Smart Contract Security Audit V1

Tuptalk Smart Contract Audit

Mar 20, 2025

https://saferico.com/

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

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
- **Name**: Tuptalk
- **Language** : Solidity
- **Contract Address**: 0x77033DecB1d3bf00384473e34B07958044a72aD3
- **Code Source**: https://etherscan.io/address/0x77033DecB1d3bf00384473e34B07958044a72aD3#code

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, 2 low, 0 very low-level issues and 0 note in all solidity files of the contract

The files:

Tuptalk.sol

Audit Score:

99% secure

File and Function Level Report

File in Scope:

Contract Name	SHA 256 hash	Contract Address
Tuptalk.sol	82f06259879d0cb33014b4c18921b2601a2d818b	0x77033DecB1d3bf00384473e34B07958044a72aD3

- Contract: Tuptalk
- Inherit: ERC20, Ownable
- Observation: All passed including security check
- Test Report: passed
- Score: passed
- Conclusion: passed

Function	Test Result	Type / Return Type	Score
balanceOf	✓	Read / public	**Passed**
allowance	✓	Read / public	**Passed**
symbol	✓	Read / public	**Passed**
decimal	✓	Read / public	**Passed**
totalSupply	✓	Read / public	**Passed**
name	✓	Read / public	**Passed**
owner	✓	Read / public	**Passed**
transferOwnership	✓	Write / public	**Passed**
renounceOwnership	✓	Write / public	**Passed**
transferFrom	✓	Write / public	**Passed**
transfer	✓	Write / public	**Passed**
approve	✓	Write / public	**Passed**

Issues Checking Status

SWC Attack Analysis

The Smart Contract Weakness Classification Registry (SWC Registry) is an implementation of the weakness classification scheme proposed in EIP-1470. It is loosely aligned to the terminologies and structure used in the Common Weakness Enumeration (CWE) for more info check https://swcregistry.io/

No.	Issue Description	Checking Status
136	Unencrypted Private Data On-Chain	**Passed**
135	Code With No Effects	**Passed**
134	Message call with hardcoded gas amount	**Passed**
133	Hash Collisions With Multiple Variable Length Arguments	**Passed**
132	Unexpected Ether balance	**Passed**
131	Presence of unused variables	**Passed**
130	Right-To-Left-Override control character (U+202E)	**Passed**
129	Typographical Error	**Passed**
128	DoS with block gas limit.	**Passed**
127	Arbitrary Jump with Function Type Variable	**Passed**
126	Insufficient Gas Griefing	**Passed**
125	Incorrect Inheritance Order	**Passed**
124	Write to Arbitrary Storage Location	**Passed**
123	Requirement Violation	**Passed**
122	Lack of Proper Signature Verification	**Passed**
121	Missing Protection against Signature Replay Attacks	**Passed**
120	Weak Sources of Randomness from Chain Attributes	**Passed**
119	Shadowing State Variables	**Passed**

118	Incorrect Constructor Name	**Passed**
117	Signature Malleability	**Passed**
116	Block values as a proxy for time	**Passed**
115	Authorization through tx.origin	**Passed**
114	Transaction Order Dependence	**Passed**
113	DoS with Failed Call	**Passed**
112	Delegatecall to Untrusted Callee	**Passed**
111	Use of Deprecated Solidity Functions	**Passed**
110	Assert Violation	**Passed**
109	Uninitialized Storage Pointer	**Passed**
108	State Variable Default Visibility	**Passed**
107	Reentrancy	**Passed**
106	Unprotected SELFDESTRUCT Instruction	**Passed**
105	Unprotected Ether Withdrawal	**Passed**
104	Unchecked Call Return Value	**Passed**
103	Floating Pragma	**Not Passed**
102	Outdated Compiler Version	**Passed**
101	Integer Overflow and Underflow	**Passed**
100	Function Default Visibility	**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:

#Pragam version not fixed

Description

It is a good practice to lock the solidity version for a live deployment (use 0.8.28 instead of ^0.8.26). 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. And avoid Solidity compiler Bugs check here

https://sepolia.etherscan.io/solcbuginfo

Remediation

Remove the ^ sign to lock the pragma version.

Status: Acknowledged.

#Missing zero address validation

When the dev wants deploy the contract the deployer will add the initial owner, he has to check for the zero address to make. Otherwise, the function will not work fine.

```
constructor(address initialOwner)
        ERC20("Tuptalk", "TUPL")
        Ownable(initialOwner)
    {
        _mint(owner(), 10000000000 * 10 ** decimals());
    }
```

Remediation

Use the require statement to check for zero addresses.

Status: Acknowledged.

Very Low:

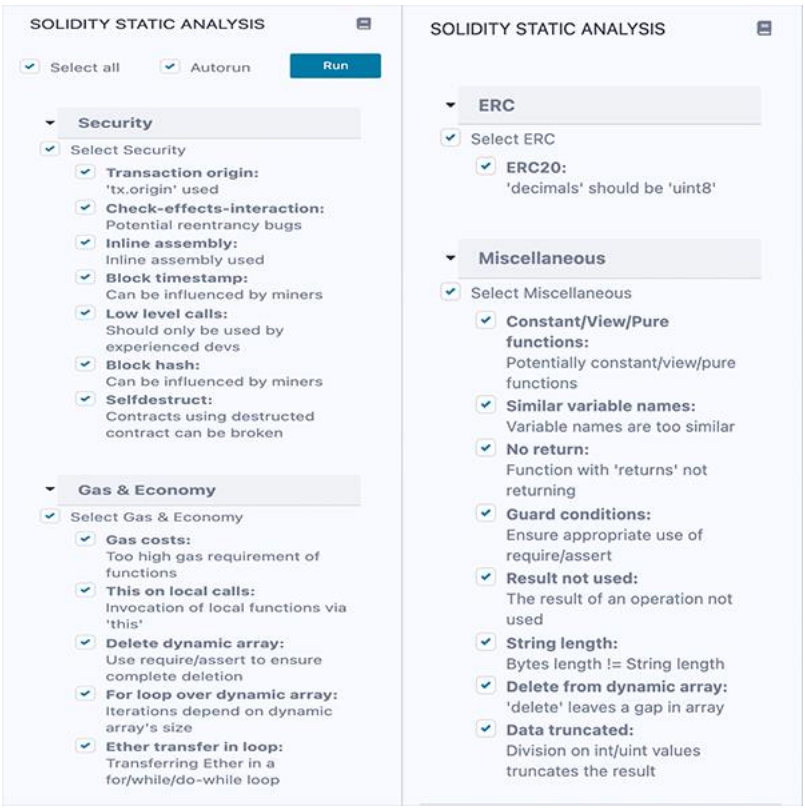
No Very Low severity vulnerabilities were found.

Notes:

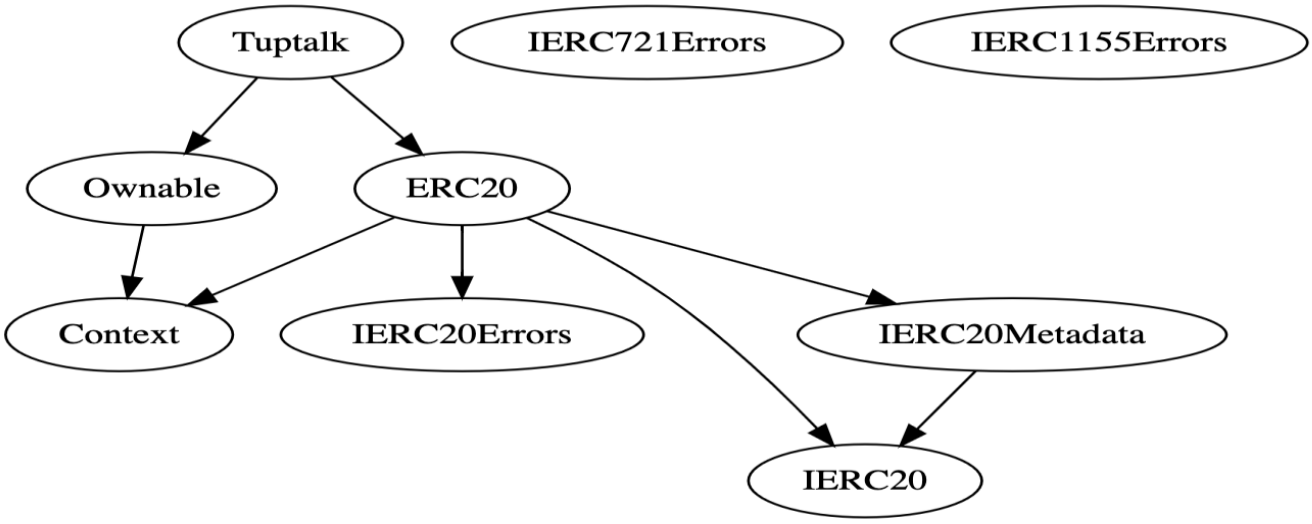
No Notes were found.

Automatic Testing

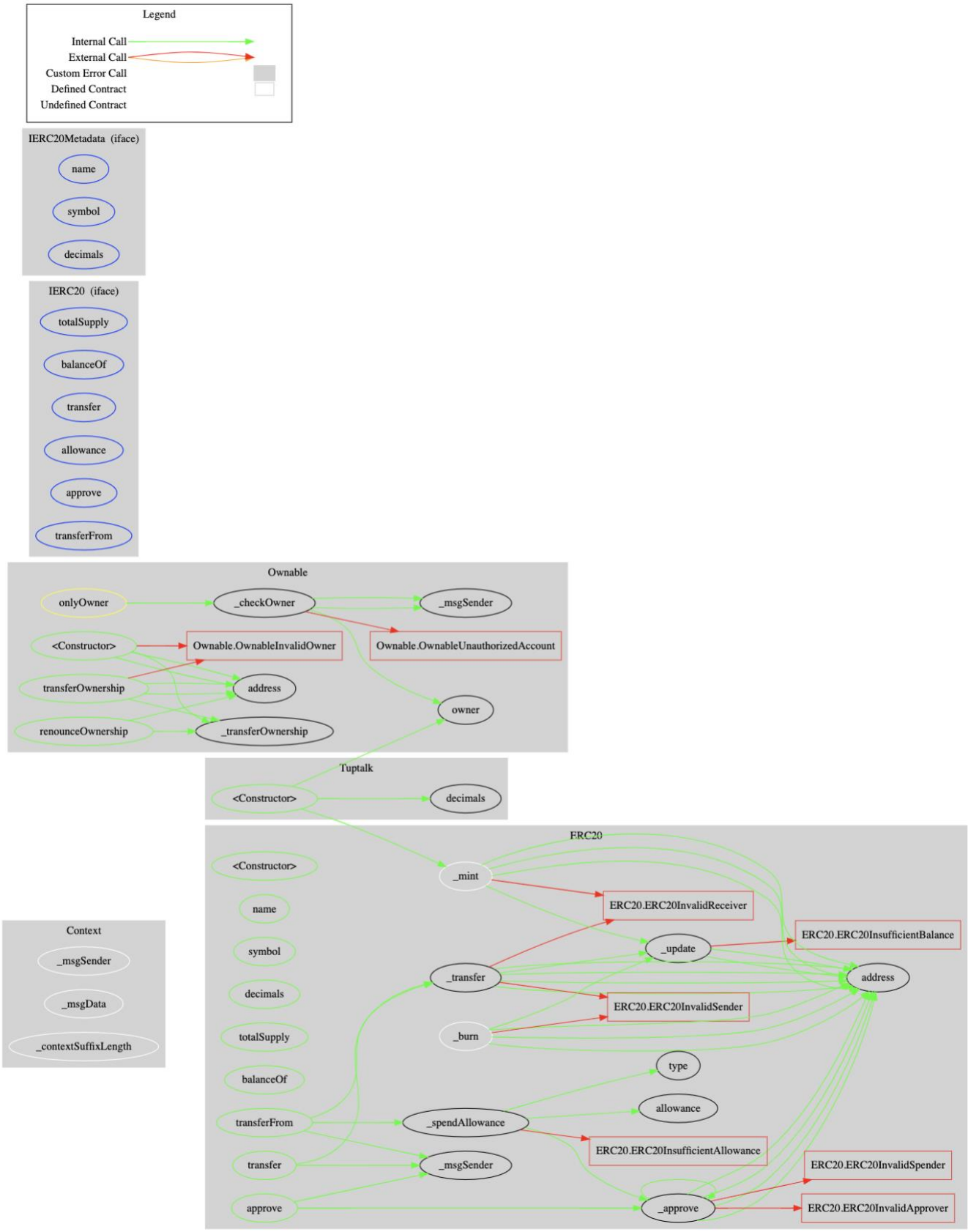
1- SOLIDITY STATIC ANALYSIS



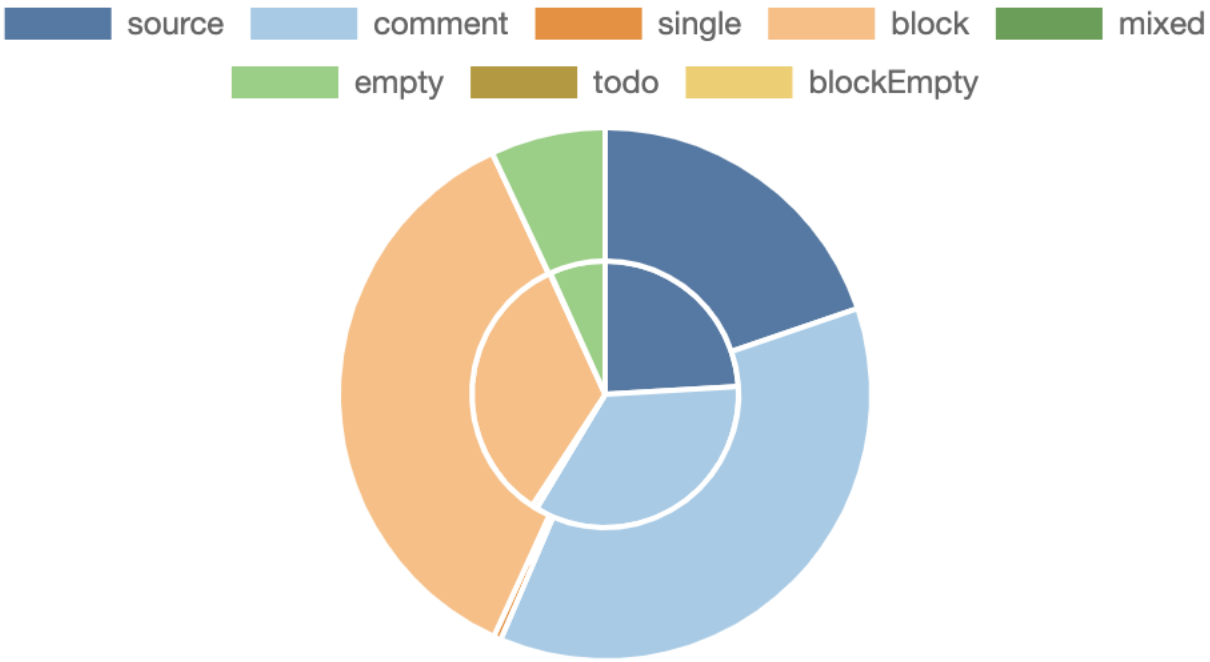
2- Inheritance graph



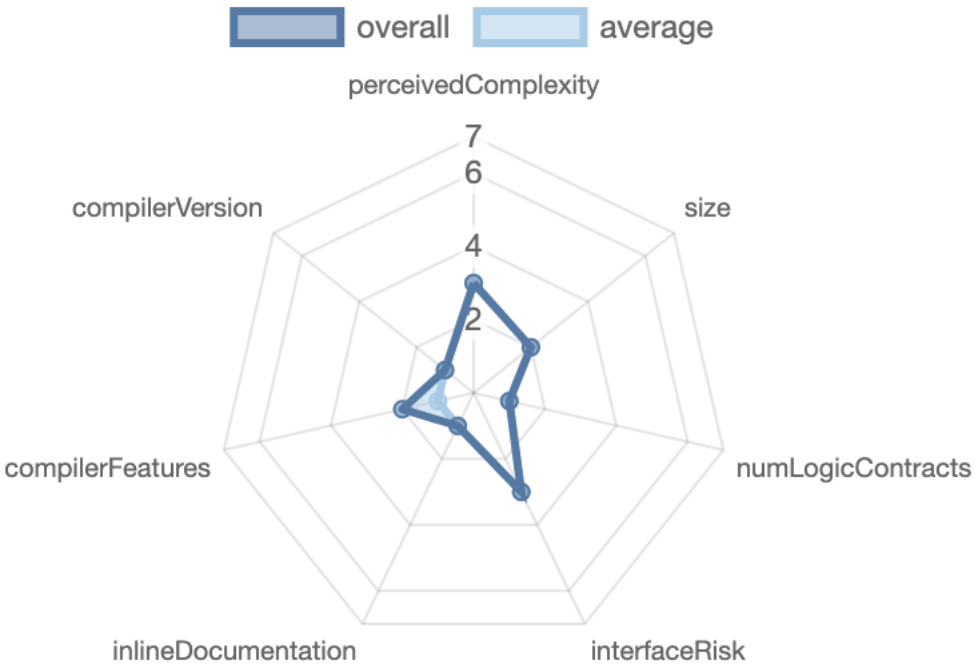
3- Call graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Ownable
onlyOwner
_checkOwner
_msgSender
<Constructor>
Ownable.OwnableInvalidOwner
Ownable.OwnableUnauthorizedAccount
transferOwnership
address
owner
renounceOwnership
_transferOwnership
Tuptalk
<Constructor>
decimals
ERC20
<Constructor>
_mint
ERC20.ERC20InvalidReceiver
name
_update
ERC20.ERC20InsufficientBalance
symbol
_transfer
address
decimals
ERC20.ERC20InvalidSender
totalSupply
_burn
type
balanceOf
allowance
transferFrom
_spendAllowance
ERC20.ERC20InsufficientAllowance
transfer
_msgSender
ERC20.ERC20InvalidSpender
approve
_approve
ERC20.ERC20InvalidApprover
Context
_msgSender
_msgData
_contextSuffixLength

Source lines



Risk level



Source units in scope

Source Units in Scope

Source Units Analyzed: `1`
Source Units in Scope: `1` **(100%)**

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝🔍🎨	Tuptalk.sol	4	5	692	629	232	400	151	🔆Σ
📝🔍🎨	**Totals**	**4**	**5**	**692**	**629**	**232**	**400**	**151**	🔆Σ

Legend: [➖]

- **Lines**: total lines of the source unit
- **nLines**: normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC**: normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines**: lines containing single or block comments
- **Complexity Score**: a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

Capabilities

Components

📝Contracts	📚Libraries	🔍Interfaces	🎨Abstract
1	0	5	3

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

🌐Public	💰Payable
21	0

External	Internal	Private	Pure	View
9	31	0	0	17

StateVariables

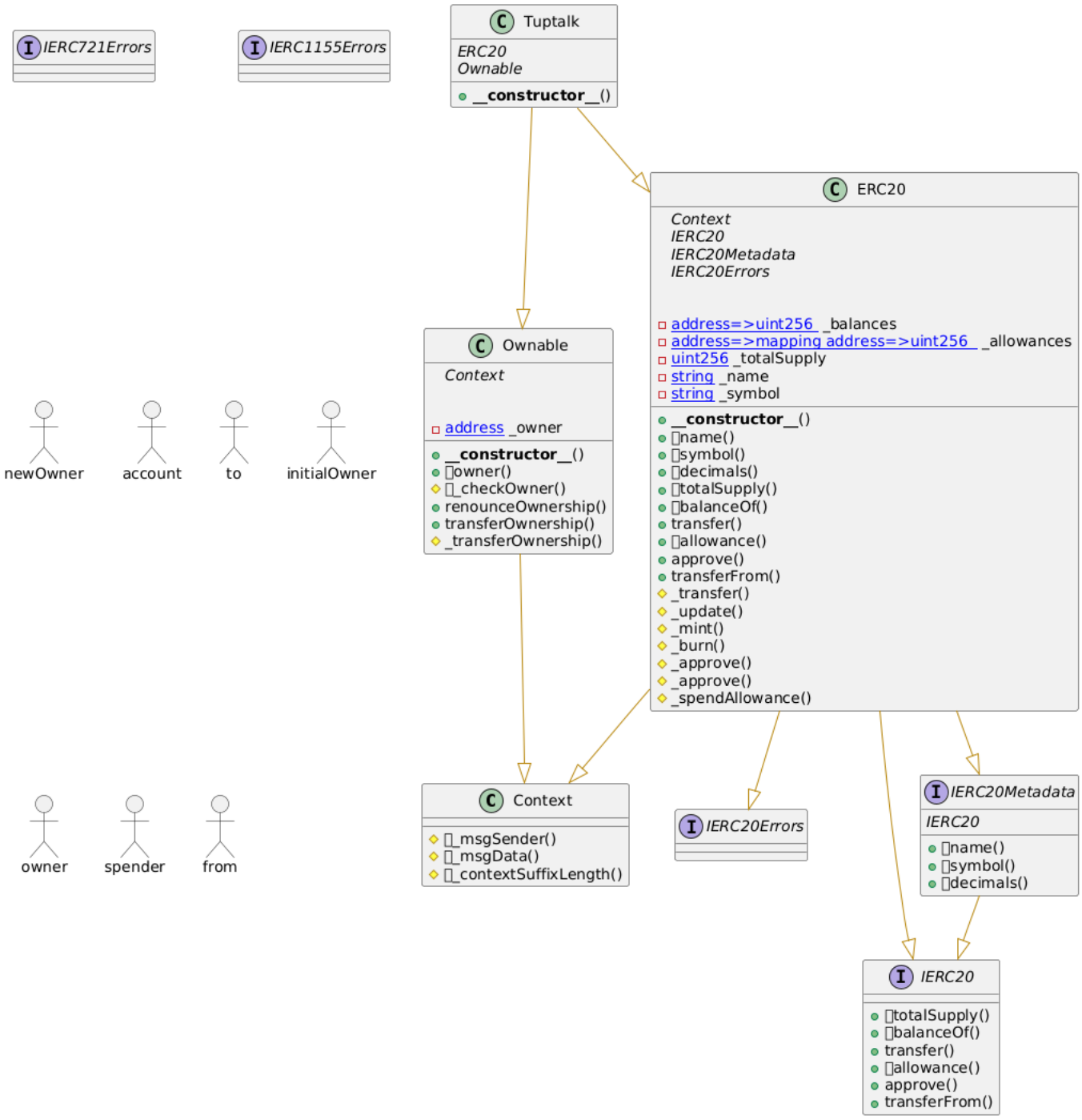
Total	🌐Public
6	0

Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥️ Uses Assembly	💣 Has Destroyable Contracts
`^0.8.26`				

📩 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧱 Uses Hash Functions	🪚 ECRecover	🌀 New/Create/Create2

Unified Modeling Language (UML)



Functions signature

Function Name	Sighash	Function Signature
owner	8da5cb5b	owner()
renounceOwnership	715018a6	renounceOwnership()
transferOwnership	f2fde38b	transferOwnership(address)
totalSupply	18160ddd	totalSupply()
balanceOf	70a08231	balanceOf(address)
transfer	a9059cbb	transfer(address,uint256)
allowance	dd62ed3e	allowance(address,address)
approve	095ea7b3	approve(address,uint256)
transferFrom	23b872dd	transferFrom(address,address,uint256)
name	06fdde03	name()
symbol	95d89b41	symbol()
decimals	313ce567	decimals()
name	06fdde03	name()
symbol	95d89b41	symbol()
decimals	313ce567	decimals()
totalSupply	18160ddd	totalSupply()
balanceOf	70a08231	balanceOf(address)
transfer	a9059cbb	transfer(address,uint256)
allowance	dd62ed3e	allowance(address,address)
approve	095ea7b3	approve(address,uint256)
transferFrom	23b872dd	transferFrom(address,address,uint256)

Automatic general report

Files Description Table

File Name	SHA-1 Hash
Tuptalk.sol	82f06259879d0cb33014b4c18921b2601a2d818b

Contracts Description Table

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
└	_msgSender	Internal 🔒		
└	_msgData	Internal 🔒		
└	_contextSuffixLength	Internal 🔒		
Ownable	Implementation	Context		
└		Public ❗	🛑	NO❗
└	owner	Public ❗		NO❗
└	_checkOwner	Internal 🔒		
└	renounceOwnership	Public ❗	🛑	onlyOwner
└	transferOwnership	Public ❗	🛑	onlyOwner
└	_transferOwnership	Internal 🔒	🛑	
IERC20Errors	Interface			
IERC721Errors	Interface			
IERC1155Errors	Interface			
IERC20	Interface			
└	totalSupply	External ❗		NO❗
└	balanceOf	External ❗		NO❗
└	transfer	External ❗	🛑	NO❗
└	allowance	External ❗		NO❗

Contract	Type	Bases		
└	approve	External ❗	🛑	NO❗
└	transferFrom	External ❗	🛑	NO❗
IERC20Metadata	Interface	IERC20		
└	name	External ❗		NO❗
└	symbol	External ❗		NO❗
└	decimals	External ❗		NO❗
ERC20	Implementation	Context, IERC20, IERC20Metadata, IERC20Errors		
└		Public ❗	🛑	NO❗
└	name	Public ❗		NO❗
└	symbol	Public ❗		NO❗
└	decimals	Public ❗		NO❗
└	totalSupply	Public ❗		NO❗
└	balanceOf	Public ❗		NO❗
└	transfer	Public ❗	🛑	NO❗
└	allowance	Public ❗		NO❗
└	approve	Public ❗	🛑	NO❗
└	transferFrom	Public ❗	🛑	NO❗
└	_transfer	Internal 🔒	🛑	
└	_update	Internal 🔒	🛑	
└	_mint	Internal 🔒	🛑	
└	_burn	Internal 🔒	🛑	
└	_approve	Internal 🔒	🛑	
└	_approve	Internal 🔒	🛑	
└	_spendAllowance	Internal 🔒	🛑	
Tuptalk	Implementation	ERC20, Ownable		

Contract	Type	Bases		
L		Public ❗		ERC20 Ownable

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.
✓ No 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.