

REPORT 605D152BC5719A00123BB6C2

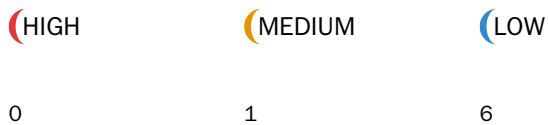
Created Thu Mar 25 2021 22:56:43 GMT+0000 (Coordinated Universal Time)
Number of analyses 1
User poopswap@outlook.com

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
209f096a-840c-4958-a674-3d2402644cc5	/contracts/pooptoken.sol	7

Started	Thu Mar 25 2021 22:56:48 GMT+0000 (Coordinated Universal Time)
Finished	Thu Mar 25 2021 23:42:36 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Vscode-Extension
Main Source File	/Contracts/Pooptoken.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/pooptoken.sol

Locations

```
9 |
10 | /// @notice Creates `_amount` token to `_to`. Must only be called by the owner (MasterPoop).
11 | function mint(address _to, uint256 _amount) public onlyOwner {
12 |     mint(_to, _amount);
13 |     moveDelegates(address(0), _delegates[_to], _amount);
14 | }
15 |
16 | /// @dev overrides transfer function to meet tokenomics of POOP
```

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/pooptoken.sol

Locations

```
113 | require(signatory != address(0), "POOP::delegateBySig: invalid signature");
114 | require(nonce == nonces[signatory]++, "POOP::delegateBySig: invalid nonce");
115 | require(now <= expiry, "POOP::delegateBySig: signature expired");
116 | return _delegate(signatory, delegatee);
117 | }
```

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/pooptoken.sol

Locations

```
135 | */
136 | function getPriorVotes(address account, uint256 blockNumber) external view returns (uint256) {
137 |     require(blockNumber < block.number, "POOP::getPriorVotes: not yet determined");
138 |
139 |     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/pooptoken.sol

Locations

```
208 | uint256 newVotes
209 | ) internal {
210 |     uint32 blockNumber = safe32(block.number, "POOP::_writeCheckpoint: block number exceeds 32 bits");
211 |
212 |     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/pooptoken.sol

Locations

```
135 | */
136 | function getPriorVotes(address account, uint256 blockNumber) external view returns (uint256) {
137 |     require(blockNumber < block.number, "POOP::getPriorVotes: not yet determined");
138 |
139 |     uint32 nCheckpoints = numCheckpoints[account];
```

LOW Potentially unbounded data structure passed to builtin.

SWC-128

Gas consumption in function "delegateBySig" in contract "PoopToken" 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/pooptoken.sol

Locations

```
104 | bytes32 s
105 | } external {
106 | bytes32 domainSeparator = keccak256(abi.encode(DOMAIN_TYPEHASH, keccak256(bytes(name[1])), getChainId(), address(this)));
107 |
108 | bytes32 structHash = keccak256(abi.encode(DELEGATION_TYPEHASH, delegatee, nonce, expiry));
```

LOW Loop over unbounded data structure.

SWC-128

Gas consumption in function "getPriorVotes" in contract "PoopToken" 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/pooptoken.sol

Locations

```
154 | uint32 lower = 0;
155 | uint32 upper = nCheckpoints - 1;
156 | while (upper > lower) {
157 |     uint32 center = upper - (upper - lower) / 2; // ceil, avoiding overflow
158 |     Checkpoint memory cp = checkpoints[account][center];
```