

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	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LOW	An outdated compiler version is used.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
2 | pragma solidity 0.4.25;
3 |
4 | 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

```
293 | require(ISparkleTimestamp(timestampAddress).hasTimestamp(_rewardAddress), 'No timestamp');
294 | // Set the specified address' locked status
295 | accounts[_rewardAddress]._isLocked = _value;
296 | // Emit event log to the block chain for future web3 use
297 | 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

Locations

```
248 |
249 | // Obtain values needed from account record before zeroing
250 | uint256 joinCount = accounts[msg.sender]._joined;
251 | uint256 collected = accounts[msg.sender]._collected;
252 | uint256 deposit = accounts[msg.sender]._balance;
```

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

```
250 | uint256 joinCount = accounts[msg.sender]._joined;
251 | uint256 collected = accounts[msg.sender]._collected;
252 | uint256 deposit = accounts[msg.sender]._balance;
253 | // Zero out the callers account record
254 | 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;
253 | // Zero out the callers account record
254 | delete accounts[msg.sender];
255 | // Carry callers program joined count over to cleared record
256 | 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

Locations

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

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 wistdraw 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

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 wistdraw 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

Locations

```
268 |  
269 | // Determine if transfer from contract address is a sucess  
270 | if(!IERC20(tokenAddress).transfer(msg.sender, deposit)) {  
271 | // No, revert indicating that the treansfer and withdraw has failed  
272 | 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

```
138 | if(maxAllowed > 0) {  
139 | // Yes, determine if the deposit amount + current balance exceed max deposit cap  
140 | if(loyaltyAccount._balance.add(_depositAmount) > maxAllowed || _depositAmount > maxAllowed) {  
141 | // Yes, revert informing that the maximum deposit cap has been exceeded  
142 | 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) {  
139 | // Yes, determine if the deposit amount + current balance exceed max deposit cap  
140 | if(loyaltyAccount._balance.add(_depositAmount) > maxAllowed || _depositAmount > maxAllowed) {  
141 | // Yes, revert informing that the maximum deposit cap has been exceeded  
142 | 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

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

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

```
146 |  
147 | // Determine if the tier selected is enabled  
148 | if(!ISparkleRewardTiers(tiersAddress).getEnabled(loyaltyAccount._tier)) {  
149 | // No, then this tier cannot be selected  
150 | 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 |  
153 | // Determine of transfer from caller has succeeded  
154 | if(ERC20(tokenAddress).transferFrom(msg.sender, this, _depositAmount)) {  
155 | // Yes, then determine if the specified address has a timestamp record  
156 | 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

Locations

```
152 |  
153 | // Determine of transfer from caller has succeeded  
154 | if(ERC20(tokenAddress).transferFrom(msg.sender, this, _depositAmount)) {  
155 | // Yes, then determine if the specified address has a timestamp record  
156 | if(ISparkleTimestamp(timestampAddress).hasTimestamp(msg.sender)) {
```

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

```
154 | if(IERC20(tokenAddress).transferFrom(msg.sender, this, _depositAmount)) {  
155 | // Yes, then determine if the specified address has a timestamp record  
156 | if(ISparkleTimestamp(timestampAddress).hasTimestamp(msg.sender)) {  
157 | // Yes, update callers account balance by deposit amount  
158 | 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)) {  
155 | // Yes, then determine if the specified address has a timestamp record  
156 | if(ISparkleTimestamp(timestampAddress).hasTimestamp(msg.sender)) {  
157 | // Yes, update callers account balance by deposit amount  
158 | 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

Locations

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

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

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

```
616 | require(_rewardAddress != address(0), "Invalid {reward}");  
617 | // Reset callers timestamp for specified address  
618 | ISparkleTimestamp(timestampAddress).resetTimestamp(_rewardAddress);  
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  
2 | pragma solidity 0.4.25;  
3 |  
4 | 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

Locations

```
358 | */  
359 | function getTimeRemaining(address _loyaltyAddress) public view whenNotPaused returns (uint256, bool, uint256) {  
360 | return ISparkleTimestamp(timestampAddress).getTimeRemaining(_loyaltyAddress);  
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

```
291 | require(_rewardAddress != address(0x0), 'Invalid {reward}');
292 | // Validate specified address has timestamp
293 | require([SparkleTimestamp(timestampAddress).hasTimestamp(_rewardAddress), 'No timestamp']);
294 | // Set the specified address' locked status
295 | 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

```
605 | require(_toAddress != address(0), "Invalid {to}");
606 | // Validate there are tokens to withdraw
607 | require(IERC20(tokenAddress).balanceOf(this) > 0, "No tokens");
608 | // Validate the transfer of tokens completed successfully
609 | 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

```
607 | require(IERC20(tokenAddress).balanceOf(this) > 0, "No tokens");
608 | // Validate the transfer of tokens completed successfully
609 | IERC20(tokenAddress).transfer(_toAddress, IERC20(tokenAddress).balanceOf(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

```
607 | require(IERC20(tokenAddress).balanceOf(this) > 0, "No tokens");
608 | // Validate the transfer of tokens completed successfully
609 | IERC20(tokenAddress).transfer(_toAddress, IERC20(tokenAddress).balanceOf(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

```
239 | require(msg.sender != address(0), 'Invalid {from}');
240 | // validate that caller has a loyalty timestamp
241 | require(!SparkleTimestamp(timestampAddress).hasTimestamp(msg.sender), 'No timestamp');
242 |
243 | // 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

```
625 | require(_rewardAddress != address(0), "Invalid {reward}");
626 | // Delete callers timestamp for specified address
627 | require(!SparkleTimestamp(timestampAddress).deleteTimestamp(_rewardAddress), 'Delete failed');
628 | }
```

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

```
127 |
128 | // Determine if caller has approved enough allowance for this deposit
129 | if(IERC20(tokenAddress).allowance(msg.sender, this) < _depositAmount) {
130 | // No, revert informing that deposit amount exceeded allowance amount
131 | revert('Exceeds allowance');
```

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(!IERC20(tokenAddress).transferFrom(treasuryAddress, msg.sender, collected)) {  
265 | // No, revert indicating that the transfer and wistdraw 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

```
268 |  
269 | // Determine if transfer from contract address is a sucess  
270 | if(!IERC20(tokenAddress).transfer(msg.sender, deposit)) {  
271 | // No, revert indicating that the treansfer and withdraw has failed  
272 | 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

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

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

```
607 | require(IERC20(tokenAddress).balanceOf(this) > 0, "No tokens");  
608 | // Validate the transfer of tokens completed successfully  
609 | IERC20(tokenAddress).transfer(_toAddress, IERC20(tokenAddress).balanceOf(this));  
610 | }
```

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

```
625 | require(_rewardAddress != address(0), "Invalid {reward}");
626 | // Delete callers timestamp for specified address
627 | require(!SparkleTimestamp(timestampAddress).deleteTimestamp(_rewardAddress), 'Delete failed');
628 | }
```

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.

SWC-123

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

contracts/SparkleLoyalty.sol

Locations

```
358 | */
359 | function getTimeRemaining(address _loyaltyAddress) public view whenNotPaused returns (uint256, bool, uint256) {
360 | return SparkleTimestamp(timestampAddress).getTimeRemaining(_loyaltyAddress);
361 | }
```

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

```
195 | require(msg.sender != address(0), 'Invalid {from}');
196 | // Validate caller has a timestamp and it has matured
197 | require(ISparkleTimestamp(timestampAddress).hasTimestamp(msg.sender), 'No record');
198 | require(ISparkleTimestamp(timestampAddress).isRewardReady(msg.sender), 'Not mature');
```

Source file

contracts/SparkleLoyalty.sol

Locations

```
14 | * @author SparkleMobile Inc.
15 | */
16 | contract SparkleLoyalty is Ownable, Pausable, ReentrancyGuard {
17 |
18 | /**
19 |  * @dev Ensure math safety through SafeMath
20 |  */
21 | using SafeMath for uint256;
22 |
23 | uint256 private gasToSendWithTX = 21317;
24 | // uint256 private baseRate = 0.00002137 * 10e7; // A full year is 365.2422 gregorian days
25 | uint256 private baseRate = 0.00013690 * 10e7; // A full year is 365.2422 gregorian days (5%)
26 |
27 | struct Account {
28 |     address _address; // Address of loyalty earner
29 |     uint256 _balance; // Balance of tokens physically deposited
30 |     uint256 _collected; // Collected value of token rewards
31 |     uint256 _claimed; // Total number of times a reward has been claimed
32 |     uint256 _joined; // Total number of times this address has joined the program
33 |     uint256 _tier; // Tier index of reward tier for this loyalty earner
34 |     bool _isLocked; // This is the locked record status. (true = no deposits, withdraws, claims)
35 | }
36 |
37 | /**
38 |  * @param tokenAddress of erc20 token used for rewards
39 |  */
40 | address private tokenAddress;
41 |
42 | /**
43 |  * @param timestampAddress of erc20 token used for rewards
44 |  */
45 | address private timestampAddress;
46 |
47 | /**
48 |  * @param treasuryAddress of token reasury used for earned rewards
49 |  */
50 | address private treasuryAddress;
51 |
52 | /**
53 |  * @param collectionAddress of ethereum account used for tier upgrade collection
54 |  */
55 | address private collectionAddress;
56 |
57 | /**
58 |  * @param rewardTiersAddress of smart contract used for tier resolution
59 |  */
```

```

60 address private tiersAddress;
61
62 /**
63  * @param minProofRequired to deposit for rewards eligibility at any tier
64  */
65 uint256 private minRequired;
66
67 /**
68  * @param maxProofAllowed allowed for deposit for rewards eligibility at any tier
69  */
70 uint256 private maxAllowed;
71
72 /**
73  * @param totalTokensClaimed of all rewards awarded
74  */
75 uint256 private totalTokensClaimed;
76
77 /**
78  * @param totalTimesClaimed
79  */
80 uint256 private totalTimesClaimed;
81
82 /**
83  * @param totalActiveAccounts count
84  */
85 uint256 private totalActiveAccounts;
86
87 /**
88  * @param Accounts mapping of user loyalty records
89  */
90 mapping(address => Account) private accounts;
91
92 /**
93  * @dev Sparkle Loyalty Rewards Program contract .cTor
94  * @param _tokenAddress of token used for proof of loyalty rewards
95  * @param _treasuryAddress of proof of loyalty token reward distribution
96  * @param _collectionAddress of ethereum account to collect tier upgrade eth
97  * @param _tiersAddress of the proof of loyalty tier rewards support contract
98  * @param _timestampAddress of the proof of loyalty timestamp support contract
99  */
100 constructor(address _tokenAddress, address _treasuryAddress, address _collectionAddress, address _tiersAddress, address _timestampAddress) public Ownable() Pausable()
101 ReentrancyGuard() {
102
103     // Initialize contract internal addresse(s)
104     tokenAddress = _tokenAddress;
105     treasuryAddress = _treasuryAddress;
106     collectionAddress = _collectionAddress;
107     tiersAddress = _tiersAddress;
108     timestampAddress = _timestampAddress;
109
110     // Initialize minimum/maximum allowed deposit limits
111     minRequired = uint256(1000).mul(10e7);
112     maxAllowed = uint256(250000).mul(10e7);
113 }
114
115 event DepositLoyaltyEvent(address, uint256, bool);
116
117 /**
118  * @dev Deposit additional tokens to a reward address loyalty balance
119  * @param _depositAmount of tokens to deposit into a reward address balance
120  * @return bool indicating the success of the deposit operation (true == success)
121  */
122 function depositLoyalty(uint _depositAmount) public whenNotPaused nonReentrant returns (bool)

```

```

123 |
124 | // Validate calling address (msg.sender)
125 | require(msg.sender != address(0), 'Invalid {from}')
126 | // Validate specified value meets minimum requirements
127 | require(_depositAmount >= minRequired, 'Minimum required')
128 |
129 | // Determine if caller has approved enough allowance for this deposit
130 | if(IERC20(tokenAddress).allowance(msg.sender, this) < _depositAmount) {
131 | // No, revert informing that deposit amount exceeded allowance amount
132 | revert('Exceeds allowance')
133 | }
134 |
135 | // Obtain a storage instance 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) {
140 | // Yes, determine if the deposit amount + current balance exceed max deposit cap
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
149 | if(!ISparkleRewardTiers(tiersAddress).getEnabled(loyaltyAccount._tier)) {
150 | // No, then this tier cannot be selected
151 | revert('Invalid tier')
152 | }
153 |
154 | // Determine if transfer from caller has succeeded
155 | if(IERC20(tokenAddress).transferFrom(msg.sender, this, _depositAmount)) {
156 | // Yes, then determine if the specified address has a timestamp record
157 | if(ISparkleTimestamp(timestampAddress).hasTimestamp(msg.sender)) {
158 | // Yes, update callers account balance by deposit amount
159 | loyaltyAccount._balance = loyaltyAccount._balance.add(_depositAmount)
160 | // Reset the callers reward timestamp
161 | resetTimestamp(msg.sender)
162 | }
163 | emit DepositLoyaltyEvent(msg.sender, _depositAmount, true)
164 | // Return success
165 | return true
166 | }
167 |
168 | // Determine if a timestamp has been added for caller
169 | if(!ISparkleTimestamp(timestampAddress).addTimestamp(msg.sender)) {
170 | // No, revert indicating there was some kind of error
171 | revert('No timestamp created')
172 | }
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 | }
181 | emit DepositLoyaltyEvent(msg.sender, _depositAmount, false)
182 | // Return success
183 | return true
184 | }
185 |

```



```

186 // Return failure
187 return false;
188 }
189
190 /**
191  * @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}');
197     // Validate caller has a timestamp and it has matured
198     require(ISparkleTimestamp(timestampAddress).hasTimestamp(msg.sender), 'No record');
199     require(ISparkleTimestamp(timestampAddress).isRewardReady(msg.sender), 'Not mature');
200
201     // Obtain the current state of the callers timestamp
202     (uint256 timeRemaining, bool isReady, uint256 rewardDate) = ISparkleTimestamp(timestampAddress).getTimeRemaining(msg.sender);
203     // Determine if the callers reward has matured
204     if(isReady) {
205         // 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];
209         // Obtain values required for caculations
210         uint256 dayCount = (timeRemaining.div(ISparkleTimestamp(timestampAddress).getTimePeriod()))+1;
211         uint256 tokenBalance = loyaltyAccount._balance.add(loyaltyAccount._collected);
212         uint256 rewardRate = ISparkleRewardTiers(tiersAddress).getRate(loyaltyAccount._tier);
213         uint256 rewardTotal = baseRate.mul(tokenBalance).mul(rewardRate).mul(dayCount).div(10e7).div(10e7);
214         // Increment collected by reward total
215         loyaltyAccount._collected = loyaltyAccount._collected.add(rewardTotal);
216         // Increment total number of times a reward has been claimed
217         loyaltyAccount._claimed = loyaltyAccount._claimed.add(1);
218         // Increment 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;
222         // Reset the callers timestamp record
223         .resetTimestamp(msg.sender);
224         // Emit event log to the block chain for future web3 use
225         emit RewardClaimedEvent(msg.sender, rewardTotal);
226         // Return success
227         return true;
228     }
229
230     // Revert opposed to returning boolean (May or may not return a txreceipt)
231     revert('Failed claim');
232 }
233
234 /**
235  * @dev Withdraw the current deposit balance + any earned loyalty rewards
236  */
237 function withdrawLoyalty() public whenNotPaused nonReentrant()
238 {
239     // Validate calling address (msg.sender)
240     require(msg.sender != address(0), 'Invalid {from}');
241     // validate that caller has a loyalty timestamp
242     require(ISparkleTimestamp(timestampAddress).hasTimestamp(msg.sender), 'No timestamp');
243
244     // Determine if the account has been locked
245     if(accounts[msg.sender]._isLocked) {
246         // Yes, revert informing that this loyalty account has been locked
247         revert('Locked');
248     }

```

```

249
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 // Decement 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
265 if(!IERC20(tokenAddress).transferFrom(treasuryAddress, msg.sender, collected)) {
266 // No, revert indicating that the transfer and wistdraw has failed
267 revert('Withdraw failed');
268 }
269
270 // Determine if transfer from contract address is a sucess
271 if(!IERC20(tokenAddress).transfer(msg.sender, deposit)) {
272 // 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 /**
281  * @dev Gets the locked status of the specified address
282  * @param _loyaltyAddress of account
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 {
290 // Validate calling address (msg.sender)
291 require(msg.sender != address(0x0), 'Invalid {from}');
292 require(_rewardAddress != address(0x0), 'Invalid {reward}');
293 // Validate specified address has timestamp
294 require(ISparkleTimestamp(timestampAddress).hasTimestamp(_rewardAddress), 'No timestamp');
295 // Set the specified address' locked status
296 accounts[_rewardAddress]._isLocked = _value;
297 // Emit event log to the block chain for future web3 use
298 emit LockedAccountEvent(_rewardAddress, _value);
299 }
300
301 /**
302  * @dev Gets the storage address value of the specified address
303  * @param _loyaltyAddress of account
304  * @return (address) indicating the address stored calls account record
305  */
306 function getLoyaltyAddress(address _loyaltyAddress) public view whenNotPaused returns (address) {
307 return accounts[_loyaltyAddress]._address;
308 }
309
310 /**
311  * @dev Get the deposit balance value of specified address

```

```

312 * @param _loyaltyAddress of account
313 * @return (uint256) indicating the balance value
314 */
315 function getDepositBalance(address _loyaltyAddress) public view whenNotPaused returns(uint256) {
316     return accounts[_loyaltyAddress]._balance;
317 }
318
319 /**
320 * @dev Get the tokens collected by the specified address
321 * @param _loyaltyAddress of account
322 * @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 * @param _loyaltyAddress of account
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
337 /**
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 */
360 function getTimeRemaining(address _loyaltyAddress) public view whenNotPaused returns (uint256, bool, uint256) {
361     return ISparkleTimestamp(timestampAddress).getTimeRemaining(_loyaltyAddress);
362 }
363
364 /**
365 * @dev Withdraw any ether that has been sent directly to the contract
366 * @param _loyaltyAddress of account
367 * @return Total number of tokens that have been claimed by users
368 * @notice Test(s) Not written
369 */
370 function getRewardTier(address _loyaltyAddress) public view whenNotPaused returns(uint256) {
371     return accounts[_loyaltyAddress]._tier;
372 }
373
374 /**

```

```

375 * @dev Select reward tier for msg.sender
376 * @param _tierSelected id of the reward tier interested in purchasing
377 * @return (bool) indicating failure/success
378 */
379 function selectRewardTier(uint256 _tierSelected) public payable whenNotPaused nonReentrant returns(bool) {
380 // Validate calling address (msg.sender)
381 require(msg.sender != address(0x0), 'Invalid {from}');
382 // Validate specified address has a timestamp
383 require(accounts[msg.sender]._address == address(msg.sender), 'No timestamp');
384 // Validate tier selection
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
390 if(ISparkleRewardTiers(tiersAddress).getRate(accounts[msg.sender]._tier) >= ISparkleRewardTiers(tiersAddress).getRate(_tierSelected)) {
391 // No, revert indicating failure
392 revert('Invalid tier');
393 }
394
395 // Determine if ether transfer for tier upgrade has completed successfully
396 if(!address(collectionAddress).call.value(ISparkleRewardTiers(tiersAddress).getPrice(_tierSelected)).gas(gasToSendWithTX)('')) {
397 // No, revert indicating reward rate is unchanged
398 revert('Rate unchanged');
399 }
400
401 // Update callers rate with the new selected rate
402 accounts[msg.sender]._tier = _tierSelected;
403 emit TierSelectedEvent(msg.sender, _tierSelected);
404 // 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 * @param _newAddress of new collection address
415 * @notice Test(s) not written
416 */
417 function setRewardTiersAddress(address _newAddress) public whenNotPaused onlyOwner nonReentrant {
418 // Validate calling address (msg.sender)
419 require(msg.sender != address(0x0), 'Invalid {from}');
420 // Validate specified address is valid
421 require(_newAddress != address(0), 'Invalid {reward}');
422 // Set tier rewards contract address
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
432 * @dev Set tier collectionm address
433 * @param _newAddress of new collection address
434 */
435 function setCollectionAddress(address _newAddress) public whenNotPaused onlyOwner nonReentrant {
436 // Validate calling address (msg.sender)
437 require(msg.sender != address(0x0), 'Invalid {from}');

```

```

438 // Validate specified address is valid
439 require(!_newAddress || address(0), "Invalid {collection}");
440 // Set tier collection address
441 collectionAddress = _newAddress;
442 emit CollectionAddressChanged(_newAddress);
443 }
444
445 function getTreasuryAddress() public view whenNotPaused returns(address) {
446     return treasuryAddress;
447 }
448
449 /**
450  * @dev Set treasury address
451  * @param _newAddress of the treasury address
452  * @notice Test(s) passed
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
459     require(!_newAddress || address(0), "Invalid {treasury}");
460     // Set current treasury contract address
461     treasuryAddress = _newAddress;
462     emit TreasuryAddressChanged(_newAddress);
463 }
464
465 function getTimestampAddress() public view whenNotPaused returns(address) {
466     return timestampAddress;
467 }
468
469 /**
470  * @dev Set the timestamp address
471  * @param _newAddress of timestamp address
472  * @notice Test(s) passed
473  */
474 function setTimestampAddress(address _newAddress) public onlyOwner whenNotPaused nonReentrant
475 {
476     // Validate calling address (msg.sender)
477     require(msg.sender != address(0), "Invalid {from}");
478     // Set current timestamp contract address
479     timestampAddress = _newAddress;
480     emit TimestampAddressChanged(_newAddress);
481 }
482
483 function getTokenAddress() public view whenNotPaused returns(address) {
484     return tokenAddress;
485 }
486
487 /**
488  * @dev Set the loyalty token address
489  * @param _newAddress of the new token address
490  * @notice Test(s) passed
491  */
492 function setTokenAddress(address _newAddress) public onlyOwner whenNotPaused nonReentrant {
493     // Validate calling address (msg.sender)
494     require(msg.sender != address(0), "Invalid {from}");
495     // Set current token contract address
496     tokenAddress = _newAddress;
497     emit TokenAddressChangedEvent(_newAddress);
498 }
499
500 function getSentGasAmount() public view whenNotPaused returns(uint256) {

```

```

501 return gasToSendWithTX
502 }
503
504 function setSentGasAmount(uint256 _amount) public onlyOwner whenNotPaused { //nonReentrant
505 // Validate calling address (msg.sender)
506 require(msg.sender != address(0), 'Invalid {from}');
507 // Set the current minimum deposit allowed
508 gasToSendWithTX = _amount;
509 emit GasSentChanged(_amount);
510 }
511
512 /**
513  * @dev Set the minimum Proof Of Loyalty amount allowed for deposit
514  * @param _minProof amount for new minimum accepted loyalty reward deposit
515  * @notice _minProof value is multiplied internally by 10e7. Do not multiply before calling!
516  */
517 function setMinProof(uint256 _minProof) public onlyOwner whenNotPaused nonReentrant
518 // Validate calling address (msg.sender)
519 require(msg.sender != address(0), 'Invalid {from}');
520 // Validate specified minimum is not lower than 1000 tokens
521 require(_minProof >= 1000, 'Invalid amount');
522 // Set the current minimum deposit allowed
523 minRequired = _minProof.mul(10e7);
524 emit MinProofChanged(minRequired);
525 }
526
527 event MinProofChanged(uint256);
528
529 /**
530  * @dev Get the minimum Proof Of Loyalty amount allowed for deposit
531  * @return Amount of tokens required for Proof Of Loyalty Rewards
532  * @notice Test(s) passed
533  */
534 function getMinProof() public view whenNotPaused returns(uint256)
535 // Return indicating minimum deposit allowed
536 return minRequired;
537 }
538
539 /**
540  * @dev Set the maximum Proof Of Loyalty amount allowed for deposit
541  * @param _maxProof amount for new maximum loyalty reward deposit
542  * @notice _maxProof value is multiplied internally by 10e7. Do not multiply before calling!
543  * @notice Smallest maximum value is 1000 + _minProof amount. (Ex: If _minProof == 1000 then smallest _maxProof possible is 2000)
544  */
545 function setMaxProof(uint256 _maxProof) public onlyOwner whenNotPaused nonReentrant
546 // Validate calling address (msg.sender)
547 require(msg.sender != address(0), 'Invalid {from}');
548 require(_maxProof >= 2000, 'Invalid amount');
549 // Set allow maximum deposit
550 maxAllowed = _maxProof.mul(10e7);
551 }
552
553 /**
554  * @dev Get the maximum Proof Of Loyalty amount allowed for deposit
555  * @return Maximum amount of tokens allowed for Proof Of Loyalty deposit
556  * @notice Test(s) passed
557  */
558 function getMaxProof() public view whenNotPaused returns(uint256)
559 // Return indicating current allowed maximum deposit
560 return maxAllowed;
561 }
562
563 /**
564  * @dev Get the total number of tokens claimed by all users

```

```

564 * @return Total number of tokens that have been claimed by users
565 * @notice Test(s) Not written
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 }
571
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) {
578 // Return indicating total number of tokens that have been claimed by all
579 return totalTimesClaimed;
580 }
581
582 /**
583 * @dev Withdraw any ether that has been sent directly to the contract
584 * @notice Tests not written
585 */
586 function withdrawEth(address _toAddress) public onlyOwner whenNotPaused nonReentrant {
587 // Validate calling address (msg.sender)
588 require(msg.sender != address(0x0), 'Invalid {from}');
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
599 * @param _toAddress to receive any stored token balance
600 * @notice Test(s) incomplete
601 */
602 function withdrawTokens(address _toAddress) public onlyOwner whenNotPaused nonReentrant {
603 // Validate calling address (msg.sender)
604 require(msg.sender != address(0x0), 'Invalid {from}');
605 // Validate specified address
606 require(_toAddress != address(0), 'Invalid {to}');
607 // Validate there are tokens to withdraw
608 require(IERC20(tokenAddress).balanceOf(this) > 0, 'No tokens');
609 // Validate the transfer of tokens completed successfully
610 IERC20(tokenAddress).transfer(_toAddress, IERC20(tokenAddress).balanceOf(this));
611 }
612
613 function _resetTimestamp(address _rewardAddress) internal {
614 // Validate calling address (msg.sender)
615 require(msg.sender != address(0x0), 'Invalid {from}');
616 // Validate specified address
617 require(_rewardAddress != address(0), 'Invalid {reward}');
618 // Reset callers timestamp for specified address
619 ISparkleTimestamp(timestampAddress).resetTimestamp(_rewardAddress);
620 }
621
622 function _deleteTimestamp(address _rewardAddress) internal {
623 // Validate calling address (msg.sender)
624 require(msg.sender != address(0x0), 'Invalid {from}16');
625 // Validate specified address
626 require(_rewardAddress != address(0), 'Invalid {reward}');

```

```

627 // Delete callers timestamp for specified address
628 require(!SparkleTimestamp(timestampAddress).deleteTimestamp(_rewardAddress), 'Delete failed');
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}');
635 // Validate specified address has a timestamp
636 require(accounts[_loyaltyAccount]._address == address(msg.sender), 'No timestamp');
637 // Update the specified loyalty address tier reward index
638 accounts[msg.sender]._tier = _tierSelected;
639 emit RewardTierChanged(_loyaltyAccount, _tierSelected);
640 }
641
642 /**
643  * @dev Event signal: Reward tiers address updated
644  */
645 event TierSelectedEvent(address, uint256);
646
647 /**
648  * @dev Event signal: Reward tiers address updated
649  */
650 event TiersAddressChanged(address);
651
652 /**
653  * @dev Event signal: Collection address updated
654  */
655 event CollectionAddressChanged(address);
656
657 /**
658  * @dev Event signal: Treasury address updated
659  */
660 event TreasuryAddressChanged(address);
661
662 /**
663  * @dev Event signal: Timestamp address updated
664  */
665 event TimestampAddressChanged(address);
666
667 /**
668  * @dev Event signal: Token address updated
669  */
670 event TokenAddressChangedEvent(address);
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 withdraw
689  */

```



```
690 event LoyaltyWithdrawnEvent(address, uint256);
691
692 /**
693  * @dev Event signal: Gas sent with call.value amount changed
694  */
695 event GasSentChanged(uint256);
696
697 /**
698  * @dev Event signal:
699  */
700 event RewardTierChanged(address, uint256);
701
```

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
2 | pragma solidity 0.4.25;
3 |
4 | import '../node_modules/openzeppelin-solidity/contracts/math/SafeMath.sol';
```