MythX

REPORT 5F45BDDB697F4700115046D8

Created	Wed Aug 26 2020 01:41:47 GMT+0000 (Coordinated Universal Time)
Number of analyses	3
User	jonahsparklemobile@gmail.com

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
c6989eef-bf85-446f-8dcb-5fa06a480cba	TimestampValidator.sol	5
6ef99165-7254-4202-8670-d9630f602fea	loyaltySettings.sol	9
0437ba42-0d9d-4583-a58e-53284c3c8f68	SparkleToken.sol	1

Analysis c6989eef-bf85-446f-8dcb-5fa06a480cba MythX

Started	Wed Aug 26 2020 01:41:53 GMT+0000 (Coordinated Universal Time)
Finished	Wed Aug 26 2020 02:27:13 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.19
Main Source File	TimestampValidator.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
2	0	3

ISSUES

HIGH

SWC-101

The arithmetic operator can overflow.

It is possible to cause an integer overflow or underflow in the arithmetic operation.

Source file

TimestampValidator.sol

Locations

```
43 uint256 currentTime = block.timestamp;
44 uint256 depositTime = block.timestamp;
45 uint256 rewardTime = block.timestamp + verifyTimeLegnth;
46 require (verifyTimeLegnth == 60, 'timestamps do not match');
47 require (currentTime == block.timestamp, 'timestamps do not match');
```

HIGH

SWC-101

The arithmetic operator can overflow.

It is possible to cause an integer overflow or underflow in the arithmetic operation.

Source file

TimestampValidator.sol

Locations

```
70 require (POT._contractCheck == contractAddress, 'contract address do not match');
71 require (POT._depositTimeCheck == POT._depositTimeCheck, 'timestamps do not match');
72 require (POT._rewardTimeCheck == POT._depositTimeCheck + verifyTimeLegnth, 'time legnths do not match');
73 require (_miner == POT._minerCheck, 'time legnths do not match');
74 require (POT._contractCheck == contractCheck, 'contract address do not match');
```

LOW

SWC-102

An outdated compiler version is used.

The compiler version specified in the pragma directive may have known bugs. It is recommended to use the latest minor release of solc 0.5 or 0.6. For more information on Solidity compiler bug reports and fixes refer to https://github.com/ethereum/solidity/releases.

Source file

TimestampValidator.sol

Locations

```
1  pragma solidity 0.4.25;
2
3  import './ReentrancyGuard.sol';
```

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

TimestampValidator.sol

Locations

```
74  require (POT._contractCheck == contractCheck, 'contract address do not match');
75  require (contractAddress == address(msg.sender), 'contract address do not match');
76  require(block.timestamp > POT._rewardTimeCheck,'Users reward has not yet been approved');
77  if (block.timestamp > POT._rewardTimeCheck){
78  return POT._timstampPassed = true;
```

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

TimestampValidator.sol

Locations

```
75  require (contractAddress == address(msg.sender), 'contract address do not match');
76  require(block.timestamp > POT._rewardTimeCheck,'Users reward has not yet been approved');
77  if (block.timestamp > POT._rewardTimeCheck){
78  return POT._timstampPassed = true;
79  } else {
80  revert('Unexspected error');
81  return false;
82  }
83  }
```

Analysis 6ef99165-7254-4202-8670-d9630f602fea MythX

Started	Wed Aug 26 2020 01:41:53 GMT+0000 (Coordinated Universal Time)
Finished	Wed Aug 26 2020 02:27:05 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.19
Main Source File	LoyaltySettings.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
2	0	7

ISSUES

HIGH The arithmetic operator can overflow.

It is possible to cause an integer overflow or underflow in the arithmetic operation.

SWC-101

Source file

loyaltySettings.sol

Locations

```
250 POL._loyaltyNeeded = true;
251 POL._rewardApproved = false;
252 POL._value = POL._value + _amount;
253 POL._rewardAmount = 0;
254 POL._loyaltyDays = 0;
```

HIGH The arithmetic operator can overflow.

It is possible to cause an integer overflow or underflow in the arithmetic operation.

SWC-101

Source file

loyaltySettings.sol

Locations

```
206 require (msg.sender == POL._miner,'miner address does not match sender address');
207 if (msg.sender == POL._miner) {
208 currentMiners -= 1;
209 delete loyaltyTimestamp[msg.sender];
210 storageDump storage SD = timestampRemoved[msg.sender];
```

LOW

SWC-102

An outdated compiler version is used.

The compiler version specified in the pragma directive may have known bugs. It is recommended to use the latest minor release of solc 0.5 or 0.6. For more information on Solidity compiler bug reports and fixes refer to https://github.com/ethereum/solidity/releases.

Source file

loyaltySettings.sol

Locations

```
1 pragma solidity 0.4.25;
2
```

LOW

SWC-107

Read of persistent state following external call

The contract account state is accessed after an external call to a fixed address. 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

loyaltySettings.sol

Locations

```
214 require (SD._timestampRemoved == true,'timestamp must be removed before tokens can be withdrawn');
215 time.removeTimestamp(_recipiant);
216 token.transferFrom(address(loyaltyfaucet),address(_recipiant),_reward);
217 token.transfer(address(_recipiant),_amount);
218 } else{
```

LOW

SWC-107

Write to persistent state following external call

The contract account state is accessed after an external call to a fixed address. 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

loyaltySettings.sol

Locations

```
215 time.removeTimestamp(_recipiant);
216 token.transferFrom(address(loyaltyfaucet),address(_recipiant),_reward);
217 token.transfer(address(_recipiant),_amount);
218 } else{
219 return false;
```

LOW

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

loyaltySettings.sol

Locations

```
257 POL._rewardTime = POL._depositTime.add(_timeLegnth);
258 time.setTimestamp(msg.sender);
259 token.transferFrom(address(msg.sender), address(this), _value);
260 } else {
261 revert ('Unexspected error ');
```

LOW

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

loyaltySettings.sol

Locations

```
214 require (SD._timestampRemoved == true,'timestamp must be removed before tokens can be withdrawn');
215 time.removeTimestamp(_recipiant);
216 token.transferFrom(address(loyaltyfaucet),address(_recipiant),_reward);
217 token.transfer(address(_recipiant),_amount);
218 } else{
```

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

loyaltySettings.sol

Locations

```
112 require(time == TimestampValidator (0x6006fea8d63f329ffb265c0907699457f9a1f52C),'Please use the correct validator address');
113 require(POL._loyaltyNeeded == true, 'User is not a loyalty holder');
114 require(block.timestamp > POL._rewardTime,'Users reward has not yet been approved');
115 require(msg.sender == POL._miner,'Users has not deposited tokens');
116 if (block.timestamp > POL._rewardTime) {
```

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

loyaltySettings.sol

Locations

```
114 require(block.timestamp > POL._rewardTime,'Users reward has not yet been approved');
115 require(msg.sender == POL._miner,'Users has not deposited tokens');
116 if (block.timestamp > POL._rewardTime) {
117 POL._rewardApproved = true;
118 dailyCounter();
119 time.checkTimestamp(POL._miner);
120 }
121 if (block.timestamp < POL._rewardTime) {
122 revert("Loyalty age not accepted");
```

Analysis 0437ba42-0d9d-4583-a58e-53284c3c8f68 MythX

Started	Wed Aug 26 2020 01:42:04 GMT+0000 (Coordinated Universal Time)
Finished	Wed Aug 26 2020 02:27:15 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.19
Main Source File	SparkleToken.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

LOW

SWC-102

An outdated compiler version is used.

The compiler version specified in the pragma directive may have known bugs. It is recommended to use the latest minor release of solc 0.5 or 0.6. For more information on Solidity compiler bug reports and fixes refer to https://github.com/ethereum/solidity/releases.

Source file

SparkleToken.sol

Locations

```
3 */
4
5 pragma solidity 0.4.25;
6
7 // File: openzeppelin-solidity/contracts/token/ERC20/IERC20.sol
```