MythX

REPORT 605D1576DC5BD6001191EEBF

Created	Thu Mar 25 2021 22:57:58 GMT+0000 (Coordinated Universal Time)
Number of analyses	1
User	poopswap@outlook.com

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
6cc63433-1e44-4c39-a559-82b51281bf37	/contracts/masterpoop.sol	32

Analysis 6cc63433-1e44-4c39-a559-82b51281bf37 MythX

Started	Thu Mar 25 2021 22:58:08 GMT+0000 (Coordinated Universal Time)
Finished	Thu Mar 25 2021 23:43:30 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Vscode-Extension
Main Source File	/Contracts/Masterpoop.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	9	23

ISSUES

MEDIUM **Function could be marked as external.**

SWC-000

The function definition of "add" 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/masterpoop.sol

Locations

```
110 massUpdatePools();
111 }
112 uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
113 totalAllocPoint = totalAllocPoint.add(_allocPoint);
114 poolExistence[_lpToken] = true;
115 poolInfo.push(
116 PoolInfo({lpToken: _lpToken, allocPoint: _allocPoint, lastRewardBlock: lastRewardBlock, accPoopPerShare: 0, depositFeeBP: _depositFeeBP})
117 );
118 }
119
120 // Update the given pool's POOP allocation point and deposit fee. Can only be called by the owner.
121 function set(
122 uint256 _pid,
123 uint256 _allocPoint,
124 uint16 _depositFeeBP,
125 bool _withUpdate
126 ) public onlyOwner {
127 require(_depositFeeBP <= 10000, "set: invalid deposit fee basis points");
128 if (_withUpdate) { if (_withUpdate) {
129 massUpdatePools();
130 }
```

MEDIUM Function could be marked as external.

SWC-000 The function definition of "set" 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/masterpoop.sol

Locations

```
129 massUpdatePools();
130 }
131 totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
132 poolInfo[_pid].allocPoint = _allocPoint;
133 poolInfo[_pid].depositFeeBP = _depositFeeBP;
134 }
135
136 // Return reward multiplier over the given _from to _to block.
137 function getMultiplier(uint256 _from, uint256 _to) public view returns (uint256) {
138 return _to.sub(_from).mul(BONUS_MULTIPLIER);
139 }
140
141 // View function to see pending POOPs on frontend.
142 function pendingPoop(uint256 _pid, address _user) external view returns (uint256) {
143 PoolInfo storage pool = poolInfo[_pid];
144 UserInfo storage user = userInfo[_pid][_user];
```

MEDIUM Function could be marked as external.

SWC-000 The function definition of "deposit" 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/masterpoop.sol

Locations

```
186 updatePool(_pid);
187 if (user.amount > 0) {
188 uint256 pending = user.amount.mul(pool.accPoopPerShare).div(1e12).sub(user.rewardDebt);
189 if (pending > 0) {
190 safePoopTransfer(msg.sender, pending);
191 }
192 }
193 if (_amount > 0) {
194 pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
195 if (pool.depositFeeBP > 0) {
196 uint256 depositFee = _amount.mul(pool.depositFeeBP).div(10000);
197 pool.lpToken.safeTransfer(feeAddress, depositFee);
198 user.amount = user.amount.add(_amount).sub(depositFee);
199 } else {
200 user.amount = user.amount.add(_amount);
201 }
202 }
203 user.rewardDebt = user.amount.mul(pool.accPoopPerShare).div(1e12);
204 emit Deposit(msg.sender, _pid, _amount);
205 }
206
207 // Withdraw LP tokens from MasterPoop.
208 function withdraw(uint256 _pid, uint256 _amount) public nonReentrant {
209 PoolInfo storage pool = poolInfo[_pid];
210 UserInfo storage user = userInfo[_pid][msg.sender];
211 require(user.amount >= _amount, "withdraw: not good");
212 updatePool(_pid);
213 uint256 pending = user.amount.mul(pool.accPoopPerShare).div(1e12).sub(user.rewardDebt);
214 if (pending > 0) {
215 safePoopTransfer(msg.sender, pending);
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "withdraw" 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/masterpoop.sol

Locations

```
211 require(user.amount >= _amount, "withdraw: not good");
212 updatePool(_pid);
213 uint256 pending = user.amount.mul(pool.accPoopPerShare).div(1e12).sub(user.rewardDebt);
214 if (pending > 0) {
215 safePoopTransfer(msg.sender, pending);
216 }
217 if (_amount > 0) {
218 user.amount = user.amount.sub(_amount);
219 pool.lpToken.safeTransfer(address(msg.sender), _amount);
220 }
221 user.rewardDebt = user.amount.mul(pool.accPoopPerShare).div(1e12);
222 emit Withdraw(msg.sender, _pid, _amount);
223 }
224
225 // Withdraw without caring about rewards. EMERGENCY ONLY.
226 function emergencyWithdraw(uint256 _pid) public nonReentrant {
227 PoolInfo storage pool = poolInfo[_pid];
228 UserInfo storage user = userInfo[_pid][msg.sender];
229 uint256 amount = user.amount;
230 user.amount = 0;
231 user.rewardDebt = 0;
232 pool.lpToken.safeTransfer(address(msg.sender), amount);
233 emit EmergencyWithdraw(msg.sender, _pid, amount);
234 }
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "emergencyWithdraw" 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/masterpoop.sol

Locations

```
231 user.rewardDebt = 0;
232 pool.lpToken.safeTransfer(address(msg.sender), amount);
233 emit EmergencyWithdraw(msg.sender, _pid, amount);
234 }
235
236 // Safe poop transfer function, just in case if rounding error causes pool to not have enough POOPs.
237 function safePoopTransfer(address _to, uint256 _amount) internal {
238 uint256 poopBal = poop.balanceOf(address(this));
239 bool transferSuccess = false;
240 if (_amount > poopBal) {
241 transferSuccess = poop.transfer(_to, poopBal);
242 } else {
243 transferSuccess = poop.transfer(_to, _amount);
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "dev" 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/masterpoop.sol

Locations

```
254
255 function setFeeAddress(address _feeAddress) public {
256 require(msg.sender == feeAddress, "setFeeAddress: NOPE!");
257 feeAddress = _feeAddress;
258 emit SetFeeAddress(msg.sender, _feeAddress);
259 }
260
261 //Pancake has to add hidden dummy pools inorder to alter the emission, here we make it simple and transparent to all.
262 function updateEmissionRate(uint256 _poopPerBlock) public onlyOwner {
263 massUpdatePools();
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "setFeeAddress" 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/masterpoop.sol

Locations

```
259 }
260
261 //Pancake has to add hidden dummy pools inorder to alter the emission, here we make it simple and transparent to all.
262 function updateEmissionRate(uint256 _poopPerBlock) public onlyOwner {
263 massUpdatePools();
264 poopPerBlock = _poopPerBlock;
265 emit UpdateEmissionRate(msg.sender, _poopPerBlock);
266 }
267 }
```

MEDIUM

Loop over unbounded data structure.

SWC-128

Gas consumption in function "massUpdatePools" in contract "MasterPoop" 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/masterpoop.sol

Locations

```
167 return;
168 }
169 uint256 lpSupply = pool.lpToken.balanceOf(address(this));
170 if (lpSupply == 0 || pool.allocPoint == 0) {
171 pool.lastRewardBlock = block.number;
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/contracts/masterpoop.sol

Locations

```
201 }
202 }
203 user.rewardDebt = user.amount.mul(pool.accPoopPerShare).div(1e12);
204 emit Deposit(msg.sender, _pid, _amount);
205 }
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/contracts/masterpoop.sol

Locations

```
202 }
203 user.rewardDebt = user.amount.mul(pool.accPoopPerShare).div(1e12);
204 emit Deposit(msg.sender, _pid, _amount);
205 }
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/contracts/masterpoop.sol

Locations

```
205 }
206
207 // Withdraw LP tokens from MasterPoop.
208 function withdraw(uint256 _pid, uint256 _amount) public nonReentrant {
209 PoolInfo storage pool = poolInfo[_pid];
210 UserInfo storage user = userInfo[_pid][msg.sender];
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/contracts/masterpoop.sol

Locations

```
205 }
206
207 // Withdraw LP tokens from MasterPoop.
208 function withdraw(uint256 _pid, uint256 _amount) public nonReentrant {
209 PoolInfo storage pool = poolInfo[_pid];
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/contracts/masterpoop.sol

Locations

```
206
207 // Withdraw LP tokens from MasterPoop.
208 function withdraw(uint256 _pid, uint256 _amount) public nonReentrant {
209 PoolInfo storage pool = poolInfo[_pid];
210 UserInfo storage user = userInfo[_pid][msg.sender];
```

LOW

SWC-107

Write to persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/contracts/masterpoop.sol

Locations

```
206
207 // Withdraw LP tokens from MasterPoop.
208 function withdraw(uint256 _pid, uint256 _amount) public nonReentrant {
209 PoolInfo storage pool = poolInfo[_pid];
210 UserInfo storage user = userInfo[_pid][msg.sender];
211 require(user.amount >= _amount, "withdraw: not good");
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/contracts/masterpoop.sol

Locations

```
209 PoolInfo storage pool = poolInfo[_pid];
210 UserInfo storage user = userInfo[_pid][msg.sender];
211 require(user.amount >= _amount, "withdraw: not good");
212 updatePool(_pid);
213 uint256 pending = user.amount.mul(pool.accPoopPerShare).div(1e12).sub(user.rewardDebt);
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/contracts/masterpoop.sol

Locations

```
208 function withdraw(uint256 _pid, uint256 _amount) public nonReentrant {
209 PoolInfo storage pool = poolInfo[_pid];
210 UserInfo storage user = userInfo[_pid][msg.sender];
211 require(user.amount >= _amount, "withdraw: not good");
212 updatePool(_pid);
213 uint256 pending = user.amount.mul(pool.accPoopPerShare).div(1e12).sub(user.rewardDebt);
```

LOW

SWC-107

Write to persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/contracts/masterpoop.sol

Locations

```
208 function withdraw(uint256 _pid, uint256 _amount) public nonReentrant {
209 PoolInfo storage pool = poolInfo[_pid];
210 UserInfo storage user = userInfo[_pid][msg.sender];
211 require(user.amount >= _amount, "withdraw: not good");
212 updatePool(_pid);
213 uint256 pending = user.amount.mul(pool.accPoopPerShare).div(1e12).sub(user.rewardDebt);
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/contracts/masterpoop.sol

Locations

```
207 // Withdraw LP tokens from MasterPoop.
208 function withdraw(uint256 _pid, uint256 _amount) public nonReentrant {
209 PoolInfo storage pool = poolInfo[_pid];
210 UserInfo storage user = userInfo[_pid][msg.sender];
211 require(user.amount >= _amount, "withdraw: not good");
212 updatePool(_pid);
```

LOW

SWC-107

Write to persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/contracts/masterpoop.sol

Locations

```
207 // Withdraw LP tokens from MasterPoop.
208 function withdraw(uint256 _pid, uint256 _amount) public nonReentrant {
209 PoolInfo storage pool = poolInfo[_pid];
210 UserInfo storage user = userInfo[_pid][msg.sender];
211 require(user.amount >= _amount, "withdraw: not good");
212 updatePool(_pid);
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/contracts/masterpoop.sol

Locations

```
226 function emergencyWithdraw(uint256 _pid) public nonReentrant {
227 PoolInfo storage pool = poolInfo[_pid];
228 UserInfo storage user = userInfo[_pid][msg.sender];
229 uint256 amount = user.amount;
230 user.amount = 0;
231 user.rewardDebt = 0;
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/contracts/masterpoop.sol

Locations

```
226 function emergencyWithdraw(uint256 _pid) public nonReentrant {
227 PoolInfo storage pool = poolInfo[_pid];
228 UserInfo storage user = userInfo[_pid][msg.sender];
229 uint256 amount = user.amount;
230 user.amount = 0;
```

LOW

SWC-107

Write to persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

/contracts/masterpoop.sol

Locations

```
226 function emergencyWithdraw(uint256 _pid) public nonReentrant {
227 PoolInfo storage pool = poolInfo[_pid];
228 UserInfo storage user = userInfo[_pid][msg.sender];
229 uint256 amount = user.amount;
230 user.amount = 0;
231 user.rewardDebt = 0;
```

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/masterpoop.sol

Locations

```
114 poolExistence[_lpToken] = true;
115 poolInfo.push(
116 PoolInfo({lpToken: _lpToken, allocPoint: _allocPoint, lastRewardBlock: lastRewardBlock, accPoopPerShare: 0, depositFeeBP: _depositFeeBP})
117 );
118 }
119
120 // Update the given pool's POOP allocation point and deposit fee. Can only be called by the owner.
121 function set(
```

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/masterpoop.sol

Locations

```
118 }
119
120 // Update the given pool's POOP allocation point and deposit fee. Can only be called by the owner.
121 function set(
122 uint256 _pid,
```

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/masterpoop.sol

Locations

```
148 uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
149 uint256 poopReward = multiplier.mul(poopPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
150 accPoopPerShare = accPoopPerShare.add(poopReward.mul(1e12).div(lpSupply));
151 }
152 return user.amount.mul(accPoopPerShare).div(1e12).sub(user.rewardDebt); return user.amount.mul(accPoopPerShare).div(1e12).sub(user.rewardDebt);
153 }
```

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/masterpoop.sol

Locations

```
153 }
154
155 // Update reward variables for all pools. Be careful of gas spending!
156 function massUpdatePools() public {
157 uint256 length = poolInfo.length;
```

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/masterpoop.sol

Locations

```
173 }
174 uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
175 uint256 poopReward = multiplier.mul(poopPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
176 poop.mint(devaddr, poopReward.div(10));
177 poop.mint(address(this), poopReward);
```

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/masterpoop.sol

Locations

```
176 poop.mint(devaddr, poopReward.div(10));
177 poop.mint(address(this), poopReward);
178 pool.accPoopPerShare = pool.accPoopPerShare.add(poopReward.mul(1e12).div(lpSupply));
179 pool.lastRewardBlock = block.number;
180 }
```

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/masterpoop.sol

Locations

```
177 poop.mint(address(this), poopReward);
178 pool.accPoopPerShare = pool.accPoopPerShare.add(poopReward.mul(1e12).div(lpSupply));
179 pool.lastRewardBlock = block.number;
180 }
```

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/masterpoop.sol

Locations

```
186 updatePool(_pid);
187 if (user.amount > 0) {
188 uint256 pending = user.amount.mul(pool.accPoopPerShare).div(1e12).sub(user.rewardDebt);
189 if (pending > 0) {
190 safePoopTransfer(msg.sender, pending);
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

/contracts/masterpoop.sol

Locations

```
173 }
174 uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
175 uint256 poopReward = multiplier.mul(poopPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
176 poop.mint(devaddr, poopReward.div(10));
177 poop.mint(address(this), poopReward);
178 pool.accPoopPerShare = pool.accPoopPerShare.add(poopReward.mul(1e12).div(lpSupply));
```