soken

SECURITY AUDIT

SOKEN.IO

SOKEN.IO

EXECUTIVE OVERVIEW

Date: 02.09.2024

Token Name: LCI

Contract Address: 0xce36b4caed3ea2ac7b8fe026d111718fb79caf8e

Contract Platform: bscscan

CONTRACT SECURITY

PUBLIC BURN

The contract was found to be using public or an external "burn" function. The function was missing access control to prevent another user from burning their tokens. Also, the burn function was found to be using a different address than msg.sender.

Recommendations:

Consider adding access control modifiers to the burn function to prevent unauthorized users from burning tokens. Use the "onlyOwner" modifier from the OpenZeppelin "Ownable" contract to restrict access. The burn function should also use "msg.sender" in the "_from" argument to ensure that only the owner can call the function.

USE OF FLOATING PRAGMA

Solidity source files indicate the versions of the compiler they can be compiled with using a pragma directive at the top of the solidity file. This can either be a floating pragma or a specific compiler version.The contract was found to be using a floating pragma which is not considered safe as it can be compiled with all the versions described.

Recommendations:

It is recommended to use a fixed pragma version, as future compiler versions may handle certain language constructions in a way the developer did not foresee.Using a floating pragma may introduce several vulnerabilities if compiled with an older version.The developers should always use the exact Solidity compiler version when designing their contracts as it may break the changes in the future.Instead of "${pragma_in_use}" use "pragma solidity ${pragma_version}", which is a stable and recommended version right now.

DEFINE CONSTRUCTOR AS PAYABLE

Developers can save around 10 opcodes and some gas if the constructors are defined as payable.However, it should be noted that it comes with risks because payable constructors can accept ETH during deployment.

Recommendations:

It is suggested to mark the constructors as payable to save some gas. Make sure it does not lead to any adverse effects in case an upgrade pattern is involved.

APPROVE FRONT-RUNNING ATTACK

The method overrides the current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account. This can be abused by a token receiver when they try to withdraw certain tokens from the sender's account. Meanwhile, if the sender decides to change the amount and sends another "approve" transaction, the receiver can notice this transaction before it's mined and can extract tokens from both transactions, therefore, ending up with tokens from both the transactions. This is a front-running attack affecting the "ERC20 Approve" function.

Recommendations:

Only use the approve function of the ERC/BEP standard to change the allowed amount to 0 or from 0 (wait till transaction is mined and approved).Token owner just needs to make sure that the first transaction actually changed allowance from N to 0, i.e., that the spender didn't manage to transfer some of N allowed tokens before the first transaction was mined. Such checking is possible using advanced blockchain explorers such as [Etherscan.io](https://etherscan.io/)Another way to mitigate the threat is to approve token transfers only to smart contracts with verified source code that does not contain logic for performing attacks like described above, and to accounts owned by the people you may trust.

OPTIMIZING ADDRESS ID MAPPING

Combining multiple address/ID mappings into a single mapping using a struct enhances storage efficiency, simplifies code, and reduces gas costs, resulting in a more streamlined and cost-effective smart contract design. It saves storage slot for the mapping and depending on the circumstances and sizes of types, it can avoid a Gsset (20000 gas) per mapping combined. Reads and subsequent writes can also be cheaper when a function requires both values and they fit in the same storage slot.

Recommendations:

It is suggested to modify the code so that multiple mappings using the address->id parameter are combined into a struct.

FUNCTIONS CAN BE IN-LINED

The internal function was called only once throughout the contract. Internal functions cost more gas due to additional "JUMP" instructions and stack operations.

Recommendations:

Creating a function for a single call is not necessary if it can be in-lined. It is recommended to implement the logic using in-line codes to save gas.

LONG NUMBER LITERALS

Solidity supports multiple rational and integer literals, including decimal fractions and scientific notations. The use of very large numbers with too many digits was detected in the code that could have been optimized using a different notation also supported by Solidity.

Recommendations:

Scientific notation in the form of "2e10" is also supported, where the mantissa can be fractional but the exponent has to be an integer. The literal "MeE" is equivalent to "M * 10**E". Examples include "2e10", "2e10", "2e-10", "2.5e1", as suggested in official solidity documentation "https://docs.soliditylang.org/en/latest/types.html#rational-and-integer-literals"

OUTDATED COMPILER VERSION

Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version.

Recommendations:

It is recommended to use a recent version of the Solidity compiler that should not be the most recent version, and it should not be an outdated version as well.
Using very old versions of Solidity prevents the benefits of bug fixes and newer security checks. Consider using the solidity version "${pragma_version}", which patches most solidity

vulnerabilities.

STORAGE VARIABLE CACHING IN MEMORY

The contract is using the state variable multiple times in the function. "SLOADs" are expensive (100 gas after the 1st one) compared to "MLOAD"/"MSTORE" (3 gas each).

Recommendations:

Storage variables read multiple times inside a function should instead be cached in the memory the first time (costing 1 "SLOAD") and then read from this cache to avoid multiple "SLOADs".

USE SCIENTIFIC NOTATION

Although the Solidity compiler can optimize exponentiation, it is recommended to prioritize idioms not reliant on compiler optimization. Utilizing scientific notation enhances code clarity, making it more self-explanatory and aligning with best practices in Solidity development.

Recommendations:

Enhance code readability by replacing exponentiation with scientific notation where applicable. This practice not only aligns with best practices but also reduces the reliance on compiler optimization, contributing to more robust and human-friendly Solidity code.

INTERNAL FUNCTIONS NEVER USED

The contract declared internal functions but was not using them in any of the functions or contracts.Since internal functions can only be called from inside the contracts, it makes no sense to have them if they are not used. This uses up gas and causes issues for auditors when understanding the contract logic.

Recommendations:

Having dead code in the contracts uses up unnecessary gas and increases the complexity of the overall smart contract. It is recommended to remove the internal functions from the contracts if they are never used.

IF-STATEMENT REFACTORING

In Solidity, we aim to write clear, efficient code that is both easy to understand and maintain. If statements can be converted to ternary operators. While using ternary operators instead of if/else statements can sometimes lead to more concise code, it's crucial to understand the trade-offs involved.

Recommendations:

To optimize your Solidity code, consider converting simple if/else statements to ternary operators, particularly for single-line arithmetic or logical operations. Utilizing ternary operators can improve code conciseness and readability. However, be mindful of code complexity and readability concerns. If the if/else statement is not single-line or involves multiple operations, retaining it for clarity is advisable.

This is a comprehensive report based on our automated and manual examination of cybersecurity vulnerabilities and framework flaws of the project's smart contract. Reading the full analysis report is essential to build your understanding of project's security level. It is crucial to take note, though we have done our best to perform this analysis and report, that you should not rely on the our research and cannot claim what it states or how we created it. Before making any judgments, you have to conduct your own independent research. We will discuss this in more depth in the following disclaimer - please read it fully.

DISCLAIMER: You agree to the terms of this disclaimer by reading this report or any portion thereof. Please stop reading this report and remove and delete any copies of this report that you download and/or print if you do not agree to these conditions. This report is for non-reliability information only and does not represent investment advice. No one shall be entitled to depend on the report or its contents, and Soken and its affiliates shall not be held responsible to you or anyone else, nor shall Soken provide any guarantee or representation to any person with regard to the accuracy or integrity of the report.
Without any terms, warranties or other conditions other than as set forth in that exclusion and Soken excludes hereby all representations, warrants, conditions and other terms (including, without limitation, guarantees implied by the law of satisfactory quality, fitness for purposes and the use of reasonable care and skills).
The report is provided as "as is" and does not contain any terms and conditions. Except as legally banned, Soken disclaims all responsibility and responsibilities and no claim against Soken is made to any amount or type of loss or damages (without limitation, direct, indirect, special, punitive, consequential or pure economic loses or losses) that may be caused by you or any other person, or any damages or damages, including without limitations (whether innocent or negligent).
Security analysis is based only on the smart contracts. No applications or operations were reviewed for security. No product code has been reviewed.

SOKEN CONTACT INFO

WEBSITE: WWW.SOKEN.IO

TELEGRAM: @SOKEN_SUPPORT

X (TWITTER): @SOKEN_TEAM