

REPORT 607F185F24923100115FC1B4

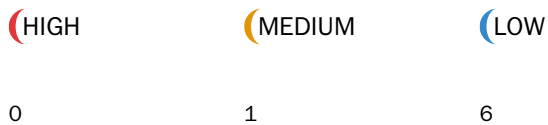
Created Tue Apr 20 2021 18:07:27 GMT+0000 (Coordinated Universal Time)
Number of analyses 1
User 607f174267db361c31bacfdc

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
4914d7d5-d96a-4ed1-8d69-76749ba274ea	/contracts/dogxtoken.sol	7

Started	Tue Apr 20 2021 18:07:32 GMT+0000 (Coordinated Universal Time)
Finished	Tue Apr 20 2021 18:09:42 GMT+0000 (Coordinated Universal Time)
Mode	Quick
Client Tool	Mythx-Vscode-Extension
Main Source File	/Contracts/Dogxtoken.Sol

DETECTED VULNERABILITIES



ISSUES

MEDIUM Function could be marked as external.

SWC-000

The function definition of "mint" is marked "public". However, it is never directly called by another function in the same contract or in any of its descendants. Consider to mark it as "external" instead.

Source file

/contracts/dogxtoken.sol

Locations

```
34 | contract DogXToken is BEP20('DogX Token', 'DOGX') {
35 |     /// @notice Creates '_amount' token to '_to'. Must only be called by the owner (MasterChef).
36 |     function mint(address _to, uint256 _amount) public onlyOwner {
37 |         mint(_to, _amount);
38 |         moveDelegates(address(0), _delegates[_to], _amount);
39 |     }
40 |
41 |     /// Copied and modified from YAM code:
42 |     // https://github.com/yam-finance/yam-protocol/blob/master/contracts/token/YAMGovernanceStorage.sol
43 |     // https://github.com/yam-finance/yam-protocol/blob/master/contracts/token/YAMGovernance.sol
```

LOW A control flow decision is made based on The block.timestamp environment variable.

SWC-116

The block.timestamp environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

/contracts/dogxtoken.sol

Locations

```
143 | require(signatory != address(0), "DogX::delegateBySig: invalid signature");
144 | require(nonce == nonces[signatory]++, "DogX::delegateBySig: invalid nonce");
145 | require(now <= expiry, "DogX::delegateBySig: signature expired");
146 | return _delegate(signatory, delegatee);
147 | }
```

LOW Potential use of "block.number" as source of randomness.

SWC-120

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

/contracts/dogxtoken.sol

Locations

```
173 | returns (uint256)
174 | {
175 |     require(blockNumber < block.number, "DogX::getPriorVotes: not yet determined");
176 |
177 |     uint32 nCheckpoints = numCheckpoints[account];
```

LOW Potential use of "block.number" as source of randomness.

SWC-120

The environment variable "block.number" looks like it might be used as a source of randomness. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

/contracts/dogxtoken.sol

Locations

```
246 | internal
247 | {
248 |     uint32 blockNumber = safe32(block.number, "DogX::_writeCheckpoint: block number exceeds 32 bits");
249 |
250 |     if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber) {
```

LOW A control flow decision is made based on The block.number environment variable.

SWC-120

The block.number environment variable is used to determine a control flow decision. Note that the values of variables like coinbase, gaslimit, block number and timestamp are predictable and can be manipulated by a malicious miner. Also keep in mind that attackers know hashes of earlier blocks. Don't use any of those environment variables as sources of randomness and be aware that use of these variables introduces a certain level of trust into miners.

Source file

/contracts/dogxtoken.sol

Locations

```
173 | returns (uint256)
174 | {
175 |     require(blockNumber < block.number, "DogX::getPriorVotes: not yet determined");
176 |
177 |     uint32 nCheckpoints = numCheckpoints[account];
178 |     if (nCheckpoints == 0) {
179 |         return 0;
```

LOW

Potentially unbounded data structure passed to builtin.

SWC-128

Gas consumption in function "delegateBySig" in contract "DogXToken" depends on the size of data structures that may grow unboundedly. Specifically the "1-st" argument to builtin "keccak256" may be able to grow unboundedly causing the builtin to consume more gas than the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Source file

/contracts/dogxtoken.sol

Locations

```
117 | abi.encode(  
118 | DOMAIN_TYPEHASH,  
119 | keccak256(bytes(name())),  
120 | getChainId(),  
121 | address(this)  
122 | )
```

LOW

Loop over unbounded data structure.

SWC-128

Gas consumption in function "getPriorVotes" in contract "DogXToken" depends on the size of data structures or values that may grow unboundedly. If the data structure grows too large, the gas required to execute the code will exceed the block gas limit, effectively causing a denial-of-service condition. Consider that an attacker might attempt to cause this condition on purpose.

Source file

/contracts/dogxtoken.sol

Locations

```
192 | uint32 lower = 0;  
193 | uint32 upper = nCheckpoints - 1;  
194 | while (upper > lower) {  
195 |     uint32 center = upper - (upper - lower) / 2; // ceil, avoiding overflow  
196 |     Checkpoint memory cp = checkpoints[account][center];  
197 |     if (cp.fromBlock == blockNumber) {
```