MythX

Analysis `9e375678-3c9e-4e9d-a785-e4da1817eb82` MythX

Started	Wed May 26 2021 20:03:16 GMT+0000 (Coordinated Universal Time)
Finished	Wed May 26 2021 20:48:41 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Remythx
Main Source File	`MasterChef.Sol`

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	23	26

ISSUES

MEDIUM

SWC-000

Function could be marked as external.

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

`SafeMath.sol`

Locations

```
* Counterpart to Solidity's `/` operator. Note: this function uses a
* `revert` opcode (which leaves remaining gas untouched) while Solidity
* uses an invalid opcode to revert (consuming all remaining gas).
*
* Requirements:
*
* - The divisor cannot be zero.
*/
function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
require(b > 0, errorMessage);
uint256 c = a / b;
// assert(a == b * c + a % b); // There is no case in which this doesn't hold

return c;
}

/**
* @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
* Reverts when dividing by zero.
*
* Counterpart to Solidity's `%` operator. This function uses a `revert`
* opcode (which leaves remaining gas untouched) while Solidity uses an
* invalid opcode to revert (consuming all remaining gas).
*
```

MEDIUM

SWC-000

Function could be marked as external.

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

SafeMath.sol

Locations

```
* Counterpart to Solidity's `%` operator. This function uses a `revert`
* opcode (which leaves remaining gas untouched) while Solidity uses an
* invalid opcode to revert (consuming all remaining gas).
*
* Requirements:
*
* - The divisor cannot be zero.
*/
function mod(uint256 a, uint256 b) internal pure returns (uint256) {
return mod(a, b, "SafeMath: modulo by zero");
}

/**
* @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
* Reverts with custom message when dividing by zero.
*
* Counterpart to Solidity's `%` operator. This function uses a `revert`
* opcode (which leaves remaining gas untouched) while Solidity uses an
* invalid opcode to revert (consuming all remaining gas).
```

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

IBEP20.sol

Locations

```
* @dev Returns the token decimals.
*/
function decimals() external view returns (uint8);

/**
* @dev Returns the token symbol.
*/
function symbol() external view returns (string memory);

/**
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "renounceOwnership" 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

`Address.sol`

Locations

```
* https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
*
* IMPORTANT: because control is transferred to `recipient`, care must be
* taken to not create reentrancy vulnerabilities. Consider using
* {ReentrancyGuard} or the
* https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-pattern[checks-effects-interactions pattern].
*/
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "transferOwnership" 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

`Address.sol`

Locations

```
* taken to not create reentrancy vulnerabilities. Consider using
* {ReentrancyGuard} or the
* https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-pattern[checks-effects-interactions pattern].
*/
function sendValue(address payable recipient, uint256 amount) internal {
require(address(this).balance >= amount, "Address: insufficient balance");

// solhint-disable-next-line avoid-low-level-calls, avoid-call-value
(bool success, ) = recipient.call{ value: amount }("");
require(success, "Address: unable to send value, recipient may have reverted");
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "symbol" 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

`KangarooToken.sol`

Locations

```

/**
* @notice Delegate votes from `msg.sender` to `delegatee`
* @param delegatee The address to delegate votes to
*/
function delegate(address delegatee) external {
function delegate(address delegatee) external {
return _delegate(msg.sender, delegatee);
}
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "decimals" 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

KangarooToken.sol

Locations

```
function delegate(address delegatee) external {
return _delegate(msg.sender, delegatee);
}

/**
* @notice Delegates votes from signatory to `delegatee`
* @param delegatee The address to delegate votes to
* @param nonce The contract state required to match the signature
* @param expiry The time at which to expire the signature
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "totalSupply" 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

KangarooToken.sol

Locations

```
* @notice Delegates votes from signatory to `delegatee`
* @param delegatee The address to delegate votes to
* @param nonce The contract state required to match the signature
* @param expiry The time at which to expire the signature
* @param v The recovery byte of the signature
* @param r Half of the ECDSA signature pair
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "transfer" 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

KangarooToken.sol

Locations

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

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "allowance" 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

KangarooToken.sol

Locations

```
);

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

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "approve" 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

KangarooToken.sol

Locations

```
abi.encodePacked(
"\x19\x01",
domainSeparator,
structHash
)
);

address signatory = ecrecover(digest, v, r, s);
require(signatory != address(0), "Kangaroo::delegateBySig: invalid signature");
require(nonce == nonces[signatory]++, "Kangaroo::delegateBySig: invalid nonce");
require(now <= expiry, "Kangaroo::delegateBySig: signature expired");
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "transferFrom" 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

`KangarooToken.sol`

Locations

```
* @notice Gets the current votes balance for `account`
* @param account The address to get votes balance
* @return The number of current votes for `account`
*/
function getCurrentVotes(address account)
external
view
returns (uint256)
{
uint32 nCheckpoints = numCheckpoints[account];
return nCheckpoints > 0 ? checkpoints[account][nCheckpoints - 1].votes : 0;
}

/**
* @notice Determine the prior number of votes for an account as of a block number
* @dev Block number must be a finalized block or else this function will revert to prevent misinformation.
* @param account The address of the account to check
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "increaseAllowance" 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

`KangarooToken.sol`

Locations

```
* @return The number of votes the account had as of the given block
*/
function getPriorVotes(address account, uint blockNumber)
external
view
returns (uint256)
{
require(blockNumber < block.number, "Kangaroo::getPriorVotes: not yet determined");

uint32 nCheckpoints = numCheckpoints[account];
if (nCheckpoints == 0) {
return 0;
```

MEDIUM

SWC-000

Function could be marked as external.

The function definition of "decreaseAllowance" 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

`KangarooToken.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];
if (cp.fromBlock == blockNumber) {
return cp.votes;
} else if (cp.fromBlock < blockNumber) {
lower = center;
} else {
```

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

`KangarooToken.sol`

Locations

```
}
}
return checkpoints[account][lower].votes;
}

function _delegate(address delegator, address delegatee)
internal
{
address currentDelegate = _delegates[delegator];
uint256 delegatorBalance = balanceOf(delegator); // balance of underlying Kangaroos (not scaled);
_delegates[delegator] = delegatee;
```

MEDIUM

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`MasterChef.sol`

Locations

```
// Return reward multiplier over the given _from to _to block.
function getMultiplier(uint256 _from, uint256 _to) public view returns (uint256) {
return _to.sub(_from).mul(BONUS_MULTIPLIER);
}

// View function to see pending Kangaroos on frontend.
function pendingKangaroo(uint256 _pid, address _user) external view returns (uint256) {
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

`MasterChef.sol`

Locations

```
totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
poolInfo[_pid].allocPoint = _allocPoint;
poolInfo[_pid].depositFeeBP = _depositFeeBP;
}

```

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

`SafeMath.sol`

Locations

```
function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
require(b > 0, errorMessage);
uint256 c = a / b;
// assert(a == b * c + a % b); // There is no case in which this doesn't hold

```

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

SafeMath.sol

Locations

```
require(b > 0, errorMessage);
uint256 c = a / b;
// assert(a == b * c + a % b); // There is no case in which this doesn't hold

return c;
```

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

MasterChef.sol

Locations

```
// Return reward multiplier over the given _from to _to block.
function getMultiplier(uint256 _from, uint256 _to) public view returns (uint256) {
return _to.sub(_from).mul(BONUS_MULTIPLIER);
}

// View function to see pending Kangaroos on frontend.
function pendingKangaroo(uint256 _pid, address _user) external view returns (uint256) {
PoolInfo storage pool = poolInfo[_pid];
```