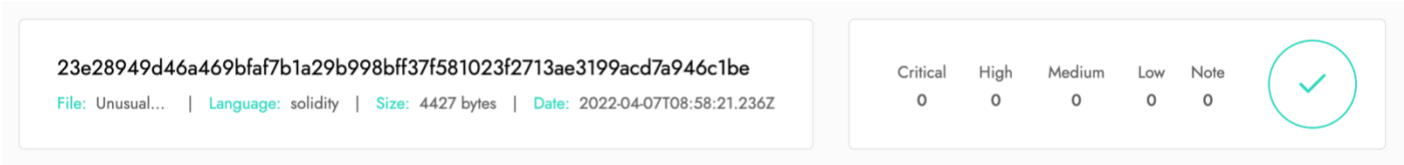
Remediation

Add License Identifier

// SPDX-License-Identifier: UNLICENSE

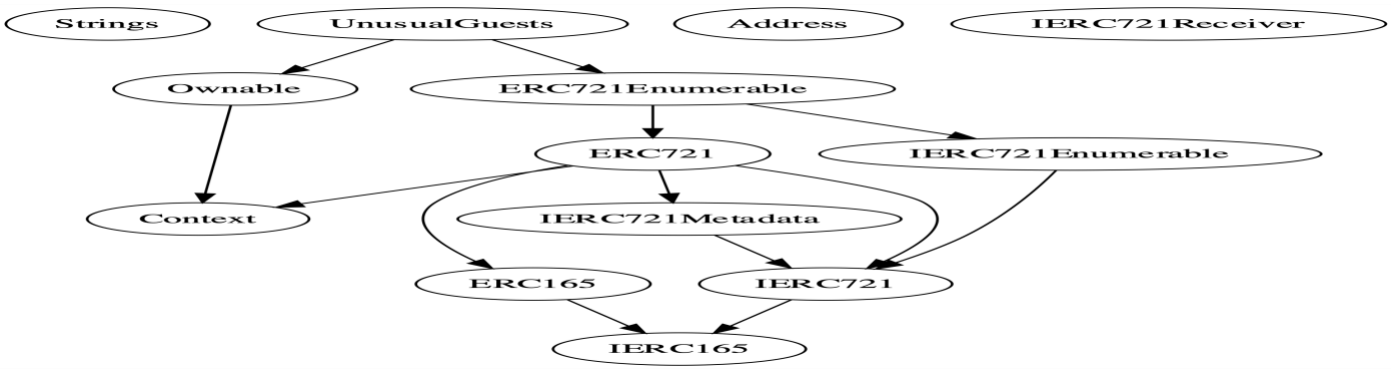
Automatic Testing

1- Check for security

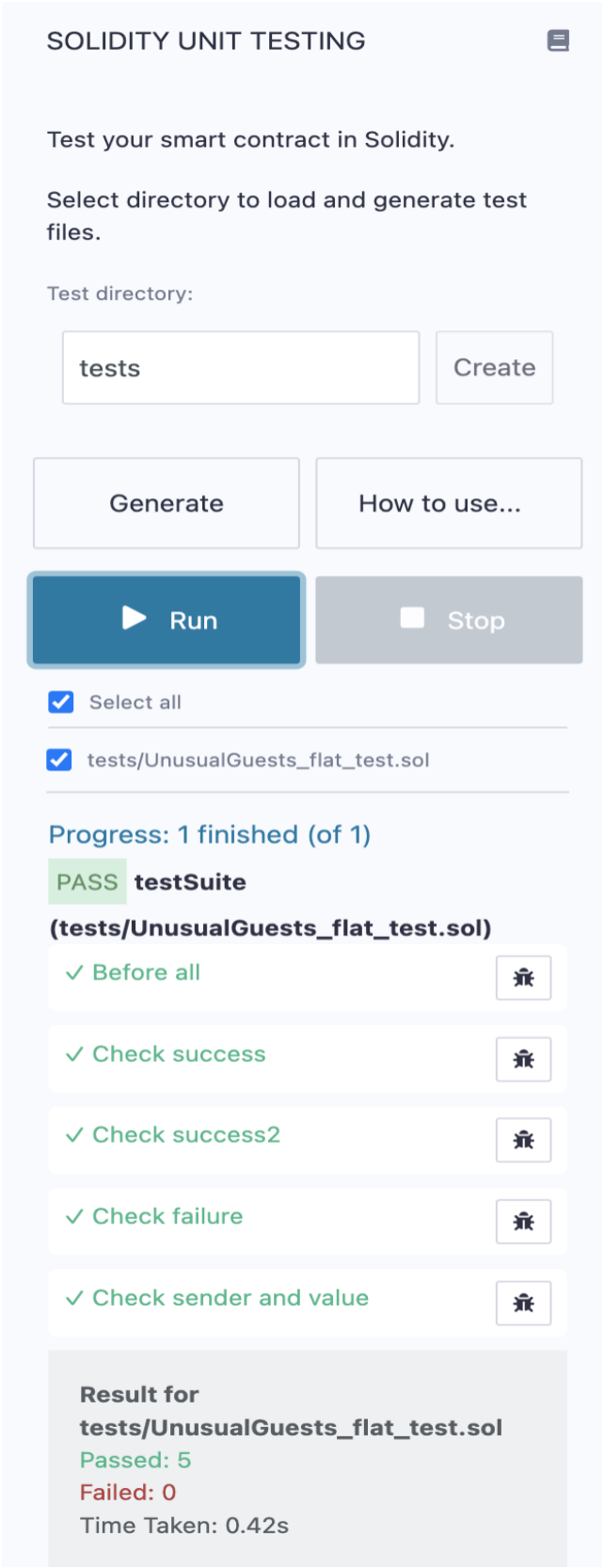


2- SOLIDITY STATIC ANALYSIS

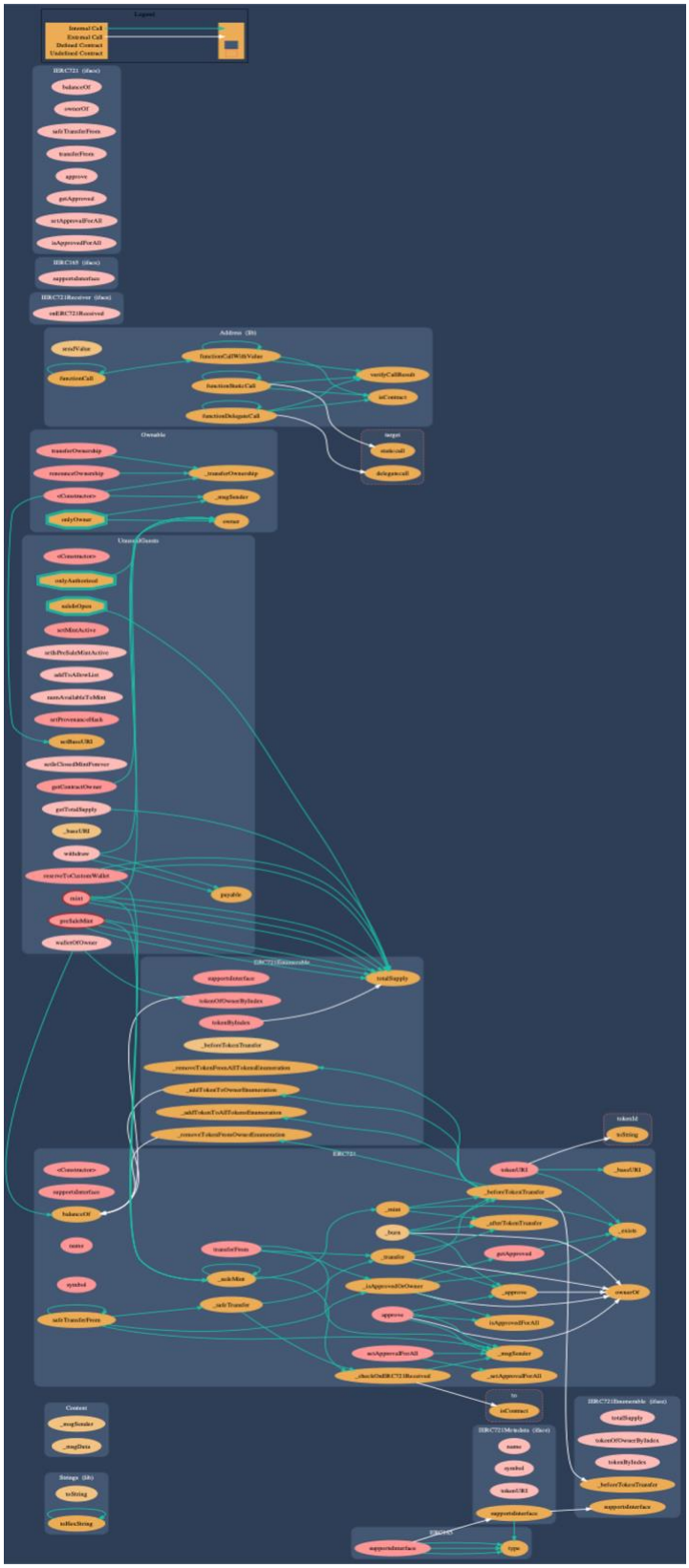
3- Inheritance graph



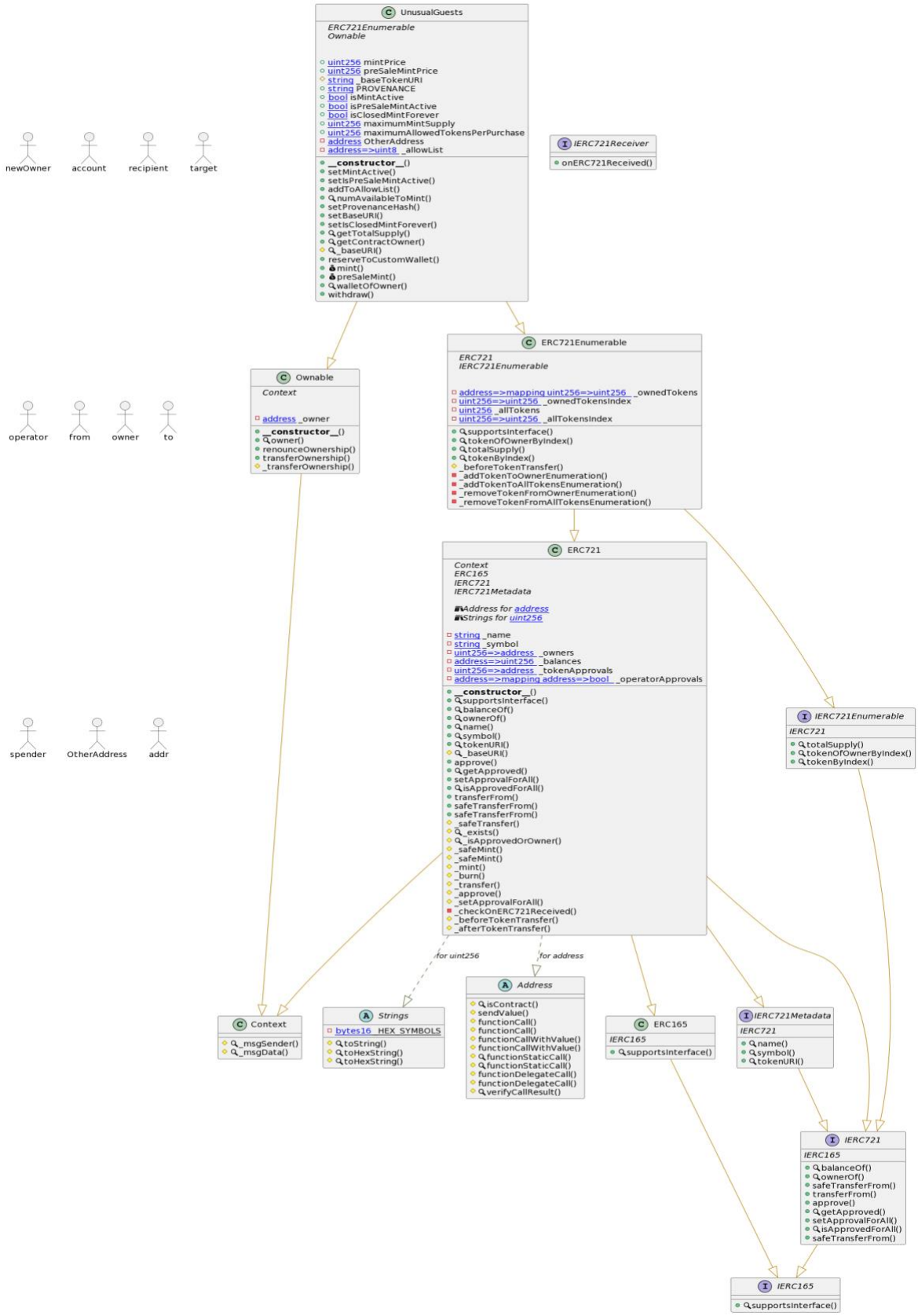
4- SOLIDITY UNIT TESTING



5- Call graph



Unified Modeling Language (UML)



Functions signature

```
Sighash   |   Function Signature
========================
10969523  =>  setProvenanceHash(string)
16279055  =>  isContract(address)
6900a3ae  =>  toString(uint256)
8fba8d5c  =>  toHexString(uint256)
63e1cbea  =>  toHexString(uint256,uint256)
119df25f  =>  _msgSender()
8b49d47e  =>  _msgData()
8da5cb5b  =>  owner()
715018a6  =>  renounceOwnership()
f2fde38b  =>  transferOwnership(address)
d29d44ee  =>  _transferOwnership(address)
24a084df  =>  sendValue(address,uint256)
a0b5ffb0  =>  functionCall(address,bytes)
241b5886  =>  functionCall(address,bytes,string)
2a011594  =>  functionCallWithValue(address,bytes,uint256)
d525ab8a  =>  functionCallWithValue(address,bytes,uint256,string)
c21d36f3  =>  functionStaticCall(address,bytes)
dbc40fb9  =>  functionStaticCall(address,bytes,string)
ee33b7e2  =>  functionDelegateCall(address,bytes)
57387df0  =>  functionDelegateCall(address,bytes,string)
946b5793  =>  verifyCallResult(bool,bytes,string)
150b7a02  =>  onERC721Received(address,address,uint256,bytes)
01ffc9a7  =>  supportsInterface(bytes4)
70a08231  =>  balanceOf(address)
6352211e  =>  ownerOf(uint256)
42842e0e  =>  safeTransferFrom(address,address,uint256)
23b872dd  =>  transferFrom(address,address,uint256)
095ea7b3  =>  approve(address,uint256)
081812fc  =>  getApproved(uint256)
a22cb465  =>  setApprovalForAll(address,bool)
e985e9c5  =>  isApprovedForAll(address,address)
b88d4fde  =>  safeTransferFrom(address,address,uint256,bytes)
18160ddd  =>  totalSupply()
2f745c59  =>  tokenOfOwnerByIndex(address,uint256)
4f6ccce7  =>  tokenByIndex(uint256)
06fdde03  =>  name()
95d89b41  =>  symbol()
c87b56dd  =>  tokenURI(uint256)
743976a0  =>  _baseURI()
24b6b8c0  =>  _safeTransfer(address,address,uint256,bytes)
f8e76cc0  =>  _exists(uint256)
4cdc9549  =>  _isApprovedOrOwner(address,uint256)
b3e1c718  =>  _safeMint(address,uint256)
6a4f832b  =>  _safeMint(address,uint256,bytes)
4e6ec247  =>  _mint(address,uint256)
9b1f9e74  =>  _burn(uint256)
30e0789e  =>  _transfer(address,address,uint256)
7b7d7225  =>  _approve(address,uint256)
8c4e3f32  =>  _setApprovalForAll(address,address,bool)
1fd01de1  =>  _checkOnERC721Received(address,address,uint256,bytes)
cad3be83  =>  _beforeTokenTransfer(address,address,uint256)
```

```
8f811a1c  =>  _afterTokenTransfer(address,address,uint256)
69025b5f  =>  _addTokenToOwnerEnumeration(address,uint256)
e03d890b  =>  _addTokenToAllTokensEnumeration(uint256)
68df0d53  =>  _removeTokenFromOwnerEnumeration(address,uint256)
4cbb4a0a  =>  _removeTokenFromAllTokensEnumeration(uint256)
ee1cc944  =>  setMintActive(bool)
3cfac570  =>  setIsPreSaleMintActive(bool)
01596309  =>  addToAllowList(address[],uint8)
c04a2836  =>  numAvailableToMint(address)
55f804b3  =>  setBaseURI(string)
932ecebc  =>  setIsClosedMintForever()
c4e41b22  =>  getTotalSupply()
442890d5  =>  getContractOwner()
77b501b9  =>  reserveToCustomWallet(address,uint256)
40c10f19  =>  mint(address,uint256)
f9f67d6c  =>  preSaleMint(uint8)
438b6300  =>  walletOfOwner(address)
3ccfd60b  =>  withdraw()
```

Automatic general report

Files Description Table

File Name	SHA-1 Hash
/Users/macbook/Desktop/smart contracts/UnusualGuests.sol	19f55f397bde9b7c438f35107a83e9fb8cccfa59

Contracts Description Table

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
Strings	Library			
└	toString	Internal 🔒		
└	toHexString	Internal 🔒		
└	toHexString	Internal 🔒		
Context	Implementation			
└	_msgSender	Internal 🔒		
└	_msgData	Internal 🔒		
Ownable	Implementation	Context		
└	<Constructor>	Public ❗	🛑	NO❗
└	owner	Public ❗		NO❗
└	renounceOwnership	Public ❗	🛑	onlyOwner
└	transferOwnership	Public ❗	🛑	onlyOwner
└	_transferOwnership	Internal 🔒	🛑	
Address	Library			
└	isContract	Internal 🔒		
└	sendValue	Internal 🔒	🛑	
└	functionCall	Internal 🔒	🛑	
└	functionCall	Internal 🔒	🛑	
└	functionCallWithValue	Internal 🔒	🛑	
└	functionCallWithValue	Internal 🔒	🛑	
└	functionStaticCall	Internal 🔒		
└	functionStaticCall	Internal 🔒		
└	functionDelegateCall	Internal 🔒	🛑	
└	functionDelegateCall	Internal 🔒	🛑	
└	verifyCallResult	Internal 🔒		
IERC721Receiver	Interface			
└	onERC721Received	External ❗	🛑	NO❗
IERC165	Interface			
└	supportsInterface	External ❗		NO❗
ERC165	Implementation	IERC165		

└	supportsInterface	Public ❗		NO❗
IERC721	Interface	IERC165		
└	balanceOf	External ❗		NO❗
└	ownerOf	External ❗		NO❗
└	safeTransferFrom	External ❗	🛑	NO❗
└	transferFrom	External ❗	🛑	NO❗
└	approve	External ❗	🛑	NO❗
└	getApproved	External ❗		NO❗
└	setApprovalForAll	External ❗	🛑	NO❗
└	isApprovedForAll	External ❗		NO❗
└	safeTransferFrom	External ❗	🛑	NO❗
IERC721Enumerable	Interface	IERC721		
└	totalSupply	External ❗		NO❗
└	tokenOfOwnerByIndex	External ❗		NO❗
└	tokenByIndex	External ❗		NO❗
IERC721Metadata	Interface	IERC721		
└	name	External ❗		NO❗
└	symbol	External ❗		NO❗
└	tokenURI	External ❗		NO❗
ERC721	Implementation	Context, ERC165, IERC721, IERC721Metadata		
└	<Constructor>	Public ❗	🛑	NO❗
└	supportsInterface	Public ❗		NO❗
└	balanceOf	Public ❗		NO❗
└	ownerOf	Public ❗		NO❗
└	name	Public ❗		NO❗
└	symbol	Public ❗		NO❗
└	tokenURI	Public ❗		NO❗
└	_baseURI	Internal 🔒		
└	approve	Public ❗	🛑	NO❗
└	getApproved	Public ❗		NO❗
└	setApprovalForAll	Public ❗	🛑	NO❗
└	isApprovedForAll	Public ❗		NO❗
└	transferFrom	Public ❗	🛑	NO❗
└	safeTransferFrom	Public ❗	🛑	NO❗
└	safeTransferFrom	Public ❗	🛑	NO❗
└	_safeTransfer	Internal 🔒	🛑	
└	_exists	Internal 🔒		
└	_isApprovedOrOwner	Internal 🔒		
└	_safeMint	Internal 🔒	🛑	
└	_safeMint	Internal 🔒	🛑	
└	_mint	Internal 🔒	🛑	
└	_burn	Internal 🔒	🛑	
└	_transfer	Internal 🔒	🛑	
└	_approve	Internal 🔒	🛑	
└	_setApprovalForAll	Internal 🔒	🛑	
└	_checkOnERC721Received	Private 🔐	🛑	
└	_beforeTokenTransfer	Internal 🔒	🛑	
└	_afterTokenTransfer	Internal 🔒	🛑	
ERC721Enumerable	Implementation	ERC721, IERC721Enumerable		
└	supportsInterface	Public ❗		NO❗

└	tokenOfOwnerByIndex	Public ❗		NO❗
└	totalSupply	Public ❗		NO❗
└	tokenByIndex	Public ❗		NO❗
└	_beforeTokenTransfer	Internal 🔒	🛑	
└	_addTokenToOwnerEnumeration	Private 🔐	🛑	
└	_addTokenToAllTokensEnumeration	Private 🔐	🛑	
└	_removeTokenFromOwnerEnumeration	Private 🔐	🛑	
└	_removeTokenFromAllTokensEnumeration	Private 🔐	🛑	
UnusualGuests	Implementation	ERC721Enumerable, Ownable		
└	<Constructor>	Public ❗	🛑	ERC721
└	setMintActive	Public ❗	🛑	onlyAuthorized
└	setIsPreSaleMintActive	External ❗	🛑	onlyAuthorized
└	addToAllowList	External ❗	🛑	onlyAuthorized
└	numAvailableToMint	External ❗		NO❗
└	setProvenanceHash	Public ❗	🛑	onlyAuthorized
└	setBaseURI	Public ❗	🛑	onlyAuthorized
└	setIsClosedMintForever	External ❗	🛑	onlyAuthorized
└	getTotalSupply	External ❗		NO❗
└	getContractOwner	Public ❗		NO❗
└	_baseURI	Internal 🔒		
└	reserveToCustomWallet	Public ❗	🛑	onlyAuthorized
└	mint	Public ❗	💸	saleIsOpen
└	preSaleMint	Public ❗	💸	saleIsOpen
└	walletOfOwner	External ❗		NO❗
└	withdraw	External ❗	🛑	onlyAuthorized

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.