

REPORT 5F42F0FC0C10D4001222E9C0




Created Sun Aug 23 2020 22:43:08 GMT+0000 (Coordinated Universal Time)  
Number of analyses 3  
User jonahsparklemobile@gmail.com

## REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
<a href="#">4115acf1-90aa-4c75-88ad-62199a1a494a</a>	contracts/SparkleTimestamp.sol	0
<a href="#">287859dc-0d53-4bda-863c-021f09f5942b</a>	contracts/SparkleLoyalty.sol	2
<a href="#">b6a808f2-eb5b-401d-934b-8d1d5287fdbd</a>	contracts/SparkleRewardTiers.sol	0

Started	Sun Aug 23 2020 22:43:18 GMT+0000 (Coordinated Universal Time)
Finished	Sun Aug 23 2020 22:58:30 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	0

ISSUES

Started	Sun Aug 23 2020 22:43:18 GMT+0000 (Coordinated Universal Time)
Finished	Sun Aug 23 2020 22:58:32 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	0	2

## ISSUES

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

```
484 | returns (uint256, bool, uint256)
485 | {
486 | (uint256 remaining, bool status, uint256 deposit) = !SparkleTimestamp(timestampAddress).getTimeRemaining(_loyaltyAddress);
487 | return (remaining, status, deposit);
488 | }
```

Source file

contracts/SparkleLoyalty.sol

Locations

```
484 | returns (uint256, bool, uint256)
485 | {
486 | (uint256 remaining, bool status, uint256 deposit) = !SparkleTimestamp(timestampAddress).getTimeRemaining(_loyaltyAddress);
487 | return (remaining, status, deposit);
488 | }
```

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

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

Source file

contracts/SparkleLoyalty.sol

Locations

```
16 | * @author SparkleMobile Inc.
17 | */
18 | contract SparkleLoyalty is Ownable, Pausable, ReentrancyGuard {
19 |
20 | /**
21 |  * @dev Ensure math safety through SafeMath
22 |  */
23 | using SafeMath for uint256;
24 |
25 | // Gas to send with certain transactions that may cost more in the future due to chain growth
26 | uint256 private gasToSendWithTX = 25317;
27 | // Base rate APR (5%) factored to 365,2422 gregorian days
28 | uint256 private baseRate = 0.00013690 * 10e7; // A full year is 365,2422 gregorian days (5%)
29 |
30 | // Account data structure
31 | struct Account {
32 |     address _address; // Loyalty reward address
33 |     uint256 _balance; // Total tokens deposited
34 |     uint256 _collected; // Total tokens collected
35 |     uint256 _claimed; // Total succesfull reward claims
36 |     uint256 _joined; // Total times address has joined
37 |     uint256 _tier; // Tier index of reward tier
38 |     bool _isLocked; // Is the account locked
39 | }
40 |
41 | // tokenAddress of erc20 token address
42 | address private tokenAddress;
43 |
44 | // timestampAddress of time stamp contract address
45 | address private timestampAddress;
46 |
47 | // treasuryAddress of token treasury address
48 | address private treasuryAddress;
49 |
50 | // collectionAddress to receive eth payed for tier upgrades
51 | address private collectionAddress;
52 |
53 | // rewardTiersAddress to resolve reward tier specifications
54 | address private tiersAddress;
55 |
56 | // minProofRequired to deposit of rewards to be eligible
57 | uint256 private minRequired;
58 |
59 | // maxProofAllowed for deposit to be eligible
60 | uint256 private maxAllowed;
61 |
```

```

62 // totalTokensClaimed of all rewards awarded
63 uint256 private totalTokensClaimed;
64
65 // totalTimesClaimed of all successfully claimed rewards
66 uint256 private totalTimesClaimed;
67
68 // totalActiveAccounts count of all currently active addresses
69 uint256 private totalActiveAccounts;
70
71 // Accounts mapping of user loyalty records
72 mapping(address => Account) private accounts;
73
74 /**
75  * @dev Sparkle Loyalty Rewards Program contract .cTor
76  * @param _tokenAddress of token used for proof of loyalty rewards
77  * @param _treasuryAddress of proof of loyalty token reward distribution
78  * @param _collectionAddress of ethereum account to collect tier upgrade eth
79  * @param _tiersAddress of the proof of loyalty tier rewards support contract
80  * @param _timestampAddress of the proof of loyalty timestamp support contract
81  */
82 constructor(address _tokenAddress, address _treasuryAddress, address _collectionAddress, address _tiersAddress, address _timestampAddress)
83 public
84 Ownable()
85 Pausable()
86 ReentrancyGuard()
87 {
88 // Initialize contract internal address(es) from params
89 tokenAddress = _tokenAddress;
90 treasuryAddress = _treasuryAddress;
91 collectionAddress = _collectionAddress;
92 tiersAddress = _tiersAddress;
93 timestampAddress = _timestampAddress;
94
95 // Initialize minimum/maximum allowed deposit limits
96 minRequired = uint256(1000).mul(10e7);
97 maxAllowed = uint256(250000).mul(10e7);
98 }
99
100 /**
101  * @dev Deposit additional tokens to a reward address loyalty balance
102  * @param _depositAmount of tokens to deposit into a reward address balance
103  * @return bool indicating the success of the deposit operation (true == success)
104  */
105 function depositLoyalty(uint _depositAmount)
106 public
107 whenNotPaused
108 nonReentrant
109 returns (bool)
110 {
111 // Validate calling address (msg.sender)
112 require(msg.sender != address(0), 'Invalid {from}!');
113 // Validate specified value meets minimum requirements
114 require(_depositAmount >= minRequired, 'Minimum required');
115
116 // Determine if caller has approved enough allowance for this deposit
117 if(!IERC20(tokenAddress).allowance(msg.sender, address(this)) < _depositAmount)
118 // No, revert informing that deposit amount exceeded allowance amount
119 revert('Exceeds allowance');
120 }
121
122 // Obtain a storage instance of callers account record
123 Account storage loyaltyAccount = accounts[msg.sender];
124

```

```

125 // Determine if there is an upper deposit cap
126 if(maxAllowed > 0) {
127     // Yes, determine if the deposit amount + current balance exceed max deposit cap
128     if(loyaltyAccount._balance.add(_depositAmount) > maxAllowed || _depositAmount > maxAllowed) {
129         // Yes, revert informing that the maximum deposit cap has been exceeded
130         revert('Exceeds cap');
131     }
132
133 }
134
135 // Determine if the tier selected is enabled
136 if(!ISparkleRewardTiers(tiersAddress).getEnabled(loyaltyAccount._tier)) {
137     // No, then this tier cannot be selected
138     revert('Invalid tier');
139 }
140
141 // Determine if transfer from caller has succeeded
142 if(IERC20(tokenAddress).transferFrom(msg.sender, address(this), _depositAmount)) {
143     // Yes, then determine if the specified address has a timestamp record
144     if(ISparkleTimestamp(timestampAddress).hasTimestamp(msg.sender)) {
145         // Yes, update callers account balance by deposit amount
146         loyaltyAccount._balance = loyaltyAccount._balance.add(_depositAmount);
147         // Reset the callers reward timestamp
148         .resetTimestamp(msg.sender);
149         //
150         emit DepositLoyaltyEvent(msg.sender, _depositAmount, true);
151         // Return success
152         return true;
153     }
154
155     // Determine if a timestamp has been added for caller
156     if(!ISparkleTimestamp(timestampAddress).addTimestamp(msg.sender)) {
157         // No, revert indicating there was some kind of error
158         revert('No timestamp created');
159     }
160
161     // Prepare loyalty account record
162     loyaltyAccount._address = address(msg.sender);
163     loyaltyAccount._balance = _depositAmount;
164     loyaltyAccount._joined = loyaltyAccount._joined.add(1);
165     // Update global account counter
166     totalActiveAccounts = totalActiveAccounts.add(1);
167     //
168     emit DepositLoyaltyEvent(msg.sender, _depositAmount, false);
169     // Return success
170     return true;
171 }
172
173 // Return failure
174 return false;
175 }
176
177 /**
178  * @dev Claim Sparkle Loyalty reward
179  */
180 function claimLoyaltyReward()
181 public
182 whenNotPaused
183 nonReentrant
184 returns(bool)
185 {
186     // Validate calling address (msg.sender)
187     require(msg.sender != address(0), 'Invalid {from}');

```

```

188 // Validate caller has a timestamp and it has matured
189 require(ISparkleTimestamp(timestampAddress).hasTimestamp(msg.sender), 'No record');
190 require(ISparkleTimestamp(timestampAddress).isRewardReady(msg.sender), 'Not mature');
191
192 // Obtain the current state of the callers timestamp
193 uint256 timeRemaining, bool isReady, uint256 rewardDate = ISparkleTimestamp(timestampAddress).getTimeRemaining(msg.sender);
194 // Determine if the callers reward has matured
195 if(isReady) {
196 // Value not used but throw unused var warning (cleanup)
197 rewardDate = 0;
198 // Yes, then obtain a storage instance of callers account record
199 Account storage loyaltyAccount = accounts[msg.sender];
200 // Obtain values required for calculations
201 uint256 dayCount = (timeRemaining.div(ISparkleTimestamp(timestampAddress).getTimePeriod()))+1;
202 uint256 tokenBalance = loyaltyAccount._balance.add(loyaltyAccount._collected);
203 uint256 rewardRate = ISparkleRewardTiers(tiersAddress).getRate(loyaltyAccount._tier);
204 uint256 rewardTotal = baseRate.mul(tokenBalance).mul(rewardRate).mul(dayCount).div(10e7).div(10e7);
205 // Increment collected by reward total
206 loyaltyAccount._collected = loyaltyAccount._collected.add(rewardTotal);
207 // Increment total number of times a reward has been claimed
208 loyaltyAccount._claimed = loyaltyAccount._claimed.add(1);
209 // Increment total number of times rewards have been collected by all
210 totalTimesClaimed = totalTimesClaimed.add(1);
211 // Increment total number of tokens claimed
212 totalTokensClaimed += rewardTotal;
213 // Reset the callers timestamp record
214 resetTimestamp(msg.sender);
215 // Emit event log to the block chain for future web3 use
216 emit RewardClaimedEvent(msg.sender, rewardTotal);
217 // Return success
218 return true;
219 }
220
221 // Revert opposed to returning boolean (May or may not return a txreceipt)
222 revert('Failed claim');
223 }
224
225 /**
226 * @dev Withdraw the current deposit balance + any earned loyalty rewards
227 */
228 function withdrawLoyalty()
229 public
230 whenNotPaused
231 nonReentrant
232 {
233 // Validate calling address (msg.sender)
234 require(msg.sender != address(0), 'Invalid {from}');
235 // validate that caller has a loyalty timestamp
236 require(ISparkleTimestamp(timestampAddress).hasTimestamp(msg.sender), 'No timestamp2');
237
238 // Determine if the account has been locked
239 if(accounts[msg.sender]._isLocked) {
240 // Yes, revert informing that this loyalty account has been locked
241 revert('locked');
242 }
243
244 // Obtain values needed from account record before zeroing
245 uint256 joinCount = accounts[msg.sender]._joined;
246 uint256 collected = accounts[msg.sender]._collected;
247 uint256 deposit = accounts[msg.sender]._balance;
248 bool isLocked = accounts[msg.sender]._isLocked;
249 // Zero out the callers account record
250 Account storage account = accounts[msg.sender];

```

```

251 account._address = _address(0x0);
252 account._balance = 0x0;
253 account._collected = 0x0;
254 account._joined = joinCount;
255 account._claimed = 0x0;
256 account._tier = 0x0;
257 // Preserve account lock even after withdraw (account always locked)
258 account._isLocked = isLocked;
259 // Decement the total number of active accounts
260 totalActiveAccounts = totalActiveAccounts.sub(1);
261
262 // Delete the callers timestamp record
263 deleteTimestamp(msg.sender);
264
265 // Determine if transfer from treasury address is a success
266 if(!IERC20(tokenAddress).transferFrom(treasuryAddress, msg.sender, collected)){
267 // No, revert indicating that the transfer and wistdraw has failed
268 revert('Withdraw failed');
269 }
270
271 // Determine if transfer from contract address is a success
272 if(!IERC20(tokenAddress).transfer(msg.sender, deposit)){
273 // No, revert indicating that the treansfer and withdraw has failed
274 revert('Withdraw failed');
275 }
276
277 // Emit event log to the block chain for future web3 use
278 emit LoyaltyWithdrawnEvent(msg.sender, deposit.add(collected));
279 }
280
281 function returnLoyaltyDeposit(address _rewardAddress)
282 public
283 whenNotPaused
284 onlyOwner
285 nonReentrant
286 {
287 // Validate calling address (msg.sender)
288 require(msg.sender != address(0), 'Invalid {from}');
289 // validate that caller has a loyalty timestamp
290 require(ISparkleTimestamp(timestampAddress).hasTimestamp(_rewardAddress, 'No timestamp2'));
291 // Validate that reward address is locked
292 require(accounts[_rewardAddress]._isLocked, 'Lock account first');
293 uint256 deposit = accounts[_rewardAddress]._balance;
294 Account storage account = accounts[_rewardAddress];
295 account._balance = 0x0;
296 // Determine if transfer from contract address is a success
297 if(!IERC20(tokenAddress).transfer(_rewardAddress, deposit)){
298 // No, revert indicating that the treansfer and withdraw has failed
299 revert('Withdraw failed');
300 }
301
302 // Emit event log to the block chain for future web3 use
303 emit LoyaltyDepositWithdrawnEvent(_rewardAddress, deposit);
304 }
305
306 function returnLoyaltyCollected(address _rewardAddress)
307 public
308 whenNotPaused
309 onlyOwner
310 nonReentrant
311 {
312 // Validate calling address (msg.sender)
313 require(msg.sender != address(0), 'Invalid {from}');

```



```

314 // validate that caller has a loyalty timestamp
315 require(!SparkleTimestamp(timestampAddress).hasTimestamp(_rewardAddress, 'No timestamp2b'));
316 // Validate that reward address is locked
317 require(accounts[_rewardAddress]._isLocked, 'Lock account first');
318 uint256 collected = accounts[_rewardAddress]._collected;
319 Account storage account = accounts[_rewardAddress];
320 account._collected = 0x0;
321 // Determine if transfer from treasury address is a success
322 if(!IERC20(tokenAddress).transferFrom(treasuryAddress, _rewardAddress, collected)) {
323 // No, revert indicating that the transfer and withdraw has failed
324 revert('Withdraw failed');
325 }
326
327 // Emit event log to the block chain for future web3 use
328 emit LoyaltyCollectedWithdrawnEvent(_rewardAddress, collected);
329 }
330
331 function removeLoyaltyAccount(address _rewardAddress)
332 public
333 whenNotPaused
334 onlyOwner
335 nonReentrant
336 {
337 // Validate calling address (msg.sender)
338 require(msg.sender != address(0), 'Invalid (from)');
339 // validate that caller has a loyalty timestamp
340 require(!SparkleTimestamp(timestampAddress).hasTimestamp(_rewardAddress, 'No timestamp2b'));
341 // Validate that reward address is locked
342 require(accounts[_rewardAddress]._isLocked, 'Lock account first');
343 uint256 joinCount = accounts[_rewardAddress]._joined;
344 Account storage account = accounts[_rewardAddress];
345 account._address = address(0x0);
346 account._balance = 0x0;
347 account._collected = 0x0;
348 account._joined = joinCount;
349 account._claimed = 0x0;
350 account._tier = 0x0;
351 account._isLocked = false;
352 // Decement the total number of active accounts
353 totalActiveAccounts = totalActiveAccounts.sub(1);
354
355 // Delete the callers timestamp record
356 deleteTimestamp(_rewardAddress);
357
358 emit LoyaltyAccountRemovedEvent(_rewardAddress);
359 }
360
361 /**
362 * @dev Gets the locked status of the specified address
363 * @param _loyaltyAddress of account
364 * @return (bool) indicating locked status
365 */
366 function isLocked(address _loyaltyAddress)
367 public
368 view
369 whenNotPaused
370 returns (bool)
371 {
372 return accounts[_loyaltyAddress]._isLocked;
373 }
374
375 function lockAccount(address _rewardAddress, bool _value)
376 public

```

```

377 onlyOwner
378 whenNotPaused
379 nonReentrant
380
381 // Validate calling address (msg.sender)
382 require(msg.sender != address(0x0), 'Invalid {from}');
383 require(_rewardAddress != address(0x0), 'Invalid {reward}');
384 // Validate specified address has timestamp
385 require(ISparkleTimestamp(timestampAddress).hasTimestamp(_rewardAddress, 'No timestamp'));
386 // Set the specified address' locked status
387 accounts[_rewardAddress]._isLocked = _value;
388 // Emit event log to the block chain for future web3 use
389 emit LockedAccountEvent(_rewardAddress, _value);
390
391
392 /**
393  * @dev Gets the storage address value of the specified address
394  * @param _loyaltyAddress of account
395  * @return (address) indicating the address stored calls account record
396  */
397 function getLoyaltyAddress(address _loyaltyAddress)
398 public
399 view
400 whenNotPaused
401 returns (address)
402 {
403     return accounts[_loyaltyAddress]._address;
404 }
405
406 /**
407  * @dev Get the deposit balance value of specified address
408  * @param _loyaltyAddress of account
409  * @return (uint256) indicating the balance value
410  */
411 function getDepositBalance(address _loyaltyAddress)
412 public
413 view
414 whenNotPaused
415 returns (uint256)
416 {
417     return accounts[_loyaltyAddress]._balance;
418 }
419
420 /**
421  * @dev Get the tokens collected by the specified address
422  * @param _loyaltyAddress of account
423  * @return (uint256) indicating the tokens collected
424  */
425 function getTokensCollected(address _loyaltyAddress)
426 public
427 view
428 whenNotPaused
429 returns (uint256)
430 {
431     return accounts[_loyaltyAddress]._collected;
432 }
433
434 /**
435  * @dev Get the total balance (deposit + collected) of tokens
436  * @param _loyaltyAddress of account
437  * @return (uint256) indicating total balance
438  */
439 function getTotalBalance(address _loyaltyAddress)

```

```

440 public
441 view
442 whenNotPaused
443 returns:uint256
444 {
445     return accounts[_loyaltyAddress]._balance.add(accounts[_loyaltyAddress]._collected)
446 }
447
448 /**
449  * @dev Get the times loyalty has been claimed
450  * @param _loyaltyAddress of account
451  * @return (uint256) indicating total time claimed
452  */
453 function getTimesClaimed(address _loyaltyAddress)
454 public
455 view
456 whenNotPaused
457 returns:uint256
458 {
459     return accounts[_loyaltyAddress]._claimed
460 }
461
462 /**
463  * @dev Get total number of times joined
464  * @param _loyaltyAddress of account
465  * @return (uint256)
466  */
467 function getTimesJoined(address _loyaltyAddress)
468 public
469 view
470 whenNotPaused
471 returns:uint256
472 {
473     return accounts[_loyaltyAddress]._joined
474 }
475
476 /**
477  * @dev Get time remaining before reward maturity
478  * @param _loyaltyAddress of account
479  * @return (uint256, bool) Indicating time remaining/past and boolean indicating maturity
480  */
481 function getTimeRemaining(address _loyaltyAddress)
482 public
483 whenNotPaused
484 returns (uint256, bool, uint256)
485 {
486     (uint256 remaining, bool status, uint256 deposit) = ISparkleTimestamp(timestampAddress).getTimeRemaining(_loyaltyAddress);
487     return (remaining, status, deposit);
488 }
489
490 /**
491  * @dev Withdraw any ether that has been sent directly to the contract
492  * @param _loyaltyAddress of account
493  * @return Total number of tokens that have been claimed by user$
494  * @notice Test(s) Not written
495  */
496 function getRewardTier(address _loyaltyAddress)
497 public
498 view whenNotPaused
499 returns:uint256
500 {
501     return accounts[_loyaltyAddress]._tier
502 }

```

```

503
504 /**
505  * @dev Select reward tier for msg.sender
506  * @param _tierSelected id of the reward tier interested in purchasing
507  * @return (bool) indicating failure/success
508  */
509 function selectRewardTier(uint256 _tierSelected)
510 public
511 payable
512 whenNotPaused
513 nonReentrant
514 returns (bool)
515 {
516     // Validate calling address (msg.sender)
517     require(msg.sender != address(0x0), 'Invalid {from}');
518     // Validate specified address has a timestamp
519     require(accounts[msg.sender]._address == address(msg.sender), 'No timestamp3');
520     // Validate tier selection
521     require(accounts[msg.sender]._tier != _tierSelected, 'Already selected');
522     // Validate that ether was sent with the call
523     require(msg.value > 0, 'No ether');
524
525     // Determine if the specified rate is > than existing rate
526     if(ISparkleRewardTiers(tiersAddress).getRate(accounts[msg.sender]._tier) >= ISparkleRewardTiers(tiersAddress).getRate(_tierSelected)) {
527         // No, revert indicating failure
528         revert('Invalid tier');
529     }
530
531     // Determine if ether transfer for tier upgrade has completed successfully
532     (bool success, ) = address(collectionAddress).call{value: ISparkleRewardTiers(tiersAddress).getPrice(_tierSelected), gas: gasToSendWithTX}('');
533     require(success, 'Rate unchanged');
534
535     // Update callers rate with the new selected rate
536     accounts[msg.sender]._tier = _tierSelected;
537     emit TierSelectedEvent(msg.sender, _tierSelected);
538     // Return success
539     return true;
540 }
541
542 function getRewardTiersAddress()
543 public
544 view
545 whenNotPaused
546 returns (address)
547 {
548     return tiersAddress;
549 }
550
551 /**
552  * @dev Set tier collection address
553  * @param _newAddress of new collection address
554  * @notice Test(s) not written
555  */
556 function setRewardTiersAddress(address _newAddress)
557 public
558 whenNotPaused
559 onlyOwner
560 nonReentrant
561 {
562     // Validate calling address (msg.sender)
563     require(msg.sender != address(0x0), 'Invalid {from}');
564     // Validate specified address is valid
565     require(_newAddress != address(0), 'Invalid {reward}');

```

```

566 // Set tier rewards contract address
567 tiersAddress = _newAddress
568 emit TiersAddressChanged(_newAddress)
569 }
570
571 function getCollectionAddress()
572 public
573 view
574 whenNotPaused
575 returns address
576 {
577     return collectionAddress
578 }
579
580 /** @notice Test(s) passed
581 * @dev Set tier collection address
582 * @param _newAddress of new collection address
583 */
584 function setCollectionAddress(address _newAddress)
585 public
586 whenNotPaused
587 onlyOwner
588 nonReentrant
589 {
590     // Validate calling address (msg.sender)
591     require(msg.sender != address(0x0), "Invalid {From}")
592     // Validate specified address is valid
593     require(_newAddress != address(0), "Invalid {collection}")
594     // Set tier collection address
595     collectionAddress = _newAddress
596     emit CollectionAddressChanged(_newAddress)
597 }
598
599 function getTreasuryAddress()
600 public
601 view
602 whenNotPaused
603 returns address
604 {
605     return treasuryAddress
606 }
607
608 /**
609 * @dev Set treasury address
610 * @param _newAddress of the treasury address
611 * @notice Test(s) passed
612 */
613 function setTreasuryAddress(address _newAddress)
614 public
615 onlyOwner
616 whenNotPaused
617 nonReentrant
618 {
619     // Validate calling address (msg.sender)
620     require(msg.sender != address(0), "Invalid {from}")
621     // Validate specified address
622     require(_newAddress != address(0), "Invalid {treasury}")
623     // Set current treasury contract address
624     treasuryAddress = _newAddress
625     emit TreasuryAddressChanged(_newAddress)
626 }
627
628 function getTimestampAddress()

```

```

629 public
630 view
631 whenNotPaused
632 returns:address
633 {}
634 return timestampAddress;
635 }
636
637 /**
638  * @dev Set the timestamp address
639  * @param _newAddress of timestamp address
640  * @notice Test(s) passed
641  */
642 function setTimestampAddress(address _newAddress)
643 public
644 onlyOwner
645 whenNotPaused
646 nonReentrant
647 {}
648 // Validate calling address (msg.sender)
649 require(msg.sender != address(0), "Invalid (from)");
650 // Set current timestamp contract address
651 timestampAddress = _newAddress;
652 emit TimestampAddressChanged(_newAddress);
653 }
654
655 function getTokenAddress()
656 public
657 view
658 whenNotPaused
659 returns:address
660 {}
661 return tokenAddress;
662 }
663
664 /**
665  * @dev Set the loyalty token address
666  * @param _newAddress of the new token address
667  * @notice Test(s) passed
668  */
669 function setTokenAddress(address _newAddress)
670 public
671 onlyOwner
672 whenNotPaused
673 nonReentrant
674 {}
675 // Validate calling address (msg.sender)
676 require(msg.sender != address(0), "Invalid (from)");
677 // Set current token contract address
678 tokenAddress = _newAddress;
679 emit TokenAddressChangedEvent(_newAddress);
680 }
681
682 function getSentGasAmount()
683 public
684 view
685 whenNotPaused
686 returns:uint256
687 {}
688 return gasToSendWithTX;
689 }
690
691 function setSentGasAmount(uint256 _amount)

```

```

692 public
693 onlyOwner
694 whenNotPaused
695 {
696     // Validate calling address (msg.sender)
697     require(msg.sender != address(0), 'Invalid {from}');
698     // Set the current minimum deposit allowed
699     gasToSendWithTX = _amount;
700     emit GasSentChanged(_amount);
701 }
702
703 /**
704  * @dev Set the minimum Proof Of Loyalty amount allowed for deposit
705  * @param _minProof amount for new minimum accepted loyalty reward deposit
706  * @notice _minProof value is multiplied internally by 10e7. Do not multiply before calling!
707  */
708 function setMinProof(uint256 _minProof)
709 public
710 onlyOwner
711 whenNotPaused
712 nonReentrant
713 {
714     // Validate calling address (msg.sender)
715     require(msg.sender != address(0), 'Invalid {from}');
716     // Validate specified minimum is not lower than 1000 tokens
717     require(_minProof >= 1000, 'Invalid amount');
718     // Set the current minimum deposit allowed
719     minRequired = _minProof.mul(10e7);
720     emit MinProofChanged(minRequired);
721 }
722
723 event MinProofChanged(uint256);
724
725 /**
726  * @dev Get the minimum Proof Of Loyalty amount allowed for deposit
727  * @return Amount of tokens required for Proof Of Loyalty Rewards
728  * @notice Test(s) passed
729  */
730 function getMinProof()
731 public
732 view
733 whenNotPaused
734 returns (uint256)
735 {
736     // Return indicating minimum deposit allowed
737     return minRequired;
738 }
739
740 /**
741  * @dev Set the maximum Proof Of Loyalty amount allowed for deposit
742  * @param _maxProof amount for new maximum loyalty reward deposit
743  * @notice _maxProof value is multiplied internally by 10e7. Do not multiply before calling!
744  * @notice Smallest maximum value is 1000 + _minProof amount. (Ex: If _minProof == 1000 then smallest _maxProof possible is 2000)
745  */
746 function setMaxProof(uint256 _maxProof)
747 public
748 onlyOwner
749 whenNotPaused
750 nonReentrant
751 {
752     // Validate calling address (msg.sender)
753     require(msg.sender != address(0), 'Invalid {from}');
754     require(_maxProof >= 2000, 'Invalid amount');
755     // Set allow maximum deposit

```

```

755 maxAllowed = _maxProof.mul(10e7);
756
757
758 /**
759  * @dev Get the maximum Proof Of Loyalty amount allowed for deposit
760  * @return Maximum amount of tokens allowed for Proof Of Loyalty deposit
761  * @notice Test(s) passed
762  */
763 function getMaxProof()
764 public
765 view
766 whenNotPaused
767 returns(uint256)
768 {
769     // Return indicating current allowed maximum deposit
770     return maxAllowed;
771 }
772
773 /**
774  * @dev Get the total number of tokens claimed by all users
775  * @return Total number of tokens that have been claimed by users
776  * @notice Test(s) Not written
777  */
778 function getTotalTokensClaimed()
779 public
780 view
781 whenNotPaused
782 returns(uint256)
783 {
784     // Return indicating total number of tokens that have been claimed by all
785     return totalTokensClaimed;
786 }
787
788 /**
789  * @dev Get total number of times rewards have been claimed for all users
790  * @return Total number of times rewards have been claimed
791  */
792 function getTotalTimesClaimed()
793 public
794 view
795 whenNotPaused
796 returns(uint256)
797 {
798     // Return indicating total number of tokens that have been claimed by all
799     return totalTimesClaimed;
800 }
801
802 /**
803  * @dev Withdraw any ether that has been sent directly to the contract
804  */
805 function withdrawEth(address _toAddress)
806 public
807 onlyOwner
808 whenNotPaused
809 nonReentrant
810 {
811     // Validate calling address (msg.sender)
812     require(msg.sender != address(0x0), 'Invalid {from}');
813     // Validate specified address
814     require(_toAddress != address(0x0), 'Invalid {to}');
815     // Validate there is ether to withdraw
816     require(address(this).balance > 0, 'No ether');
817     // Determine if ether transfer of stored ether has completed successfully

```



```

818 // require(address(_toAddress).call.value(address(this).balance).gas(gasToSendWithTX()), 'Withdraw failed');
819 bool success = address(_toAddress).call.value(address(this).balance, gas, gasToSendWithTX(''));
820 require(success, 'Withdraw failed');
821 }
822
823 /**
824  * @dev Withdraw any ether that has been sent directly to the contract
825  * @param _toAddress to receive any stored token balance
826  */
827 function withdrawTokens(address _toAddress)
828 public
829 onlyOwner
830 whenNotPaused
831 nonReentrant
832 {
833 // Validate calling address (msg.sender)
834 require(msg.sender != address(0x0), 'Invalid {from}');
835 // Validate specified address
836 require(_toAddress != address(0), "Invalid {to}");
837 // Validate there are tokens to withdraw
838 uint256 balance = IERC20(tokenAddress).balanceOf(address(this));
839 require(balance != 0, "No tokens");
840
841 // Validate the transfer of tokens completed successfully
842 if(IERC20(tokenAddress).transfer(_toAddress, balance)) {
843 emit TokensWithdrawn(_toAddress, balance);
844 }
845 }
846
847 /**
848  * @dev Override loyalty account tier by contract owner
849  * @param _loyaltyAccount loyalty account address to tier override
850  * @param _tierSelected reward tier to override current tier value
851  * @return (bool) indicating success status
852  */
853 function overrideRewardTier(address _loyaltyAccount, uint256 _tierSelected)
854 public
855 whenNotPaused
856 onlyOwner
857 nonReentrant
858 returns (bool)
859 {
860 // Validate calling address (msg.sender)
861 require(msg.sender != address(0x0), 'Invalid {from}');
862 require(_loyaltyAccount != address(0x0), 'Invalid {account}');
863 // Validate specified address has a timestamp
864 require(accounts[_loyaltyAccount]._address == address(_loyaltyAccount), 'No timestamp4');
865 // Update the specified loyalty address tier reward index
866 accounts[_loyaltyAccount]._tier = _tierSelected;
867 emit RewardTierChanged(_loyaltyAccount, _tierSelected);
868 }
869
870 /**
871  * @dev Reset the specified loyalty account timestamp
872  * @param _rewardAddress of the loyalty account to perform a reset
873  */
874 function _resetTimestamp(address _rewardAddress)
875 internal
876 {
877 // Validate calling address (msg.sender)
878 require(msg.sender != address(0x0), 'Invalid {from}');
879 // Validate specified address
880 require(_rewardAddress != address(0), "Invalid {reward}");

```

```




881 // Reset callers timestamp for specified address
882 require(!SparkleTimestamp(timestampAddress).resetTimestamp(_rewardAddress, 'Reset failed'));
883 emit ResetTimestampEvent(_rewardAddress);
884 }
885
886 /**
887  * @dev Delete the specified loyalty account timestamp
888  * @param _rewardAddress of the loyalty account to perform the delete
889  */
890 function _deleteTimestamp(address _rewardAddress)
891 internal
892 {
893 // Validate calling address (msg.sender)
894 require(msg.sender != address(0x0), 'Invalid {from}16');
895 // Validate specified address
896 require(_rewardAddress != address(0), 'Invalid {reward}*');
897 // Delete callers timestamp for specified address
898 require(!SparkleTimestamp(timestampAddress).deleteTimestamp(_rewardAddress, 'Delete failed'));
899 emit DeleteTimestampEvent(_rewardAddress);
900 }
901
902 /**
903  * @dev Event signal: Treasury address updated
904  */
905 event TreasuryAddressChanged(address);
906
907 /**
908  * @dev Event signal: Timestamp address updated
909  */
910 event TimestampAddressChanged(address);
911
912 /**
913  * @dev Event signal: Token address updated
914  */
915 event TokenAddressChangedEvent(address);
916
917 /**
918  * @dev Event signal: Timestamp reset
919  */
920 event ResetTimestampEvent(address _rewardAddress);
921
922 /**
923  * @dev Event signal: Timestamp deleted
924  */
925 event DeleteTimestampEvent(address _rewardAddress);
926
927 /**
928  * @dev Event signal: Loyalty deposited event
929  */
930 event DepositLoyaltyEvent(address, uint256, bool);
931
932 /**
933  * @dev Event signal: Reward claimed successfully for address
934  */
935 event RewardClaimedEvent(address, uint256);
936
937 /**
938  * @dev Event signal: Loyalty withdrawn
939  */
940 event LoyaltyWithdrawnEvent(address, uint256);
941
942 /**
943  * @dev Event signal: Account locked/unlocked

```

```
944 */
945 event LockedAccountEvent(address _rewardAddress, bool _locked);
946
947 /**
948  * @dev Event signal: Loyalty deposit balance withdrawn
949  */
950 event LoyaltyDepositWithdrawnEvent(address, uint256);
951
952 /**
953  * @dev Event signal: Loyalty collected balance withdrawn
954  */
955 event LoyaltyCollectedWithdrawnEvent(address, uint256);
956
957 /**
958  * @dev Event signal: Loyalty account removed
959  */
960 event LoyaltyAccountRemovedEvent(address);
961
962 /**
963  * @dev Event signal: Gas sent with call.value amount updated
964  */
965 event GasSentChanged(uint256);
966
967 /**
968  * @dev Event signal: Reward tiers address updated
969  */
970 event TierSelectedEvent(address, uint256);
971
972 /**
973  * @dev Event signal: Reward tiers address updated
974  */
975 event TiersAddressChanged(address);
976
977 /**
978  * @dev Event signal: Reward tier has been updated
979  */
980 event RewardTierChanged(address, uint256);
981
982 /**
983  * @dev Event signal: Collection address updated
984  */
985 event CollectionAddressChanged(address);
986
987 /**
988  * @dev Event signal: All stored tokens have been removed
989  */
990 event TokensWithdrawn(address, uint256);
991
```

Started	Sun Aug 23 2020 22:43:18 GMT+0000 (Coordinated Universal Time)
Finished	Sun Aug 23 2020 22:58:29 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	0

ISSUES