MythX

REPORT 60764B18AA487E0012F32B79

Created	Wed Apr 14 2021 01:53:28 GMT+0000 (Coordinated Universal Time)
Number of analyses	1
User	contact@balldefi.com

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
47357e6e-0d95-4a67-bda0-295ee7baf1c2	BallToken.sol	20

Analysis 47357e6e-0d95-4a67-bda0-295ee7baf1c2 MythX

Started	Wed Apr 14 2021 01:53:38 GMT+0000 (Coordinated Universal Time)
Finished	Wed Apr 14 2021 02:09:48 GMT+0000 (Coordinated Universal Time)
Mode	Standard
Client Tool	Remythx
Main Source File	BallToken.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	13	7

ISSUES

MEDIUM Function could be marked as external.

SWC-000

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
BallToken.sol
Locations

```
72  * thereby removing any functionality that is only available to the owner.
73  */
74  function renounceOwnership() public virtual onlyOwner {
75  emit OwnershipTransferred(_owner, address(0));
76  _owner = address(0);
77  }
78
79  /**
```

MEDIUM

Function could be marked as external.

SWC-000

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

BallToken.sol

Locations

```
81  * Can only be called by the current owner.
82  */
83  function transferOwnership(address newOwner) public virtual onlyOwner {
84  require(newOwner != address(0), "Ownable: new owner is the zero address");
85  emit OwnershipTransferred(_owner, newOwner);
86  _owner = newOwner;
87  }
88  }
89
```

MEDIUM

Function could be marked as external.

SWC-000

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

BallToken.sol

Locations

```
406 * name.
407 */
408 function symbol() public override view returns (string memory) {
409 return _symbol;
410 }
411
412 /**
```

MEDIUM

Function could be marked as external.

SWC-000

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

BallToken.sol

Locations

```
413 * @dev Returns the number of decimals used to get its user representation.
414 */
415 function decimals() public override view returns (uint8) {
416 return _decimals;
417 }
418
419 /**
```

MEDIUM

Function could be marked as external.

SWC-000

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

BallToken.sol

Locations

```
420 * @dev See {BEP20-totalSupply}.
421 */
422 function totalSupply() public override view returns (uint256) {
423 return _totalSupply;
424 }
425
426 /**
```

MEDIUM

Function could be marked as external.

SWC-000

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

BallToken.sol

Locations

```
439 * - the caller must have a balance of at least `amount`.
440 */
441 function transfer(address recipient, uint256 amount) public override returns (bool) {
442 _transfer(_msgSender(), recipient, amount);
443 return true;
444 }
445
446 /**
```

MEDIUM

Function could be marked as external.

SWC-000

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

BallToken.sol

Locations

```
447 * @dev See {BEP20-allowance}.
448 */
449 function allowance(address owner, address spender) public override view returns (uint256) {
450 return _allowances[owner][spender];
451 }
452
453 /**
```

MEDIUM

Function could be marked as external.

SWC-000

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

BallToken.sol

Locations

```
458 * - `spender` cannot be the zero address.
459 */
460 function approve(address spender, uint256 amount) public override returns (bool) {
461 _approve(_msgSender(), spender, amount);
462 return true;
463 }
464
465 /**
```

MEDIUM

Function could be marked as external.

SWC-000

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

BallToken.sol

Locations

```
475 * `amount`.
476 */
477 function transferFrom (address sender, address recipient, uint256 amount) public override returns (bool) {
478 _transfer(sender, recipient, amount);
479 _approve(
480 sender,
481 _msgSender(),
482 _allowances[sender][_msgSender()].sub(amount, 'BEP20: transfer amount exceeds allowance')
483 );
484 return true;
485 }
486
487 /**
```

MEDIUM

Function could be marked as external.

SWC-000

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

BallToken.sol

Locations

```
497 * - `spender` cannot be the zero address.
498 */
499 function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
500 _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
501 return true;
502 }
503
504 /**
```

MEDIUM

Function could be marked as external.

SWC-000

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

BallToken.sol

Locations

```
516 * `subtractedValue`.
517 */
518 function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
519 _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, 'BEP20: decreased allowance below zero'));
520 return true;
521 }
522
523 /**
```

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

BallToken.sol

Locations

```
529 * - `msg.sender` must be the token owner
530 */
531 function mint(uint256 amount) public onlyOwner returns (bool) {
532 _mint(_msgSender(), amount);
533 return true;
534 }
535
536 /**
```

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

BallToken.sol

Locations

```
634 contract BallToken is BEP20('Ball Token', 'BALL') {
635 /// @notice Creates `_amount` token to `_to`. Must only be called by the owner (MasterChef).
636 function mint(address _to, uint256 _amount) public onlyOwner {
637 _mint(_to, _amount);
638 _moveDelegates(address(0), _delegates[_to], _amount);
639 }
640
641 // Copied and modified from YAM code:
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is """>=0.6.0<0.8.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

BallToken.sol

Locations

```
1 // SPDX-License-Identifier: MIT
2
3 pragma solidity >=0.6.0 <0.8.0;
4
5 /*
```

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

BallToken.sol

Locations

```
743 require(signatory != address(0), "BALL::delegateBySig: invalid signature");
744 require(nonce == nonces[signatory]++, "BALL::delegateBySig: invalid nonce");
745 require(now <= expiry, "BALL::delegateBySig: signature expired");
746 return _delegate(signatory, delegatee);
747 }
```

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

BallToken.sol

Locations

```
773 returns (uint256)
774 {
775 require(blockNumber < block.number, "BALL::getPriorVotes: not yet determined");
776
777 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

BallToken.sol

Locations

```
846 internal
847 {
848 uint32 blockNumber = safe32(block.number, "BALL::_writeCheckpoint: block number exceeds 32 bits");
849
850 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

BallToken.sol

Locations

```
773 returns (uint256)
774 {
775 require(blockNumber < block.number, "BALL::getPriorVotes: not yet determined");
776
777 uint32 nCheckpoints = numCheckpoints[account];
```

LOW

SWC-128

Potentially unbounded data structure passed to builtin.

Gas consumption in function "delegateBySig" in contract "BallToken" 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

BallToken.sol

Locations

```
717 abi.encode(
718 DOMAIN_TYPEHASH,
719 keccak256(bytes(name())),
720 getChainId(),
721 address(this)
```

LOW

SWC-128

Loop over unbounded data structure.

Gas consumption in function "getPriorVotes" in contract "BallToken" 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

BallToken.sol

Locations

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