

REPORT 5F45534573DC670011A7107E

Created Tue Aug 25 2020 18:07:01 GMT+0000 (Coordinated Universal Time)

Number of analyses 3

User jonahsparklemobile@gmail.com

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
3eaeafe6-f238-41dc-b72b-7a7ce84e3747	contracts/SparkleTimestamp.sol	1
50395820-0661-4dd8-b1f6-fa7336feb028	contracts/SparkleLoyalty.sol	36
7aaa564a-b0a3-47ea-8635-d616ebfa9bef	contracts/SparkleRewardTiers.sol	1

Started Tue Aug 25 2020 18:07:08 GMT+0000 (Coordinated Universal Time)

Finished Tue Aug 25 2020 18:22:20 GMT+0000 (Coordinated Universal Time)

Mode Standard

Client Tool Mythx-Cli-0.6.19

Main Source File Contracts/SparkleTimestamp.Sol

DETECTED VULNERABILITIES

(HIGH	(MEDIUM	(LOW
•	0	4
0	0	1

ISSUES

LOW An outdated compiler version is used.

SWC-102

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

contracts/SparkleTimestamp.sol

Locations

```
1 /// SWC-103: Floating Pragma
```

pragma solidity 0.4.25;

import "../node_modules/openzeppelin-solidity/contracts/math/SafeMath.sol";

Started Tue Aug 25 2020 18:07:08 GMT+0000 (Coordinated Universal Time)

Finished Tue Aug 25 2020 18:22:35 GMT+0000 (Coordinated Universal Time)

Mode Standard

Client Tool Mythx-Cli-0.6.19

Main Source File Contracts/SparkleLoyalty.Sol

DETECTED VULNERABILITIES

(HIGH	(MEDIUM	(LOW
0	19	17

ISSUES

MEDIUM Write to persistent state following external call

SWC-107

The contract account state is accessed after an external call to a user defined 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

contracts/SparkleLoyalty.sol

Locations

```
require(ISparkleTimestamp(timestampAddress).hasTimestamp(_rewardAddress), 'No timstamp');

// Set the specified address' locked status

accounts _rewardAddress _isLocked = __value;

// Emit event log to the block chain for future web3 use

emit LockedAccountEvent(_rewardAddress, __value);
```

MEDIUM Read of persistent state following external call

SWC-107

The contract account state is accessed after an external call to a user defined 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

contracts/SparkleLoyalty.sol

```
// Obtain values needed from account record before zeroing
uint256 joinCount = accounts msg sender __joined;
uint256 collected = accounts[msg.sender]._collected;
uint256 deposit = accounts[msg.sender]._balance;
```

Read of persistent state following external call

SWC-107

The contract account state is accessed after an external call to a user defined 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

contracts/SparkleLoyalty.sol

Locations

```
250 | uint256 joinCount = accounts[msg.sender]._joined
     uint256 collected = accounts[msg.sender]._collected;
     uint256 deposit = accounts[msg_sender]__balance
252
     // Zero out the callers account record
253
     delete accounts[msg.sender];
```

MEDIUM Write to persistent state following external call

SWC-107

The contract account state is accessed after an external call to a user defined 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

contracts/SparkleLoyalty.sol

Locations

```
252 | uint256 deposit = accounts[msg.sender]._balance;
     // Zero out the callers account record
     delete accounts[msg.sender];
254
     // Carry callers program joined count over to cleared record
     accounts[msg.sender]._joined = joinCount;
```

MEDIUM Write to persistent state following external call

SWC-107

The contract account state is accessed after an external call to a user defined 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

contracts/SparkleLoyalty.sol

```
256 | accounts[msg.sender]._joined = joinCount;
     // Decement the totak number of active accounts
257
     totalActiveAccounts -= 1;
259
     // Delete the callers timestamp record
```

Read of persistent state following external call

SWC-107

The contract account state is accessed after an external call to a user defined 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

contracts/SparkleLoyalty.sol

Locations

```
// Determine if transfer from treasury address is a success

if(!IERC20(tokenAddress).transferFrom(treasuryAddress, msg.sender, collected)) {

// No, revert indicating that the transfer and wisthdraw has failed

revert('Withdraw failed');
```

MEDIUM

Read of persistent state following external call

SWC-107

The contract account state is accessed after an external call to a user defined 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

contracts/SparkleLoyalty.sol

Locations

```
262
263 // Determine if transfer from treasury address is a success
264 if(!IERC20(tokenAddress).transferFrom(treasuryAddress, msg.sender, collected)) {
265 // No, revert indicating that the transfer and wisthdraw has failed
266 revert('Withdraw failed');
```

MEDIUM

Read of persistent state following external call

SWC-107

The contract account state is accessed after an external call to a user defined 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

contracts/SparkleLoyalty.sol

```
// Determine if transfer from contract address is a sucess
if(!IERC20(tokenAddress).transfer(msg.sender, deposit)) {
// No, revert indicating that the treansfer and withdraw has failed
revert('Withdraw failed');
```

Read of persistent state following external call

SWC-107

The contract account state is accessed after an external call to a user defined 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

contracts/SparkleLoyalty.sol

Locations

```
138 | if(maxAllowed > 0) {
     \ensuremath{//} Yes, determine if the deposit amount + current balance exceed max deposit cap
139
     if(loyaltyAccount._balance.add(_depositAmount) > maxAllowed || _depositAmount > maxAllowed) {
     // Yes, revert informing that the maximum deposit cap has been exceeded
141
     revert('Exceeds cap');
```

MEDIUM

Read of persistent state following external call

SWC-107

The contract account state is accessed after an external call to a user defined 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

contracts/SparkleLoyalty.sol

Locations

```
138 | if(maxAllowed > 0) {
                                              // Yes, determine if the deposit amount + current balance exceed max deposit cap
                                           if(loyaltyAccount.\_balance.add(\_depositAmount) > maxAllowed \mid \mid \_depositAmount > \frac{maxAllowed}{maxAllowed}) \mid \{ (a,b,c) \mid (a
                                           // Yes, revert informing that the maximum deposit cap has been exceeded
                                           revert('Exceeds cap');
```

MEDIUM Read of persistent state following external call

SWC-107

The contract account state is accessed after an external call to a user defined 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

contracts/SparkleLoyalty.sol

```
146
  // Determine if the tier selected is enabled
  // No, then this tier cannot be selected
149
  revert('Invalid tier');
```

Write to persistent state following external call

SWC-107

The contract account state is accessed after an external call to a user defined 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

contracts/SparkleLoyalty.sol

Locations

```
146
   \ensuremath{//} Determine if the tier selected is enabled
147
   // No, then this tier cannot be selected
149
   revert('Invalid tier');
```

MEDIUM

Read of persistent state following external call

SWC-107

The contract account state is accessed after an external call to a user defined 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

contracts/SparkleLoyalty.sol

Locations

```
152
    // Determine of transfer from caller has succeeded
    if(IERC20(tokenAddress).transferFrom(msg.sender, this, _depositAmount)) {
154
    // Yes, thend determine if the specified address has a timestamp record
    if(ISparkleTimestamp(timestampAddress).hasTimestamp(msg.sender)) {
```

MEDIUM Write to persistent state following external call

SWC-107

The contract account state is accessed after an external call to a user defined 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

contracts/SparkleLoyalty.sol

```
152
     // Determine of transfer from caller has succeeded
153
     if(IERC20(tokenAddress).transferFrom(msg sender, this, _depositAmount)) {
     // Yes, thend determine if the specified address has a timestamp record
155
     \textbf{if}(\textbf{ISparkleTimestamp}(\textbf{timestampAddress}).\textbf{hasTimestamp}(\textbf{msg.sender})) \ \{
```

Read of persistent state following external call

SWC-107

The contract account state is accessed after an external call to a user defined 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

contracts/SparkleLoyalty.sol

Locations

```
154 | if(IERC20(tokenAddress).transferFrom(msg.sender, this, _depositAmount)) {
     \ensuremath{//} Yes, thend determine if the specified address has a timestamp record
155
    if(ISparkleTimestamp(timestampAddress).hasTimestamp(msg.sender)) {
    // Yes, update callers account balance by deposit amount
157
     loyaltyAccount._balance = loyaltyAccount._balance.add(_depositAmount);
```

MEDIUM Write to persistent state following external call

SWC-107

The contract account state is accessed after an external call to a user defined 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

contracts/SparkleLoyalty.sol

Locations

```
 154 \ \ | \ \ if(IERC20(tokenAddress).transferFrom(msg.sender, \ this, \ \_depositAmount)) \ \{
     // Yes, thend determine if the specified address has a timestamp record
    if(ISparkleTimestamp(timestampAddress).hasTimestamp(msg.sender)) {
156
    // Yes, update callers account balance by deposit amount
    loyaltyAccount._balance = loyaltyAccount._balance.add(_depositAmount);
```

MEDIUM Read of persistent state following external call

SWC-107

The contract account state is accessed after an external call to a user defined 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

contracts/SparkleLoyalty.sol

```
156 | if(ISparkleTimestamp(timestampAddress).hasTimestamp(msg.sender)) {
    // Yes, update callers account balance by deposit amount
    loyaltyAccount._balance = loyaltyAccount _balance.add(_depositAmount);
    // Reset the callers reward timestamp
159
    _resetTimestamp(msg.sender);
```

Write to persistent state following external call

SWC-107

The contract account state is accessed after an external call to a user defined 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

contracts/SparkleLoyalty.sol

Locations

```
156
    if(ISparkleTimestamp(timestampAddress).hasTimestamp(msg.sender)) {
    // Yes, update callers account balance by deposit amount
157
    loyaltyAccount._balance = loyaltyAccount._balance.add(_depositAmount)
158
    // Reset the callers reward timestamp
159
     _resetTimestamp(msg.sender);
```

MEDIUM SWC-107

Read of persistent state following external call

The contract account state is accessed after an external call to a user defined 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

contracts/SparkleLoyalty.sol

Locations

```
616 | require(_rewardAddress != address(0), "Invalid {reward}");
     // Reset callers timestamp for specified address
     ISparkleTimestamp(timestampAddress).resetTimestamp(_rewardAddress);
618
619
```

LOW

An outdated compiler version is used.

SWC-102

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

contracts/SparkleLoyalty.sol

Locations

```
1 /// SWC-103: Floating Pragma
   pragma solidity 0.4.25;
   import "../node_modules/openzeppelin-solidity/contracts/math/SafeMath.sol";
```

LOW

A call to a user-supplied address is executed.

SWC-107

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

Source file

contracts/SparkleLoyalty.sol

```
358
       function getTimeRemaining(address _loyaltyAddress) public view whenNotPaused returns (uint256, bool, uint256) {
      return \  \  {\color{blue} \textbf{ISparkleTimestamp(timestampAddress)}}, get \\ \\ {\color{blue} \textbf{TimeRemaining(\_loyaltyAddress)}}; \\ \\
360
361
```

LOW A call to a user-supplied address is executed.

SWC-107

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

Source file

contracts/SparkleLoyalty.sol

Locations

```
require(_rewardAddress != address(0x0), 'Invalid {reward}');

// Validate specified address has timestamp

require(_ISparkleTimestamp timestampAddress, hasTimestamp(_rewardAddress, 'No timstamp');

// Set the specified address' locked status

accounts[_rewardAddress]._isLocked = _value;
```

LOW A call to a user-supplied address is executed.

SWC-107

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

Source file

contracts/SparkleLoyalty.sol

Locations

```
require(_toAddress != address(0), "Invalid {to}");

// Validate there are tokens to withdraw

require(_IERC20 tokenAddress) balanceOf(this) > 0, "No tokens");

// Validate the transfer of tokens completed successfully

IERC20(tokenAddress).transfer(_toAddress, IERC20(tokenAddress).balanceOf(this));
```

LOW A call to a user-supplied address is executed.

SWC-107

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

Source file

contracts/SparkleLoyalty.sol

Locations

```
require(IERC20(tokenAddress).balanceOf(this) > 0, "No tokens");

// Validate the transfer of tokens completed successfully

IERC20(tokenAddress).transfer(_toAddress, __terc20_tokenAddress__balanceOf(this_));

// Only the complete of tokens completed successfully

// IERC20(tokenAddress).transfer(_toAddress, __terc20_tokenAddress__balanceOf(this_));

// Only the complete of tokens completed successfully

// Only the complete of tokens c
```

LOW A call to a user-supplied address is executed.

SWC-107

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

Source file

contracts/SparkleLoyalty.sol

```
require(IERC20(tokenAddress).balanceOf(this) > 0, "No tokens");

// Validate the transfer of tokens completed successfully

IERC20(tokenAddress).transfer(_toAddress_leaceOf(this));

610 }
```

LOW

A call to a user-supplied address is executed.

SWC-107

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

Source file

contracts/SparkleLoyalty.sol

Locations

```
require(msg.sender != address(0), 'Invalid {from}');

// validate that caller has a loyalty timestamp
require([SparkleTimestamp:timestampAddress: hasTimestamp:msg.sender], 'No timestamp');

// Determine if the account has been locked
```

LOW

A call to a user-supplied address is executed.

SWC-107

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

Source file

contracts/SparkleLoyalty.sol

Locations

```
require(_rewardAddress != address(0), "Invalid {reward}");

// Delete callers timestamp for specified address
require(_ISparkleTimestampAddress)_deleteTimestamp(_rewardAddress)_, 'Delete failed');

// Delete callers timestampAddress)_deleteTimestamp(_rewardAddress)_, 'Delete failed');
```

LOW

A call to a user-supplied address is executed.

SWC-107

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

Source file

contracts/SparkleLoyalty.sol

LOW A call to a user-supplied address is executed.

SWC-107

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

Source file

contracts/SparkleLoyalty.sol

Locations

```
262
263 // Determine if transfer from treasury address is a success
264 if(! IFRC20LtokenAddress transferFrom treasuryAddress msg sender collected.) {
265 // No, revert indicating that the transfer and wisthdraw has failed
266 revert('Withdraw failed');
```

LOW A call to a user-supplied address is executed.

SWC-107

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

Source file

contracts/SparkleLoyalty.sol

Locations

```
// Determine if transfer from contract address is a sucess
if(!IERCZ0:tokenAddress transfer msg sender deposit) {
// No, revert indicating that the treansfer and withdraw has failed
revert('Withdraw failed');
```

LOW A call to a user-supplied address is executed.

SWC-107

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

Source file

contracts/SparkleLoyalty.sol

Locations

```
require(_rewardAddress != address(0), "Invalid {reward}");

// Reset callers timestamp for specified address

ISparkleTimestamp(timestampAddress) resetTimestamp(_rewardAddress);

3
```

LOW Multiple calls are executed in the same transaction.

SWC-113

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

contracts/SparkleLoyalty.sol

LOW

Multiple calls are executed in the same transaction.

SWC-113

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

contracts/SparkleLoyalty.sol

Locations

```
require(_rewardAddress != address(0), "Invalid {reward}");

// Delete callers timestamp for specified address

require(_ISparkleTimestamp(timestampAddress) deleteTimestamp(_rewardAddress), 'Delete failed');

}
```

LOW Multiple calls are executed in the same transaction.

SWC-113

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

contracts/SparkleLoyalty.sol

Locations

```
146
147 // Determine if the tier selected is enabled
148 if(! SparkleRewardTiers tiersAddress _getEnabled(loyaltyAccount _tier)) {
149 // No, then this tier cannot be selected
150 revert('Invalid tier');
```

LOW 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).

SWC-123

Source file

contracts/SparkleLoyalty.sol

LOW 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).

SWC-123

Source file

contracts/SparkleLoyalty.sol

Locations

```
require(msg.sender != address(0), 'Invalid {from}');

// Validate caller has a timestamp and it has matured

require(ISparkleTimestamptimestampAddress) hasTimestamp msg.sender), 'No record');

require(ISparkleTimestamp(timestampAddress).isRewardReady(msg.sender), 'Not mature');
```

Source file

contracts/SparkleLoyalty.sol

```
14 * @author SparkleMobile Inc.
15
    contract SparkleLoyalty is Ownable, Pausable, ReentrancyGuard {
16
17
18
     * @dev Ensure math safety through SafeMath
19
20
    using SafeMath for uint256;
21
22
    uint256 private gasToSendWithTX = 21317;
23
        uint256 private baseRate = 0.00082137 * 10e7; // A full year is 365.2422 gregorian days
     uint256 private baseRate = 0.00013690 * 10e7; // A full year is 365.2422 gregorian days (5%)
25
27
    struct Account {
     address _address; // Address of loyalty earner
28
     uint256 _balance; // Balance of tokens physically deposited
29
    uint256 _collected; // Collected value of token reward
30
    uint256 _claimed: // Total number of times a reward has been claimed
uint256 _joined: // Total number of times this address has joined the program
uint256 _tier: // Tier index of reward tier for this loyaly earner
32
33
    bool _isLocked: // This is the locked record status. (true = no deposits, withdraws, claims)
34
35
36
     * @param tokenAddress of erc20 token used for rewards
38
39
     address private tokenAddress;
41
     * @param timestampAddress of erc20 token used for rewards
43
     address private timestampAddress;
45
47
     * @param treasuryAddress of token reeasury used for earned rewards
48
49
     address private treasuryAddress;
50
52
     * @param collectionAddress of ethereum account used for tier upgrade collection
53
54
     address private collectionAddress;
57
          aram rewardTiersAddress of smart contractused for tier resolution
```

```
address private tiersAddress;
61
62
63
      * @param minProofRequired to deposit for rewards eligibility at any tier
64
      uint256 private minRequired
65
66
68
        <code>Oparam maxProofAllowed allowed for deposit for rewards eligibility at any tier</code>
70
      uint256 private maxAllowed;
71
72
 73
        <sup>*</sup> @param totalTokensClaimed of all rewards awarded
74
 75
      uint256 private totalTokensClaimed;
77
      * @param totalTimesClaimed
79
80
      uint256 private totalTimesClaimed
81
82
      * @param totalActiveAccounts count
83
84
85
      uint256 private totalActiveAccounts;
86
87
      * @param Accounts mapping of user loyalty records
88
89
90
     mapping(address => Account) private accounts;
91
92
      * @dev Sparkle Loyalty Rewards Program contract .cTor
93
     \mbox{\ensuremath{^{\star}}} <code>Bparam</code> _tokenAddress of token used for proof of loyalty rewards
95
     * Oparam _collectionAddress of ethereum account to collect tier upgrade eth

* Oparam _tiersAddress of the proof of loyalty tier rewards support contract

* Oparam _timestampAddress of the proof of loyalty timestamp support contract
97
98
99
      constructor/address _tokenAddress, address _treasuryAddress, address _collectionAddress, address _tiersAddress address _timestampAddress) public Ownable() Pausable()
100
102
103
     // Initialize contract internal addresse(s)
     tokenAddress = _tokenAddress;
104
      treasuryAddress = _treasuryAddress;
     collectionAddress = _collectionAddress;
106
107
      tiersAddress = _tiersAddress;
108
     timestampAddress = _timestampAddress;
109
      // Initialize minimum/maximum allowed deposit limits
110
     minRequired = uint256(1000).mul(10e7);
112
     maxAllowed = uint256(250000).mul(10e7);
113
114
115
      event DepositLoyaltyEvent(address, uint256, bool);
116
117
     • Odev Deposit additional tokens to a reward address loyalty balance
• Oparam _depositAmount of tokens to deposit into a reward address balance
• Oreturn bool indicating the success of the deposit operation (true == success)
118
120
121
          nction depositLoyalty(uint _depositAmount)    public whenNotPaused nonReentrant returns (bool)
```

```
// Validate calling address (msg.sender)
124
      require(msg sender != address(0), 'Invalid {from}1');
// Validate specified value meets minimum requirements
125
126
      require(_depositAmount >= minRequired, 'Minimum required');
     // Determine if caller has approved enough allowance for this deposit

if(IERC20(tokenAddress).allowance(msg_sender_this) < __depositAmount) |

// No, rever informing that deposit amount exceeded allowance amount
129
130
131
132
      revert('Exceeds allowance');
133
134
135
      // Obtain a storage instsance of callers account record
136
      Account storage loyaltyAccount = accounts[msg sender];
137
138
      // Determine if there is an upper deposit cap
139
     if(maxAllowed > 0) {
      // Yes, determine if the deposit amount + current balance exceed max deposit cap
140
141
      if(loyaltyAccount._balance add(_depositAmount) > maxAllowed || _depositAmount > maxAllowed) {
142
      // Yes, revert informing that the maximum deposit cap has been exceeded
143
      revert('Exceeds cap');
144
145
146
147
148
     // Determine if the tier selected is enabled
     if([![SparkleRewardTiers(tiersAddress).getEnabled(loyaltyAccount_tier)).
149
150
      // No, then this tier cannot be selected
151
      revert('Invalid tier');
152
153
154
      // Determine of transfer from caller has succeeded
      if(IERC20(tokenAddress), transferFrom(msg sender, this _depositAmount)) {
// Yes, thend determine if the specified address has a timestamp record
156
     if(ISparkleTimestamp(timestampAddress).hasTimestamp(msg_sender)) |
// Yes, update callers account balance by deposit amount
157
158
159
     loyaltyAccount._balance = loyaltyAccount._balance add(_depositAmount);
160
      // Reset the callers reward timestamp
161
      _resetTimestamp(msg.sender);
162
      emit DepositLoyaltyEvent(msg.sender, _depositAmount, true);
163
164
165
      return true;
166
167
      // Determine if a timestamp has been added for caller
168
     if(!ISparkleTimestamp timestampAddress addTimestamp(msg sender)) {
// No, revert indicating there was some kind of error
169
170
171
      revert('No timestamp created');
173
174
      // Prepare loyalty account record
175
     loyaltyAccount _address = msg sender;
176
     loyaltyAccount _balance = _depositAmount:
177
      loyaltyAccount._joined = 1;
178
      // Update global account counter
179
      totalActiveAccounts += 1;
180
     emit DepositLoyaltyEvent(msg_sender; _depositAmount, false);
// Return success
181
182
183
      return true;
184
```

```
return false;
188
189
190
      * @dev Claim Sparkle Loyalty reward
192
193
      function claimLoyaltyReward() public whenNotPaused nonReentrant returns(bool)
194
195
      // Validate calling address (msg.sender)
196
      require(msg sender != address(0), 'Invalid {from}');
     // Validate caller has a timestamp and it has matured
require(ISparkleTimestamp(timestampAddress), hasTimestamp(msg.sender), 'No record');
require(ISparkleTimestamp(timestampAddress), isRewardReady(msg.sender), 'Not mature');
197
198
199
200
201
      // Obtain the current state of the callers timestamp
202
      (uint256 timeRemaining bool isReady, uint256 rewardDate) = ISparkleTimestamp(timestampAddress).getTimeRemaining(msg.sender);
203
      if(isReady) {
204
      // Value not used but throw unused var warning (cleanup)
206
      rewardDate = 0;
207
      // Yes, then obtain a storage instance of callers account record
208
      Account storage loyaltyAccount = accounts[msg.sender];
      // Obtain values required for caculations
uint256 dayCount = (timeRemaining div(TSparkleTimestamp(timestampAddress) getTimePeriod())) add(1);
209
210
211
      uint256 tokenBalance = loyaltyAccount _balance add(loyaltyAccount _collected);
212
      uint256 rewardRate = ISparkleRewardTiers(tiersAddress).getRate(loyaltyAccount._tier);
      uint256 rewardTotal = baseRate mul(tokenBalance mul(rewardRate).mul(dayCount).div(10e7).div(10e7);
214
215
      loyaltyAccount._collected = loyaltyAccount._collected.add(rewardTotal);
216
      // Increment total number of times a reward has been claimed
217
      loyaltyAccount__claimed = loyaltyAccount__claimed_add(
218
       // Incrememtn total number of times rewards have been collected by all
219
      totalTimesClaimed = totalTimesClaimed.add(1);
220
      // Increment total number of tokens claimed
221
      totalTokensClaimed += rewardTotal;
     // Reset the callers timestamp record
_resetTimestamp(msg sender);
// Emit event log to the block chain for future web3 use
223
224
      emit RewardClaimedEvent(msg sender, rewardTotal);
// Return success
225
226
      return true;
228
229
230
      // Revert opposed to returning boolean (May or may not return a txreceipt)
231
      revert('Failed claim');
232
234
235
        Odev Withdraw the current deposit balance + any earned loyalty rewards
236
237
      function withdrawLoyalty() public whenNotPaused nonReentrant()
238
239
         Validate calling address (msg.sender)
      require(msg sender |= address(0), 'Invalid {from}');
240
241
242
      \frac{\text{require}(\text{ISparkleTimestamp}(\textbf{timestampAddress}).}{\text{hasTimestamp}(\textbf{msg.sender})}, \text{ 'No timestamp'});}
243
244
245
      if(accounts[msg.sender]._isLocked) {
246
      // Yes, revert informing that this loyalty account has been locked
247
     revert('Locked');
```

```
250
              // Obtain values needed from account record before zeroing
251
              uint256 joinCount = accounts[msg sender]._joined
252
              uint256 collected = accounts[msg.sender]._collected;
253
              uint256 deposit = accounts[msg.sender]._balance
254
               // Zero out the callers account record
255
              delete accounts[msg.sender];
256
              // Carry callers program joined count over to cleared record
257
              accounts[msg_sender]._joined = joinCount
258
                                 ment the totak number of active accounts
259
              totalActiveAccounts -= 1;
260
261
              // Delete the callers timestamp record
262
              _deleteTimestamp(msg.sender);
263
264
              // Determine if transfer from treasury address is a success
            \label{lem:if_lemon} \textbf{if}(!\texttt{IERC20}(\textbf{tokenAddress}), \texttt{transferFrom}(\textbf{treasuryAddress}, \textbf{msg}, \textbf{sender}, \textbf{collected})) = \texttt{lemonts}(\texttt{lemonts}) + \texttt{lemonts}(\texttt{lem
265
266
267
             revert('Withdraw failed');
268
269
270
             // Determine if transfer from contract address is a sucess
271
              \underline{if(!}\underline{IERC20(tokenAddress).transfer(msg.sender,\ deposit
               // No, revert indicating that the treansfer and withdraw has failed
273
              revert('Withdraw failed');
274
275
276
              // Emit event log to the block chain for future web3 use
277
              emit LoyaltyWithdrawnEvent(msg.sender, deposit.add(collected));
278
279
280
              * @dev Gets the locked status of the specified address
* @param _loyaltyAddress of account
281
283
             * @return (bool) indicating locked status
284
285
              function isLocked(address _loyaltyAddress) public view whenNotPaused returns (bool) [
286
             return accounts[_loyaltyAddress]._isLocked;
287
288
289
              function lockAccount(address _rewardAddress, bool _value)    public onlyOwner whenNotPaused nonReentrant {
             // Validate calling address (msg.sender)
require(msg sender [!= address(0x0), 'Invalid {from}'))
290
291
292
             require(_rewardAddress != address(0x0), 'Invalid {reward}');
293
294
             \begin{tabular}{ll} \textbf{require} (ISparkleTimestamp(\textbf{timestampAddress}), has Timestamp(\_\textbf{rewardAddress}), 'No timestamp'); \\ \end{tabular}
295
              // Set the specified address' locked status
296
              accounts[_rewardAddress]._isLocked = _value;
             // Emit event log to the block chain for future web3 use
emit LockedAccountEvent(_rewardAddress _value)
297
298
299
300
301
302
             * @dev Gets the storage address value of the specified address
303
304
              * @return (address) indicating the address stored calls acc
305
306
             function getLoyaltyAddress(address _loyaltyAddress) public view whenNotPaused returns(address) {
307
             return accounts[_loyaltyAddress]._address;
308
309
310
                        lev Get the deposit balance value of specified address
```

```
Oparam loyaltyAddress of accountOperation (uint256) indicating the balance value
314
315
      function getDepositBalance(address _loyaltyAddress) public view whenNotPaused returns(uint256) {
316
      return accounts[_loyaltyAddress]._balance;
318
320
321
      * @return (uint256) indicating the tokens collected
323
324
      function getTokensCollected(address _loyaltyAddress) public view whenNotPaused returns(uint256) {
325
      return accounts[_loyaltyAddress]._collected;
326
327
328
329
        @dev Get the total balance (deposit + collected) of tokens
330
331
      * @return (uint256) indicating total balance
332
333
      function getTotalBalance(address _loyaltyAddress) public view whenNotPaused returns(uint256) {
334
      return accounts[_loyaltyAddress]._balance.add(accounts[_loyaltyAddress]._collected);
335
336
338
      * @dev Get the times loyalty has been claimed
339
      * @param _loyaltyAddress of account
340
      * @return (uint256) indicating total time claimed
341
342
      function getTimesClaimed(address _loyaltyAddress) public view whenNotPaused returns(uint256) {
343
      return accounts[_loyaltyAddress]._claimed;
344
345
346
347
      * @dev Get total number of times joined
348
      * @param _loyaltyAddress of account
349
      * @return (uint256)
350
351
      function getTimesJoined(address _loyaltyAddress) public view whenNotPaused returns(uint256) {
352
      return accounts[_loyaltyAddress]._joined;
353
354
355
356
      * @dev Get time remaining before reward maturity
357
      * @param _loyaltyAddress of account
358
      * @return (uint256, bool) Indicating time remaining/past and boolean indicating maturity
359
      function getTimeRemaining address _loyaltyAddress | public view whenNotPaused returns (uint256 bool uint256) |
return ISparkleTimestamp/timestampAddress | getTimeRemaining(_loyaltyAddress |
360
361
362
363
364
365
      * @dev Withdraw any ether that has been sent directly to the contract
366
      * @param _loyaltyAddress of account
      * @return Total number of tokens that have been claimed by users
* @notice Test(s) Not written
367
368
369
370
      function getRewardTier(address _loyaltyAddress) public view whenNotPaused returns(uint256) {
371
      return accounts[_loyaltyAddress]._tier;
373
374
```

```
Odev Select reward tier for msg.senderOparam _tierSelected id of the reward tier interested in purchasing
376
377
      * @return (bool) indicating failure/success
378
379
       function selectRewardTier(uint256 _tierSelected) public payable whenNotPaused nonReentrant returns(bool) {
380
      // Validate calling address (msg.sender)
require(msg.sender != address 0x0), 'Invalid {From}');
381
382
       // Validate specified address has a timestamp
383
      require(accounts[msg_sender],_address == address(msg_sender), 'No timestamp');
384
385
      require(accounts[msg.sender]._tier != _tierSelected, 'Already selected');
386
      // Validate that ether was sent with the call
387
      require(msg.value > 0, 'No ether');
388
389
      // Determine if the specified rate is > than existing rate

if(ISparkleRewardTiers(tiersAddress),getRate(accounts/msg_sender)_tier) >= ISparkleRewardTiers(tiersAddress),getRate(_tierSelected)) |
390
391
      // No, revert indicating failure
392
393
394
395
      // Determine if ether transfer for tier upgrade has completed successfully
      if(!address:collectionAddress:.call.value(ISparkleRewardTiers(tiersAddress:.getPrice(_tierSelected:).gas(gasToSendWithTX)('')) = 
// No, revert indicating reward rate is unchanged
396
397
398
      revert('Rate unchanged');
399
400
401
      // Update callers rate with the new selected rate
402
      accounts[msg.sender]._tier_= _tierSelected;
      emit TierSelectedEvent(msg sender, _tierSelected)
403
      // Return success
405
      return true;
406
407
408
      function getRewardTiersAddress() public view whenNotPaused returns(address) {
409
      return tiersAddress;
410
411
412
413
      * @dev Set tier collectionm address
414
        Oparam _newAddress of new collection address
Onotice Test(s) not written
415
416
      function setRewardTiersAddress(address _newAddress) public whenNotPaused onlyOwner nonReentrant ///
/// Validate calling address (msg.sender)
417
418
419
      require(msg.sender |= address(0x0), 'Invalid {From}');
420
      require:_newAddress != address:0). 'Invalid {reward}')...

// Set tier rewards contract address
421
422
423
      tiersAddress = _newAddress;
424
      emit TiersAddressChanged(_newAddress);
425
426
427
      function \ \ getCollectionAddress() \ \ public \ \ view \ \ \ whenNotPaused \ \ returns(address) \ \ \{ \ \ \ \ \ \}
428
      return collectionAddress;
429
430
431
      /** @notice Test(s) passed
      * @dev Set tier collectionm address

* @param _newAddress of new collection address
432
433
434
      function setCollectionAddress(address_newAddress) public whenNotPaused onlyOwner nonReentrant (
// Validate calling address (msg.sender)
435
436
437
      require(msg.sender != address(0x0), 'Invalid {From}');
```

```
// Validate specified address is valid require:_newAddress [:= address:0]. 'Invalid (collection)'),
439
440
       // Set tier collection address
441
      collectionAddress = _newAddress;
       emit CollectionAddressChanged(_newAddress);
443
444
      function getTreasuryAddress() public view whenNotPaused returns(address) {
445
446
      return treasuryAddress;
447
448
449
      * @dev Set treasury address
450
451
        @param _newAddress of the treasury address
452
453
454
      function setTreasuryAddress(address _newAddress) public onlyOwner whenNotPaused nonReentrant
455
456
       // Validate calling address (msg.sender)
457
      require(msg.sender != address(0), "Invalid {from}");
458
      // Validate specified address
      require(_newAddress != address(0), "Invalid {treasury}");
// Set current treasury contract address
459
460
461
      treasuryAddress = _newAddress;
emit TreasuryAddressChanged(_newAddress.;
463
464
465
      function getTimestampAddress() public view whenNotPaused returns(address) {
466
      return timestampAddress;
468
469
      * @dev Set the timestamp address
* @param _newAddress of timestamp address
470
471
472
      * @notice Test(s) passed
473
474
      function setTimestampAddress(address _newAddress) public onlyOwner whenNotPaused nonReentrant
475
476
      // Validate calling address (msg.sender)
      require(msg.sender != address(0), "Invalid {from}");
// Set current timestamp contract address
477
478
479
      timestampAddress = _newAddress
emit TimestampAddress(_newAddress);
480
481
482
483
      function getTokenAddress() public view whenNotPaused returns(address) {
484
      return tokenAddress;
485
486
487
      • Odev Set the loyalty token address
• Oparam _newAddress of the new token address
• Onotice Test(s) passed
488
489
490
491
      function setTokenAddress(address _newAddress public onlyOwner whenNotPaused nonReentrant :

// Validate calling address (msg.sender)

require(msg.sender != address 0), "Invalid {from}");
492
493
495
       // Set current token contract address
496
      tokenAddress = _newAddress;
497
      emit TokenAddressChangedEvent(_newAddress);
498
499
          nction getSentGasAmount() public view whenNotPaused returns(uint256) {
```

```
return gasToSendWithTX;
502
503
504
       function setSentGasAmount(uint256 _amount) public onlyOwner whenNotPaused [ //nonReentrant {
   // Validate calling address (msg.sender)
       require(msg.sender != address(0), 'Invalid {from}');

// Set the current minimum deposit allowed
506
507
508
       gasToSendWithTX = _amount;
509
510
512
       * @dev Set the minimum Proof Of Loyalty amount allowed for deposit
514
       * @notice _minProof value is multiplied internally by 10e7. Do not multiply before calling!
516
      function setMinProof(uint256 _minProof) public onlyOwner whenNotPaused nonReentrant | // Validate calling address (msg.sender)
518
      require(msg_sender_!= address(0), 'Invalid {from}');

// Validate specified minimum is not lower than 1000 tokens
require(_minProof >= 1000, 'Invalid amount');
519
520
521
       // Set the current minimum deposit allowed
minRequired = __minProof mul(10e7 ...
523
524
       emit MinProofChanged(minRequired);
525
526
527
528
529
       * @dev Get the minimum Proof Of Loyalty amount allowed for dep
530
       * @return Amount of tokens required for Proof Of Loyalty Rewards
       * @notice Test(s) passed
532
533
       function getMinProof() public view whenNotPaused returns(uint256) |
// Return indicating minimum deposit allowed
534
535
       return minRequired;
536
537
538
539
       * @dev Set the maximum Proof Of Loyalty amount allowed for deposit
       * @param _maxProof amount for new maximum loyalty reward deposit

* @notice _maxProof value is multiplied internally by 10e7. Do not multiply before calling!
540
541
542
       * @notice Smallest maximum value is 1000 + _minProof amount. (Ex: If _minProof == 1000 then smallest _maxProof possible is 2000)
543
       function setMaxProof(uint256 _maxProof) public onlyOwner whenNotPaused nonReentrant =
544
545
       require(msg.sender != address(0), 'Invalid {from}');
546
547
       require(_maxProof >= 2000, 'Invalid amount');
548
       // Set allow maximum deposit
549
       maxAllowed = _maxProof.mul(10e7);
550
551
552
553
      * @dev Get the maximum Proof Of Loyalty amount allowed for deposit
554
       * @return Maximum amount of tokens allowed for Proof Of Loyalty dep
555
556
557
       function getMaxProof() public view whenNotPaused returns(uint256) {
// Return indicating current allowed maximum deposit
558
559
       return maxAllowed;
560
561
562
563
            lev Get the total number of tokens claimed by all users
```

```
    @return Total number of tokens that have been claimed by users
    @notice Test(s) Not written

565
566
567
      function getTotalTokensClaimed() public view whenNotPaused returns(uint256) {
568
       // Return indicating total number of tokens that have been claimed by all
569
      return totalTokensClaimed;
570
572
573
       * @dev Get total number of times rewards have been claimed for all users
574
       * @return Total number of times rewards have been claimed
575
      * @notice Test(s) Not written
576
577
      function getTotalTimesClaimed() public view whenNotPaused returns(uint256) // Return indicating total number of tokens that have been claimed by all
578
      return totalTimesClaimed;
580
581
582
      583
584
      * @notice Tests not written
585
586
      function withdrawEth(address _toAddress) public onlyOwner whenNotPaused nonReentrant {
      // Validate calling address (msg.sender)
require(msg sender != address(0x0), 'Invalid (from)');
587
588
589
       // Validate specified address
590
      require(_toAddress |= address(0x0), 'Invalid {to}');
591
       // Validate there is ether to withdraw
592
      require(address(this).balance > 0, 'No ether');
593
      // Determine if ether transfer of stored ether has completed successfully
594
      require(address(_toAddress).call.value(address(this).balance).gas(gasToSendWithTX)(), 'Withdraw failed');
595
596
597
598
      * @dev Withdraw any ether that has been sent directly to the contract

* @param _toAddress to receive any stored token balance
599
600
601
      function withdrawTokens(address _toAddress _public onlyOwner whenNotPaused nonReentrant =
602
      // Validate calling address (msg.sender)
require(msg.sender != address(0x0), 'Invalid (from)')
603
604
605
       // Validate specified address
      // Validate specified address (0), "Invalid (to)")

// Validate there are tokens to withdraw
606
607
      require(IERC20(tokenAddress) balanceOf(this) > 0, "No tokens");
// Validate the transfer of tokens completed successfully
608
609
610
      IERC20(tokenAddress).transfer(_toAddress, IERC20(tokenAddress).balanceOf(this));
611
612
613
      function _resetTimestamp(address _rewardAddress) internal {
614
      // Validate calling address (msg.sender)
require(msg.sender != address(0x0), 'Invalid (from)'):
616
       // Validate specified address
      require(_rewardAddress != address(0), "Invalid {reward}");
617
618
619
       ISparkleTimestamp(timestampAddress);resetTimestamp(_rewardAddress);
620
621
622
      function _deleteTimestamp(address _rewardAddress) internal {
623
      // Validate calling address (msg.sender)
require:msg sender != address(0x0), 'Invalid (from)16')
624
625
      // Validate specified address
626
      require(_rewardAddress != address(0), "Invalid {reward}");
```

```
// Delete callers timestamp for specified address
require(ISparkleTimestamp(timestampAddress), 'Delete failed');
628
629
630
631
       function overrideRewardTier(address _loyaltyAccount, uint256 _tierSelected) public whenNotPaused onlyOwner nonReentrant returns(bool)
632
633
       // Validate calling address (msg.sender)
634
      require(msg_sender != address(0x0), 'Invalid {from}');
// Validate specified address has a timestamp
635
636
      require|accounts|_loyaltyAccount) _address == address msg sender), 'No timestamp');
// Update the specified loyalty address tier reward index
637
638
      accounts[msg.sender]._tier = _tierSelected/
639
      emit RewardTierChanged(_loyaltyAccount, _tierSelected);
640
641
643
      * @dev Event signal: Reward tiers address updated
644
645
646
647
648
      * @dev Event signal: Reward tiers address updated
649
      event TiersAddressChanged(address);
650
651
652
653
         @dev Event signal: Collection address updated
654
655
656
657
658
        @dev Event signal: Treasury address updated
659
660
661
662
663
       * @dev Event signal: Timestamp address updated
664
      event TimestampAddressChanged(address);
666
667
668
       * @dev Event signal: Token address updated
669
670
671
672
673
       * @dev Event signal: Account locked/unlocked
674
675
      event LockedAccountEvent(address _rewardAddress, bool _locked);
676
677
678
       * @dev Event signal: Timestamp deleted
679
680
      event DeleteTimestampEvent(address _rewardAddress);
681
682
683
      * @dev Event signal: Reward claimed successfully for address
684
685
      event RewardClaimedEvent(address, uint256);
686
687
688
      * @dev Event signal: Loyalty withdrawn
689
```

```
event LoyaltyWithdrawnEvent address, wint256)

event LoyaltyWithdrawnEvent address, wint256)

event signal; Gas sent with call.value amount changed

event GasSentChanged(wint256);

event GasSentChanged(wint256);
```

Started Tue Aug 25 2020 18:07:18 GMT+0000 (Coordinated Universal Time)

Finished Tue Aug 25 2020 18:22:29 GMT+0000 (Coordinated Universal Time)

Mode Standard

Client Tool Mythx-Cli-0.6.19

Main Source File Contracts/SparkleRewardTiers.Sol

DETECTED VULNERABILITIES

(HIGH	(MEDIUM	(LOW
0	0	1

ISSUES

LOW An outdated compiler version is used.

SWC-102

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

contracts/SparkleRewardTiers.sol

Locations

```
1 /// SWC-103: Floating Pragma
```

pragma solidity 0.4.25;

import '../node_modules/openzeppelin-solidity/contracts/math/SafeMath.sol';