Smart Contract Security Audit V1

Digital UYS Token

20/2/2022

https://saferico.com/

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

Table of Contents

Table of Contents

Background

Project Information

- Token Information
- Executive Summary

File and Function Level Report

- **File in Scope:**

Issues Checking Status

- Severity Definitions
- Audit Findings

Automatic testing

- Testing proves
- Inheritance graph
- Call graph

Unified Modeling Language (UML)

Functions signature

Automatic general report

Conclusion

Disclaimer

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**: 0x19102DBf368C7BE2cdA93BcA8Cf4c6D59FC39A5f
- **Code:**

https://rinkeby.etherscan.io/address/0x19102dbf368c7be2cda93bca8cf4c6d59fc39a5f#code

Token Information

- Name: UYS
- MAX Supply: 8,888,888,888
- Holders:
- Total transactions:

Contracts address deployed to test net (ETH)

Digital UYS Smart contract on ETH test net to test write functions by the auditor.

https://rinkeby.etherscan.io/address/0x7cf8de5111b1b9bc5a3d0717479bdb64c1dfb2f9

Executive Summary

According to our assessment, the customer`s solidity smart contract is **Well-Secured**. because all critical issues fixed in version 2.

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 1 critical, 0 high, 0 medium, 2 low, 0 very low-level issues and 2 notes in all solidity files of the contract

The files:

DigitalUYS.sol

File and Function Level Report

File in Scope:

Contract Name	SHA 256 hash	Contract Address
DigitalUYS.sol	**a9284007d1ac43084571521b93c84a25c75a299ee8ab9cae361717b533bcdc7f**	0x19102DBf368C7BE2cdA93BcA8Cf4c6D59FC39A5f

- Contract: DigitalUYS
- Inherit: Context, IERC20 , 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**
balanceOf	✓	Read / public	**Passed**
totalSupply	✓	Read / public	**Passed**
decimal	✓	Read / public	**Passed**
allowance	✓	Read / public	**Passed**
owner	✓	Read / public	**Passed**
getOwner	✓	Read / public	**Passed**
_claimAddressList	✓	Read / public	**Passed**
_ownerTokenList	✓	Read / public	**Passed**
approve	✓	Write / public	**Passed**
transferFrom	✓	Write / public	**Passed**
transfer	✓	Write / public	**Passed**

burn	✓	Write / public	**Passed**
decreaseAllowance	✓	Write / public	**Passed**
increaseAllowance	✓	Write / public	**Passed**
withdraw	✓	Write / public	**Passed**
transferOwnership	✓	Write / public	**Passed**
renounceOwnership	✓	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	Timestamp dependence.	**Passed**
6	Integer Overflow and Underflow.	**Passed**
7	DoS with Revert.	**Passed**
8	DoS with block gas limit.	**Passed with notes**
9	Methods execution permissions.	**Passed**
10	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**
11	The impact of the exchange rate on the logic.	**Passed**
12	Private user data leaks.	**Passed**
13	Malicious Event log.	**Passed**
14	Scoping and Declarations.	**Passed**
15	Uninitialized storage pointers.	**Passed**
16	Arithmetic accuracy.	**Passed**
17	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:

#The owner can burn all total supply

Description

The owner has the ability to burn all total supply include investors tokens without their permission, if he used the burn function ; this represents a risk for the users because in that case their funds will be burned with his/her permission in the smart contract.

```
function burn() public onlyOwner returns (bool) {
         for(uint i = 0; i < _ownerTokenList.length; i++){
             _burn(_ownerTokenList[i], _balances[_ownerTokenList[i]]);
         }
        return true;
    }
```

Remediation

There are two possibilities to remediate the risk. Make this function internal which no one can control it . The second one is making burn function burn from the total supply not from user's funds which is recommended by the auditor.

Status: Closed. Fixed in version2.

High:

No High severity vulnerabilities were found

Medium:

No Medium severity vulnerabilities were found

Low:

#Multiple pragma statements

Line	Pragma
7	pragma solidity ^0.8.0;
11	pragma solidity ^0.8.0;
35	pragma solidity ^0.8.0;
117	pragma solidity ^0.8.0;
121	pragma solidity ^0.8.0;
338	pragma solidity ^0.8.0;
437	pragma solidity ^0.8.0;
664	pragma solidity ^0.8.0;

Description

There are multiple pragma statements in the code. Only the compiler version 0.8.2 will work with the code, but keeping only one pragma statement helps in maintaining readability of the code.

Remediation

Keep a single pragma statement.

Status: Closed. Fixed In version 2

Functions that do not have a function visibility

Description

Functions that do not have a function visibility type specified are public by default. This can lead to a vulnerability if a developer forgot to set the visibility and a malicious user is able to make unauthorized or unintended state changes.

```
  constructor () {
        _name = 'Digital UYS';
        _symbol = 'UYS';
        _totalSupply = 8_888_888_888*10**decimals();
        _balances[msg.sender] = _totalSupply;
        _ownerTokenList.push(_owner);
        emit Transfer(address(0), msg.sender, _totalSupply);
        _burn(msg.sender, 4_444_444_444*10**decimals());
    }
```

Remediation

Make this function like that

```
_ constructor () {
        _name = 'Digital UYS';
        _symbol = 'UYS';
        _totalSupply = 8888888888*10**9;
        _balances[msg.sender] = _totalSupply;
        _ownerTokenList.push(_owner);
        emit Transfer(address(0), msg.sender, _totalSupply);
        _burn(msg.sender, 4444444444*10**9);
    }
```

Status: Closed. Fixed in version2.

Very Low:

No Very Low severity vulnerabilities were found.

Notes:

#Naming Conventions

Description
The contract follows a consistent naming convention where we are private variables with leading"_" and public variables without it. But we have missed to comply to the condition for certain variable names "__ownerTokenList" which is public

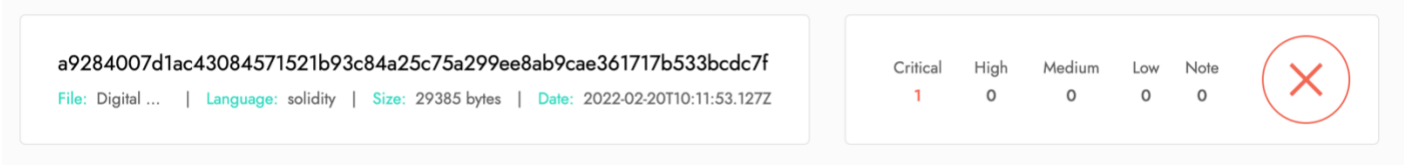
Remediation
Remove "_" from external variable names and add it to private variable names
Status: Closed. Fixed in version2.

#Compiler version is old
Description
The compiler being used was released a year ago. It's recommended to use more recent compiler version, there can be benefits like reduction in bytecode size etc.

Status: Acknowledged

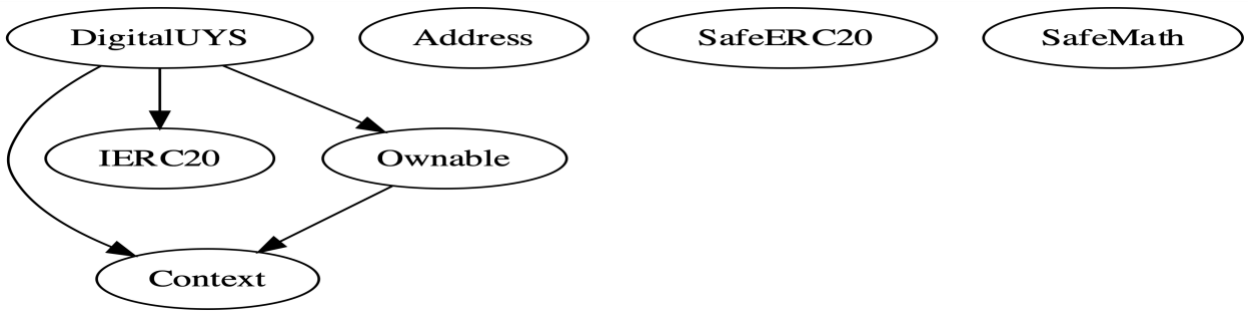
Automatic Testing

1- Check for security

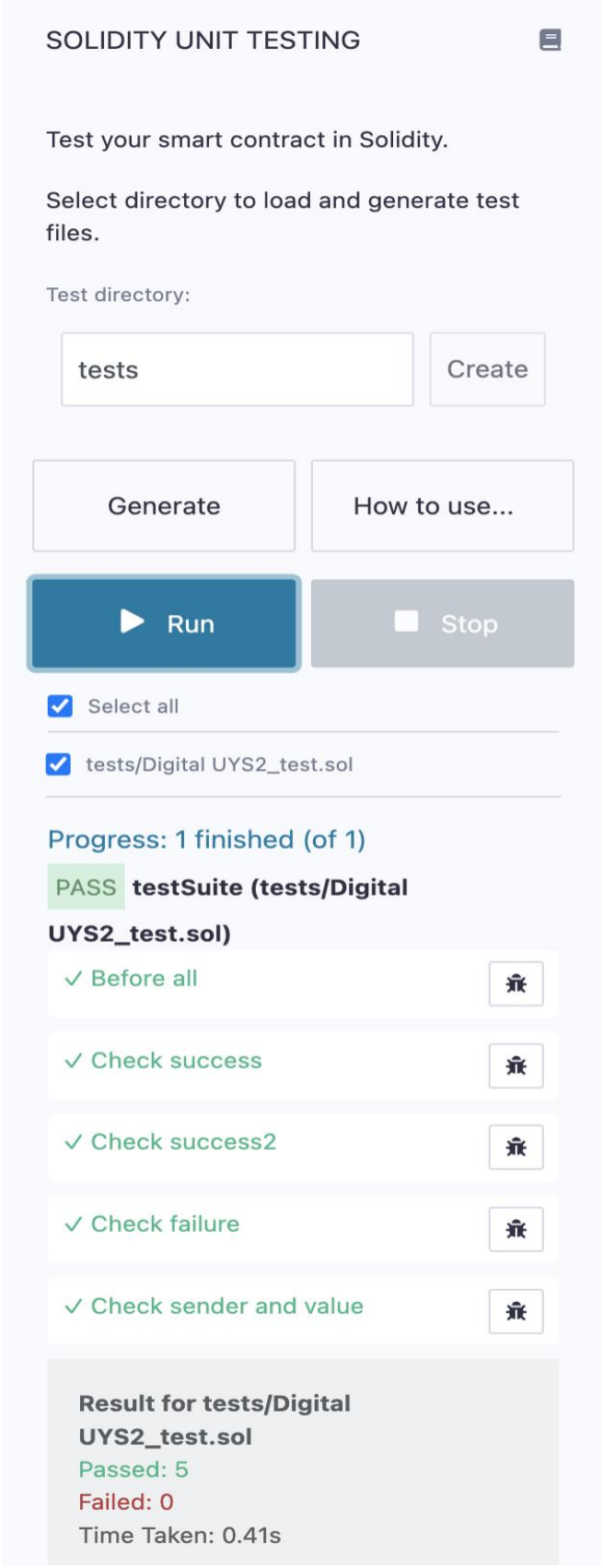


2- SOLIDITY STATIC ANALYSIS

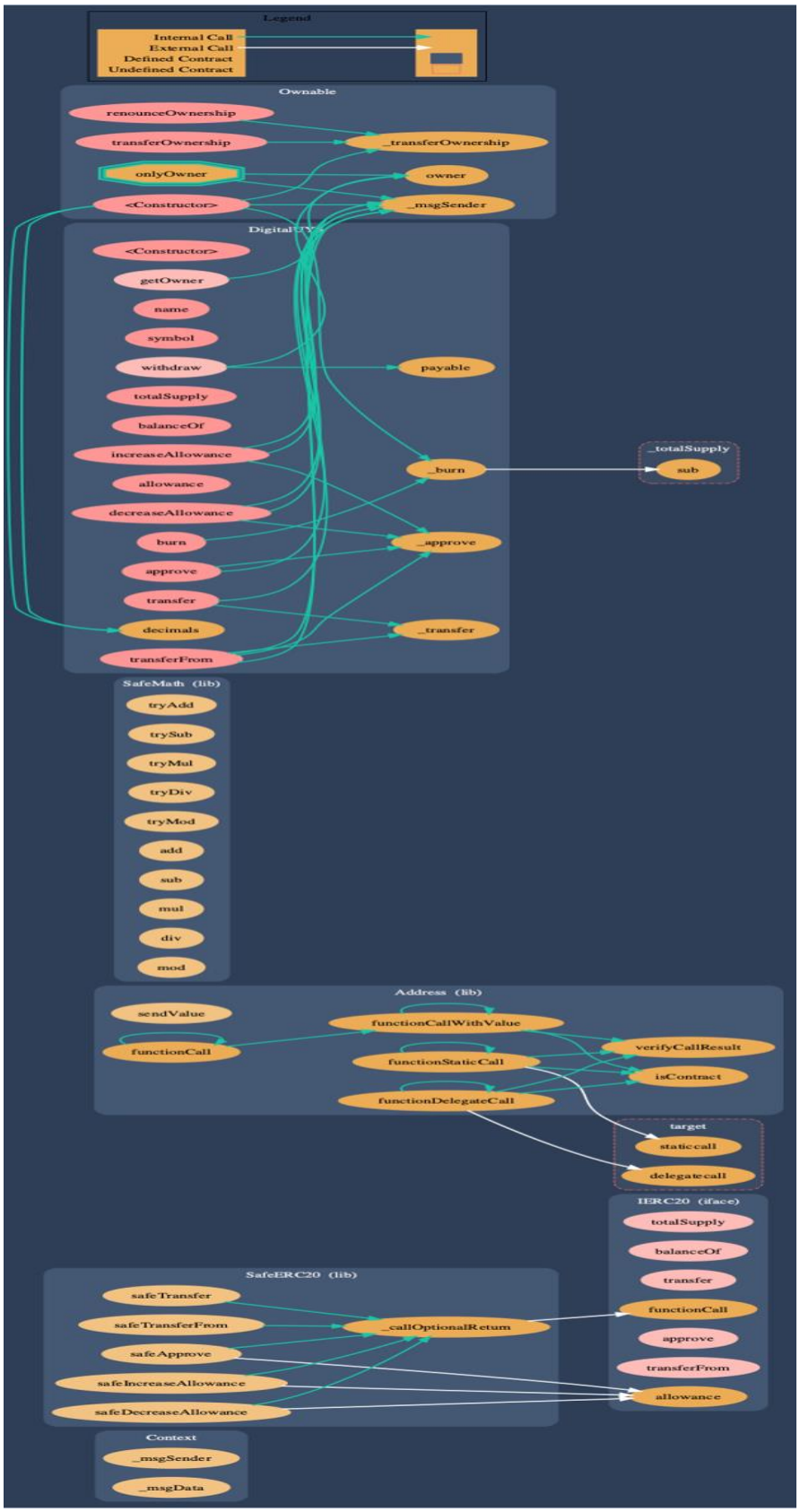
3- Inheritance graph



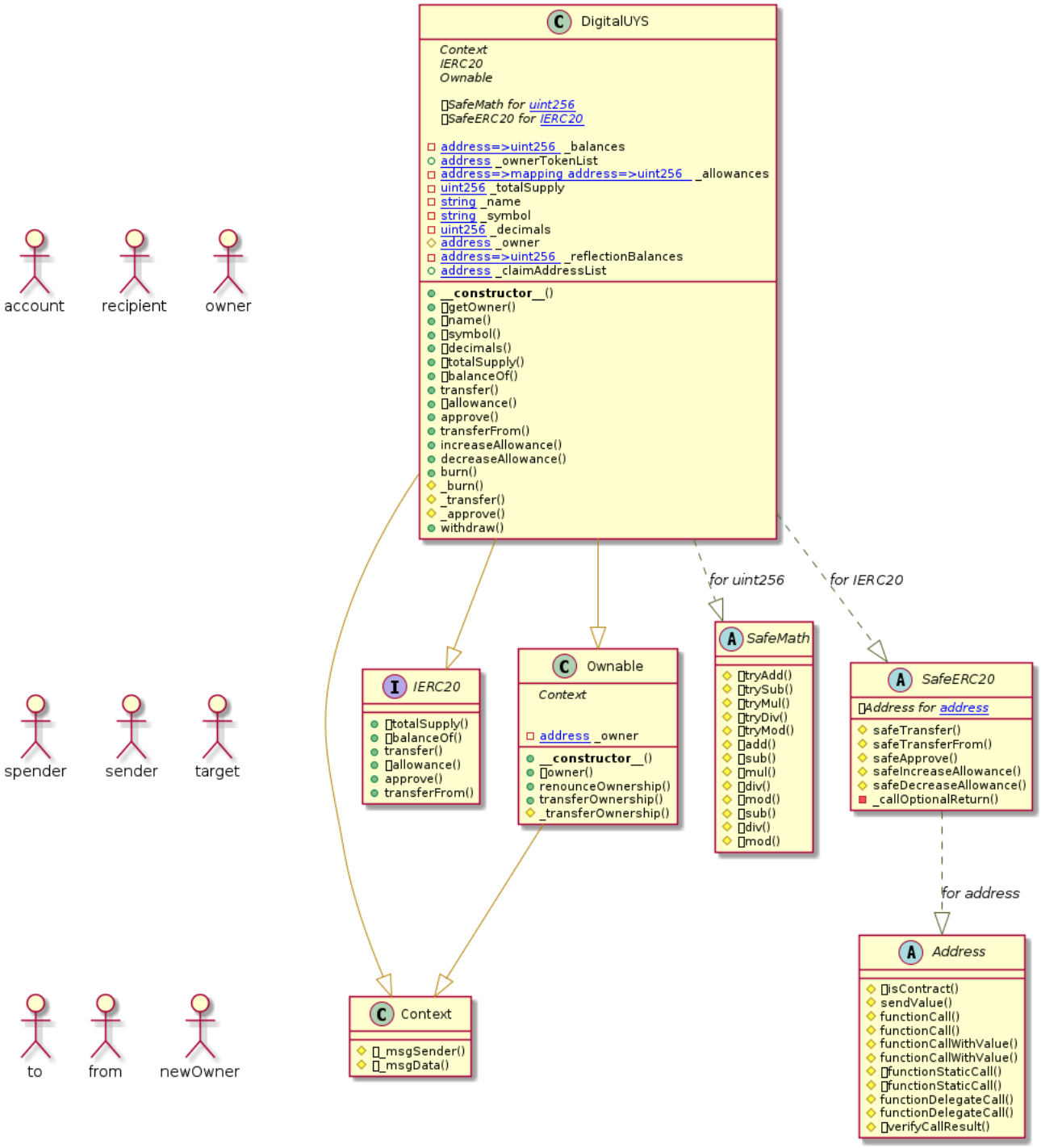
4- SOLIDITY UNIT TESTING



5- Call graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Ownable
renounceOwnership
transferOwnership
onlyOwner
<Constructor>
_transferOwnership
owner
_msgSender
DigitalHY
<Constructor>
getOwner
name
symbol
withdraw
totalSupply
balanceOf
increaseAllowance
allowance
decreaseAllowance
burn
approve
transfer
decimals
transferFrom
payable
_burn
_approve
_transfer
_totalSupply
sub
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
add
sub
mul
div
mod
Address (lib)
sendValue
functionCall
functionCallWithValue
functionStaticCall
functionDelegateCall
verifyCallResult
isContract
target
staticcall
delegatecall
IERC20 (iface)
totalSupply
balanceOf
transfer
functionCall
approve
transferFrom
allowance
SafeERC20 (lib)
safeTransfer
safeTransferFrom
safeApprove
safeIncreaseAllowance
safeDecreaseAllowance
_callOptionalReturn
Context
_msgSender
_msgData

Unified Modeling Language (UML)



Functions signature

```
16279055  =>  isContract(address)
39509351  =>  increaseAllowance(address,uint256)
119df25f  =>  _msgSender()
8b49d47e  =>  _msgData()
18160ddd  =>  totalSupply()
70a08231  =>  balanceOf(address)
a9059cbb  =>  transfer(address,uint256)
dd62ed3e  =>  allowance(address,address)
095ea7b3  =>  approve(address,uint256)
23b872dd  =>  transferFrom(address,address,uint256)
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)
d0c407e1  =>  safeTransfer(IERC20,address,uint256)
5beae096  =>  safeTransferFrom(IERC20,address,address,uint256)
d6dcec8d  =>  safeApprove(IERC20,address,uint256)
390cc046  =>  safeIncreaseAllowance(IERC20,address,uint256)
5164ffed  =>  safeDecreaseAllowance(IERC20,address,uint256)
becc5a20  =>  _callOptionalReturn(IERC20,bytes)
884557bf  =>  tryAdd(uint256,uint256)
a29962b1  =>  trySub(uint256,uint256)
6281efa4  =>  tryMul(uint256,uint256)
736ecb18  =>  tryDiv(uint256,uint256)
38dc0867  =>  tryMod(uint256,uint256)
771602f7  =>  add(uint256,uint256)
b67d77c5  =>  sub(uint256,uint256)
c8a4ac9c  =>  mul(uint256,uint256)
a391c15b  =>  div(uint256,uint256)
f43f523a  =>  mod(uint256,uint256)
e31bdc0a  =>  sub(uint256,uint256,string)
b745d336  =>  div(uint256,uint256,string)
71af23e8  =>  mod(uint256,uint256,string)
8da5cb5b  =>  owner()
715018a6  =>  renounceOwnership()
f2fde38b  =>  transferOwnership(address)
d29d44ee  =>  _transferOwnership(address)
893d20e8  =>  getOwner()
06fdde03  =>  name()
95d89b41  =>  symbol()
313ce567  =>  decimals()
a457c2d7  =>  decreaseAllowance(address,uint256)
44df8e70  =>  burn()
6161eb18  =>  _burn(address,uint256)
30e0789e  =>  _transfer(address,address,uint256)
104e81ff  =>  _approve(address,address,uint256)
3ccfd60b  =>  withdraw()
```

Automatic general report

Files Description Table

File Name	SHA-1 Hash
/Users/macbook/Desktop/smart contracts/Digital UYS2.sol	98bfdb2c3ec7fc3096ad9363050a5e977c9d117a

Contracts Description Table

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
└	_msgSender	Internal 🔒		
└	_msgData	Internal 🔒		
IERC20	Interface			
└	totalSupply	External ❗		NO❗
└	balanceOf	External ❗		NO❗
└	transfer	External ❗	🛑	NO❗
└	allowance	External ❗		NO❗
└	approve	External ❗	🛑	NO❗
└	transferFrom	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 🔒		
SafeERC20	Library			
└	safeTransfer	Internal 🔒	🛑	
└	safeTransferFrom	Internal 🔒	🛑	
└	safeApprove	Internal 🔒	🛑	
└	safeIncreaseAllowance	Internal 🔒	🛑	
└	safeDecreaseAllowance	Internal 🔒	🛑	
└	_callOptionalReturn	Private 🔐	🛑	
SafeMath	Library			
└	tryAdd	Internal 🔒		
└	trySub	Internal 🔒		

└	tryMul	Internal 🔒		
└	tryDiv	Internal 🔒		
└	tryMod	Internal 🔒		
└	add	Internal 🔒		
└	sub	Internal 🔒		
└	mul	Internal 🔒		
└	div	Internal 🔒		
└	mod	Internal 🔒		
└	sub	Internal 🔒		
└	div	Internal 🔒		
└	mod	Internal 🔒		

Ownable	Implementation	Context		
└	<Constructor>	Public ❗	🛑	NO❗
└	owner	Public ❗		NO❗
└	renounceOwnership	Public ❗	🛑	onlyOwner
└	transferOwnership	Public ❗	🛑	onlyOwner
└	_transferOwnership	Internal 🔒	🛑	

DigitalUYS	Implementation	Context, IERC20, Ownable		
└	<Constructor>	Public ❗	🛑	NO❗
└	getOwner	External ❗		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❗
└	increaseAllowance	Public ❗	🛑	NO❗
└	decreaseAllowance	Public ❗	🛑	NO❗
└	burn	Public ❗	🛑	onlyOwner
└	_burn	Internal 🔒	🛑	
└	_transfer	Internal 🔒	🛑	
└	_approve	Internal 🔒	🛑	
└	withdraw	External ❗	🛑	onlyOwner

Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

Conclusion

The contracts are written systematically. Team found no critical issues after the fix the problems. 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.