SMART CONTRACT AUDIT REPORT

For

CA

Prepared By: SFI Team

Prepared for: Crypto Accessories

Prepared on: 31/10/2021

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 498 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 0.8.4;” which is a good way to define the compiler version.

```
pragma solidity 0.8.4;
```

- **Address library: -**
  - CA is using Address library it is a good thing.

```
library Address {
    function isContract(address account) internal view returns (bool) {
        uint256 size;
        assembly {
            size := extcodesize(account)}
        return size > 0;
    function sendValue(address payable recipient, uint256 amount)
internal {
        require(address(this).balance >= amount, "Address: insufficient
balance");
        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Address: unable to send value, recipient may
have reverted");
    }
```

- **Good required condition in functions: -**
  - Here you are checking transferOwnership function

```
function transferOwnership(address newOwner) public
  virtual onlyOwner {
        require(newOwner != address(0), "Ownable:
  new owner is the zero address");
        _setOwner(newOwner);
      }
```

  - Here you are add to white list function

```
function addToWhitelist(address[] memory newusers, uint
numeration) external { // numeration of whitelist
        for(uint256 i=0; i < newusers.length; i++){
            if (numeration == 1) {
                require(whitelist2[newusers[i]] == 0,
"Whitelist: the user is already on the whitelist2"); //
                whitelist1[newusers[i]] = 1;
            } else if (numeration == 2) {
                require(whitelist1[newusers[i]] == 0,
"Whitelist: the user is already on the whitelist1");
                whitelist2[newusers[i]] = 1;}}}
```

- Here you are Using interface IERC165

```
interface IERC1155 is IERC165{
    event TransferSingle(address indexed operator,
  address indexed from, address indexed to, uint256
  id, uint256 value);
    event TransferBatch(address indexed
  operator,address indexed from,address indexed
  to,uint256[] ids,uint256[] values);
    event ApprovalForAll(address indexed account,
  address indexed operator, bool approved);
    event URI(string value, uint256 indexed id);
    function balanceOf(address account, uint256 id)
  external view returns (uint256);
    function balanceOfBatch(address[] calldata
  accounts, uint256[] calldata ids) external view
  returns (uint256[] memory);
    function setApprovalForAll(address operator,
  bool approved) external;
    function isApprovedForAll(address account,
  address operator) external view returns (bool);
    function safeTransferFrom(address from,address
  to,uint256 id,uint256 amount,bytes calldata data)
  external;
    function safeBatchTransferFrom(address
  from,address to,uint256[] calldata ids,uint256[]
  calldata amounts,bytes calldata data) external;
}
```

- Here you are Using mint function

```
function mint(
        address _to,
        uint256 _quantity
    ) public payable saleEnabled {
        require( _price * _quantity <= msg.value,
  "Need more money to buy tokens");
        require( _minted + _quantity <=
  TOTAL_TOKENS, "Max tokens reached");
        if (whitelist1[_to] == 1) {
            require( _balances[1][_to] + _quantity
  <= MAX_TOKENS_PER_ADDR1LIST, "Max tokens per
  address reached");
        } else if (whitelist2[_to] == 1) {
            require( _balances[1][_to] + _quantity
  <= MAX_TOKENS_PER_ADDR2LIST, "Max tokens per
  address reached");}
        _mint(_to, 1, _quantity, "");
        uint256 half_amount = msg.value / 2;
        payable(_owner1).transfer(half_amount)}
```

• 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

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/v0.8.4/security-considerations.html#re-entrancy

#Inline assembly

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

In detail

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/v0.8.4/assembly.html

#For loop over dynamic array:

```
for (uint256 i = 0; i < accounts.length; ++i) {
            batchBalances[i] = balanceOf(accounts[i], ids[i]);
        }
for(uint256 i=0; i < newusers.length; i++){
            if (numeration == 1) {
                require(whitelist2[newusers[i]] == 0, "Whitelist: the
user is already on the whitelist2"); //
                whitelist1[newusers[i]] = 1;
            } else if (numeration == 2) {
                require(whitelist1[newusers[i]] == 0, "Whitelist: the
user is already on the whitelist1");
                whitelist2[newusers[i]] = 1;
            }
        }
for(uint256 i=0; i < newusers.length; i++){
            if (numeration == 1) {
                whitelist1[newusers[i]] = 0;
            } else if (numeration == 2) {
                whitelist2[newusers[i]] = 0;
            }
        }
```

In detail
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.

https://docs.soliditylang.org/en/v0.8.4/security-considerations.html#gas-limit-and-loops

- **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
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/v0.8.4/control-structures.html#external-function-calls

#NO Return

```
function supportsInterface(bytes4
interfaceId) external view returns (bool);


    function balanceOf(address account,
uint256 id) external view returns (uint256);

 function balanceOfBatch(address[] calldata
accounts, uint256[] calldata ids) external
view  returns (uint256[] memory);

 function isApprovedForAll(address account,
address operator) external view returns
(bool);

 function isApprovedForAll(address account,
address operator) external view returns
(bool);
```

In detail
IERC1155.balanceOfBatch(address[],uint256[]): Defines a return type but never explicitly returns a value.

Testing proves:

1- Check for security

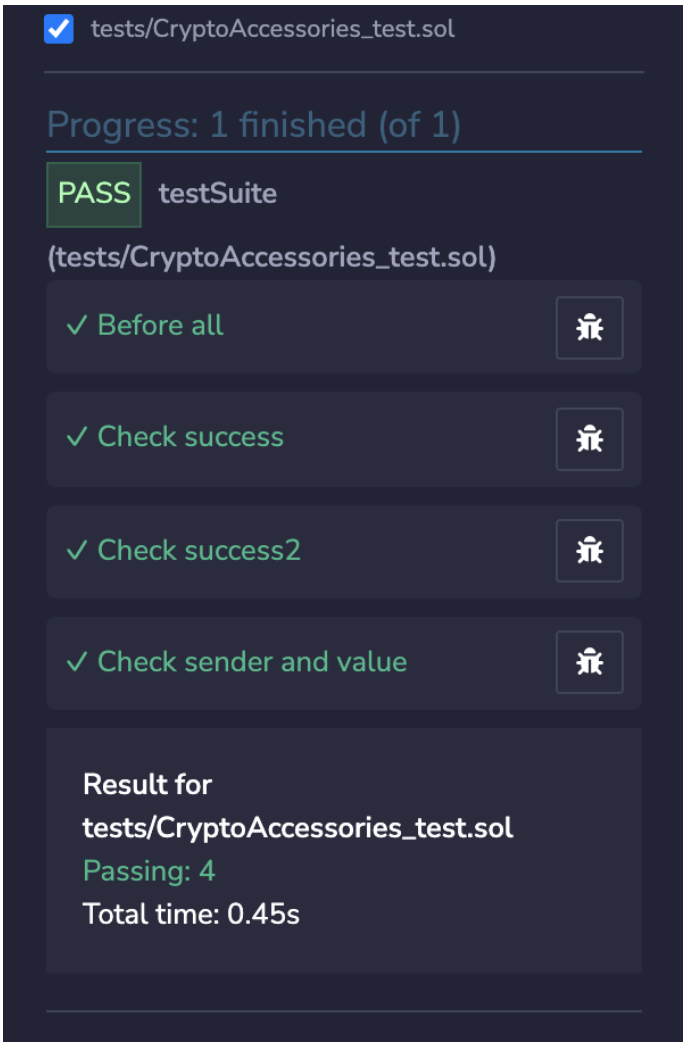
8cd50879677d9222e761a879a1fb21520f7cdc4189ed8502d4c7dd66f7d703db

File: CryptoA... | Language: solidity | Size: 17980 bytes | Date: 2021-10-31T12:54:12.034Z

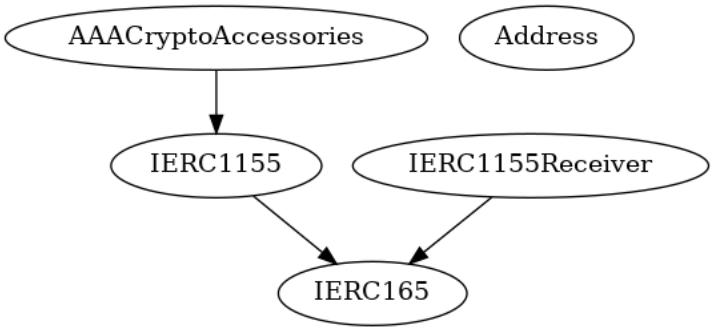
Critical	High	Medium	Low	Note
0	0	0	3	2

2- SOLIDITY STATIC ANALYSIS

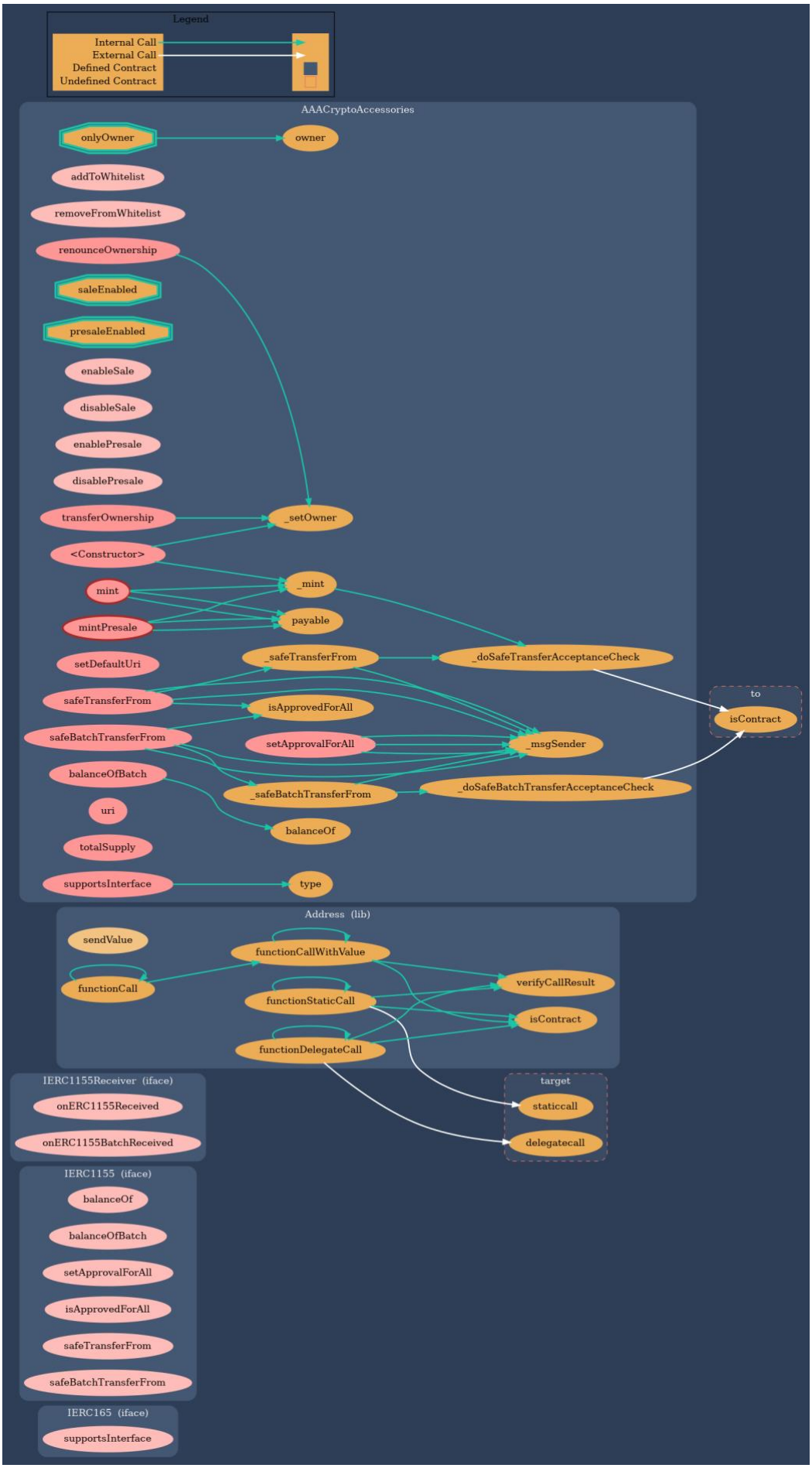
3- SOLIDITY UNIT TESTING



4- Inheritance graph



5- Call graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
AAACryptoAccessories
onlyOwner
owner
addToWhitelist
removeFromWhitelist
renounceOwnership
saleEnabled
presaleEnabled
enableSale
disableSale
enablePresale
disablePresale
transferOwnership
_setOwner
<Constructor>
mint
_mint
payable
mintPresale
setDefaultUri
_safeTransferFrom
_doSafeTransferAcceptanceCheck
safeTransferFrom
isApprovedForAll
to
isContract
safeBatchTransferFrom
setApprovalForAll
_msgSender
balanceOfBatch
_safeBatchTransferFrom
_doSafeBatchTransferAcceptanceCheck
uri
balanceOf
totalSupply
supportsInterface
type
Address (lib)
sendValue
functionCallWithValue
functionCall
verifyCallResult
functionStaticCall
isContract
functionDelegateCall
target
staticcall
delegatecall
IERC1155Receiver (iface)
onERC1155Received
onERC1155BatchReceived
IERC1155 (iface)
balanceOf
balanceOfBatch
setApprovalForAll
isApprovedForAll
safeTransferFrom
safeBatchTransferFrom
IERC165 (iface)
supportsInterface

Automatic general report

Files Description Table

File Name	SHA-1 Hash
/Users/macbook/Desktop/smart contracts/CryptoAccessories.sol	3fe25c88e2bab881f56331b48c7b3972f659681c

Contracts Description Table

Contract	Type	Bases		
	└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
IERC165	Interface			
	└	supportsInterface	External ❗		NO❗
IERC1155	Interface	IERC165		
	└	balanceOf	External ❗		NO❗
	└	balanceOfBatch	External ❗		NO❗
	└	setApprovalForAll	External ❗	🛑	NO❗
	└	isApprovedForAll	External ❗		NO❗
	└	safeTransferFrom	External ❗	🛑	NO❗
	└	safeBatchTransferFrom	External ❗	🛑	NO❗
IERC1155Receiver	Interface	IERC165		
	└	onERC1155Received	External ❗	🛑	NO❗
	└	onERC1155BatchReceived	External ❗	🛑	NO❗
Address	Library			
	└	isContract	Internal 🔒		
	└	sendValue	Internal 🔒	🛑	
	└	functionCall	Internal 🔒	🛑	
	└	functionCall	Internal 🔒	🛑	
	└	functionCallWithValue	Internal 🔒	🛑	
	└	functionCallWithValue	Internal 🔒	🛑	
	└	functionStaticCall	Internal 🔒		

| └ | functionStaticCall | Internal 🔒 | | |

| └ | functionDelegateCall | Internal 🔒 | 🛑 | |

| └ | functionDelegateCall | Internal 🔒 | 🛑 | |

| └ | verifyCallResult | Internal 🔒 | | |

||||||

| **AAACryptoAccessories** | Implementation | IERC1155 |||

| └ | <Constructor> | Public ❗ | 🛑 |NO❗ |

| └ | addToWhitelist | External ❗ | 🛑 |NO❗ |

| └ | removeFromWhitelist | External ❗ | 🛑 |NO❗ |

| └ | owner | Public ❗ | |NO❗ |

| └ | enableSale | External ❗ | 🛑 | onlyOwner |

| └ | disableSale | External ❗ | 🛑 | onlyOwner |

| └ | enablePresale | External ❗ | 🛑 | onlyOwner |

| └ | disablePresale | External ❗ | 🛑 | onlyOwner |

| └ | renounceOwnership | Public ❗ | 🛑 | onlyOwner |

| └ | transferOwnership | Public ❗ | 🛑 | onlyOwner |

| └ | _setOwner | Private 🔐 | 🛑 | |

| └ | mint | Public ❗ | 💲 | saleEnabled |

| └ | mintPresale | Public ❗ | 💲 | presaleEnabled |

| └ | _mint | Internal 🔒 | 🛑 | |

| └ | setDefaultUri | Public ❗ | 🛑 | onlyOwner |

| └ | safeTransferFrom | Public ❗ | 🛑 |NO❗ |

| └ | safeBatchTransferFrom | Public ❗ | 🛑 |NO❗ |

| └ | _safeTransferFrom | Internal 🔒 | 🛑 | |

| └ | isApprovedForAll | Public ❗ | |NO❗ |

| └ | setApprovalForAll | Public ❗ | 🛑 |NO❗ |

| └ | balanceOfBatch | Public ❗ | |NO❗ |

| └ | balanceOf | Public ❗ | |NO❗ |

| └ | uri | Public ❗ | |NO❗ |

| └ | totalSupply | Public ❗ | |NO❗ |

| └ | _safeBatchTransferFrom | Internal 🔒 | 🛑 | |

| └ | _doSafeTransferAcceptanceCheck | Private 🔐 | 🛑 | |

| └ | _doSafeBatchTransferAcceptanceCheck | Private 🔐 | 🛑 | |

| └ | _msgSender | Internal 🔒 | | |

| └ | supportsInterface | Public ❗ | |NO❗ |

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 **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, 3 low issues, and 2 notes.