EtherAuthority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project:	ELF Token
Website:	aelf.com
Platform:	Ethereum
Language:	Solidity
Date:	February 16th, 2024

Table of contents

`

Introduction

As part of EtherAuthority's community smart contracts audit initiatives, the Elf Token smart contract from aelf.com was audited extensively. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on February 16th, 2024.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

- The `AElfToken` contract is a robust implementation of an ERC20 token with added functionalities for minting, burning, time-locked tokens, and transfer restrictions.
- It uses the `SafeMath` library for safe arithmetic operations and incorporates ownership control via the `Ownable` contract.
- The contract also features detailed events and modifiers to ensure secure and efficient token management.
- The ELF Token is an ERC20-based smart contract that offers various functions such as transferring tokens, updating ELF Multisig addresses, and setting lock durations for minting, withdrawing, burning, and minting tokens.

Audit scope

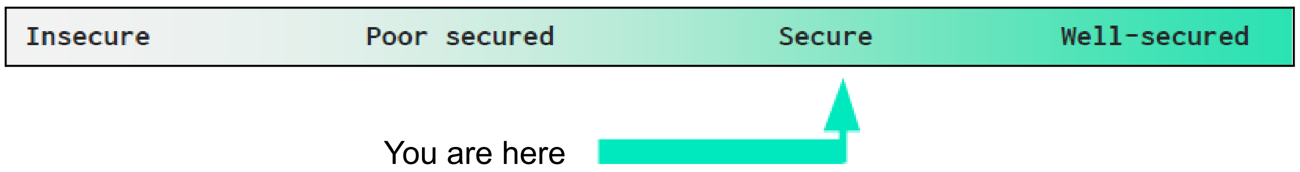
Name	**Code Review and Security Analysis Report for ELF Token Smart Contract**
Platform	**Ethereum**
Language	**Solidity**
File	AElfToken.sol
Smart Contract Code	0xbf2179859fc6d5bee9bf9158632dc51678a4100e
Audit Date	February 16th, 2024

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: ELF Token • Symbol: ELF • Decimals: 18 • Total Supply Cap:1 billion	**YES, This is valid.**
Ownership Control: • The current owner can transfer the ownership.	**YES, This is valid.** **We suggest renouncing ownership once the ownership functions are not needed. This is to make the smart contract 100% decentralized.**

Audit Summary

According to the standard audit assessment, the Customer`s solidity-based smart contracts are **“Secured”.** This token contract does contain owner control, which does not make it fully decentralized.



We used various tools like Slither, Solhint, and Remix IDE. At the same time, this finding is based on a critical analysis of the manual audit.

All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit Overview section. The general overview is presented in the AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 0 high, 0 medium, 0 low, and 9 very low level issues.

Investor Advice: A technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner-controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	The solidity version is not specified	Passed
	Solidity version is too old	Moderated
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Business Risk Analysis

Category	Result
● Buy Tax	0%
● Sell Tax	0%
● Cannot Buy	No
● Cannot Sell	No
● Max Tax	0%
● Modify Tax	No
● Fee Check	Not Detected
● Is Honeypot	Not Detected
● Trading Cooldown	Not Detected
● Can Pause Trade?	Not Detected
● Pause Transfer?	Not Detected
● Max Tax?	No
● Is it Anti-whale?	Not Detected
● Is Anti-bot?	Not Detected
● Is it a Blacklist?	Yes
● Blacklist Check	Yes
● Can Mint?	No
● Is it a Proxy?	No
● Can Take Ownership?	Yes
● Hidden Owner?	Not Detected
● Self Destruction?	Not Detected
● Auditor Confidence	High

Overall Audit Result: PASSED

Code Quality

This audit scope has 1 smart contract. Smart contracts contain Libraries, Smart contracts, inherits, and Interfaces. This is a compact and well-written smart contract.

The libraries in ELF Token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties/methods can be reused many times by other contracts in the ELF Token.

The EtherAuthority team has no scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are not well commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given an ELF Token smart contract code in the form of an Etherscan web link.

As mentioned above, code parts are not well commented on. but the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries used in this smart contract infrastructure that are based on well-known industry standard open-source projects.

Apart from libraries, its functions are not used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	canMint	modifier	Missing required error message	Refer Audit Findings
3	only	modifier	Missing required error message	Refer Audit Findings
4	nonZeroAddress	modifier	Missing required error message	Refer Audit Findings
5	canTransfer	modifier	Missing required error message	Refer Audit Findings
6	transfer	write	Missing required error message	Refer Audit Findings
7	balanceOf	read	Passed	No Issue
8	transferFrom	write	Missing required error message	Refer Audit Findings
9	approve	write	can Transfer	No Issue
10	allowance	read	Passed	No Issue
11	increaseApproval	write	canTransfer	No Issue
12	decreaseApproval	write	canTransfer	No Issue
13	setTransferable	write	Centralized, Missing required error message	Refer Audit Findings
14	disableSetTransferable	write	Centralized	Refer Audit Findings
15	setAElfDevMultisig	write	Centralized	Refer Audit Findings
16	setAElfCommunityMultisig	write	Centralized	Refer Audit Findings
17	setDurationOfLock	write	Centralized, Missing required error message	Refer Audit Findings
18	getLockTokens	read	Passed	No Issue
19	approveMintTokens	write	Centralized, Missing required error message	Refer Audit Findings
20	withdrawMintTokens	write	Centralized, Missing required error message	Refer Audit Findings
21	mintTokens	write	Missing required error message	Refer Audit Findings
22	mintTokensWithinTime	write	Missing required error message	Refer Audit Findings
23	transferForMultiAddresses	write	canTransfer	No Issue
24	burnTokens	write	Missing required error message	Refer Audit Findings
25	finishMinting	write	canMint	No Issue
26	getCurrentBlockNumber	read	Passed	No Issue
27	balanceOf	read	Passed	No Issue

28	transfer	write	Passed	No Issue
29	allowance	read	Passed	No Issue
30	transferFrom	write	Passed	No Issue
31	approve	write	Passed	No Issue
32	Ownable	write	Passed	No Issue
33	onlyOwner	modifier	Missing required error message	Refer Audit Findings
34	transferOwnership	write	Centralized, Missing required error message	Refer Audit Findings

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial
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 a significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can’t affect smart contract execution and can be ignored.

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

No High severity vulnerabilities were found.

Medium

No Medium-severity vulnerabilities were found.

Low

No Low Severity vulnerabilities were found.

Very Low / Informational / Best practices:

(1) Use the latest solidity version:

```
pragma solidity ^0.4.18;
```

Using the latest solidity will prevent any compiler-level bugs.

Resolution: We suggest using the latest solidity compiler version.

(2) Missing SPDX license identifier:
Solidity's new specification requires a valid SPDX license identifier to be included in every smart contract file.

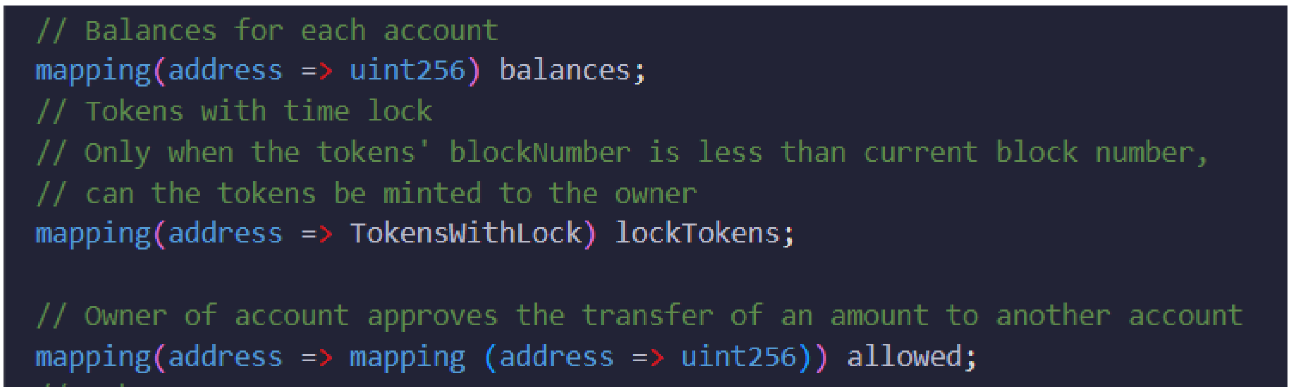
Resolution: Please add a comment for the appropriate SPDX license identifier.

(3) Code style:
The assert function mul(), sub(), add() should only be used to test for internal errors and to check invariants. The required function should be used to ensure valid conditions, such as

inputs, or contract state variables are met, or to validate return values from calls to external contracts.

Resolution: We suggest using the require condition instead of assert in the mentioned functions.

(4) Visibility not defined:



```
// Balances for each account
mapping(address => uint256) balances;
// Tokens with time lock
// Only when the tokens' blockNumber is less than current block number,
// can the tokens be minted to the owner
mapping(address => TokensWithLock) lockTokens;

// Owner of account approves the transfer of an amount to another account
mapping(address => mapping (address => uint256)) allowed;
```

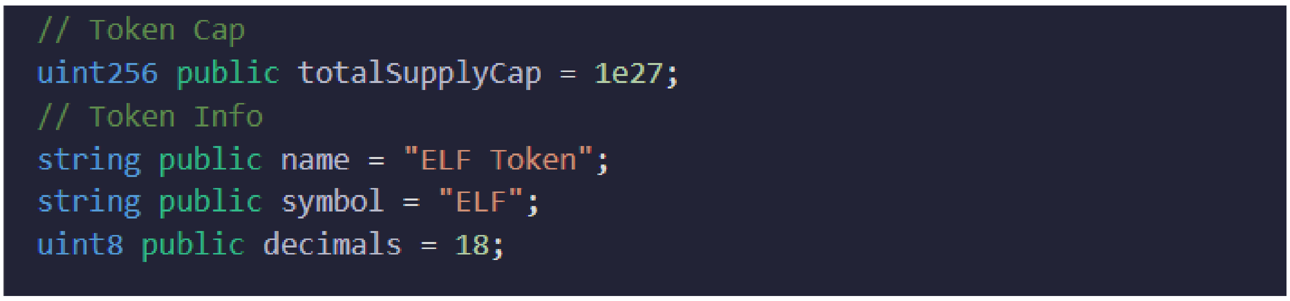
The linked variable declaration does not have a visibility specifier explicitly set.

<u>Variables like:</u>

- balances
- LockToken
- allowed

Resolution: Inconsistencies in the default visibility the Solidity compilers impose can cause issues in the functionality of the codebase. We suggest that the visibility specifier for the linked variable is explicitly set.

(5) Declare Constant Variables:

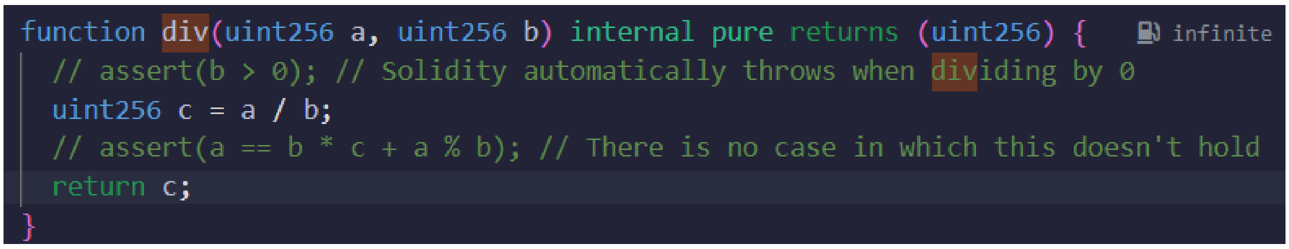


```
// Token Cap
uint256 public totalSupplyCap = 1e27;
// Token Info
string public name = "ELF Token";
string public symbol = "ELF";
uint8 public decimals = 18;
```

The variables totalSupplyCap, name, symbol, and decimals are unchanged throughout the contract.

Resolution: We suggest setting these variables as constant variables.

(6) Logical issue:


```
function div(uint256 a, uint256 b) internal pure returns (uint256) {    infinite
    // assert(b > 0); // Solidity automatically throws when dividing by 0
    uint256 c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
    return c;
}
```

The linked div() function performs divisions without proper division by zero checks.

Resolution: We suggest performing proper division by zero checks before performing division to avoid unexpected exceptions.

(7) Visibility can be external over the public:

Any functions which are not called internally should be declared as external. This saves some gas and is considered a good practice.

https://ethereum.stackexchange.com/questions/19380/external-vs-public-best-practices

(8) Centralized:

The onlyOwner has an owner authority:

- transferOwnership

The aelfDevMultisig has to mint unlimited tokens and change the address of the aelfDevMultisig of these functions:

- setTransferable
- disableSetTransferable
- setAElfDevMultisig
- mintTokens
- mintTokensWithinTime
- finishMinting.

The aelfCommunityMultisig has owner authority on:

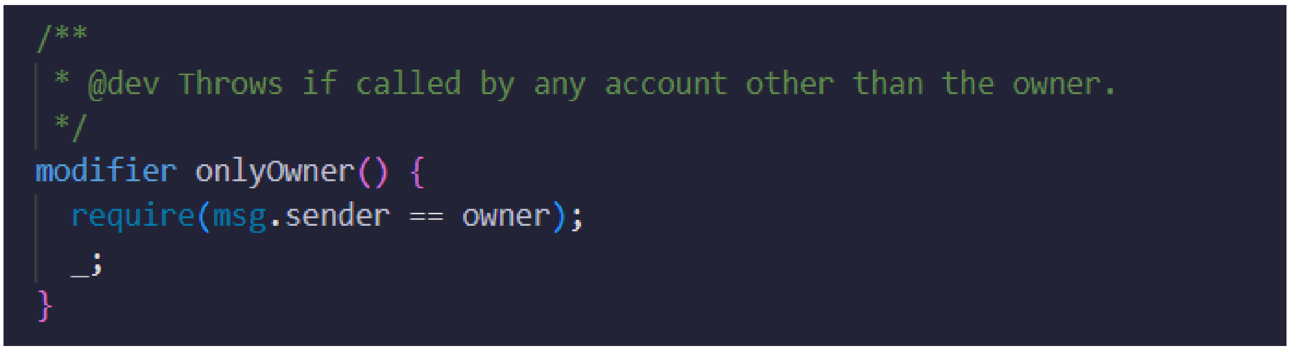
- setAElfCommunityMultisig
- setDurationOfLock
- approveMintTokens

- withdrawMintTokens

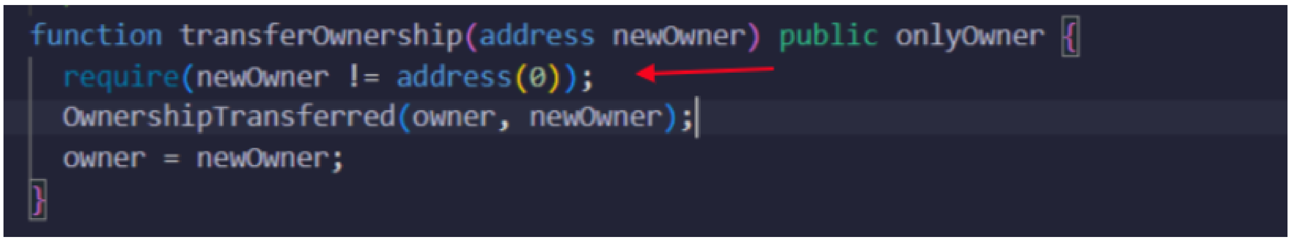
Any compromise to these accounts may allow the hacker to manipulate the project through these functions.

Resolution: We suggest carefully managing these account's private keys to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices.

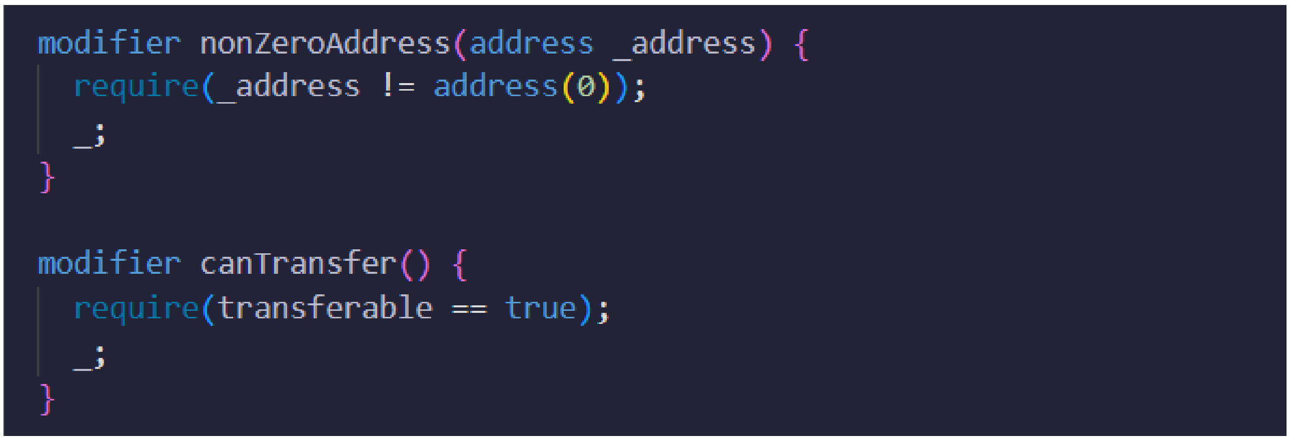
(9) Missing required error message:



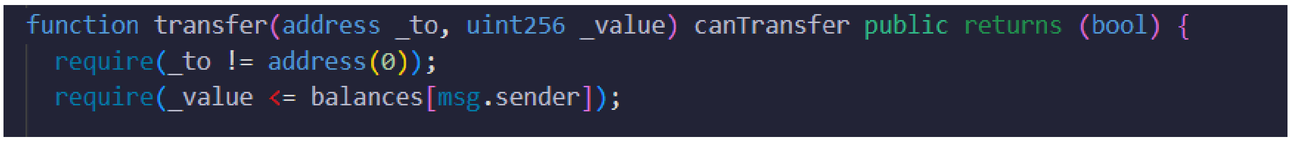
```
/**
 * @dev Throws if called by any account other than the owner.
 */
modifier onlyOwner() {
  require(msg.sender == owner);
  _;
}
```



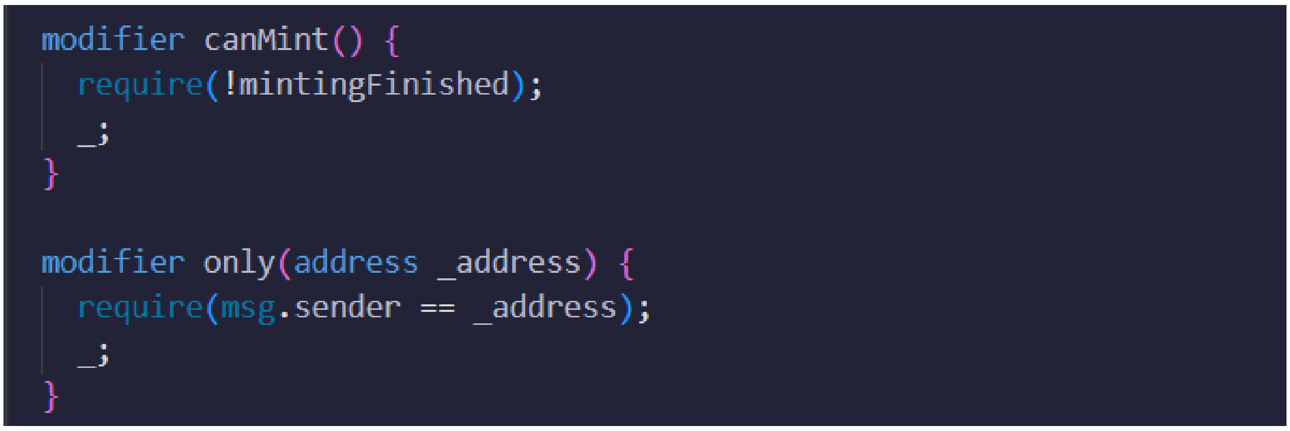
```
function transferOwnership(address newOwner) public onlyOwner {
  require(newOwner != address(0));
  OwnershipTransferred(owner, newOwner);
  owner = newOwner;
}
```



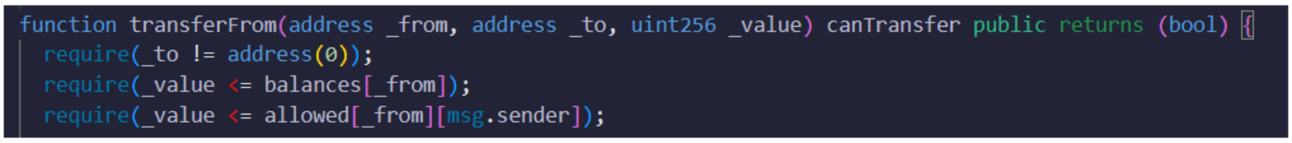
```
modifier nonZeroAddress(address _address) {
  require(_address != address(0));
  _;
}

modifier canTransfer() {
  require(transferable == true);
  _;
}
```



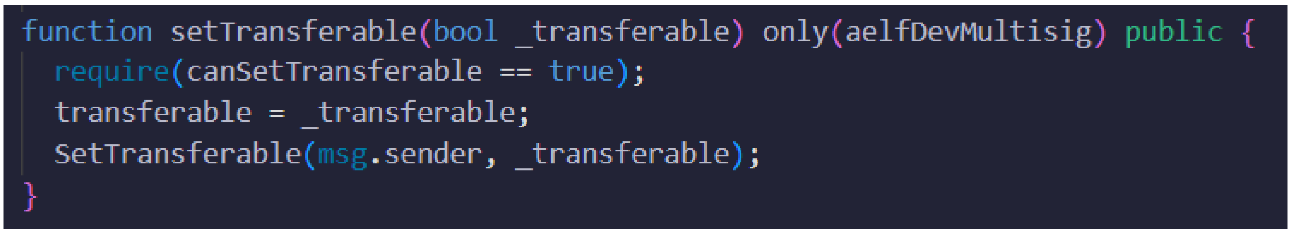
```
function transfer(address _to, uint256 _value) canTransfer public returns (bool) {
  require(_to != address(0));
  require(_value <= balances[msg.sender]);
```



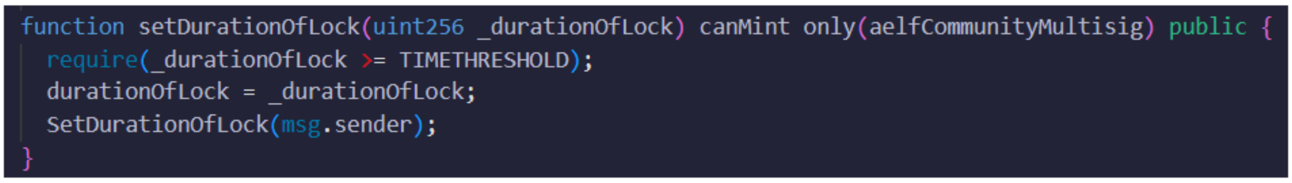
```
modifier canMint() {
  require(!mintingFinished);
  _;
}

modifier only(address _address) {
  require(msg.sender == _address);
  _;
}
```



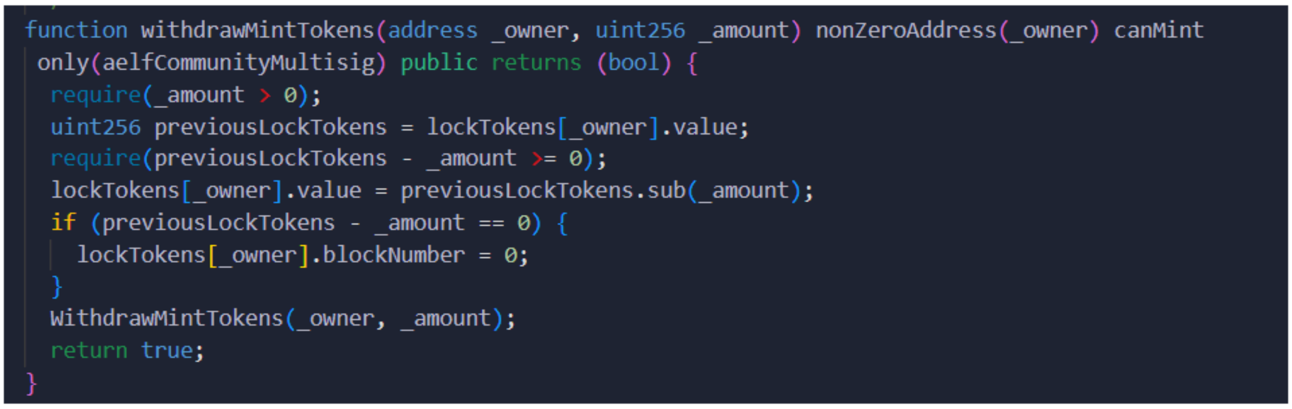
```
function transferFrom(address _from, address _to, uint256 _value) canTransfer public returns (bool) {
  require(_to != address(0));
  require(_value <= balances[_from]);
  require(_value <= allowed[_from][msg.sender]);
```



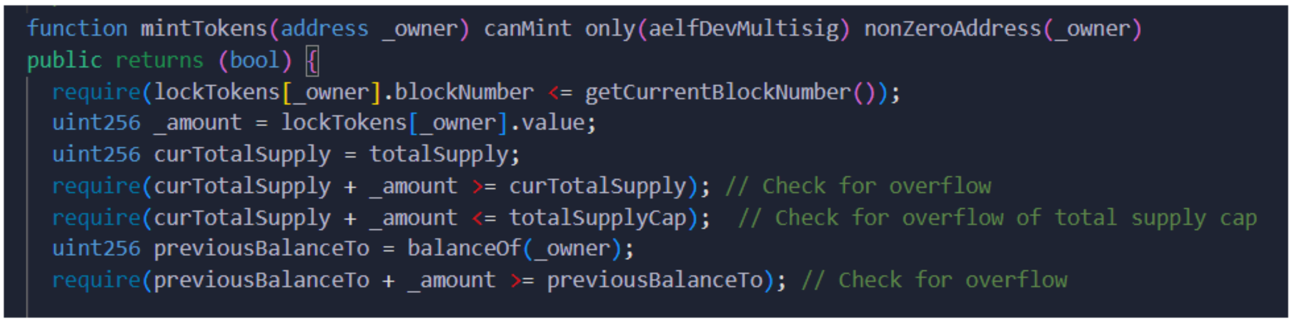
```
function setTransferable(bool _transferable) only(aelfDevMultisig) public {
  require(canSetTransferable == true);
  transferable = _transferable;
  SetTransferable(msg.sender, _transferable);
}
```



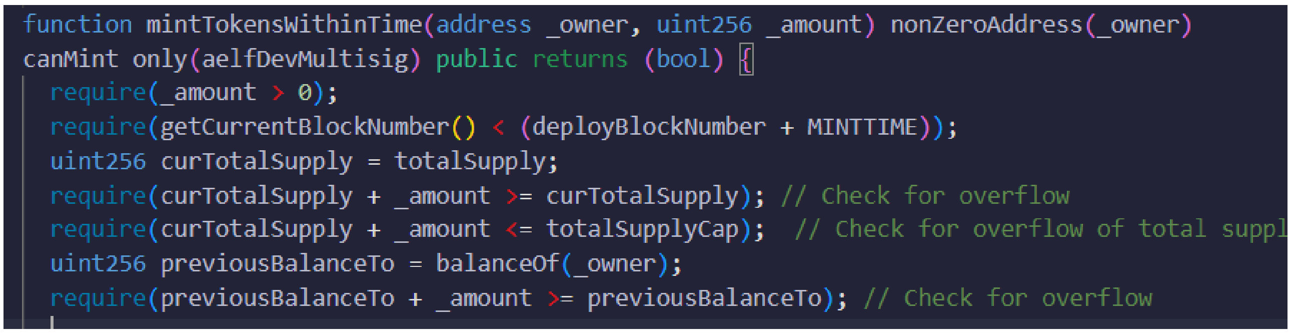
```
function setDurationOfLock(uint256 _durationOfLock) canMint only(aelfCommunityMultisig) public {
  require(_durationOfLock >= TIMETHRESHOLD);
  durationOfLock = _durationOfLock;
  SetDurationOfLock(msg.sender);
}
```



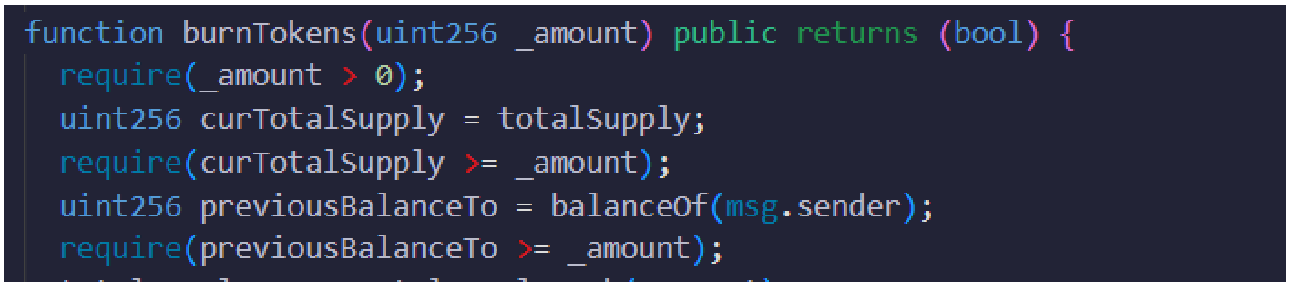
```
function withdrawMintTokens(address _owner, uint256 _amount) nonZeroAddress(_owner) canMint
 only(aelfCommunityMultisig) public returns (bool) {
  require(_amount > 0);
  uint256 previousLockTokens = lockTokens[_owner].value;
  require(previousLockTokens - _amount >= 0);
  lockTokens[_owner].value = previousLockTokens.sub(_amount);
  if (previousLockTokens - _amount == 0) {
    lockTokens[_owner].blockNumber = 0;
  }
  WithdrawMintTokens(_owner, _amount);
  return true;
}
```



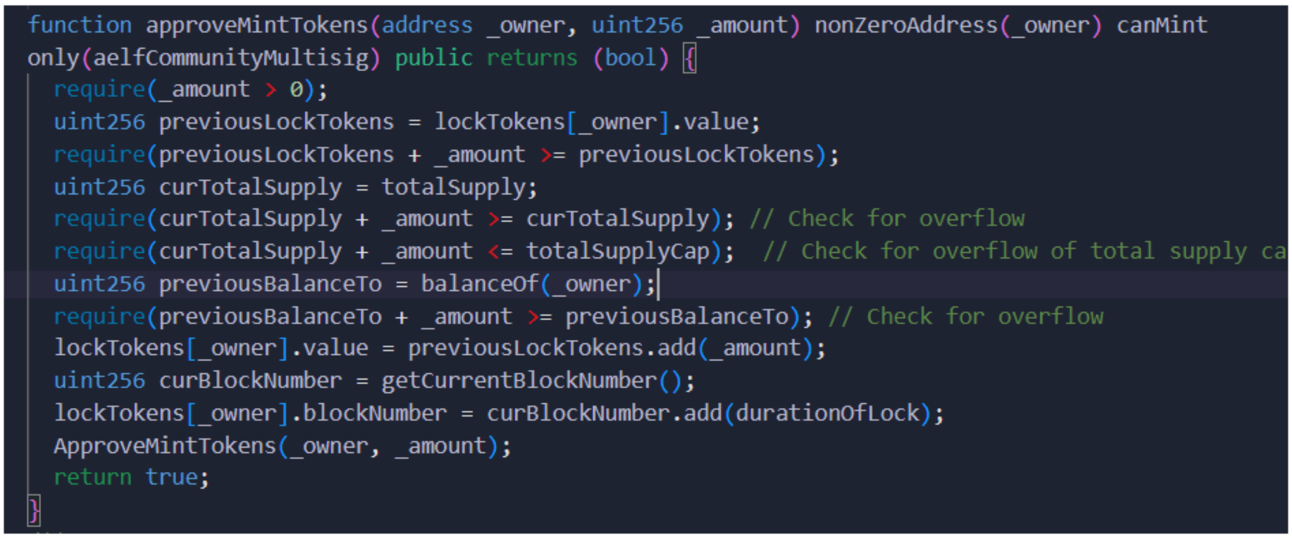
```
function mintTokens(address _owner) canMint only(aelfDevMultisig) nonZeroAddress(_owner)
public returns (bool) {
  require(lockTokens[_owner].blockNumber <= getCurrentBlockNumber());
  uint256 _amount = lockTokens[_owner].value;
  uint256 curTotalSupply = totalSupply;
  require(curTotalSupply + _amount >= curTotalSupply); // Check for overflow
  require(curTotalSupply + _amount <= totalSupplyCap);  // Check for overflow of total supply cap
  uint256 previousBalanceTo = balanceOf(_owner);
  require(previousBalanceTo + _amount >= previousBalanceTo); // Check for overflow
```



```
function mintTokensWithinTime(address _owner, uint256 _amount) nonZeroAddress(_owner)
canMint only(aelfDevMultisig) public returns (bool) {
    require(_amount > 0);
    require(getCurrentBlockNumber() < (deployBlockNumber + MINTTIME));
    uint256 curTotalSupply = totalSupply;
    require(curTotalSupply + _amount >= curTotalSupply); // Check for overflow
    require(curTotalSupply + _amount <= totalSupplyCap);  // Check for overflow of total suppl
    uint256 previousBalanceTo = balanceOf(_owner);
    require(previousBalanceTo + _amount >= previousBalanceTo); // Check for overflow
```



```
function burnTokens(uint256 _amount) public returns (bool) {
    require(_amount > 0);
    uint256 curTotalSupply = totalSupply;
    require(curTotalSupply >= _amount);
    uint256 previousBalanceTo = balanceOf(msg.sender);
    require(previousBalanceTo >= _amount);
```



```
function approveMintTokens(address _owner, uint256 _amount) nonZeroAddress(_owner) canMint
only(aelfCommunityMultisig) public returns (bool) {
    require(_amount > 0);
    uint256 previousLockTokens = lockTokens[_owner].value;
    require(previousLockTokens + _amount >= previousLockTokens);
    uint256 curTotalSupply = totalSupply;
    require(curTotalSupply + _amount >= curTotalSupply); // Check for overflow
    require(curTotalSupply + _amount <= totalSupplyCap);  // Check for overflow of total supply ca
    uint256 previousBalanceTo = balanceOf(_owner);
    require(previousBalanceTo + _amount >= previousBalanceTo); // Check for overflow
    lockTokens[_owner].value = previousLockTokens.add(_amount);
    uint256 curBlockNumber = getCurrentBlockNumber();
    lockTokens[_owner].blockNumber = curBlockNumber.add(durationOfLock);
    ApproveMintTokens(_owner, _amount);
    return true;
}
```

There is an error message required.

Resolution: We suggest setting relevant error messages to identify the failure of the transaction.

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet's private key would be compromised, then it would create trouble. The following are Admin functions:

Ownable.sol

- transferOwnership: Current owner can transfer ownership of the contract to a new account.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

Conclusion

We were given a contract code in the form of Etherscan web links. And we have used all possible tests based on given objects as files. We observed 9 Informational issues in the smart contracts. but those are not critical. So, **it's good to go for the production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover the maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

The security state of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of the systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:
In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:
Our audit techniques included manual code analysis, user interface interaction, and white box penetration testing. We look at the project's web site to get a high-level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, and then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this, we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

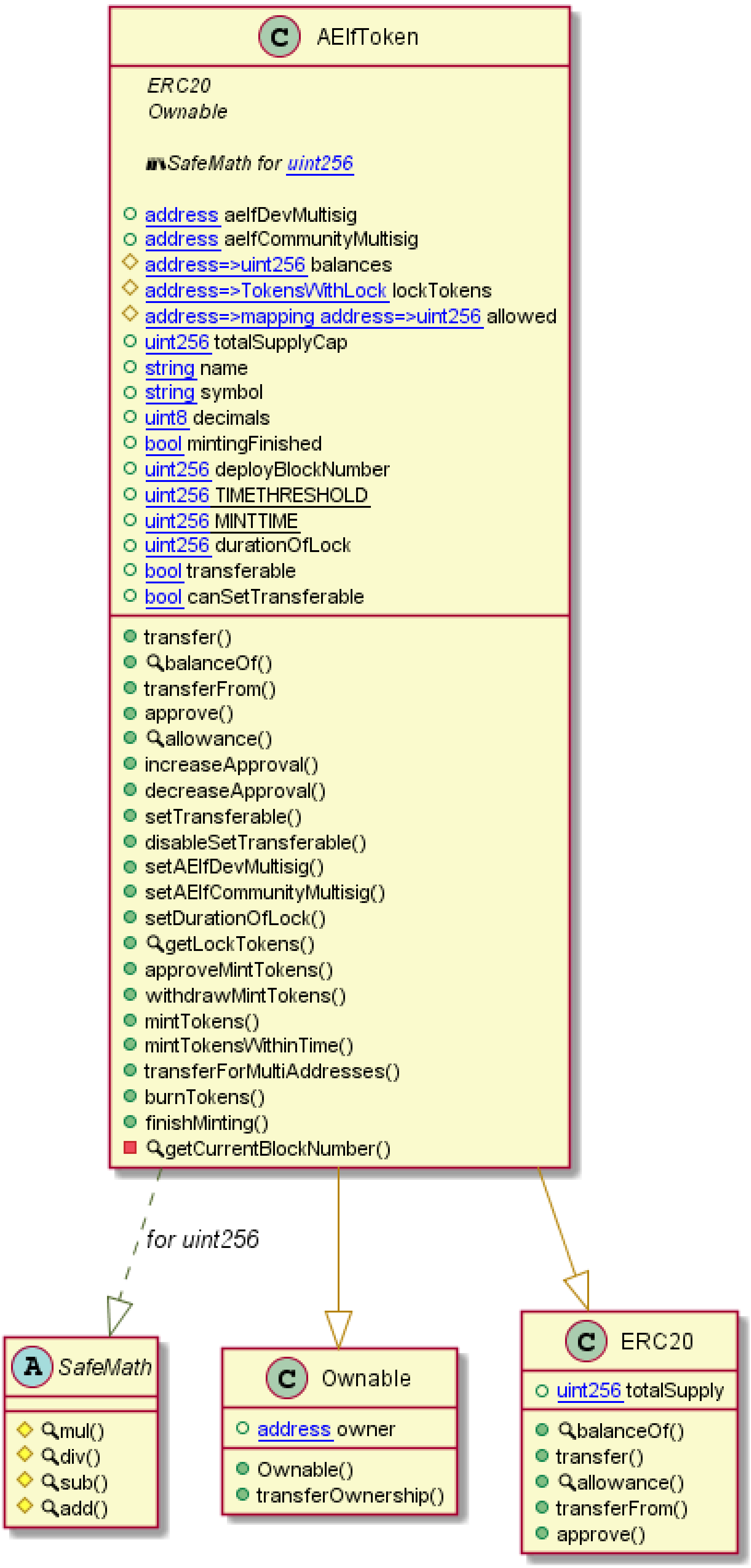
Due to the fact that the total number of test cases is unlimited, the audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee the explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - ELF Token

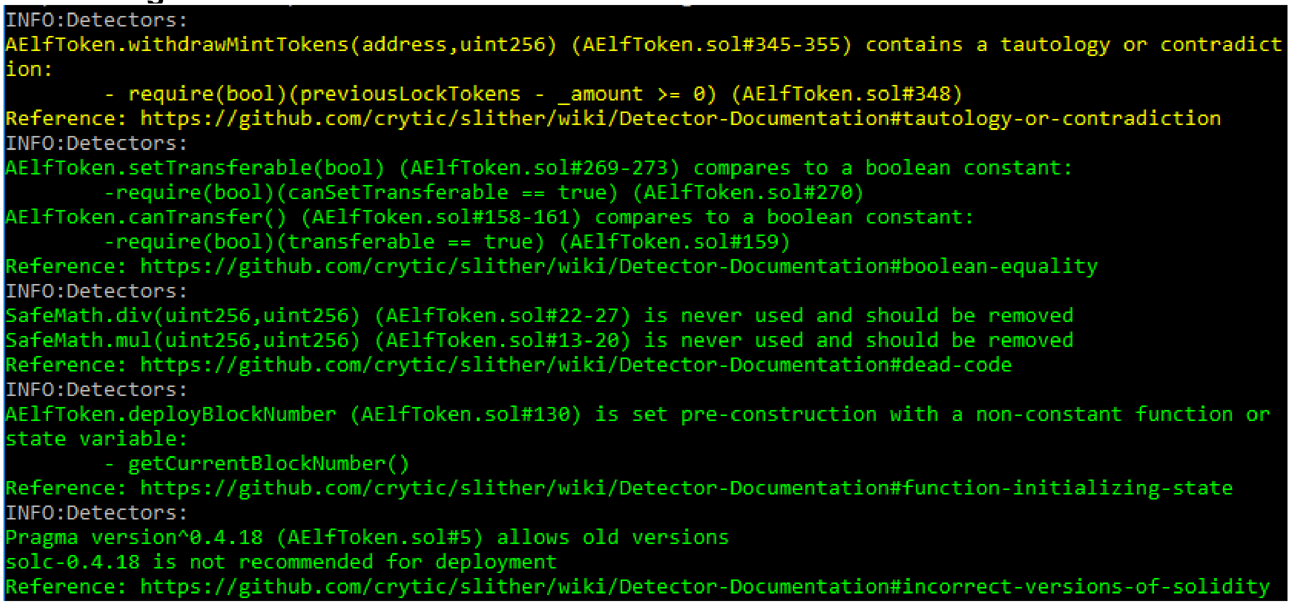


Slither Results Log

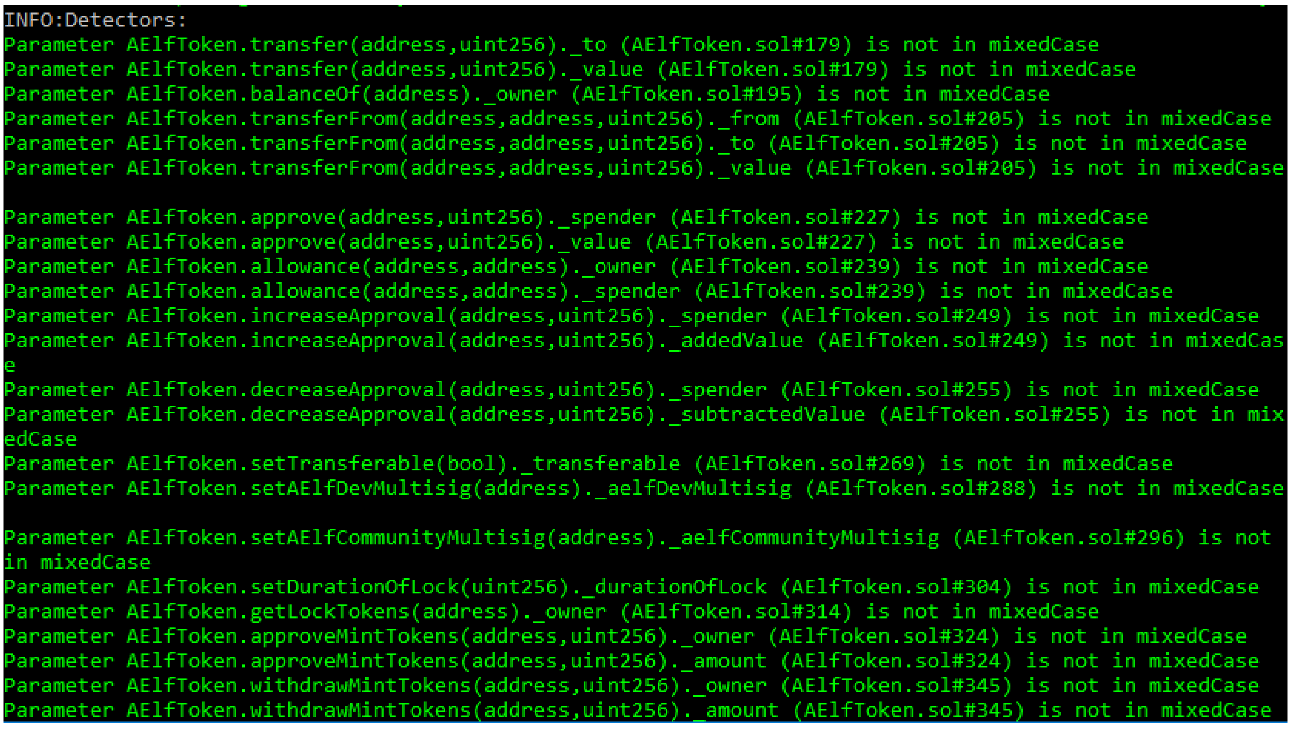
Slither is a Solidity static analysis framework that uses vulnerability detectors, displays contract details, and provides an API for writing custom analyses. It helps developers identify vulnerabilities, improve code comprehension, and prototype custom analyses quickly. The analysis includes a report with warnings and errors, allowing developers to quickly prototype and fix issues.

We did the analysis of the project altogether. Below are the results.

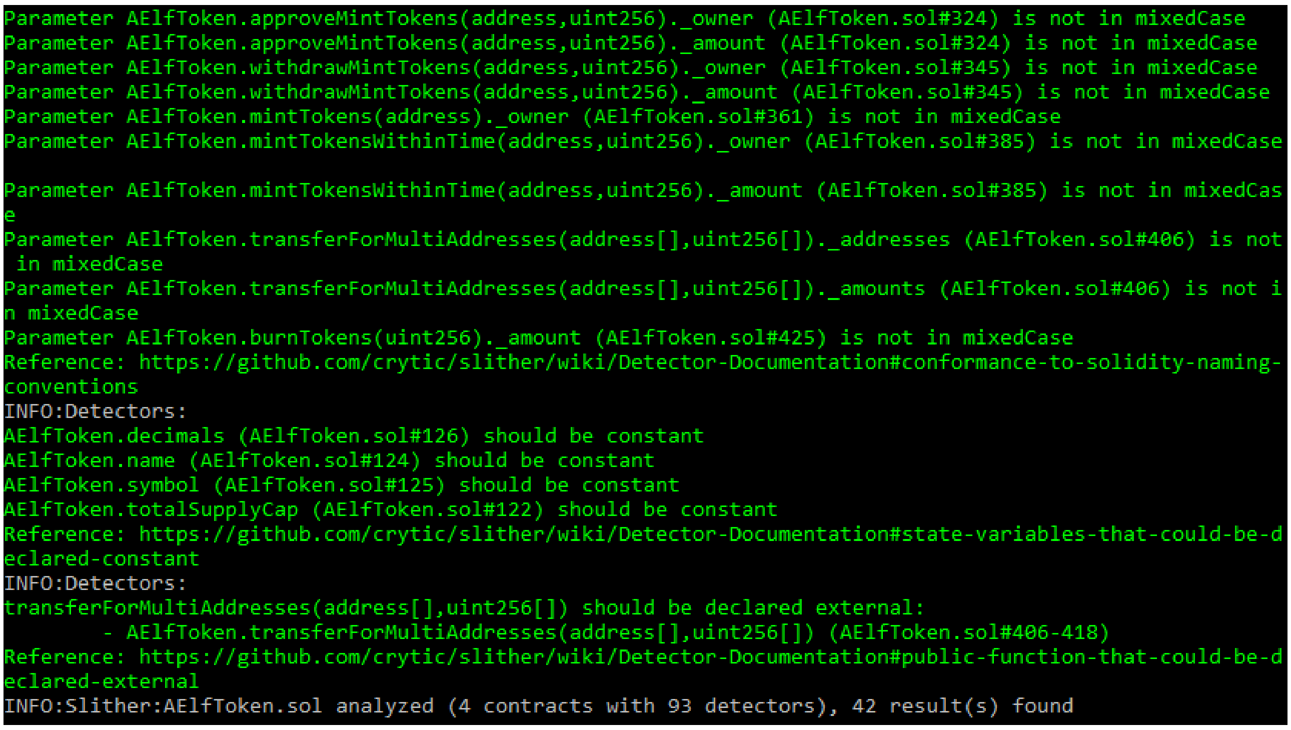
Slither Log >> AElfToken.sol



```
INFO:Detectors:
AElfToken.withdrawMintTokens(address,uint256) (AElfToken.sol#345-355) contains a tautology or contradict
ion:
        - require(bool)(previousLockTokens - _amount >= 0) (AElfToken.sol#348)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#tautology-or-contradiction
INFO:Detectors:
AElfToken.setTransferable(bool) (AElfToken.sol#269-273) compares to a boolean constant:
        -require(bool)(canSetTransferable == true) (AElfToken.sol#270)
AElfToken.canTransfer() (AElfToken.sol#158-161) compares to a boolean constant:
        -require(bool)(transferable == true) (AElfToken.sol#159)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality
INFO:Detectors:
SafeMath.div(uint256,uint256) (AElfToken.sol#22-27) is never used and should be removed
SafeMath.mul(uint256,uint256) (AElfToken.sol#13-20) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
AElfToken.deployBlockNumber (AElfToken.sol#130) is set pre-construction with a non-constant function or
state variable:
        - getCurrentBlockNumber()
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#function-initializing-state
INFO:Detectors:
Pragma version^0.4.18 (AElfToken.sol#5) allows old versions
solc-0.4.18 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```



```
INFO:Detectors:
Parameter AElfToken.transfer(address,uint256)._to (AElfToken.sol#179) is not in mixedCase
Parameter AElfToken.transfer(address,uint256)._value (AElfToken.sol#179) is not in mixedCase
Parameter AElfToken.balanceOf(address)._owner (AElfToken.sol#195) is not in mixedCase
Parameter AElfToken.transferFrom(address,address,uint256)._from (AElfToken.sol#205) is not in mixedCase
Parameter AElfToken.transferFrom(address,address,uint256)._to (AElfToken.sol#205) is not in mixedCase
Parameter AElfToken.transferFrom(address,address,uint256)._value (AElfToken.sol#205) is not in mixedCase

Parameter AElfToken.approve(address,uint256)._spender (AElfToken.sol#227) is not in mixedCase
Parameter AElfToken.approve(address,uint256)._value (AElfToken.sol#227) is not in mixedCase
Parameter AElfToken.allowance(address,address)._owner (AElfToken.sol#239) is not in mixedCase
Parameter AElfToken.allowance(address,address)._spender (AElfToken.sol#239) is not in mixedCase
Parameter AElfToken.increaseApproval(address,uint256)._spender (AElfToken.sol#249) is not in mixedCase
Parameter AElfToken.increaseApproval(address,uint256)._addedValue (AElfToken.sol#249) is not in mixedCas
e
Parameter AElfToken.decreaseApproval(address,uint256)._spender (AElfToken.sol#255) is not in mixedCase
Parameter AElfToken.decreaseApproval(address,uint256)._subtractedValue (AElfToken.sol#255) is not in mix
edCase
Parameter AElfToken.setTransferable(bool)._transferable (AElfToken.sol#269) is not in mixedCase
Parameter AElfToken.setAElfDevMultisig(address)._aelfDevMultisig (AElfToken.sol#288) is not in mixedCase

Parameter AElfToken.setAElfCommunityMultisig(address)._aelfCommunityMultisig (AElfToken.sol#296) is not
in mixedCase
Parameter AElfToken.setDurationOfLock(uint256)._durationOfLock (AElfToken.sol#304) is not in mixedCase
Parameter AElfToken.getLockTokens(address)._owner (AElfToken.sol#314) is not in mixedCase
Parameter AElfToken.approveMintTokens(address,uint256)._owner (AElfToken.sol#324) is not in mixedCase
Parameter AElfToken.approveMintTokens(address,uint256)._amount (AElfToken.sol#324) is not in mixedCase
Parameter AElfToken.withdrawMintTokens(address,uint256)._owner (AElfToken.sol#345) is not in mixedCase
Parameter AElfToken.withdrawMintTokens(address,uint256)._amount (AElfToken.sol#345) is not in mixedCase
```

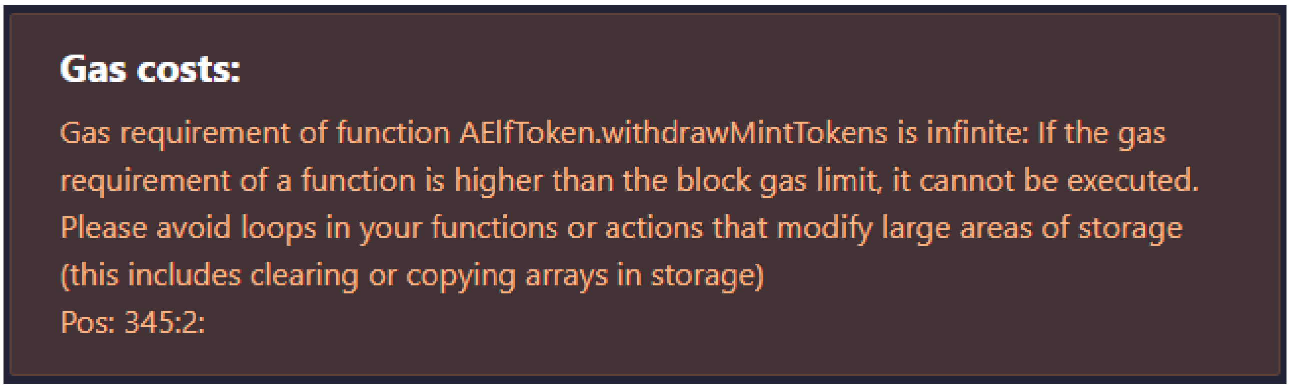


```
Parameter AElfToken.approveMintTokens(address,uint256)._owner (AElfToken.sol#324) is not in mixedCase
Parameter AElfToken.approveMintTokens(address,uint256)._amount (AElfToken.sol#324) is not in mixedCase
Parameter AElfToken.withdrawMintTokens(address,uint256)._owner (AElfToken.sol#345) is not in mixedCase
Parameter AElfToken.withdrawMintTokens(address,uint256)._amount (AElfToken.sol#345) is not in mixedCase
Parameter AElfToken.mintTokens(address)._owner (AElfToken.sol#361) is not in mixedCase
Parameter AElfToken.mintTokensWithinTime(address,uint256)._owner (AElfToken.sol#385) is not in mixedCase

Parameter AElfToken.mintTokensWithinTime(address,uint256)._amount (AElfToken.sol#385) is not in mixedCas
e
Parameter AElfToken.transferForMultiAddresses(address[],uint256[])._addresses (AElfToken.sol#406) is not
 in mixedCase
Parameter AElfToken.transferForMultiAddresses(address[],uint256[])._amounts (AElfToken.sol#406) is not i
n mixedCase
Parameter AElfToken.burnTokens(uint256)._amount (AElfToken.sol#425) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-
conventions
INFO:Detectors:
AElfToken.decimals (AElfToken.sol#126) should be constant
AElfToken.name (AElfToken.sol#124) should be constant
AElfToken.symbol (AElfToken.sol#125) should be constant
AElfToken.totalSupplyCap (AElfToken.sol#122) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-d
eclared-constant
INFO:Detectors:
transferForMultiAddresses(address[],uint256[]) should be declared external:
        - AElfToken.transferForMultiAddresses(address[],uint256[]) (AElfToken.sol#406-418)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-d
eclared-external
INFO:Slither:AElfToken.sol analyzed (4 contracts with 93 detectors), 42 result(s) found
```

Solidity Static Analysis

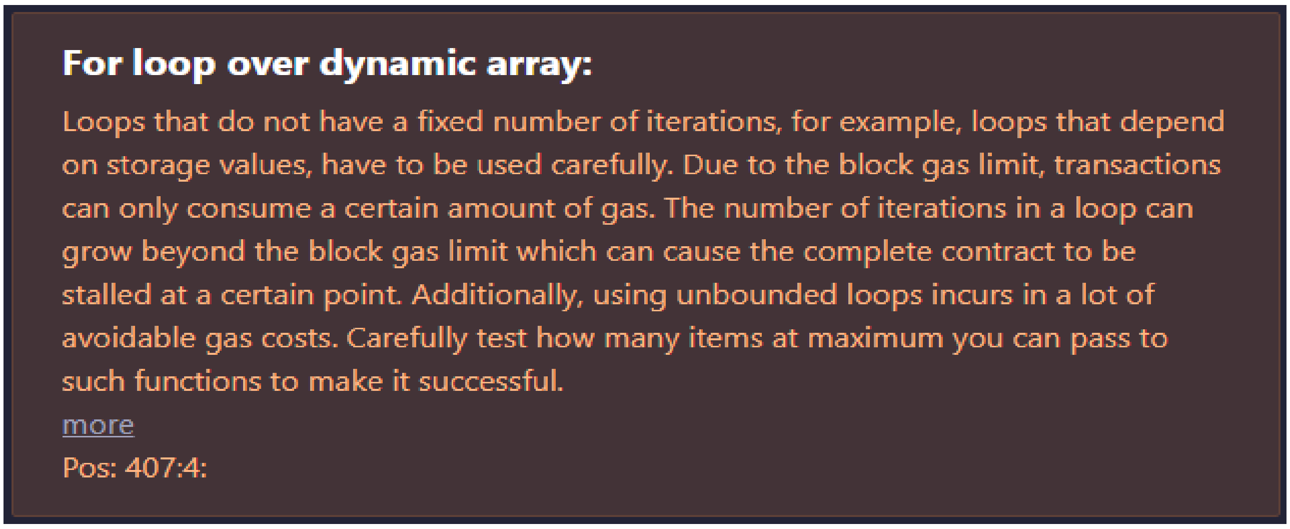
Static code analysis is used to identify many common coding problems before a program is released. It involves examining the code manually or using tools to automate the process. Static code analysis tools can automatically scan the code without executing it.

AElfToken.sol



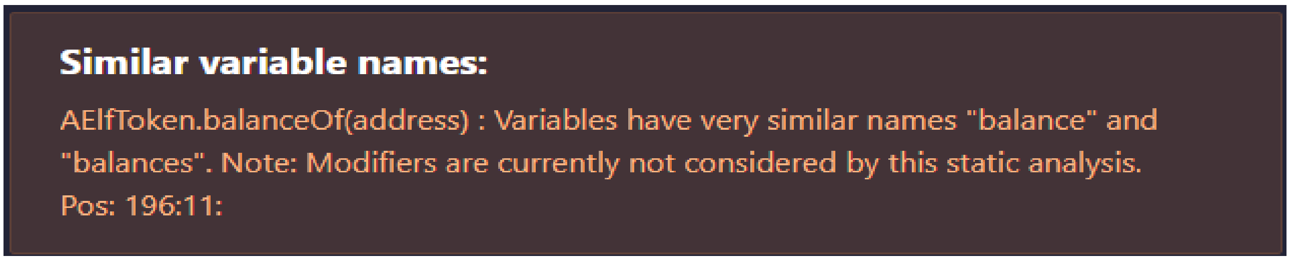
Gas costs:

Gas requirement of function AElfToken.withdrawMintTokens 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)
Pos: 345:2:



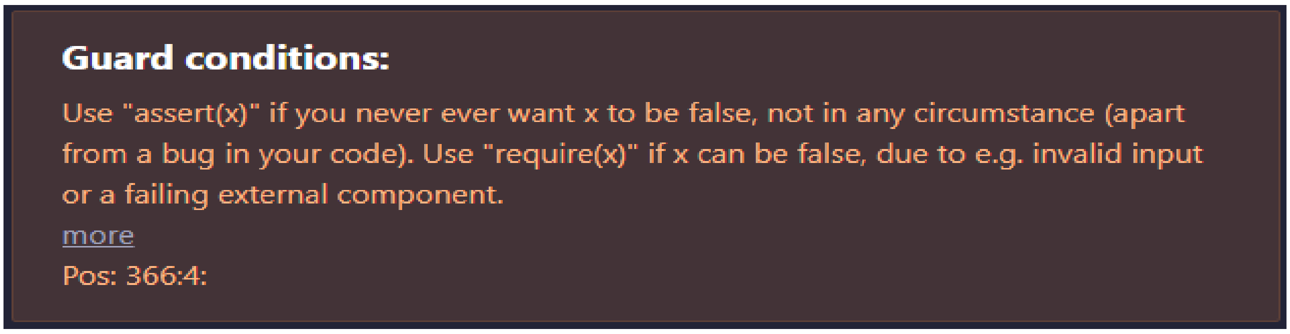
For loop over dynamic array:

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.
more
Pos: 407:4:



Similar variable names:

AElfToken.balanceOf(address) : Variables have very similar names "balance" and "balances". Note: Modifiers are currently not considered by this static analysis.
Pos: 196:11:



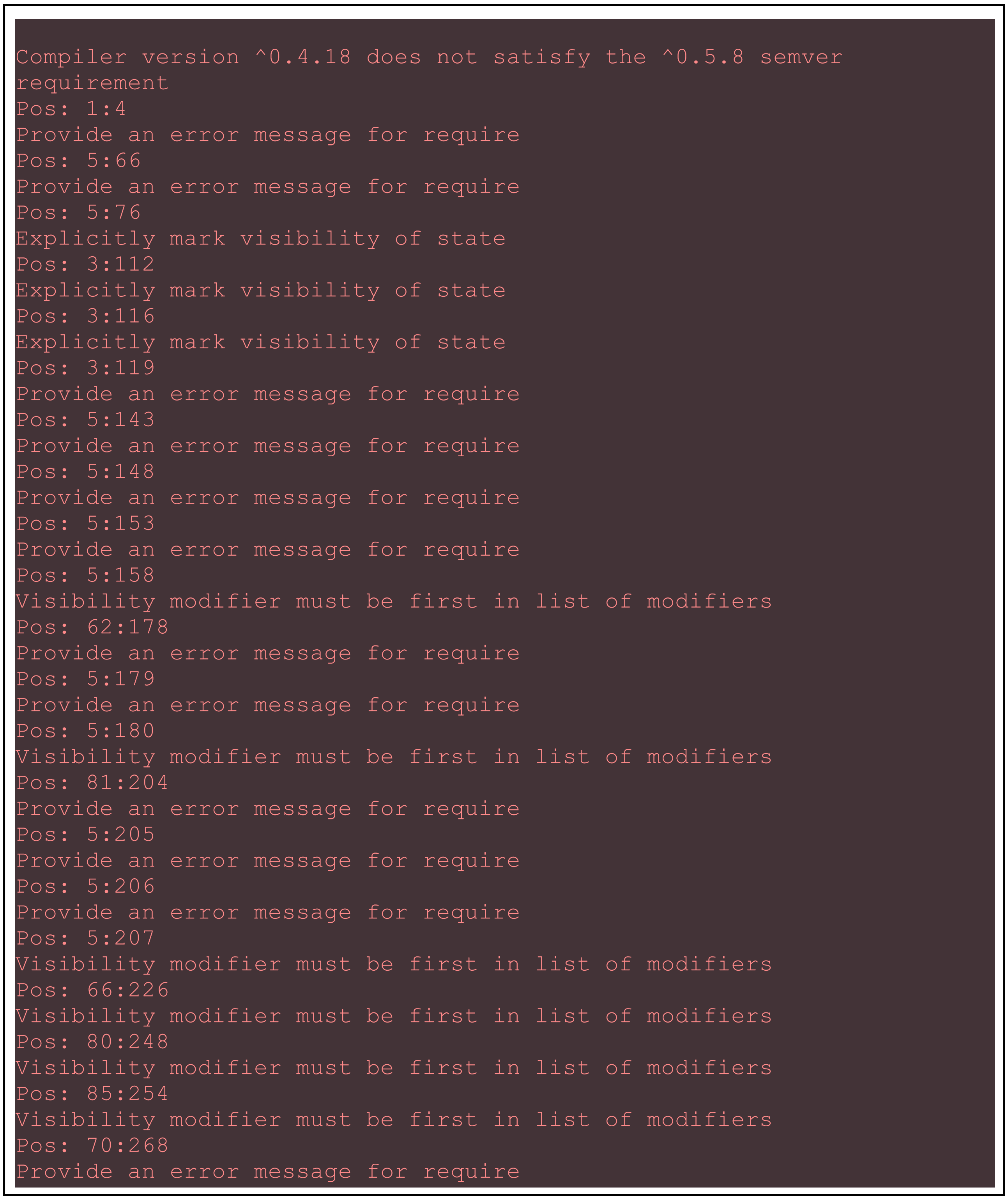
Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.
more
Pos: 366:4:

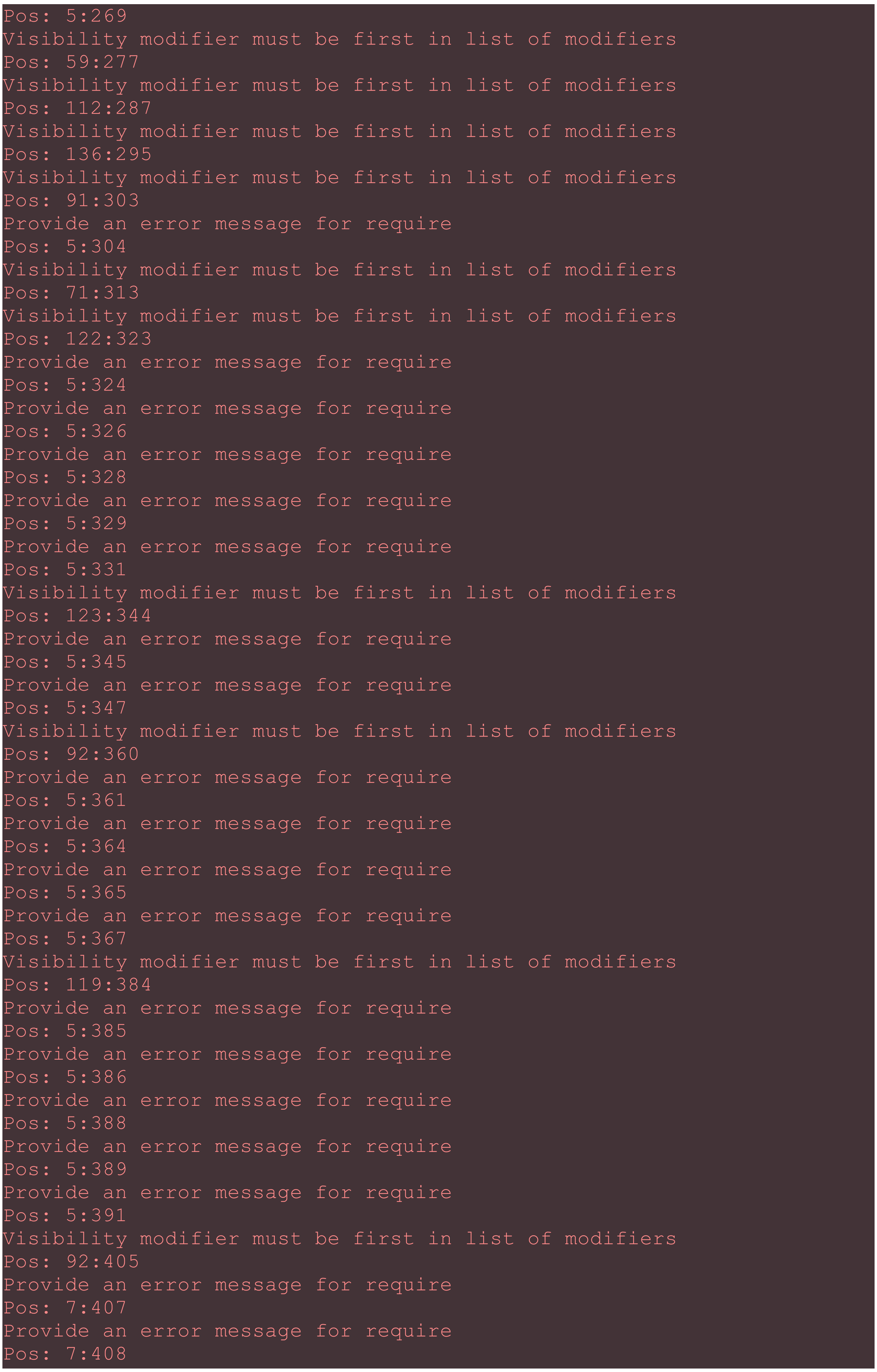
Solhint Linter

Linters are the utility tools that analyze the given source code and report programming errors, bugs, and stylistic errors. For the Solidity language, there are some linter tools available that a developer can use to improve the quality of their Solidity contracts.

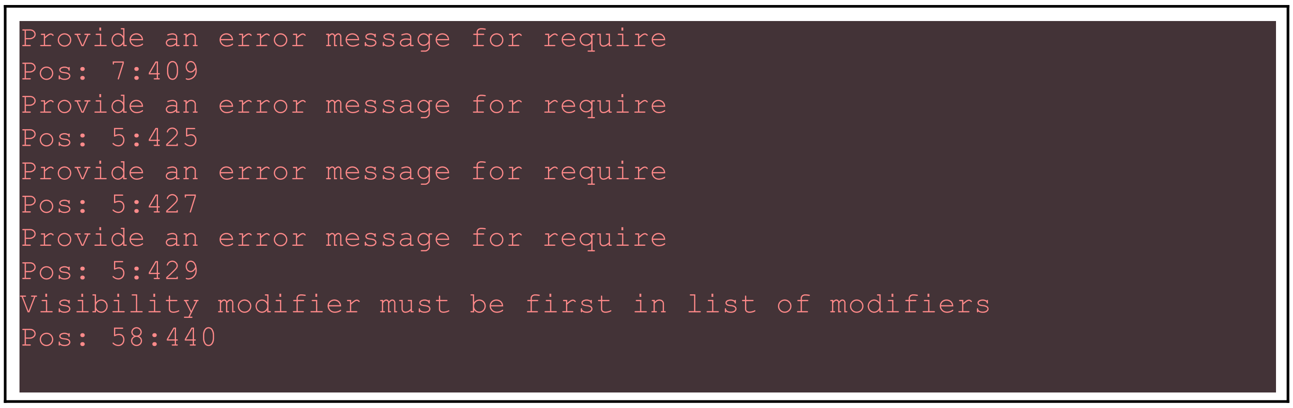
AElfToken.sol



```
Compiler version ^0.4.18 does not satisfy the ^0.5.8 semver
requirement
Pos: 1:4
Provide an error message for require
Pos: 5:66
Provide an error message for require
Pos: 5:76
Explicitly mark visibility of state
Pos: 3:112
Explicitly mark visibility of state
Pos: 3:116
Explicitly mark visibility of state
Pos: 3:119
Provide an error message for require
Pos: 5:143
Provide an error message for require
Pos: 5:148
Provide an error message for require
Pos: 5:153
Provide an error message for require
Pos: 5:158
Visibility modifier must be first in list of modifiers
Pos: 62:178
Provide an error message for require
Pos: 5:179
Provide an error message for require
Pos: 5:180
Visibility modifier must be first in list of modifiers
Pos: 81:204
Provide an error message for require
Pos: 5:205
Provide an error message for require
Pos: 5:206
Provide an error message for require
Pos: 5:207
Visibility modifier must be first in list of modifiers
Pos: 66:226
Visibility modifier must be first in list of modifiers
Pos: 80:248
Visibility modifier must be first in list of modifiers
Pos: 85:254
Visibility modifier must be first in list of modifiers
Pos: 70:268
Provide an error message for require
```



```
Pos: 5:269
Visibility modifier must be first in list of modifiers
Pos: 59:277
Visibility modifier must be first in list of modifiers
Pos: 112:287
Visibility modifier must be first in list of modifiers
Pos: 136:295
Visibility modifier must be first in list of modifiers
Pos: 91:303
Provide an error message for require
Pos: 5:304
Visibility modifier must be first in list of modifiers
Pos: 71:313
Visibility modifier must be first in list of modifiers
Pos: 122:323
Provide an error message for require
Pos: 5:324
Provide an error message for require
Pos: 5:326
Provide an error message for require
Pos: 5:328
Provide an error message for require
Pos: 5:329
Provide an error message for require
Pos: 5:331
Visibility modifier must be first in list of modifiers
Pos: 123:344
Provide an error message for require
Pos: 5:345
Provide an error message for require
Pos: 5:347
Visibility modifier must be first in list of modifiers
Pos: 92:360
Provide an error message for require
Pos: 5:361
Provide an error message for require
Pos: 5:364
Provide an error message for require
Pos: 5:365
Provide an error message for require
Pos: 5:367
Visibility modifier must be first in list of modifiers
Pos: 119:384
Provide an error message for require
Pos: 5:385
Provide an error message for require
Pos: 5:386
Provide an error message for require
Pos: 5:388
Provide an error message for require
Pos: 5:389
Provide an error message for require
Pos: 5:391
Visibility modifier must be first in list of modifiers
Pos: 92:405
Provide an error message for require
Pos: 7:407
Provide an error message for require
Pos: 7:408
```



```
Provide an error message for require
Pos: 7:409
Provide an error message for require
Pos: 5:425
Provide an error message for require
Pos: 5:427
Provide an error message for require
Pos: 5:429
Visibility modifier must be first in list of modifiers
Pos: 58:440
```

Software analysis result:

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

EtherAuthority