MythX

REPORT 607F1959951091001 18FDE79

Created	Tue Apr 20 2021 18:11:37 GMT+0000 (Coordinated Universal Time)
Number of analyses	1
User	607f174267db361c31bacfdc

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
a143a9e5-f40e-45b8-ac13-aedbec772ac7	/contracts/masterchef.sol	26

Analysis a143a9e5-f40e-45b8-ac13-aedbec772ac7 MythX

Started	Tue Apr 20 2021 18:11:42 GMT+0000 (Coordinated Universal Time)
Finished	Tue Apr 20 2021 18:13:55 GMT+0000 (Coordinated Universal Time)
Mode	Quick
Client Tool	Mythx-Vscode-Extension
Main Source File	/Contracts/Masterchef.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	9	17

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/masterchef.sol
Locations

```
139 function add(uint256 _allocPoint, IBEP20 _lpToken, uint16 _depositFeeBP, bool _withUpdate
140 ) public onlyOwner nonDuplicated(_lpToken) {
141 require(_depositFeeBP <= MAXIMUM_DEPOSIT_FEE_BP, "add: invalid deposit fee basis points");
142 if (_withUpdate) {
143 massUpdatePools();
144 }
145 uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
146 totalAllocPoint = totalAllocPoint.add(_allocPoint);
147 poolExistence[_lpToken] = true;
148 poolInfo.push(
149 PoolInfo({
150 lpToken: _lpToken,
151 allocPoint: _allocPoint,
152 lastRewardBlock: lastRewardBlock,
153 accDogXPerShare: 0,
154 depositFeeBP: _depositFeeBP
155 })
156 );
157 poolIdForLpAddress[_lpToken] = poolInfo.length - 1;
158 }
159
160 // Update the given pool's DogX allocation point and deposit fee. Can only be called by the owner.
161 function set(uint256 _pid, uint256 _allocPoint, uint16 _depositFeeBP, bool _withUpdate) public onlyOwner {
162 require(_depositFeeBP <= MAXIMUM_DEPOSIT_FEE_BP, "set: invalid deposit fee basis points");
163 if (_withUpdate) {
```

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/masterchef.sol

Locations

```
160 // Update the given pool's DogX allocation point and deposit fee. Can only be called by the owner.
161 function set(uint256 _pid, uint256 _allocPoint, uint16 _depositFeeBP, bool _withUpdate) public onlyOwner {
162 require(_depositFeeBP <= MAXIMUM_DEPOSIT_FEE_BP, "set: invalid deposit fee basis points");
163 if (_withUpdate) {
164 massUpdatePools();
165 }
166 totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(
167 _allocPoint
168 );
169 poolInfo[_pid].allocPoint = _allocPoint;
170 poolInfo[_pid].depositFeeBP = _depositFeeBP;
171 }
172
173 // Return reward multiplier over the given _from to _to block.
174 function getMultiplier(uint256 _from, uint256 _to) public pure returns (uint256)
175 {
176 return _to.sub(_from);
177 }
```

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/masterchef.sol

Locations

```
236 function deposit(uint256 _pid, uint256 _amount) public nonReentrant {
237 PoolInfo storage pool = poolInfo[_pid];
238 UserInfo storage user = userInfo[_pid][msg.sender];
239 updatePool(_pid);
240 if (user.amount > 0) {
241 uint256 pending =
242 user.amount.mul(pool.accDogXPerShare).div(1e12).sub(
243 user.rewardDebt
244 );
245 if (pending > 0) {
246 safeDogXTransfer(msg.sender, pending);
247 }
248 }
249 if (_amount > 0) {
250 pool.lpToken.safeTransferFrom(address(msg.sender),address(this),_amount);
251 if (pool.depositFeeBP > 0) {
252 uint256 depositFee = _amount.mul(pool.depositFeeBP).div(10000);
253 user.amount = user.amount.add(_amount).sub(depositFee);
254 pool.lpToken.safeTransfer(feeAddress, depositFee);
255 } else {
256 user.amount = user.amount.add(_amount);
257 }
258 }
259 user.rewardDebt = user.amount.mul(pool.accDogXPerShare).div(1e12);
260 emit Deposit(msg.sender, _pid, _amount);
261 }
262
263 // Withdraw LP tokens from MasterChef.
264 function withdraw(uint256 _pid, uint256 _amount) public nonReentrant {
265 PoolInfo storage pool = poolInfo[_pid];
266 UserInfo storage user = userInfo[_pid][msg.sender];
267 require(user.amount >= _amount, "withdraw: not good");
268 updatePool(_pid);
```

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/masterchef.sol

Locations

```
264 function withdraw(uint256 _pid, uint256 _amount) public nonReentrant {
265 PoolInfo storage pool = poolInfo[_pid];
266 UserInfo storage user = userInfo[_pid][msg.sender];
267 require(user.amount >= _amount, "withdraw: not good");
268 updatePool(_pid);
269 uint256 pending =
270 user.amount.mul(pool.accDogXPerShare).div(1e12).sub(
271 user.rewardDebt
272 );
273 if (pending > 0) {
274 safeDogXTransfer(msg.sender, pending);
275 }
276 if (_amount > 0) {
277 user.amount = user.amount.sub(_amount);
278 pool.lpToken.safeTransfer(address(msg.sender), _amount);
279 }
280 user.rewardDebt = user.amount.mul(pool.accDogXPerShare).div(1e12);
281 emit Withdraw(msg.sender, _pid, _amount);
282 }
283
284 // Withdraw without caring about rewards. EMERGENCY ONLY.
285 function emergencyWithdraw(uint256 _pid) public nonReentrant {
286 PoolInfo storage pool = poolInfo[_pid];
287 UserInfo storage user = userInfo[_pid][msg.sender];
288 pool.lpToken.safeTransfer(address(msg.sender), user.amount);
```

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/masterchef.sol

Locations

```
286 PoolInfo storage pool = poolInfo[_pid];
287 UserInfo storage user = userInfo[_pid][msg.sender];
288 pool.lpToken.safeTransfer(address(msg.sender), user.amount);
289 emit EmergencyWithdraw(msg.sender, _pid, user.amount);
290 user.amount = 0;
291 user.rewardDebt = 0;
292 }
293
294 // Safe DogX transfer function, just in case if rounding error causes pool to not have enough DogXs.
295 function safeDogXTransfer(address _to, uint256 _amount) internal {
296 uint256 dogXBal = dogX.balanceOf(address(this));
297 bool transferSuccess = false;
```

MEDIUM

Function could be marked as external.

SWC-000

The function definition of "setDevAddress" 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/masterchef.sol

Locations

```
308 require(_devaddr != address(0), "dev: invalid address");
309 require(msg.sender == devAddr, "dev: wut?");
310 devAddr = _devaddr;
311 emit SetDevAddress(msg.sender, _devaddr);
312 }
313
314 // Update fee address by the previous fee address.
315 function setFeeAddress(address _feeAddress) public {
316 require(_feeAddress != address(0), "setFeeAddress: invalid address");
317 require(msg.sender == feeAddress, "setFeeAddress: FORBIDDEN");
318 feeAddress = _feeAddress;
```

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/masterchef.sol

Locations

```
315 function setFeeAddress(address _feeAddress) public {
316 require(_feeAddress != address(0), "setFeeAddress: invalid address");
317 require(msg.sender == feeAddress, "setFeeAddress: FORBIDDEN");
318 feeAddress = _feeAddress;
319 emit SetFeeAddress(msg.sender, _feeAddress);
320 }
321
322 //Pancake has to add hidden dummy pools inorder to alter the emission, here we make it simple and transparent to all.
323 function updateEmissionRate(uint256 _dogXPerBlock) public onlyOwner {
324 massUpdatePools();
325 dogXPerBlock = _dogXPerBlock;
```

MEDIUM

Loop over unbounded data structure.

SWC-128

Gas consumption in function "massUpdatePools" in contract "MasterChef" 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/masterchef.sol

Locations

```
210
211 // Update reward variables of the given pool to be up-to-date.
212 function updatePool(uint256 _pid) public {
213 PoolInfo storage pool = poolInfo[_pid];
214 if (block.number <= pool.lastRewardBlock) {
215 return;
```

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/masterchef.sol

Locations

```
292 }
293
294 // Safe DogX transfer function, just in case if rounding error causes pool to not have enough DogXs.
295 function safeDogXTransfer(address _to, uint256 _amount) internal {
296 uint256 dogXBal = dogX.balanceOf(address(this));
297 bool transferSuccess = false;
```

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/masterchef.sol

Locations

```
293
294 // Safe DogX transfer function, just in case if rounding error causes pool to not have enough DogXs.
295 function safeDogXTransfer(address _to, uint256 _amount) internal {
296 uint256 dogXBal = dogX.balanceOf(address(this));
297 bool transferSuccess = false;
```

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/masterchef.sol

Locations

```
293
294 // Safe DogX transfer function, just in case if rounding error causes pool to not have enough DogXs.
295 function safeDogXTransfer(address _to, uint256 _amount) internal {
296 uint256 dogXBal = dogX.balanceOf(address(this));
297 bool transferSuccess = false;
```

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/masterchef.sol

Locations

```
251 if (pool.depositFeeBP > 0) {
252 uint256 depositFee = _amount.mul(pool.depositFeeBP).div(10000);
253 user.amount = user.amount.add(_amount).sub(depositFee);
254 pool.lpToken.safeTransfer(feeAddress, depositFee);
255 } else {
256 user.amount = user.amount.add(_amount);
```

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/masterchef.sol

Locations

```
259 user.rewardDebt = user.amount.mul(pool.accDogXPerShare).div(1e12);
260 emit Deposit(msg.sender, _pid, _amount);
261 }
262
263 // Withdraw LP tokens from MasterChef.
264 function withdraw(uint256 _pid, uint256 _amount) public nonReentrant {
265 PoolInfo storage pool = poolInfo[_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/masterchef.sol

Locations

```
258 }
259 user.rewardDebt = user.amount.mul(pool.accDogXPerShare).div(1e12);
260 emit Deposit(msg.sender, _pid, _amount);
261 }
262
263 // Withdraw LP tokens from MasterChef.
264 function withdraw(uint256 _pid, uint256 _amount) public nonReentrant {
265 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/masterchef.sol

Locations

```
262
263 // Withdraw LP tokens from MasterChef.
264 function withdraw(uint256 _pid, uint256 _amount) public nonReentrant {
265 PoolInfo storage pool = poolInfo[_pid];
266 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/masterchef.sol

Locations

```
262
263 // Withdraw LP tokens from MasterChef.
264 function withdraw(uint256 _pid, uint256 _amount) public nonReentrant {
265 PoolInfo storage pool = poolInfo[_pid];
266 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/masterchef.sol

Locations

```
262
263 // Withdraw LP tokens from MasterChef.
264 function withdraw(uint256 _pid, uint256 _amount) public nonReentrant {
265 PoolInfo storage pool = poolInfo[_pid]; PoolInfo storage pool = poolInfo[_pid];
266 UserInfo storage user = userInfo[_pid][msg.sender];
267 require(user.amount >= _amount, "withdraw: not good");
```

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/masterchef.sol

Locations

```
146 totalAllocPoint = totalAllocPoint.add(_allocPoint);
147 poolExistence[_lpToken] = true;
148 poolInfo.push(
149 PoolInfo({
150 lpToken: _lpToken,
151 allocPoint: _allocPoint,
```

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/masterchef.sol

Locations

```
148 poolInfo.push(
149 PoolInfo({
150 lpToken: _lpToken,
151 allocPoint: _allocPoint,
152 lastRewardBlock: lastRewardBlock,
```

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/masterchef.sol

Locations

```
190 uint256 multiplier =
191 getMultiplier(pool.lastRewardBlock, block.number);
192 uint256 dogXReward =
193 multiplier.mul(dogXPerBlock).mul(pool.allocPoint).div(
194 totalAllocPoint
```

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/masterchef.sol

Locations

```
194 totalAllocPoint
195 );
196 accDogXPerShare = accDogXPerShare.add(
197 dogXReward.mul(1e12).div(lpSupply)
198 );
```

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/masterchef.sol

Locations

```
216 }
217 uint256 lpSupply = pool.lpToken.balanceOf(address(this));
218 if (lpSupply == 0 || pool.allocPoint == 0) {
219 pool.lastRewardBlock = block.number;
220 return;
```

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/masterchef.sol

Locations

```
222 uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
223 uint256 dogXReward =
224 multiplier.mul(dogXPerBlock).mul(pool.allocPoint).div(
225 totalAllocPoint
226 );
```

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/masterchef.sol

Locations

```
225 totalAllocPoint
226 );
227 dogX.mint(devAddr, dogXReward.div(10));
228 dogX.mint(address(this), dogXReward);
229 pool.accDogXPerShare = pool.accDogXPerShare.add(
```

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/masterchef.sol

Locations

```
235 // Deposit LP tokens to MasterChef for DogX allocation.
236 function deposit(uint256 _pid, uint256 _amount) public nonReentrant {
237 PoolInfo storage pool = poolInfo[_pid];
238 UserInfo storage user = userInfo[_pid][msg.sender];
239 updatePool(_pid);
```