Byte Detective By: Kronobit

Smart Contracts Audit

https://kronobit.org Email: kronobit.coin@gmail.com

Project Name:	Smart Contract Address:
LitteMoose	0x17d8C396a55D6c1c2130EfC23f4936b56860Df9C
Block chain token:	**Website:**
BSC	https:// lmoosecoin.com
Telegram:	**Discord:**
@lmooseglobal	

LOGO

File: Moose. Sol **Language:** solidity **Size:** 32279 bytes **Date:** 2021-11-13T21:09:50.534ZLP
Contract owner = 0x0000000000000000000000000000000000000000

- contract code (owner part) OK

Token Status:

Total supply: 100000000000000000000000000 **Holders:** 16,054 Buy Tax: 10% sell Tax: 10%

Burned tokens: 8,774,384,282,977,150,000,000,000.434806802 (87.7438 %)

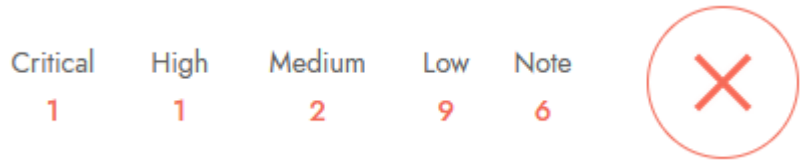
LP address = 0x2cb54a980c158b7271f0afa1778693f5efd7211f
Burned or locked : Burn Address (79.5279%)
Burned or locked : 0x0000000000000000000000000000000000000000 (0.0000%)
dev address : 0xb1b9... (15.89%)
dev address : 0x8adb... (1.14%)
dev address : 0x58d9... (0.74%)
dev address : 0xdd51... (0.58%)
dev address : 0x5554... (0.42%)

TOP HOLDERS INFO

- address 0xb2f5... has 0.2304 % tokens supply
- address 0x10f3... has 0.1011 % tokens supply
- address 0xf9e4... has 0.0895 % tokens supply
- address 0x55a8... has 0.0874 % tokens supply

Honeypot check:

✔ **Honeypot tests passed.** Our system was able to buy and sell it successfully.



Critical Security vulnerability Found

```
⊖  A security vulnerability has been detected.

59
60      function sub(uint256 a, uint256 b) internal pure returns (uint256) {
61          return sub(a, b, "SafeMath: subtraction overflow");
```

Report:

One of the major dangers of calling external contracts is that they can take over the control flow. In the reentrancy attack (a.k.a. recursive call attack), a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways.

High Security vulnerability Found

```
A security vulnerability has been detected.

135         // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
136         (bool success, ) = recipient.call{value: amount}("");
137         require(success, "Address: unable to send value, recipient may have reverted");
```

Report:

The return value of a message call is not checked. Execution will resume even if the called contract throws an exception. If the call fails accidentally or an attacker forces the call to fail, this may cause unexpected behavior in the subsequent program logic.

Medium Security Vulnerability Found

```
A security vulnerability has been detected.

133     function sendValue(address payable recipient, uint256 amount) internal {
134         require(address(this).balance >= amount, "Address: insufficient balance");
135         // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
```

```
A security vulnerability has been detected.

165     ) internal returns (bytes memory) {
166         require(address(this).balance >= value, "Address: insufficient balance for call");
167         return _functionCallWithValue(target, data, value, errorMessage);
```

Report:

Contracts can behave erroneously when they strictly assume a specific Ether balance. It is always possible to forcibly send ether to a contract (without triggering its fallback function), using self-destruct, or by mining to the account. In the worst-case scenario, this could lead to DOS conditions that might render the contract unusable.

Low Security Vulnerability Found

```
A security vulnerability has been detected.

12
13  pragma solidity ^0.8.0;
```

Report:

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Final report:

Unused variables are allowed in Solidity and they do not pose a direct security issue. It is best practice though to avoid them as they can:

- Cause an increase in computations (and unnecessary gas consumption)
- indicate bugs or malformed data structures and they are generally a sign of poor code quality
- cause code noise and decrease readability of the code
- LittleMoose token needs some corrections at future

Littlemose token is 82% safe 

The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.