ALINK.A

# Smart Contract Audit Report

SYSFIXED

# TABLE OF CONTENTS

# AUDITED DETAILS

## Audited Project

| Project name | Token ticker | Blockchain |
|---|---|---|
| ALINK.A | ALINK | Binance Smart Chain |

## Addresses

| Contract address | 0x78e624070871831842730b43f77467af3e8b580c |
|---|---|
| Contract deployer address | 0x0fde63927d31beb6689E1307F7544aC9B3d0984a |

## Project Website

https://alink.ai/

## Codebase

https://bscscan.com/address/0x78e624070871831842730b43f77467af3e8b580c#code

# SUMMARY

ALINK is an open and decentralized platform that provides access to AI services through blockchain technology. The network allows developers to publish their AI services and make them available to anyone with an internet connection. These services can range from simple algorithms to complete end-to-end solutions. They can be used across various domains, including image/video, speech, text, time series, bio-AI, and network analysis. Developers can also create autonomous AI agents that can interact with other services on the network. Payments for these services can be made using the native ALINK token. The ALINK platform has several critical components that work together to facilitate the decentralized network of AI services. The design of these components is based on creating a functional, scalable, and extensible system that is open and compliant with regulatory and legal requirements. To minimize dependence on the blockchain, the platform uses tools to abstract all blockchain interactions and implements a multi-party escrow contract and unidirectional atomic channels for payments. In addition, the platform abstracts as much of the network as possible to simplify the process of providing AI services through the web. This is achieved using a single exible tool, the daemon, which provides scalability, robustness, distribution, and management features to the entire community. To maintain compliance with regulations while still being open, the platform has implemented a separate marketplace separate from the fully decentralized registry of AI services currently available on the blockchain.

## ▌Contract Summary

**Documentation Quality**

ALINK.A provides a very good documentation with standard of solidity base code.

- The technical description is provided clearly and structured and also dont have any high risk issue.

**Code Quality**

The Overall quality of the basecode is standard.

- Standard solidity basecode and rules are already followed by ALINK.A with the discovery of several low issues.

**Test Coverage**

Test coverage of the project is 100% ( Through Codebase )

## ▌Audit Findings Summary

- SWC-100 SWC-108 | Explicitly define visibility for all state variables on lines 96 and 97.

- SWC-103 | Pragma statements can be allowed to float when a contract is intended on lines 5.
- SWC-107 | It is recommended to use a reentrancy lock, reentrancy weaknesses detected on lines 209

- SWC-110 SWC-123 | It is recommended to use of revert(), assert(), and require() in Solidity, and the new REVERT opcode in the EVM on lines 209 and 192.
- SWC-111 | It is recommended to use alternatives to the deprecated constructions on lines 35, 36, 37, 116, 124 and 179.

# CONCLUSION

We have audited the ALINK.A project released on February 2023 to discover issues and identify potential security vulnerabilities in ALINK.A Project. This process is used to find technical issues and security loopholes which might be found in the smart contract.

The security audit report provides satisfactory results with low-risk issues.

The issues found in the ALINK.A smart contract code does not pose a considerable risk. The writing of the contract is close to the standard of writing contracts in general. The low-risk issues are some floating pragma is set, a call to a user-supplied address is executed, the "constant" state mutability modifier is deprecated, and requirement violation. A floating pragma is set, current pragma Solidity directive is ""^0.4.24"". Specifying a fixed compiler version is recommended to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code. Requirement violation, the Requirement was violated in a nested call, and the call was reverted as a result. Ensure valid inputs are provided to the nested call (for instance, via passed arguments). Use of the "constant" state mutability modifier is deprecated, use of "constant" as a state mutability modifier in function "totalSupply" is disallowed as of Solidity version 0.5.0. Use "view" instead.

# AUDIT RESULT

| Article | Category | Description | Result |
|---|---|---|---|
| Default Visibility | SWC-100 SWC-108 | Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously. | ISSUE FOUND |
| Integer Overflow and Underflow | SWC-101 | If unchecked math is used, all math operations should be safe from overflows and underflows. | PASS |
| Outdated Compiler Version | SWC-102 | It is recommended to use a recent version of the Solidity compiler. | PASS |
| Floating Pragma | SWC-103 | Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly. | ISSUE FOUND |
| Unchecked Call Return Value | SWC-104 | The return value of a message call should be checked. | PASS |
| Unprotected Ether Withdrawal | SWC-105 | Due to missing or insufficient access controls, malicious parties can withdraw from the contract. | PASS |
| SELFDESTRUCT Instruction | SWC-106 | The contract should not be self-destructible while it has funds belonging to users. | PASS |
| Reentrancy | SWC-107 | Check effect interaction pattern should be followed if the code performs recursive call. | ISSUE FOUND |
| Uninitialized Storage Pointer | SWC-109 | Uninitialized local storage variables can point to unexpected storage locations in the contract. | PASS |
| Assert Violation | SWC-110 SWC-123 | Properly functioning code should never reach a failing assert statement. | ISSUE FOUND |
| Deprecated Solidity Functions | SWC-111 | Deprecated built-in functions should never be used. | ISSUE FOUND |
| Delegate call to Untrusted Callee | SWC-112 | Delegatecalls should only be allowed to trusted addresses. | PASS |

| DoS (Denial of Service) | SWC-113 SWC-128 | Execution of the code should never be blocked by a specific contract state unless required. | PASS |
|---|---|---|---|
| Race Conditions | SWC-114 | Race Conditions and Transactions Order Dependency should not be possible. | PASS |
| Authorization through tx.origin | SWC-115 | tx.origin should not be used for authorization. | PASS |
| Block values as a proxy for time | SWC-116 | Block numbers should not be used for time calculations. | PASS |
| Signature Unique ID | SWC-117 SWC-121 SWC-122 | Signed messages should always have a unique id. A transaction hash should not be used as a unique id. | PASS |
| Incorrect Constructor Name | SWC-118 | Constructors are special functions that are called only once during the contract creation. | PASS |
| Shadowing State Variable | SWC-119 | State variables should not be shadowed. | PASS |
| Weak Sources of Randomness | SWC-120 | Random values should never be generated from Chain Attributes or be predictable. | PASS |
| Write to Arbitrary Storage Location | SWC-124 | The contract is responsible for ensuring that only authorized user or contract accounts may write to sensitive storage locations. | PASS |
| Incorrect Inheritance Order | SWC-125 | When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order. The rule of thumb is to inherit contracts from more /general/ to more /specific/. | PASS |
| Insufficient Gas Griefing | SWC-126 | Insufficient gas griefing attacks can be performed on contracts which accept data and use it in a sub-call on another contract. | PASS |
| Arbitrary Jump Function | SWC-127 | As Solidity doesnt support pointer arithmetics, it is impossible to change such variable to an arbitrary value. | PASS |

| | | | |
|---|---|---|---|
| Typographical Error | SWC-129 | A typographical error can occur for example when the intent of a defined operation is to sum a number to a variable. | PASS |
| Override control character | SWC-130 | Malicious actors can use the Right-To-Left-Override unicode character to force RTL text rendering and confuse users as to the real intent of a contract. | PASS |
| Unused variables | SWC-131 SWC-135 | Unused variables are allowed in Solidity and they do not pose a direct security issue. | PASS |
| Unexpected Ether balance | SWC-132 | Contracts can behave erroneously when they strictly assume a specific Ether balance. | PASS |
| Hash Collisions Variable | SWC-133 | Using abi.encodePacked() with multiple variable length arguments can, in certain situations, lead to a hash collision. | PASS |
| Hardcoded gas amount | SWC-134 | The transfer() and send() functions forward a fixed amount of 2300 gas. | PASS |
| Unencrypted Private Data | SWC-136 | It is a common misconception that private type variables cannot be read. | PASS |

# SMART CONTRACT ANALYSIS

| Started | Tuesday Feb 07 2023 05:12:50 GMT+0000 (Coordinated Universal Time) |
|---|---|
| Finished | Wednesday Feb 08 2023 13:13:40 GMT+0000 (Coordinated Universal Time) |
| Mode | Standard |
| Main Source File | ALINK.sol |

## Detected Issues

| ID | Title | Severity | Status |
|---|---|---|---|
| SWC-103 | A FLOATING PRAGMA IS SET. | low | acknowledged |
| SWC-107 | A CALL TO A USER-SUPPLIED ADDRESS IS EXECUTED. | low | acknowledged |
| SWC-107 | A CALL TO A USER-SUPPLIED ADDRESS IS EXECUTED. | low | acknowledged |
| SWC-108 | STATE VARIABLE VISIBILITY IS NOT SET. | low | acknowledged |
| SWC-108 | STATE VARIABLE VISIBILITY IS NOT SET. | low | acknowledged |
| SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED. | low | acknowledged |
| SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED. | low | acknowledged |
| SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED. | low | acknowledged |
| SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED. | low | acknowledged |
| SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED. | low | acknowledged |
| SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED. | low | acknowledged |

| SWC-123 | REQUIREMENT VIOLATION. | low | acknowledged |
|---------|------------------------|-----|--------------|
| SWC-123 | REQUIREMENT VIOLATION. | low | acknowledged |

# SWC-103 | A FLOATING PRAGMA IS SET.
LINE 5

## low SEVERITY

The current pragma Solidity directive is ""^0.4.24"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

## Source File

- ALINK.sol

## Locations

```
4
5    pragma solidity ^0.4.24;
6
7
8    // ---------------------------------------------------------------------------
9
```

# SWC-107 | A CALL TO A USER-SUPPLIED ADDRESS IS EXECUTED.

LINE 209

## low SEVERITY

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

## Source File

- ALINK.sol

## Locations

```
208    function transferAnyERC20Token(address tokenAddress, uint tokens) public onlyOwner
returns (bool success) {
209    return ERC20Interface(tokenAddress).transfer(owner, tokens);
210    }
211    }
212
```

# SWC-107 | A CALL TO A USER-SUPPLIED ADDRESS IS EXECUTED.
LINE 192

## low SEVERITY

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

## Source File

- ALINK.sol

## Locations

```
191    emit Approval(msg.sender, spender, tokens);
192    ApproveAndCallFallBack(spender).receiveApproval(msg.sender, tokens, this, data);
193    return true;
194    }
195
196
```

# SWC-108 | STATE VARIABLE VISIBILITY IS NOT SET.
LINE 96

## low SEVERITY

It is best practice to set the visibility of state variables explicitly. The default visibility for "balances" is internal. Other possible visibility settings are public and private.

## Source File

- ALINK.sol

## Locations

```
95
96   mapping(address => uint) balances;
97   mapping(address => mapping(address => uint)) allowed;
98
99
100
```

# SWC-108 | STATE VARIABLE VISIBILITY IS NOT SET.
LINE 97

## low SEVERITY

It is best practice to set the visibility of state variables explicitly. The default visibility for "allowed" is internal. Other possible visibility settings are public and private.

## Source File

- ALINK.sol

## Locations

```
96    mapping(address => uint) balances;
97    mapping(address => mapping(address => uint)) allowed;
98
99
100   // -----------------------------------------------------------------------
101
```

# SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.

LINE 35

## low SEVERITY

Using "constant" as a state mutability modifier in function "totalSupply" is disallowed as of Solidity version 0.5.0. Use "view" instead.

## Source File

- ALINK.sol

## Locations

```
34    contract ERC20Interface {
35    function totalSupply() public constant returns (uint);
36    function balanceOf(address tokenOwner) public constant returns (uint balance);
37    function allowance(address tokenOwner, address spender) public constant returns
(uint remaining);
38    function transfer(address to, uint tokens) public returns (bool success);
39
```

# SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.

LINE 36

## low SEVERITY

Using "constant" as a state mutability modifier in function "balanceOf" is disallowed as of Solidity version 0.5.0. Use "view" instead.

## Source File

- ALINK.sol

## Locations

```
35    function totalSupply() public constant returns (uint);
36    function balanceOf(address tokenOwner) public constant returns (uint balance);
37    function allowance(address tokenOwner, address spender) public constant returns
(uint remaining);
38    function transfer(address to, uint tokens) public returns (bool success);
39    function approve(address spender, uint tokens) public returns (bool success);
40
```

# SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.

LINE 37

## low SEVERITY

Using "constant" as a state mutability modifier in function "allowance" is disallowed as of Solidity version 0.5.0. Use "view" instead.

## Source File

- ALINK.sol

## Locations

```
36    function balanceOf(address tokenOwner) public constant returns (uint balance);
37    function allowance(address tokenOwner, address spender) public constant returns
(uint remaining);
38    function transfer(address to, uint tokens) public returns (bool success);
39    function approve(address spender, uint tokens) public returns (bool success);
40    function transferFrom(address from, address to, uint tokens) public returns (bool
success);
41
```

# SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.

LINE 116

## low SEVERITY

Using "constant" as a state mutability modifier in function "totalSupply" is disallowed as of Solidity version 0.5.0. Use "view" instead.

## Source File

- ALINK.sol

## Locations

```
115    // ----------------------------------------------------------------------
116    function totalSupply() public constant returns (uint) {
117    return _totalSupply  - balances[address(0)];
118    }
119
120
```

**SYSFIXED**

# SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.

LINE 124

## low SEVERITY

Using "constant" as a state mutability modifier in function "balanceOf" is disallowed as of Solidity version 0.5.0. Use "view" instead.

## Source File

- ALINK.sol

## Locations

```
123    // ----------------------------------------------------------------------
124    function balanceOf(address tokenOwner) public constant returns (uint balance) {
125    return balances[tokenOwner];
126    }
127
128
```

# SWC-111 | USE OF THE "CONSTANT" STATE MUTABILITY MODIFIER IS DEPRECATED.

LINE 179

## low SEVERITY

Using "constant" as a state mutability modifier in function "allowance" is disallowed as of Solidity version 0.5.0. Use "view" instead.

## Source File

- ALINK.sol

## Locations

```
178   // -----------------------------------------------------------------------
179   function allowance(address tokenOwner, address spender) public constant returns
(uint remaining) {
180   return allowed[tokenOwner][spender];
181   }
182
183
```

# SWC-123 | REQUIREMENT VIOLATION.

LINE 209

## low SEVERITY

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

## Source File

- ALINK.sol

## Locations

```
208    function transferAnyERC20Token(address tokenAddress, uint tokens) public onlyOwner
returns (bool success) {
209    return ERC20Interface(tokenAddress).transfer(owner, tokens);
210    }
211    }
212
```

# SWC-123 | REQUIREMENT VIOLATION.
LINE 192

## low SEVERITY

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

## Source File

- ALINK.sol

## Locations

```
191    emit Approval(msg.sender, spender, tokens);
192    ApproveAndCallFallBack(spender).receiveApproval(msg.sender, tokens, this, data);
193    return true;
194    }
195
196
```

# DISCLAIMER

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to, or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Sysfixed's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Sysfixed to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model, or legal compliance.

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 us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the below disclaimer below – please make sure to read it in full.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

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 Sysfixed and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (Sysfixed) owe no duty of care.

# SYSFIXED

# ABOUT US

Sysfixed is a blockchain security certification organization established in 2021 with the objective to provide smart contract security services and verify their correctness in blockchain-based protocols. Sysfixed automatically scans for security vulnerabilities in Ethereum and other EVM-based blockchain smart contracts. Sysfixed a comprehensive range of analysis techniques—including static analysis, dynamic analysis, and symbolic execution—can accurately detect security vulnerabilities to provide an in-depth analysis report. With a vibrant ecosystem of world-class integration partners that amplify developer productivity, Sysfixed can be utilized in all phases of your project's lifecycle. Our team of security experts is dedicated to the research and improvement of our tools and techniques used to fortify your code.