LuckyToken

Smart Contract Audit Report
Prepared for Lucky Lion

Date Issued: Sep 8, 2021
Project ID: AUDIT2021022
Version: v1.0
Confidentiality Level: Public

inspeX
CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2021022
Version	v1.0
Client	Lucky Lion
Project	LuckyToken
Auditor(s)	Weerawat Pawanawiwat Patipon Suwanbol
Author	Weerawat Pawanawiwat
Reviewer	Pongsakorn Sommalai
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
1.0	Sep 8, 2021	Full report	Weerawat Pawanawiwat

Contact Information

Company	Inspex
Phone	(+66) 90 888 7186
Telegram	t.me/inspexco
Email	audit@inspex.co

Table of Contents

1. Executive Summary

As requested by Lucky Lion, Inspex team conducted an audit to verify the security posture of the LuckyToken smart contract on Sep 6, 2021. During the audit, Inspex team examined the smart contract and the overall operation within the scope to understand the overview of LuckyToken smart contract. 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.

LuckyToken is an ERC20 token contract that inherits OpenZeppelin's Ownable contract, and the minting of $LUCKY is purely controlled by the contract owner. Even when the smart contract itself is safe, if the contract owner is compromised, potential harm can still be done and cause effects to the token holders. From the documentation, the LuckyToken contract will be owned by a MasterChef contract, but since the contract is not yet deployed at the time of the audit, Inspex cannot verify the ownership of the LuckyToken contract. Inspex suggests that the contracts related to the minting of $LUCKY should be further assessed to ensure the security of LuckyLion's platform as a whole.

1.1. Audit Result

In the initial audit, Inspex found 1 high and 2 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 LuckyToken smart contract has 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

Lucky Lion is the latest addition to the portfolio of APAC's leading iGaming brands with over 200,000 loyal monthly active users, allowing players to yield users' tokens on the decentralized yield farm, play industry leading iGaming, and stake the reward through the revenue sharing pool to earn even more amazing rewards.

LuckyToken is the ERC20 contract for $LUCKY. It is the main token distributed as a reward for the platform users and used for earning profit from the revenue sharing pool.

Scope Information:

Project Name	LuckyToken
Website	https://luckylion.io/
Smart Contract Type	Ethereum Smart Contract
Chain	Binance Smart Chain
Programming Language	Solidity

Audit Information:

Audit Method	Whitebox
Audit Date	Sep 6, 2021
Reassessment Date	Sep 8, 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 contract was audited and reassessed by Inspex in detail:

Initial Audit: (Commit: de9b9e8432f3550f509d5682cbd26583be310c35)

Contract	Location (URL)
LuckyToken	https://github.com/LuckyLionIO/Lucky-Token/blob/de9b9e8432/contracts/LuckyToken.sol

Reassessment: (Commit: c811a6afe35b69d0eb2e621bbec0a15772f7ea98)

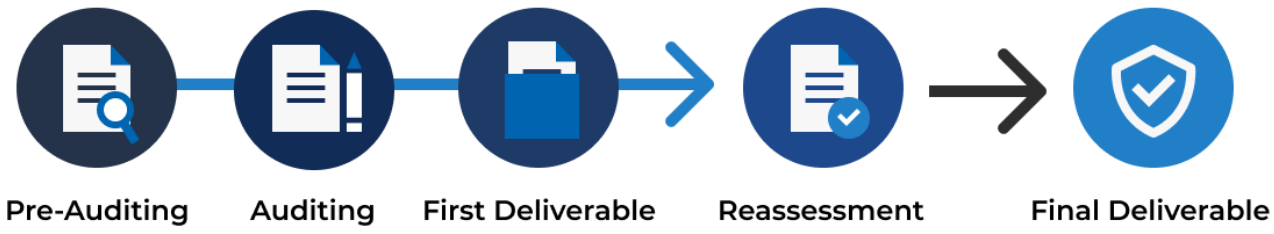
Contract	Location (URL)
LuckyToken	https://github.com/LuckyLionIO/Lucky-Token/blob/c811a6afe3/contracts/LuckyToken.sol

The assessment scope covers only the in-scope smart contract and the smart contracts that it is inherited 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
Advanced
Business Logic Flaw
Ownership Takeover
Broken Access Control
Broken Authentication
Upgradable Without Timelock
Improper Kill-Switch Mechanism
Improper Front-end Integration
Insecure Smart Contract Initiation

Denial of Service
Improper Oracle Usage
Memory Corruption
Best Practice
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

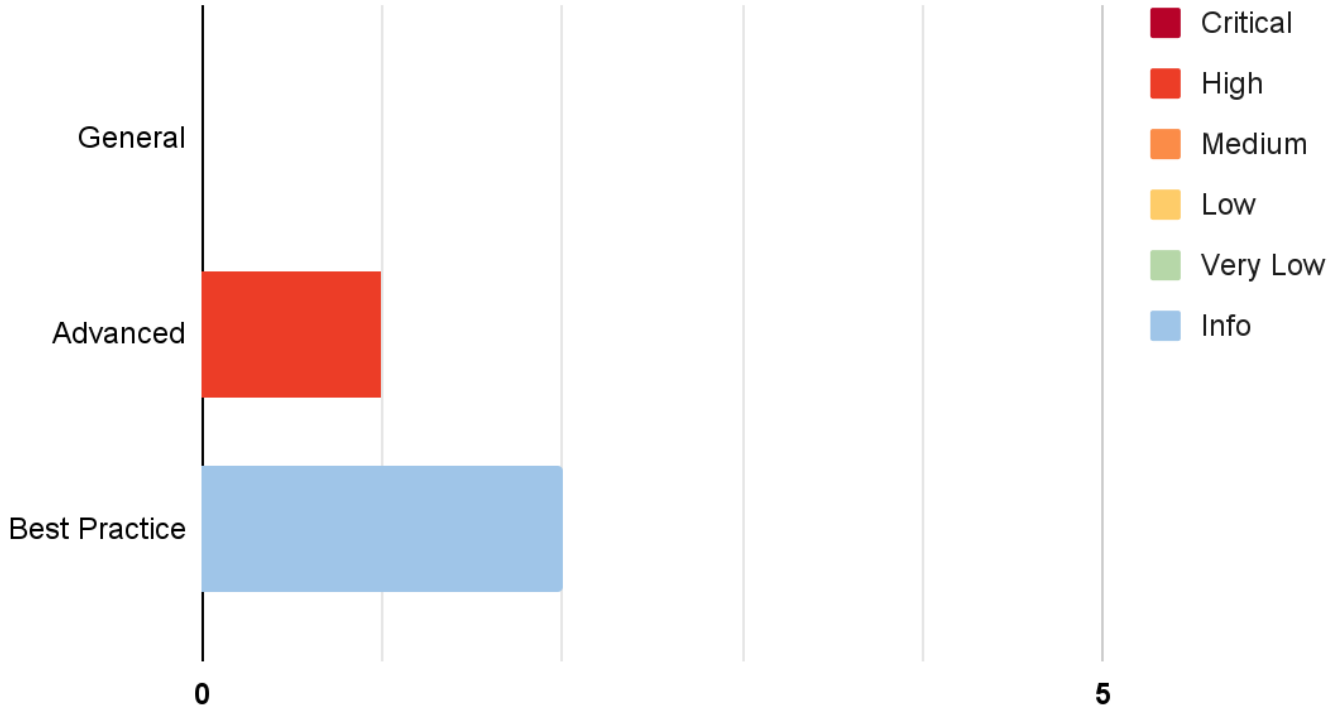
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

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**.

Impact \ Likelihood	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	Improper Token Minting	Advanced	**High**	Resolved
IDX-002	Improper Function Visibility	Best Practice	**Info**	Resolved
IDX-003	Inexplicit Solidity Compiler Version	Best Practice	**Info**	Resolved

* The mitigations or clarifications by Lucky Lion can be found in Chapter 5.

5. Detailed Findings Information

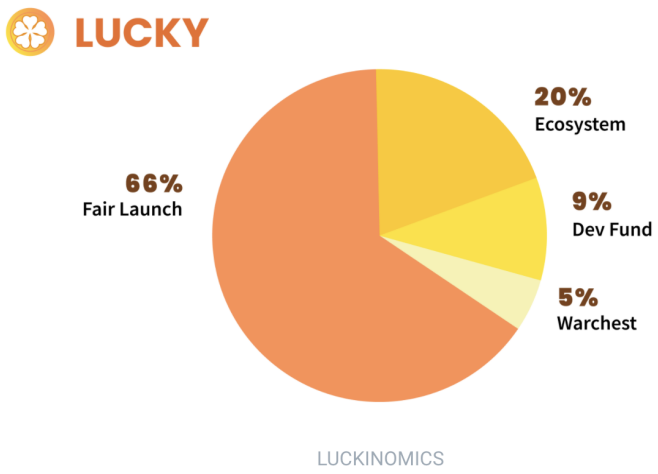
5.1. Improper Token Minting

ID	IDX-001
Target	LuckyToken
Category	Advanced Smart Contract Vulnerability
CWE	CWE-840: Business Logic Errors
Risk	**Severity:** High **Impact:** Medium $LUCKY is minted without conforming to the tokenomics defined in the documentation. **Likelihood:** High It is very likely that this portion will be minted since it is defined in the contract constructor.
Status	Resolved The Lucky Lion team has resolved this issue by detailing the affected $LUCKY portion in the documentation on https://docs.luckylion.io/tokenomics and clarifying that at fair launch, the 1% minted (1,000,000 $LUCKY) is used for seeding the Lucky Lion's LUCKY-BNB pool. 1,000,000 $LUCKY will be equal to $30,000 USD worth of $BNB. Then, after launching the farm, 65% of the total supply will be distributed to all users deposited in yield farming LP pools with 9% of the tokens vesting to the team over a two-year period. Over this time, $LUCKY is going to be released with a decaying emission schedule. In total, there will be 100 million $LUCKY.

5.1.1. Description

According to Lucky Lion's tokenomics (https://docs.luckylion.io/tokenomics), it is stated that 66% of $LUCKY will be distributed using the fair launch mechanism.

LUCKY Tokenomics aka Luckinomics



Fair Launch Token Distribution

66% of our total supply will be distributed to all LUCKY Token holders with only 9% of the tokens vesting to the team over a two-year period. Over this time, LUCKY Tokens are going to be released with a decaying emissions schedule. In total, there will be **100 million $LUCKY.** To incentivize early adopters, there will be a special multiplier bonus rewards period. Please see chart below for details.

However, in the contract constructor at line 30, 1% of the total cap is minted to the contract owner's wallet instead of being minted and distributed gradually through the fair launch mechanism.

LuckyToken.sol

```
14  uint256 private constant FAIR_LAUNCH = 1 * 1000000 * 10**18; //1 * 1000000 *
    10**_decimals;
15  uint256 private constant WAR_CHEST = 5 * 1000000 * 10**18;
16  uint256 private constant ECOSYSTEM = 20 * 1000000 * 10**18;
17  uint256 private constant CAP = 100 * 1000000 * 10**18; //max supply
18
19  address Owner;
20  address WarChest;
21  address Ecosystem;
22
23  constructor (address _Owner, address _Warchest, address _Ecosystem)
    ERC20(_name, _symbol) {
24      //set wallet address
25      Owner = _Owner;
26      WarChest = _Warchest;
27      Ecosystem = _Ecosystem;
28
29      //mint to Owner's Wallet for Fairlunch
30      _mint(Owner, FAIR_LAUNCH);
31      //mint to WarChest's Wallet
```

```
32      _mint(WarChest, WAR_CHEST);
33      //mint to Ecosystem's Wallet
34      _mint(Ecosystem, ECOSYSTEM);
35      //transfer to real owner
36      transferOwnership(_Owner);
37  }
```

The result indicates that the $LUCKY is minted without conforming to the tokenomics defined in the documentation.

5.1.2. Remediation

Inspex suggests removing the improper token minting from the contract constructor.

5.2. Improper Function Visibility

ID	IDX-002
Target	LuckyToken
Category	Smart Contract Best Practice
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The Lucky Lion team has resolved this issue in commit `c811a6afe35b69d0eb2e621bbec0a15772f7ea98` by updating the visibility of `mint()` function and removing the `decimals()` function.

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 `decimals()` function of the `LuckyToken` contract is set to `public` and it is never called from any internal function.

LuckyToken.sol

```
39 function decimals() public view virtual override returns (uint8) {
40     return _decimals;
41 }
```

The following table contains all functions that have `public` visibility and are never called from any internal function.

Contract	Function
LuckyToken (L:39)	decimals()
LuckyToken (L:50)	mint()

5.2.2. Remediation

Inspex suggests changing all functions' visibility to external if they are not called from any internal function as shown in the following example:

LuckyToken.sol

```
39 function decimals() external view virtual override returns (uint8) {
40     return _decimals;
41 }
```

5.3. Inexplicit Solidity Compiler Version

ID	IDX-003
Target	LuckyToken
Category	Smart Contract Best Practice
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved The Lucky Lion team has resolved this issue as suggested in commit `c811a6afe35b69d0eb2e621bbec0a15772f7ea98`.

5.3.1. Description

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

LuckyToken.sol

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity ^0.8.0;
```

5.3.2. Remediation

Inspex suggests fixing the solidity compiler to the latest stable version. At the time of the audit, the latest stable version of Solidity compiler in major 0.8 is v0.8.7.

LuckyToken.sol

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity 0.8.7;
```

6. Appendix

6.1. About Inspex

Inspex 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	https://inspex.co
Twitter	@InspexCo
Facebook	https://www.facebook.com/InspexCo
Telegram	@inspex_announcement

6.2. References

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

inspeX

CYBERSECURITY PROFESSIONAL SERVICE