MythX

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

Analysis 209f096a-840c-4958-a674-3d2402644cc5

MythX

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

HIGH	MEDIUM	LOW
0	1	6

ISSUES

MEDIUM SWC-000 Function could be marked as external.

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

```

/// @notice Creates `_amount` token to `_to`. Must only be called by the owner (MasterPoop).
function mint(address _to, uint256 _amount) public onlyOwner {
_mint(_to, _amount);
_moveDelegates(address(0), _delegates[_to], _amount);
}

/// @dev overrides transfer function to meet tokenomics of POOP
```

LOW SWC-116 A control flow decision is made based on The block.timestamp environment variable.

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

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

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

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

```
*/
function getPriorVotes(address account, uint256 blockNumber) external view returns (uint256) {
require(blockNumber < block.number, "POOP::getPriorVotes: not yet determined");

uint32 nCheckpoints = numCheckpoints[account];
```

LOW

SWC-120

Potential use of "block.number" as source of randonmness.

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

```
uint256 newVotes
) internal {
uint32 blockNumber = safe32(block.number, "POOP::_writeCheckpoint: block number exceeds 32 bits");

if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber) {
```

LOW

SWC-120

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

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

```
*/
function getPriorVotes(address account, uint256 blockNumber) external view returns (uint256) {
require(blockNumber < block.number, "POOP::getPriorVotes: not yet determined");

uint32 nCheckpoints = numCheckpoints[account];
```

LOW

SWC-128

Potentially unbounded data structure passed to builtin.

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

```
bytes32 s
) external {
bytes32 domainSeparator = keccak256(abi.encode(DOMAIN_TYPEHASH, keccak256(bytes(name())), getChainId(), address(this)));

bytes32 structHash = keccak256(abi.encode(DELEGATION_TYPEHASH, delegatee, nonce, expiry));
```

LOW

SWC-128

Loop over unbounded data structure.

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

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