f68a38dc6b2a9af2d9f99cac8028f4d3c69f00aa017555b7ecc5f33112085aa1

File: dogemoon.sol | Language:solidity | Size:35185 bytes | Date:2021-05-10T11:05:49.386Z

Critical	High	Medium	Low	Note
0	0	1	5	17

Issues

Severity	Issue	Analyzer	Code Lines
Medium	SWC-102	Achilles	7
Low	SOOHO-SOL-001	Achilles	265, 270, 271, 325, 325
Note	SWC-108	Achilles	525
Note	SWC-116	Achilles	885, 900
Note	SWC-131	Achilles	473, 473, 473, 478, 478, 478, 483, 483, 483, 483, 488, 488, 88

Code

1. SWC-102 / lines: 7 Medium Achilles

A security vulnerability has been detected.

```
6
7    pragma solidity ^0.6.12;
8
```

In detail

Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version.

2. SOOHO-SOL-001 / lines: 265 Low Achilles

A security vulnerability has been detected.

```
264      function getPair(address tokenA, address tokenB) external view returns (address pair);
265      function allPairs(uint) external view returns (address pair);
266      function allPairsLength() external view returns (uint);
```

In detail

Empty function parameter name

The name of function parameter is empty.

3. SOOHO-SOL-001 / lines: 270 Low Achilles

A security vulnerability has been detected.

```
269
270      function setFeeTo(address) external;
271      function setFeeToSetter(address) external;
```

In detail

Empty function parameter name
The name of function parameter is empty.

4. SOOHO-SOL-001 / lines: 271 Low Achilles

A security vulnerability has been detected.

```
    function setFeeTo(address) external;
    function setFeeToSetter(address) external;
}
```

In detail

Empty function parameter name
The name of function parameter is empty.

5. SOOHO-SOL-001 / lines: 325 Low Achilles

A security vulnerability has been detected.

```

    function initialize(address, address) external;
}
```

In detail

Empty function parameter name
The name of function parameter is empty.

6. SOOHO-SOL-001 / lines: 325 Low Achilles

A security vulnerability has been detected.

```

    function initialize(address, address) external;
}
```

In detail

Empty function parameter name
The name of function parameter is empty.

7. SWC-108 / lines: 525 Note Achilles

A security vulnerability has been detected.

```

    bool inSwapAndLiquify;
    bool public swapAndLiquifyEnabled = true;
```

In detail

Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.

8. SWC-116 / lines: 885 Note Achilles

A security vulnerability has been detected.

```
            address(this),
            block.timestamp
        );
```

In detail

Contracts often need access to the current timestamp to trigger time-dependent events. As Ethereum is decentralized, nodes can synchronize time only to some degree. Moreover, malicious miners can alter the timestamp of their blocks, especially if they can gain advantages by doing so. However, miners can't set timestamp smaller than the previous one (otherwise the block will be rejected), nor can they set the timestamp too far ahead in the future. Taking all of the above into consideration, developers can't rely on the preciseness of the provided timestamp.

9. SWC-116 / lines: 900 Note Achilles

A security vulnerability has been detected.

```
            owner(),
            block.timestamp
        );
```

In detail

Contracts often need access to the current timestamp to trigger time-dependent events. As Ethereum is decentralized, nodes can synchronize time only to some degree. Moreover, malicious miners can alter the timestamp of their blocks, especially if they can gain advantages by doing so. However, miners can't set timestamp smaller than the previous one (otherwise the block will be rejected), nor can they set the timestamp too far ahead in the future. Taking all of the above into consideration, developers can't rely on the preciseness of the provided timestamp.

10. SWC-131 / lines: 473 Note Achilles

A security vulnerability has been detected.

```
library TransferHelper {
    function safeApprove(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('approve(address,uint256)')));
```

In detail

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

11. SWC-131 / lines: 473 Note Achilles

A security vulnerability has been detected.

```
library TransferHelper {
    function safeApprove(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('approve(address,uint256)')));
```

In detail

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

12. SWC-131 / lines: 473 Note Achilles

A security vulnerability has been detected.

```
library TransferHelper {
    function safeApprove(address token, address to, uint value) internal {
        // bytes4(keccak256(bytes('approve(address,uint256)')));
```

In detail

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

13. SWC-131 / lines: 478 Note Achilles

A security vulnerability has been detected.

```
478     function safeTransfer(address token, address to, uint value) internal {
479         // bytes4(keccak256(bytes('transfer(address,uint256)')));
```

In detail

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

14. SWC-131 / lines: 478 Note Achilles

`A security vulnerability has been detected.`

```
477
478     function safeTransfer(address token, address to, uint value) internal {
479         // bytes4(keccak256(bytes('transfer(address,uint256)')));
```

In detail

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

15. SWC-131 / lines: 478 Note Achilles

`A security vulnerability has been detected.`

```
477
478     function safeTransfer(address token, address to, uint value) internal {
479         // bytes4(keccak256(bytes('transfer(address,uint256)')));
```

In detail

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

16. SWC-131 / lines: 483 Note Achilles

`A security vulnerability has been detected.`

```
482
483     function safeTransferFrom(address token, address from, address to, uint value) internal {
484         // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
```

In detail

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

17. SWC-131 / lines: 483 Note Achilles

`A security vulnerability has been detected.`

```
482
483     function safeTransferFrom(address token, address from, address to, uint value) internal {
484         // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
```

In detail

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

18. SWC-131 / lines: 483 Note Achilles

A security vulnerability has been detected.

```
    function safeTransferFrom(address token, address from, address to, uint value) internal {
        // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
```

In detail

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

19. SWC-131 / lines: 483 Note Achilles

A security vulnerability has been detected.

```
    function safeTransferFrom(address token, address from, address to, uint value) internal {
        // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
```

In detail

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

20. SWC-131 / lines: 488 Note Achilles

A security vulnerability has been detected.

```
    function safeTransferETH(address to, uint value) internal {
        require(success, 'TransferHelper: ETH_TRANSFER_FAILED');
```

In detail

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

21. SWC-131 / lines: 488 Note Achilles

A security vulnerability has been detected.

```
    function safeTransferETH(address to, uint value) internal {
        require(success, 'TransferHelper: ETH_TRANSFER_FAILED');
```

In detail

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

22. SWC-131 / lines: 889 Achilles

A security vulnerability has been detected.

```
888
889        function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
890            // approve token transfer to cover all possible scenarios
```

In detail

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

23. SWC-131 / lines: 507

Achilles

```
A security vulnerability has been detected.
```

```
506
507        uint256 private constant MAX = ~uint256(0);
508        uint256 private _tTotal = 1000000000 * 10**6 * 10**9;
```

In detail

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