SMART CONTRACT AUDIT REPORT

for

LEONS Token

Prepared By: Xiaomi Huang

PeckShield

February 1, 2023

Document Properties

Client	Leonicornswap
Title	Smart Contract Audit Report
Target	LEONS Token
Version	1.0
Author	Xiaotao Wu
Auditors	Xiaotao Wu, Xuxian Jiang
Reviewed by	Patrick Lou
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author	Description
1.0	February 1, 2023	Xiaotao Wu	Final Release
1.0-rc	January 31, 2023	Xiaotao Wu	Release Candidate

Contact

For more information about this document and its contents, please contact PeckShield Inc.

Name	Xiaomi Huang
Phone	+86 183 5897 7782
Email	contact@peckshield.com

Contents

1 Introduction 4

1.1 About LEONS Token 4

1.2 About PeckShield 5

1.3 Methodology 5

1.4 Disclaimer 7

2 Findings 8

2.1 Summary 8

2.2 Key Findings 9

3 BEP20 Compliance Checks 10

4 Detailed Results 13

4.1 Meaningful Events For Important State Changes 13

4.2 Trust Issue Of Admin Keys 14

5 Conclusion 17

References 18

1 Introduction

Given the opportunity to review the design document and related source code of the LEONS token contract, we outline in the report our systematic method to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistency between smart contract code and the documentation, and provide additional suggestions or recommendations for improvement. Our results show that the given version of the smart contract can be further improved due to the presence of certain issues related to BEP20-compliance, security, or performance. This document outlines our audit results.

1.1 About LEONS Token

LEONS token is the main platform token of Leonicornswap that can be used for various features like Staking, Farming, games, NFT marketplace, Initial Farm Offerings, etc. It's basically a BEP20 token with extra features like whitelisting to allow unlimited wallet transfers, blacklisting in not allowing wallet transfers, as well as per-day transfer limit. The basic information of the audited LEONS token contract is as follows:

Table 1.1: Basic Information of LEONS Token

Item	Description
Issuer	Leonicornswap
Website	https://www.leonicornswap.com/
Type	BEP20 Token Contract
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	February 1, 2023

In the following, we show the bscscan link to the contract address with the verified source code used in this audit:

- https://testnet.bscscan.com/address/0x0c5140223717b2bf5b90e7c6b4db8a75ebf3a84a#code

1.2 About PeckShield

PeckShield Inc. [6] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystem by offering top-notch, industry-leading services and products (including the service of smart contract auditing). We are reachable at Telegram (https://t.me/peckshield), Twitter (http://twitter.com/peckshield), or Email (contact@peckshield.com).

1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [5]:

- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact measures the technical loss and business damage of a successful attack;
- Severity demonstrates the overall criticality of the risk;

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact and can be classified into four categories accordingly, i.e., *Critical*, *High*, *Medium*, *Low* shown in Table 1.2.

Table 1.2: Vulnerability Severity Classification

Impact \ **Likelihood**	*High*	*Medium*	*Low*
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

We perform the audit according to the following procedures:

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.

- BEP20 Compliance Checks: We then manually check whether the implementation logic of the audited smart contract(s) follows the standard BEP20 specification and other best practices.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

Table 1.3: The Full List of Check Items

Category	Check Item
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead of Transfer
	Costly Loop
	(Unsafe) Use of Untrusted Libraries
	(Unsafe) Use of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
	Approve / TransferFrom Race Condition
BEP20 Compliance Checks	Compliance Checks (Section 3)
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

To evaluate the risk, we go through a list of check items and each would be labeled with a severity category. For one check item, if our tool does not identify any issue, the contract is considered safe regarding the check item. For any discovered issue, we might further deploy contracts on our private testnet and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.3.

1.4 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Last but not least, this security audit should not be used as investment advice.

2 Findings

2.1 Summary

Here is a summary of our findings after analyzing the LEONS token contract. During the first phase of our audit, we study the smart contract source code and run our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by our tool. We further manually review business logics, examine system operations, and place BEP20-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	# of Findings	
Critical	0	
High	0	
Medium	1	■
Low	0	
Informational	1	■
Total	2	

Moreover, we explicitly evaluate whether the given contracts follow the standard BEP20 specification and other known best practices, and validate its compatibility with other similar BEP20 tokens and current DeFi protocols. The detailed BEP20 compliance checks are reported in Section 3. After that, we examine a few identified issues of varying severities that need to be brought up and paid more attention to. (The findings are categorized in the above table.) Additional information can be found in the next subsection, and the detailed discussions are in Section 4.

2.2 Key Findings

Overall, a minor BEP20 compliance issue was found and our detailed checklist can be found in Section 3. Overall, there is no critical or high severity issue, although the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 1 medium-severity vulnerability and 1 informational recommendation.

Table 2.1: Key LEONS Token Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Informational	Meaningful Events For Important State Changes	Coding Practices	Resolved
PVE-002	Medium	Trust Issue Of Admin Keys	Security Features	Mitigated

Besides recommending specific countermeasures to mitigate the above issue(s), we also emphasize that it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms need to kick in at the very moment when the contracts are being deployed in mainnet. Please refer to Section 3 for our detailed compliance checks and Section 4 for elaboration of reported issues.

3 BEP20 Compliance Checks

The BEP20 specification defines a list of API functions (and relevant events) that each token contract is expected to implement (and emit). The failure to meet these requirements means the token contract cannot be considered to be BEP20-compliant. Naturally, as the first step of our audit, we examine the list of API functions defined by the BEP20 specification and validate whether there exist any inconsistency or incompatibility in the implementation or the inherent business logic of the audited contract(s).

Table 3.1: Basic `View-Only` Functions Defined in The BEP20 Specification

Item	Description	Status
name()	Is declared as a public view function	✓
	Returns a string, for example "Tether USD"	✓
symbol()	Is declared as a public view function	✓
	Returns the symbol by which the token contract should be known, for example "USDT". It is usually 3 or 4 characters in length	✓
decimals()	Is declared as a public view function	✓
	Returns decimals, which refers to how divisible a token can be, from 0 (not at all divisible) to 18 (pretty much continuous) and even higher if required	✓
totalSupply()	Is declared as a public view function	✓
	Returns the number of total supplied tokens, including the total minted tokens (minus the total burned tokens) ever since the deployment	✓
balanceOf()	Is declared as a public view function	✓
	Anyone can query any address' balance, as all data on the blockchain is public	✓
allowance()	Is declared as a public view function	✓
	Returns the amount which the spender is still allowed to withdraw from the owner	✓
getOwner()	Is declared as a public view function	–
	Returns the bep20 token owner which is necessary for binding with bep2 token.	–

Our analysis shows that there is a minor BEP20 inconsistency or incompatibility issue found in the audited `LEONS` token contract. Specifically, the `getOwner()` function is an extended method of `EIP20` and is currently not defined. Tokens that do not implement this method will not be able to flow across the `Binance Chain` and `Binance Smart Chain` (`BSC`). In the surrounding two tables, we outline the respective list of basic `view-only` functions (Table 3.1) and key `state-changing` functions (Table 3.2) according to the widely-adopted BEP20 specification.

Table 3.2: Key `State-Changing` Functions Defined in The BEP20 Specification

Item	Description	Status
transfer()	Is declared as a public function	✓
	Returns a boolean value which accurately reflects the token transfer status	✓
	Reverts if the caller does not have enough tokens to spend	✓
	Allows zero amount transfers	✓
	Emits Transfer() event when tokens are transferred successfully (include 0 amount transfers)	✓
	Reverts while transferring to zero address	✓
transferFrom()	Is declared as a public function	✓
	Returns a boolean value which accurately reflects the token transfer status	✓
	Reverts if the spender does not have enough token allowances to spend	✓
	Updates the spender's token allowances when tokens are transferred successfully	✓
	Reverts if the from address does not have enough tokens to spend	✓
	Allows zero amount transfers	✓
	Emits Transfer() event when tokens are transferred successfully (include 0 amount transfers)	✓
	Reverts while transferring from zero address	✓
	Reverts while transferring to zero address	✓
approve()	Is declared as a public function	✓
	Returns a boolean value which accurately reflects the token approval status	✓
	Emits Approval() event when tokens are approved successfully	✓
	Reverts while approving to zero address	✓
Transfer() event	Is emitted when tokens are transferred, including zero value transfers	✓
	Is emitted with the from address set to $address(0x0)$ when new tokens are generated	✓
Approval() event	Is emitted on any successful call to approve()	✓

In addition, we perform a further examination on certain features that are permitted by the BEP20 specification or even further extended in follow-up refinements and enhancements, but not required for implementation. These features are generally helpful, but may also impact or bring certain incompatibility with current DeFi protocols. Therefore, we consider it is important to highlight them as well. This list is shown in Table 3.3.

Table 3.3: Additional `Opt-in` Features Examined in Our Audit

Feature	Description	Opt-in
Deflationary	Part of the tokens are burned or transferred as fee while on transfer()/transferFrom() calls	—
Rebasing	The balanceOf() function returns a re-based balance instead of the actual stored amount of tokens owned by the specific address	—
Pausable	The token contract allows the owner or privileged users to pause the token transfers and other operations	—
Blacklistable	The token contract allows the owner or privileged users to blacklist a specific address such that token transfers and other operations related to that address are prohibited	✓
Mintable	The token contract allows the owner or privileged users to mint tokens to a specific address	✓
Burnable	The token contract allows the owner or privileged users to burn tokens of a specific address	✓

4 Detailed Results

4.1 Meaningful Events For Important State Changes

- ID: PVE-001
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: Leonicornswap
- Category: Coding Practices [4]
- CWE subcategory: CWE-563 [2]

Description

In Ethereum, the event is an indispensable part of a contract and is mainly used to record a variety of runtime dynamics. In particular, when an event is emitted, it stores the arguments passed in transaction logs and these logs are made accessible to external analytics and reporting tools. Events can be emitted in a number of scenarios. One particular case is when system-wide parameters or settings are being changed. Another case is when tokens are being minted, transferred, or burned.

In the following, we use the Leonicornswap contract as an example. While examining the events that reflect the Leonicornswap dynamics, we notice there is a lack of emitting related events to reflect important state changes. Specifically, when the setOperator()/whitelistAccount()/unWhitelistAccount()/blacklistAccount()/unBlacklistAccount() are being called, there are no corresponding events being emitted to reflect the occurrence of setOperator()/whitelistAccount()/unWhitelistAccount()/blacklistAccount()/unBlacklistAccount().

```
    function setOperator(address _operator) external onlyOwner {
        operator = _operator;
    }
```

Listing 4.1: Leonicornswap::setOperator()

```
    function whitelistAccount(address account) external onlyAuthorized {
        _whitelisted[account] = true;
    }

```

```
    function unWhitelistAccount(address account) external onlyAuthorized {
        delete _whitelisted[account];
    }
```

Listing 4.2: `Leonicornswap::whitelistAccount()/unWhitelistAccount()`

```
    function blacklistAccount(address account) external onlyAuthorized {
        _blacklisted[account] = true;
    }

    function unBlacklistAccount(address account) external onlyAuthorized {
        delete _blacklisted[account];
    }
```

Listing 4.3: `Leonicornswap::blacklistAccount()/unBlacklistAccount()`

Recommendation Properly emit the related events when the above-mentioned functions are being invoked.

Status This issue has been resolved as the team confirms that it will be addressed during the main net deployment.

4.2 Trust Issue Of Admin Keys

- ID: PVE-002
- Severity: Medium
- Likelihood: Low
- Impact: High
- Target: `Leonicornswap`
- Category: Security Features [3]
- CWE subcategory: CWE-287 [1]

Description

In the `LEONS Token` implementation, there are two privileged accounts, i.e., `owner`, and `operator`. These accounts play a critical role in governing and regulating the system-wide operations (e.g., mint more `LEONS` tokens into circulation, set `operator`, set `transferLimit`, whiteList/unwhitelist account, blacklist/unblacklist account, recover tokens from the `Leonicornswap` contract, etc.). Our analysis shows that these privileged accounts need to be scrutinized. In the following, we use the `Leonicornswap` contract as an example and show the representative functions potentially affected by the privileges of the `owner/operator` accounts.

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

    function setOperator(address _operator) external onlyOwner {
```

```
        operator = _operator;
    }

    function setTransferLimit(uint256 _limit) external onlyAuthorized {
        transferLimit = _limit;
        emit TransferLimitSet(_limit);
    }

    function whitelistAccount(address account) external onlyAuthorized {
        _whitelisted[account] = true;
    }

    function unWhitelistAccount(address account) external onlyAuthorized {
        delete _whitelisted[account];
    }

    function blacklistAccount(address account) external onlyAuthorized {
        _blacklisted[account] = true;
    }

    function unBlacklistAccount(address account) external onlyAuthorized {
        delete _blacklisted[account];
    }

    function recoverWrongTokens(
        address token,
        address to,
        uint256 amount
    ) external onlyAuthorized {
        if (token == address(this))
           this.transfer(to, amount);
        else
           IERC20(token).transfer(to, amount);
    }
```

Listing 4.4: Example Privileged Operations in `Leonicornswap`

We understand the need of the privileged functions for contract maintenance, but at the same time the extra power to the owner may also be a counter-party risk to the protocol users. It is worrisome if the privileged `owner` account is a plain EOA account. Note that a multi-sig account could greatly alleviate this concern, though it is still far from perfect. Specifically, a better approach is to eliminate the administration key concern by transferring the role to a community-governed DAO.

Recommendation Promptly transfer the privileged account to the intended `DAO`-like governance contract. All changes to privileged operations may need to be mediated with necessary timelocks. Eventually, activate the normal on-chain community-based governance life-cycle and ensure the intended trustless nature and high-quality distributed governance.

Status This issue has been mitigated as the team confirms that multi-sig will be planned for the privileged `owner` account.

5 | Conclusion

In this security audit, we have examined the `LEONS` token design and implementation. During our audit, we first checked all respects related to the compatibility of the BEP20 specification and other known BEP20 pitfalls/vulnerabilities. We then proceeded to examine other areas such as coding practices and business logics. Overall, although no critical level vulnerabilities were discovered, we identified two issues that need to be promptly addressed. In the meantime, as disclaimed in Section 1.4, we appreciate any constructive feedbacks or suggestions about our findings, procedures, audit scope, etc.

References

[1] MITRE. CWE-287: Improper Authentication. https://cwe.mitre.org/data/definitions/287.html.

[2] MITRE. CWE-563: Assignment to Variable without Use. https://cwe.mitre.org/data/definitions/563.html.

[3] MITRE. CWE CATEGORY: 7PK - Security Features. https://cwe.mitre.org/data/definitions/254.html.

[4] MITRE. CWE CATEGORY: Bad Coding Practices. https://cwe.mitre.org/data/definitions/1006.html.

[5] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.

[6] PeckShield. PeckShield Inc. https://www.peckshield.com.