SMART CONTRACT AUDIT REPORT

For

EMC

Prepared By: SFI Team

Prepared for: E l y s i u m

Prepared on: 23/10/2021

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 154 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

- **Good required condition in functions: -**

  - Here you are transferring to the address .

```
function transfer(address _to, uint256 _value)
  public returns(bool success) {
        _transfer(msg.sender, _to, _value);
        return true;
       }
```

  - Here you are checking the approve function

```
function approve(address _spender, uint256 _value)
  public returns (bool success) {
        allowance[msg.sender][_spender] = _value;
        emit Approval(msg.sender, _spender, _value);

        return true;
    }
```

  - Here you are Checking transfer ownership to another address function

```
function transferOwnership(address newOwner) public returns
(bool success) {
        require(newOwner != address(0));
        require(newOwner != owner);
        require(msg.sender == owner);


        emit OwnershipTransferred(owner, newOwner);
        owner = newOwner;


        return success;
    }
```

  - Here you are checking Mint function only owner can do it

```
function mint(uint256 _value) public returns (bool success) {
        require(msg.sender == owner);
        require(totalSupply + _value <= maxSupply);

        balanceOf[owner] += _value;
        totalSupply += _value;

        emit Mint(owner, _value);

        return true;}
```

• 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

#Gas costs:

```
  string public name; // token name
    string public symbol; // token symbol
```

In detail

Gas requirement of function Elysium.name is infinite:
If the gas requirement of a function is higher than the block gas limit, it cannot be executed.
Please avoid loops in your functions or actions that modify large areas of storage
(this includes clearing or copying arrays in storage)

ERC20 standard violation

```
function approve(address _spender, uint256 _value) public returns (bool succ
 allowance[msg.sender][_spender] = _value;

 Approval(msg.sender, _spender, _value);
```

```
return true;}
```

In detail
Approval event is not triggered after successful operation with allowances (see https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md
MUST trigger on any successful call to approve(address _spender, uint256 _value)).

#outdated compiler

```
pragma solidity 0.5.17;
```

In detail
Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version.

Notes

```
uint8 devFeePercentage;
```

In detail
Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

#

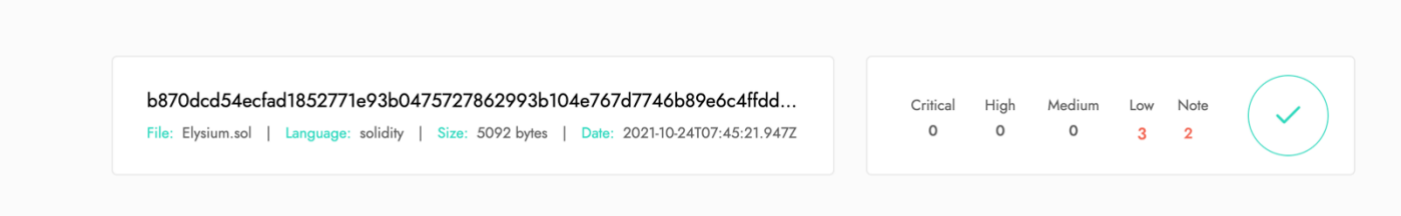
Data truncated:

```
uint256 devFeeAmount = _value/100*devFeePercentage;
```

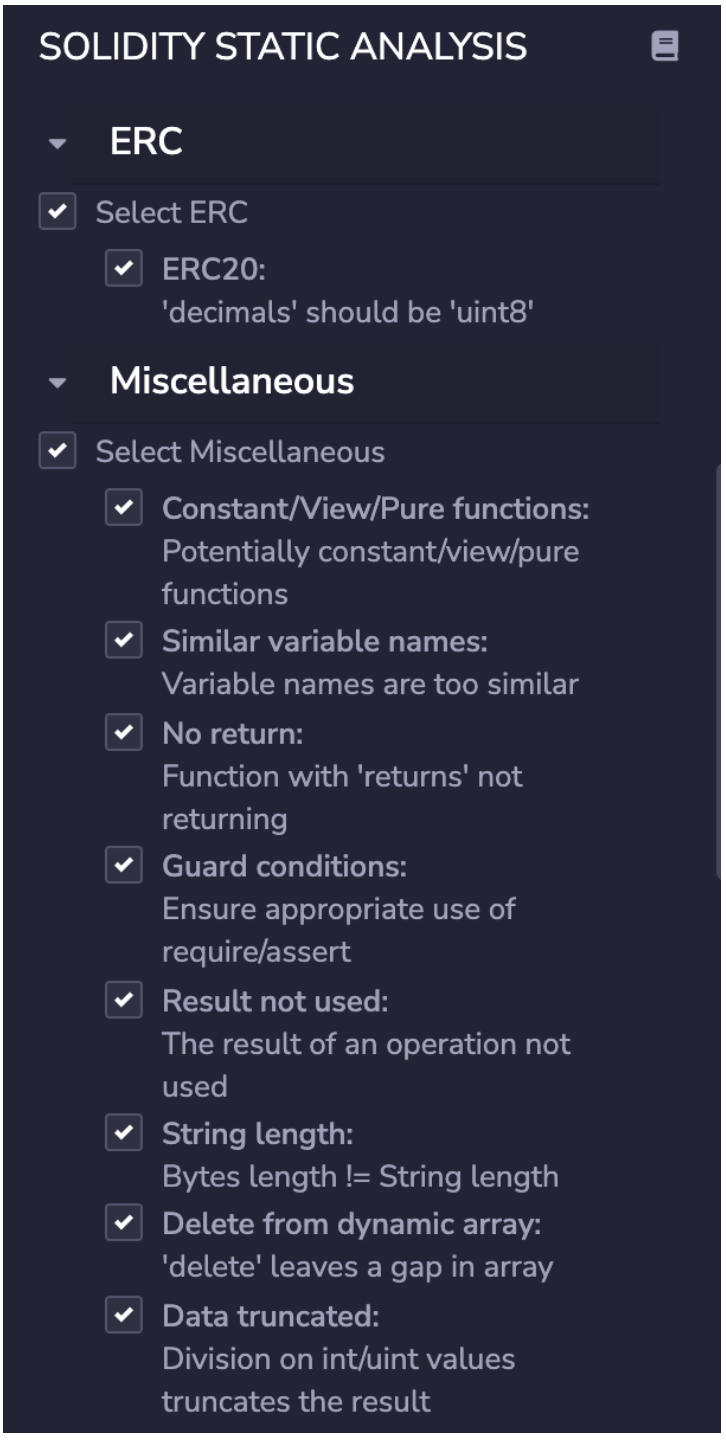
Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.

Testing proves:

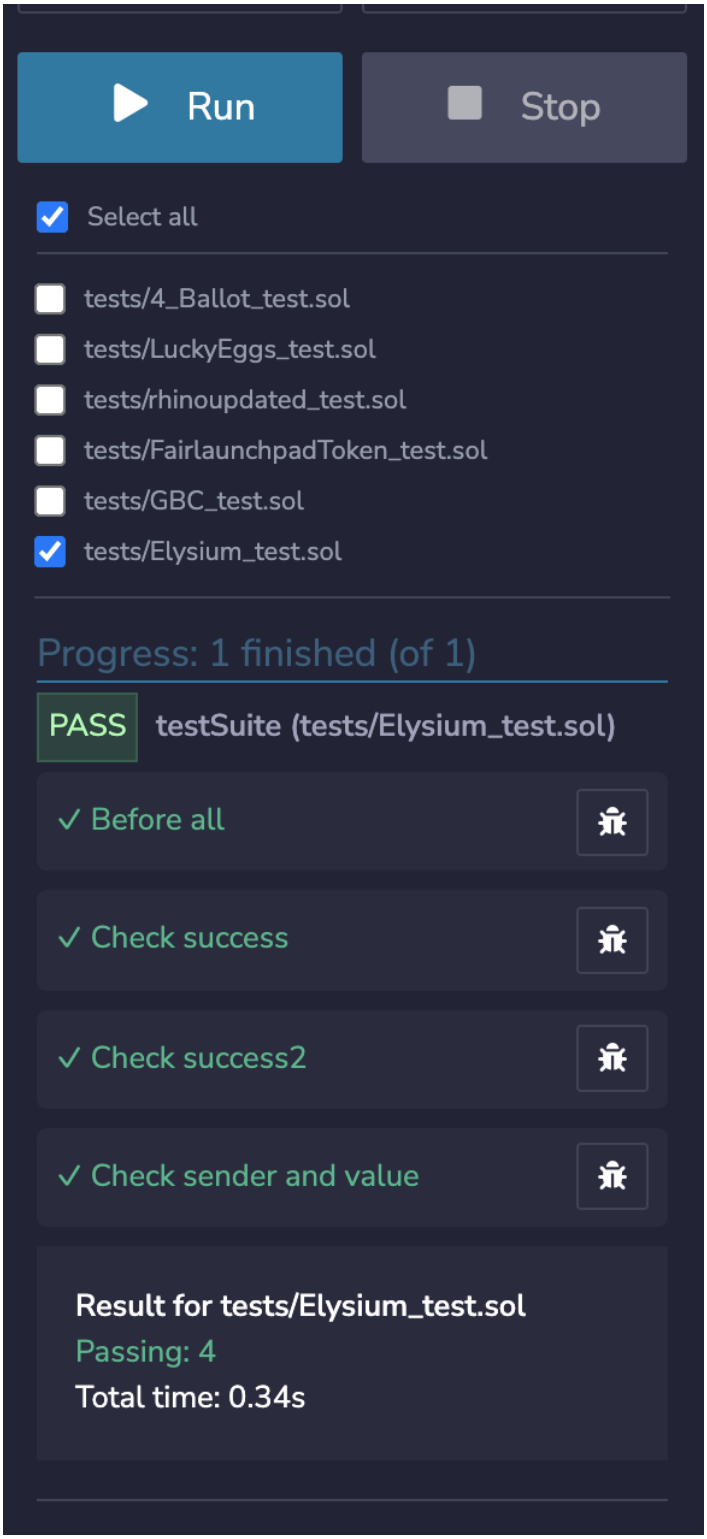
1- Check for security



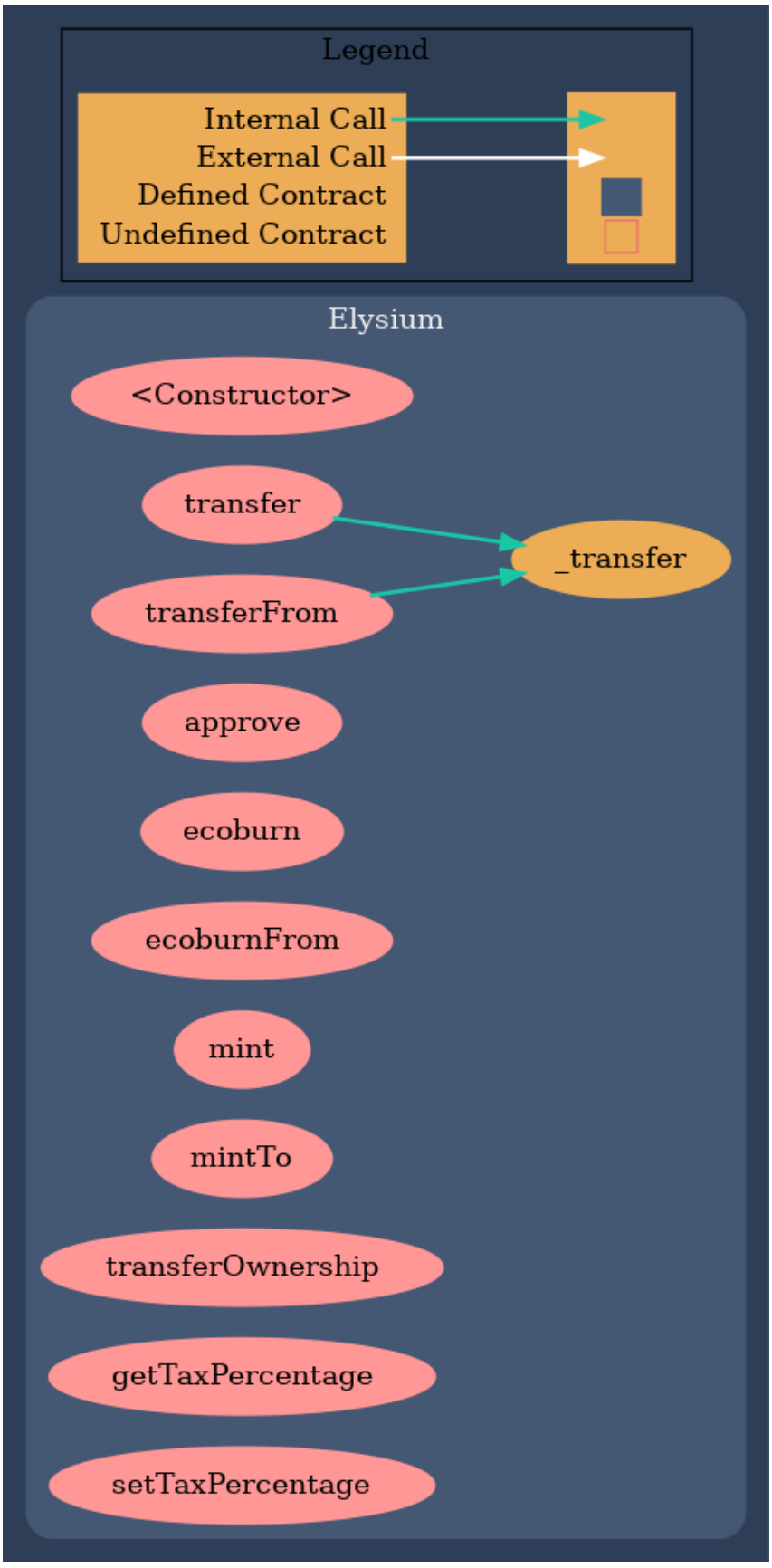
2- SOLIDITY STATIC ANALYSIS



3- SOLIDITY UNIT TESTING



4- Call graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Elysium
<Constructor>
transfer
_transfer
transferFrom
approve
ecoburn
ecoburnFrom
mint
mintTo
transferOwnership
getTaxPercentage
setTaxPercentage

Automatic general report

Files Description Table

File Name	SHA-1 Hash
/Users/macbook/Desktop/smart contracts/Elysium.sol	e066cf222bd65247b78b4f3c9e0916b35e05d4ce

Contracts Description Table

Contract	Type	Bases		
└	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
Elysium	Implementation			
└	<Constructor>	Public	🛑	NO
└	_transfer	Internal	🛑	
└	transfer	Public	🛑	NO
└	transferFrom	Public	🛑	NO
└	approve	Public	🛑	NO
└	ecoburn	Public	🛑	NO
└	ecoburnFrom	Public	🛑	NO
└	mint	Public	🛑	NO
└	mintTo	Public	🛑	NO
└	transferOwnership	Public	🛑	NO
└	getTaxPercentage	Public		NO
└	setTaxPercentage	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.