

# **LINKS Token**

## **Smart Contract Audit Report**

### **Prepared for GetLinks**

---



**Date Issued:** Nov 24, 2021  
**Project ID:** AUDIT2021036  
**Version:** v2.0  
**Confidentiality Level:** Public

## Report Information

Project ID	AUDIT2021036
Version	v2.0
Client	GetLinks
Project	LINKS Token
Auditor(s)	Weerawat Pawanawiwat Patipon Suwanbol
Author	Patipon Suwanbol
Reviewer	Suvicha Buakhom
Confidentiality Level	Public

## Version History

Version	Date	Description	Author(s)
2.0	Nov 24, 2021	Update executive summary	Weerawat Pawanawiwat
1.0	Nov 23, 2021	Full report	Patipon Suwanbol

## Contact Information

Company	Inspex
Phone	(+66) 90 888 7186
Telegram	<a href="https://t.me/inspexco">t.me/inspexco</a>
Email	<a href="mailto:audit@inspex.co">audit@inspex.co</a>

# Table of Contents

<b>1. Executive Summary</b>	<b>1</b>
1.1. Audit Result	1
1.2. Disclaimer	1
<b>2. Project Overview</b>	<b>2</b>
2.1. Project Introduction	2
2.2. Scope	3
<b>3. Methodology</b>	<b>4</b>
3.1. Test Categories	4
3.2. Audit Items	5
3.3. Risk Rating	6
<b>4. Summary of Findings</b>	<b>7</b>
<b>5. Detailed Findings Information</b>	<b>9</b>
5.1 Missing Timestamp Validation	9
5.2 Improper Function Visibility	12
5.3 Inexplicit Solidity Compiler Version	13
<b>6. Appendix</b>	<b>14</b>
6.1 About Inspect	14
6.2 References	15

## 1. Executive Summary

As requested by GetLinks, Inspex team conducted an audit to verify the security posture of the LINKS Token smart contracts on Nov 12, 2021. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of LINKS Token smart contracts. Static code analysis, dynamic analysis, and manual review were done in conjunction to identify smart contract vulnerabilities together with technical & business logic flaws that may be exposed to the potential risk of the platform and the ecosystem. Practical recommendations are provided according to each vulnerability found and should be followed to remediate the issue.

On the creation of the \$LINKS contract, 250 million tokens are minted, completely filling the maximum cap. Since \$LINKS is not currently used in any other smart contract other than the presale, there is no burning of \$LINKS to reduce the supply, so there is no risk of manual token minting by the contract owner. However, in the future, if new contracts are deployed with the functionality to burn \$LINKS for any purpose, it should be known that the contract owner can mint \$LINKS for the amount burned to fill the reduced token cap.

### 1.1. Audit Result

In the initial audit, Inspex found 3 info-severity issues. With the project team's prompt response in resolving the issues found by Inspex, all issues were resolved in the reassessment. Therefore, Inspex trusts that LINKS Token smart contracts have high-level protections in place to be safe from most attacks.



### 1.2. Disclaimer

This security audit is not produced to supplant any other type of assessment and does not guarantee the discovery of all security vulnerabilities within the scope of the assessment. However, we warrant that this audit is conducted with goodwill, professional approach, and competence. Since an assessment from one single party cannot be confirmed to cover all possible issues within the smart contract(s), Inspex suggests conducting multiple independent assessments to minimize the risks. Lastly, nothing contained in this audit report should be considered as investment advice.

## 2. Project Overview

### 2.1. Project Introduction

GetLinks is a decentralized remuneration network platform where the users can earn, learn, and give back to the community.

LINKS token is a utility token that can be used across the platform to access all platform's provided features.

#### Scope Information:

Project Name	LINKS Token
Website	<a href="https://getlinks.io/">https://getlinks.io/</a>
Smart Contract Type	Ethereum Smart Contract
Chain	Binance Smart Chain
Programming Language	Solidity

#### Audit Information:

Audit Method	Whitebox
Audit Date	Nov 12, 2021
Reassessment Date	Nov 18, 2021

The audit method can be categorized into two types depending on the assessment targets provided:

1. **Whitebox:** The complete source code of the smart contracts are provided for the assessment.
2. **Blackbox:** Only the bytecodes of the smart contracts are provided for the assessment.

## 2.2. Scope

The following smart contracts were audited and reassessed by Inspect in detail:

### Initial Audit:

Contract	Location (URL)
GetLinksToken	<a href="https://public.inspex.co/audit/GetLinks_LinksToken/GetLinksToken.sol">https://public.inspex.co/audit/GetLinks_LinksToken/GetLinksToken.sol</a>
GetLinksTokenPresale	<a href="https://public.inspex.co/audit/GetLinks_LinksToken/GetLinksTokenPresale.sol">https://public.inspex.co/audit/GetLinks_LinksToken/GetLinksTokenPresale.sol</a>

Please note that the contracts were initially in GetLink's private repository. The files have been uploaded to Inspect's storage for public access.

### Reassessment: (Commit: 60759c738f436e31e615b698775be1a732d4095b)

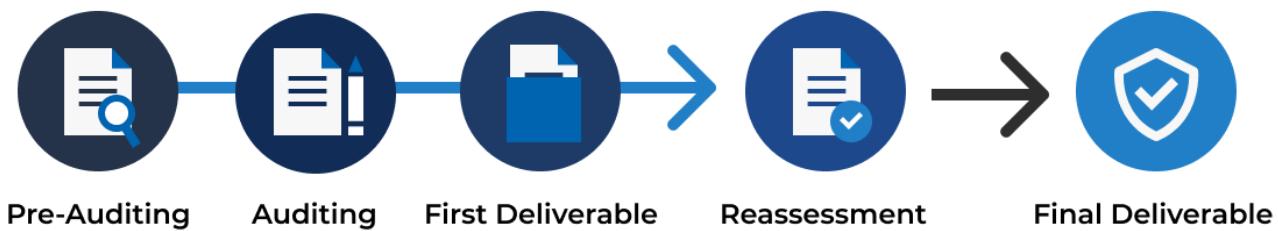
Contract	Location (URL)
GetLinksToken	<a href="https://github.com/LinksToken/Contracts/blob/60759c738f/contracts/GetLinksToken.sol">https://github.com/LinksToken/Contracts/blob/60759c738f/contracts/GetLinksToken.sol</a>
GetLinksTokenPresale	<a href="https://github.com/LinksToken/Contracts/blob/60759c738f/contracts/GetLinksTokenPresale.sol">https://github.com/LinksToken/Contracts/blob/60759c738f/contracts/GetLinksTokenPresale.sol</a>

The assessment scope covers only the in-scope smart contracts and the smart contracts that they inherit from.

### 3. Methodology

Inspex conducts the following procedure to enhance the security level of our clients' smart contracts:

1. **Pre-Auditing:** Getting to understand the overall operations of the related smart contracts, checking for readiness, and preparing for the auditing
2. **Auditing:** Inspecting the smart contracts using automated analysis tools and manual analysis by a team of professionals
3. **First Deliverable and Consulting:** Delivering a preliminary report on the findings with suggestions on how to remediate those issues and providing consultation
4. **Reassessment:** Verifying the status of the issues and whether there are any other complications in the fixes applied
5. **Final Deliverable:** Providing a full report with the detailed status of each issue



#### 3.1. Test Categories

Inspex smart contract auditing methodology consists of both automated testing with scanning tools and manual testing by experienced testers. We have categorized the tests into 3 categories as follows:

1. **General Smart Contract Vulnerability (General)** - Smart contracts are analyzed automatically using static code analysis tools for general smart contract coding bugs, which are then verified manually to remove all false positives generated.
2. **Advanced Smart Contract Vulnerability (Advanced)** - The workflow, logic, and the actual behavior of the smart contracts are manually analyzed in-depth to determine any flaws that can cause technical or business damage to the smart contracts or the users of the smart contracts.
3. **Smart Contract Best Practice (Best Practice)** - The code of smart contracts is then analyzed from the development perspective, providing suggestions to improve the overall code quality using standardized best practices.

### 3.2. Audit Items

The following audit items were checked during the auditing activity.

General
Reentrancy Attack
Integer Overflows and Underflows
Unchecked Return Values for Low-Level Calls
Bad Randomness
Transaction Ordering Dependence
Time Manipulation
Short Address Attack
Outdated Compiler Version
Use of Known Vulnerable Component
Deprecated Solidity Features
Use of Deprecated Component
Loop with High Gas Consumption
Unauthorized Self-destruct
Redundant Fallback Function
Insufficient Logging for Privileged Functions
Invoking of Unreliable Smart Contract
Use of Upgradable Contract Design
Advanced
Business Logic Flaw
Ownership Takeover
Broken Access Control
Broken Authentication
Improper Kill-Switch Mechanism

Improper Front-end Integration
Insecure Smart Contract Initiation
Denial of Service
Improper Oracle Usage
Memory Corruption
<b>Best Practice</b>
Use of Variadic Byte Array
Implicit Compiler Version
Implicit Visibility Level
Implicit Type Inference
Function Declaration Inconsistency
Token API Violation
Best Practices Violation

### 3.3. Risk Rating

OWASP Risk Rating Methodology[1] is used to determine the severity of each issue with the following criteria:

- **Likelihood:** a measure of how likely this vulnerability is to be uncovered and exploited by an attacker.
- **Impact:** a measure of the damage caused by a successful attack

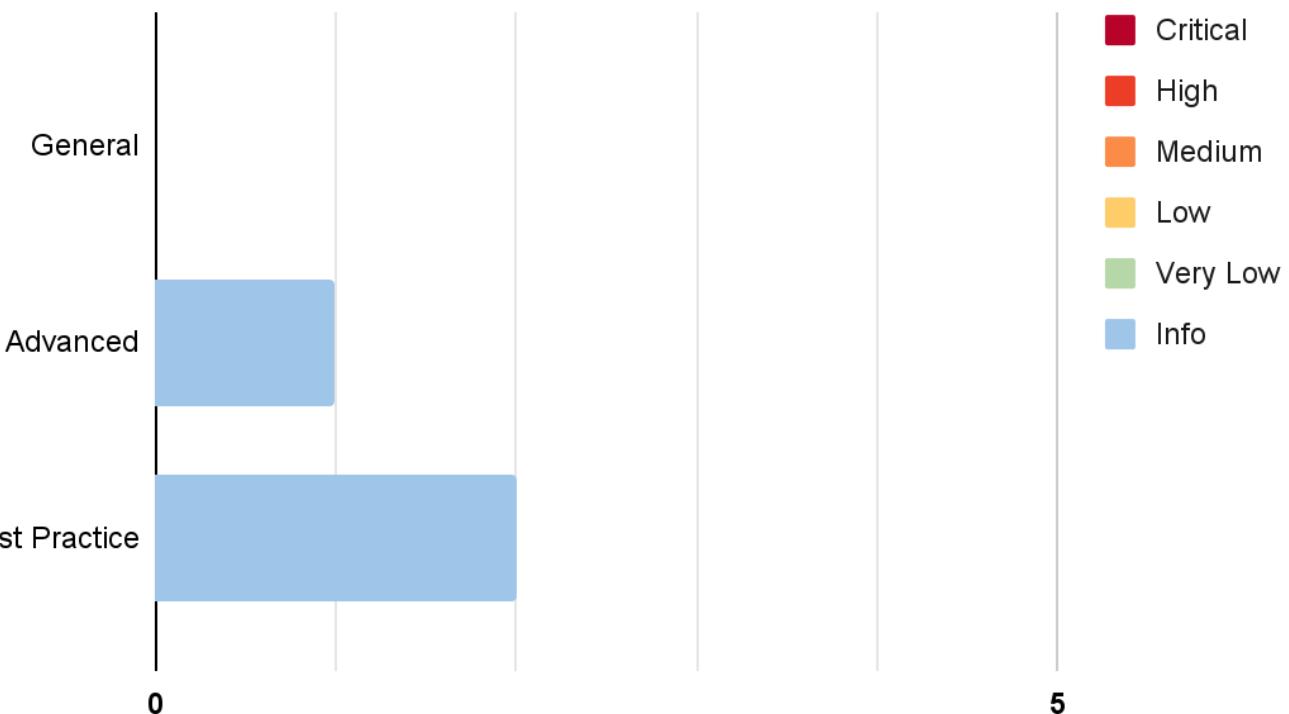
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

**Severity** is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would be categorized as **Info**.

Likelihood Impact	Low	Medium	High
Low	Very Low	Low	Medium
Medium	Low	Medium	High
High	Medium	High	Critical

## 4. Summary of Findings

From the assessments, Inspex has found 3 issues in three categories. The following chart shows the number of the issues categorized into three categories: **General**, **Advanced**, and **Best Practice**.



The statuses of the issues are defined as follows:

Status	Description
Resolved	The issue has been resolved and has no further complications.
Resolved *	The issue has been resolved with mitigations and clarifications. For the clarification or mitigation detail, please refer to Chapter 5.
Acknowledged	The issue's risk has been acknowledged and accepted.
No Security Impact	The best practice recommendation has been acknowledged.

---

The information and status of each issue can be found in the following table:

ID	Title	Category	Severity	Status
IDX-001	Missing Timestamp Validation	Advanced	Info	Resolved
IDX-002	Improper Function Visibility	Best Practice	Info	Resolved
IDX-003	Inexplicit Solidity Compiler Version	Best Practice	Info	Resolved

## 5. Detailed Findings Information

### 5.1 Missing Timestamp Validation

ID	IDX-001
Target	GetLinksTokenPresale
Category	Advanced Smart Contract Vulnerability
CWE	CWE-20: Improper Input Validation
Risk	<b>Severity:</b> Info <b>Impact:</b> None <b>Likelihood:</b> None
Status	<b>Resolved</b> GetLinks team has resolved this issue as suggested in commit <a href="#">7044b5f7cf700f806c953e7af203975af07c1e9</a> by adding an input validation logic into the contract's constructor.

#### 5.1.1 Description

The `GetLinksTokenPresale` contract allows the users to buy \$LINKS in a specific time range (presale). The users will be able to claim the tokens after the presale has ended.

The \$LINKS is required to be sent to this contract by the owner before the presale begins due to the `onlyBeforeStartTime()` modifier.

#### GetLinksTokenPresale.sol

```

61 function increaseSupply(uint256 amount) external onlyBeforeStartTime {
62     getlinksToken.safeTransferFrom(msg.sender, address(this), amount);
63     totalSupply += amount;
64
65     emit TotalSupplyIncreased(msg.sender, amount, totalSupply);
66 }
```

This modifier checks that the timestamp must be before the start time.

#### GetLinksTokenPresale.sol

```

158 modifier onlyBeforeStartTime() {
159     require(
160         block.timestamp < startTime,
161         "Presale: only before the start time"
162     );
```

```
163     -;
164 }
```

To buy \$LINK during the presale, users can do it through the `fund()` function.

### GetLinksTokenPresale.sol

```
68 function fund(uint256 amount) external onlyDuringPresalePeriod {
69     busdToken.safeTransferFrom(msg.sender, address(this), amount);
70     _userInfo[msg.sender].fundInUsd += amount;
71     totalFundInUsd += amount;
72
73     emit Fund(msg.sender, amount);
74 }
```

The presale period is checked by the `onlyDuringPresalePeriod()` modifier.

### GetLinksTokenPresale.sol

```
166 modifier onlyDuringPresalePeriod() {
167     require(
168         block.timestamp >= startTime && block.timestamp < endTime,
169         "Presale: Only during the pre-sale period"
170     );
171     -;
172 }
```

However, in the contract constructor, there is no input validation to check whether the `_startTime` has passed or not. If `_startTime` has passed, the owner will not be able to register the \$LINKS token to the contract, resulting in \$LINKS being unable to be bought by the platform's users.

### GetLinksTokenPresale.sol

```
39 constructor(
40     address _busdToken,
41     address _getlinksToken,
42     address _treasuryAddress,
43     uint256 _startTime,
44     uint256 _endTime
45 ) {
46     require(
47         _busdToken != address(0) &&
48             _getlinksToken != address(0) &&
49                 _treasuryAddress != address(0),
50         "Presale: Cannot be zero address"
51     );
52     require(_startTime < _endTime, "Presale: Time");
53 }
```

```

54     busdToken = IERC20(_busdToken);
55     getlinksToken = IERC20(_getlinksToken);
56     treasuryAddress = _treasuryAddress;
57     startTime = _startTime;
58     endTime = _endTime;
59 }
```

However, although the `_startTime` is less than `block.timestamp` (the current time has passed the `_startTime`), the platform can simply redeploy the contract again.

### 5.1.2 Remediation

Inspex suggests implementing input validation on both `_startTime` and `_endTime` parameters to ensure that `_startTime` has not passed (`block.timestamp` must be less than `_startTime`).

#### GetLinksTokenPresale.sol

```

39 constructor(
40     address _busdToken,
41     address _getlinksToken,
42     address _treasuryAddress,
43     uint256 _startTime,
44     uint256 _endTime
45 ) {
46     require(
47         _busdToken != address(0) &&
48         _getlinksToken != address(0) &&
49         _treasuryAddress != address(0),
50         "Presale: Cannot be zero address"
51     );
52     require(block.timestamp < _startTime && _startTime < _endTime, "Presale:
53 Time");
54     busdToken = IERC20(_busdToken);
55     getlinksToken = IERC20(_getlinksToken);
56     treasuryAddress = _treasuryAddress;
57     startTime = _startTime;
58     endTime = _endTime;
59 }
```

## 5.2 Improper Function Visibility

ID	IDX-002
Target	GetLinksToken
Category	Smart Contract Best Practice
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	<b>Severity:</b> Info <b>Impact:</b> None <b>Likelihood:</b> None
Status	<b>Resolved</b> GetLinks team has resolved this issue as suggested in commit <code>7044b5f7cf700f806c953e7af203975af07c1e9</code> by changing the function's visibility to optimize the gas usage.

### 5.2.1 Description

Functions with public visibility copy calldata to memory when being executed, while external functions can read directly from calldata. Memory allocation uses more resources (gas) than reading directly from calldata.

The following source code shows that the `mint()` function of the `GetLinkstoken` contract is set to public and it is never called from any internal function.

#### GetLinksToken.sol

```

16 function mint(address to, uint256 amount) public onlyOwner {
17     _mint(to, amount);
18 }
```

### 5.2.2 Remediation

Inspex suggests changing the `mint()` function's visibility to external if they are not called from any internal function as shown in the following example:

#### GetLinksToken.sol

```

16 function mint(address to, uint256 amount) external onlyOwner {
17     _mint(to, amount);
18 }
```

## 5.3 Inexplicit Solidity Compiler Version

ID	IDX-003
Target	GetLinksToken GetLinksTokenPresale
Category	Smart Contract Best Practice
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	<b>Severity:</b> Info <b>Impact:</b> None <b>Likelihood:</b> None
Status	<b>Resolved</b> GetLinks team has resolved this issue as suggested in commit <code>7044b5f7cf700f806c953e7af203975af07c1e9</code> by changing the Solidity compiler version to an explicit version.

### 5.3.1 Description

The Solidity compiler versions declared in the smart contracts were not explicit. Each compilation may be done using different compiler versions, which may potentially result in compatibility issues.

#### GetLinksToken.sol

```

1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.2;

```

The following table represents a list of contracts that specifies an inexplicit solidity compiler version.

Contract	Version
GetLinksToken	^0.8.2
GetLinksTokenPresale	^0.8.2

### 5.3.2 Recommendation

Inspex suggests fixing the solidity compiler to the latest stable version. At the time of the audit, the latest stable versions of Solidity compiler in major 0.8 is v0.8.10, for example:

#### GetLinksToken.sol

```

1 // SPDX-License-Identifier: MIT
2 pragma solidity 0.8.10;

```

## 6. Appendix

### 6.1. About Inspect



## CYBERSECURITY PROFESSIONAL SERVICE

Inspect is formed by a team of cybersecurity experts highly experienced in various fields of cybersecurity. We provide blockchain and smart contract professional services at the highest quality to enhance the security of our clients and the overall blockchain ecosystem.

#### Follow Us On:

Website	<a href="https://inspex.co">https://inspex.co</a>
Twitter	<a href="https://twitter.com/InspexCo">@InspexCo</a>
Facebook	<a href="https://www.facebook.com/InspexCo">https://www.facebook.com/InspexCo</a>
Telegram	<a href="https://t.me/inspex_announcement">@inspex_announcement</a>

## 6.2. References

- [1] “OWASP Risk Rating Methodology.” [Online]. Available:  
[https://owasp.org/www-community/OWASP\\_Risk\\_Rating\\_Methodology](https://owasp.org/www-community/OWASP_Risk_Rating_Methodology). [Accessed: 08-May-2021]



**inspex**

CYBERSECURITY PROFESSIONAL SERVICE