

STEPN Audit

Copyright © 2022 by Verilog. All rights reserved.

April 4, 2022

by **Verilog Audit**



STEPN Audit

VERILOG

This report presents our engineering engagement with STEPN, a Game-Fi/Social-Fi Web3 application. Users could acquire STEPN NFT sneakers and earn rewards by engaging in outdoor activities. STEPN tokens have two tokens: **GST** & **GMT**.

Table of Content

- [STEPN Audit](#)
 - [Table of Content](#)
 - [Project Summary](#)
 - [Service Scope](#)
 - [GMT & GST Token](#)
 - [Deployment on Solana](#)
 - [Deployment on BNB Chain](#)
 - [Privileged Roles](#)

- Findings & Improvement Suggestions
 - Critical
 - Major
 - Medium
 - Minor
 - Informational
- Reference Code
 - GMT BNB Chain Implementation Code
 - GST Token Contract
 - STEPNNFT nft contract
- Disclaimer

Project Summary

STEPN is a Game-Fi/Social-Fi project with a dual-token system (GMT and GST) and an NFT system (STEPN Sneaker). Users acquire the STEPN Sneaker to participate in the move-and-earn program and earn GST and/or GMT. GMT and GST can be used to upgrade Sneakers and increase the rate of earning. GMT is the governance token, and it is currently deployed on Solana and BNB Chain. GST is the unlimited-supply reward token, and it is currently deployed on Solana.

Service Scope

The review was conducted over 3 days, from April 1st to April 3rd, 2022 by the Verilog team. Our review focused on the **main branch** (<https://github.com/stepnxyz/bnbcontracts>), specifically, commit hash **c07ff86e5e8060de0cd10d6842405964c2cc5d13**

(<https://github.com/stepnxyz/bnbcontracts/tree/c07ff86e5e8060de0cd10d6842405964c2cc5d13>).

Our auditing service for STEPN includes the following two stages:

- Pre-Audit Consulting Service
- Audit Service

1. Pre-Audit Consulting Service

As a part of the pre-audit service, the Verilog team worked closely with the STEPN development team to discuss potential vulnerability and smart contract development best practices in a timely fashion. Verilog team is very appreciative of establishing an

efficient and effective communication channel with the STEPN team, as new findings are often exchanged promptly and fixes were deployed quickly, during the preliminary report stage.

2. Audit Service

The Verilog team conducted a thorough study of the STEPN code. The list of findings, along with the severity and solution, is available under the section **Findings & Improvement Suggestions**.

GMT & GST Token

Below is the summary of GMT & GST token info:

Network	Token	Token Address
Solana	GMT	<u>7i5KKsX2weiTkryZjA4ZwSuXGhs5eJBEjY8vVxR4pfRx</u> (https://solscan.io/token/7i5KKsX2weiTkryZjA4ZwSuXGhs5eJBEjY8vVxR4pfRx).
Solana	GST	<u>AFbX8oGjGpmVFywbVouvhQSRmiW2aR1mohfahi4Y2AdB</u> (https://solscan.io/token/AFbX8oGjGpmVFywbVouvhQSRmiW2aR1mohfahi4Y2AdB).
BNB	GMT	<u>0x3019BF2a2eF8040C242C9a4c5c4BD4C81678b2A1</u> (https://bscscan.com/address/0x3019BF2a2eF8040C242C9a4c5c4BD4C81678b2A1).
BNB	GST	n/a (coming soon)

Deployment on Solana

STEPN dev team used Solana Token Program to create both **GMT** & **GST** tokens. Token Program defines a common implementation for Fungible and Non Fungible tokens.

The document of Token Program: <https://spl.solana.com/token> (<https://spl.solana.com/token>).

The Github of Token Program: <https://github.com/solana-labs/solana-program-library>.
(<https://github.com/solana-labs/solana-program-library>).

As can be seen from the Solscan, both GMT & GST tokens used Solana Token Program:

The image displays two screenshots from the Solscan interface, showing the details for the GMT and GST tokens. Both tokens are confirmed to use the 'Token Program' as their Owner Program, which is highlighted with a red box in the 'Profile Summary' section of each token's page.

Token GMT

Market Overview:

- Max Total Supply: 6,000,000,000.00
- Holders: 8,484
- Website: <https://stepn.com/>
- Social Channels: [\[Icon\]](#)

Profile Summary:

- Token name: GMT (GMT)
- Token address: 7i5KKsX2weiTkry7jA4ZwSuXGhs5eJBEjY8vVxR4pfRx
- Owner Program: **Token Program** (highlighted)
- Decimals: 9

Token GST

Market Overview:

- Max Total Supply: 10,000,000.00
- Holders: 111,833
- Website: <https://stepn.com/>
- Social Channels: [\[Icon\]](#)

Profile Summary:

- Token name: GST (GST)
- Token address: AFbX8oGjGpmVFywbVouvhQSRmiW2aR1mohfahi4Y2AdB
- Owner Program: **Token Program** (highlighted)
- Authority: STEPnQ2UGeGSzCyGvR2nMQAzf8xuejwqebd84wcksCK
- Decimals: 9

Besides, the STEPn project dev team already turned off the emission right for GMT Token as can be seen in the screenshot, GMT token has a fixed supply.

The image shows the 'Overview' section of the GMT token page on Solscan. The 'Fixed Supply' is highlighted with a red box, indicating that the token has a fixed supply of 6,000,000,000.000000000.

GMT

Overview

Address: 7i5KKsX2weiTkry7jA4ZwSuXGhs5eJBEjY8vVxR4pfRx

Fixed Supply: 6,000,000,000.000000000

Website: <https://stepn.com/>

Decimals: 9

Deployment on BNB Chain

Currently, only the GMT token has been deployed on BNB Chain, and the token has been implemented by the Binance Bridge team. The contract deployment address is [0x3019BF2a2eF8040C242C9a4c5c4BD4C81678b2A1](https://bscscan.com/address/0x3019BF2a2eF8040C242C9a4c5c4BD4C81678b2A1) (<https://bscscan.com/address/0x3019BF2a2eF8040C242C9a4c5c4BD4C81678b2A1>).

Below is the summary of GMT token on BNB Chain:

Title	Info
Network	BNB Chain
Token	GMT
Upgradable Contract?	Yes
Proxy Address	<u>0x3019BF2a2eF8040C242C9a4c5c4BD4C81678b2A1</u> (https://bscscan.com/address/0x3019BF2a2eF8040C242C9a4c5c4BD4C81678b2A1).
Implementation Address	<u>0xba5fe23f8a3a24bed3236f05f2fcf35fd0bf0b5c</u> (https://bscscan.com/address/0xba5fe23f8a3a24bed3236f05f2fcf35fd0bf0b5c).

Implementation Contract Source Code can be found in **[GMT BNB Implementation](#)**.
In summary, the implemented smart contracts follow the ERC20 standards.

Privileged Roles

1. GreenSatoshiToken.sol :
 - a. Owner can mint() any amount of GST tokens to any address without limitations.
2. STEPNNFT.sol :
 - a. Owner can setBaseURI() , mint() .

Findings & Improvement Suggestions

Informational Minor Medium Major Critical

	Total	Acknowledged	Resolved
Critical	0	0	0
Major	0	0	0
Medium	2	2	0
Minor	0	0	0
Informational	0	0	0

Critical

none ;)

Major

none ;)

Medium

1. Centralization Risks on GreenSatoshiToken.sol . Medium

```
1 function mint(address to, uint256 amount) public onlyOwner {  
2     _mint(to, amount);  
3 }
```

Description: Owner of this smart contract can mint tokens to certain addresses, Private key leaks may result in the unlimited token supply issue.

Recommendation: uses a multisig wallet to prevent a single point of failure.

Feedback from Project Team: As disclosed in the STEPn whitepaper, GST has an unlimited supply therefore we have to enable the function to mint an unlimited amount of GST.

2. Centralization Risks on STEPnNFT.sol . Medium

```
1 function setBaseURI(string memory buri) public onlyOwner {  
2     require(bytes(buri).length > 0, "wrong base uri");  
3     _buri = buri;  
4 }  
5  
6 function mint(address to, uint256 tokenId) public onlyOwner {  
7     _safeMint(to, tokenId);  
8 }
```

Description: Owner of this smart contract can mint tokens to certain addresses, Private key leaks may result in the unlimited NFT supply issue. Owner of this smart contract can change the base url of the NFT.

Recommendation: uses multisig wallet to prevent a single point of failure.

Feedback from Project Team: STEPn's NFT sneaker also has an unlimited supply, therefore we have to beagle the function to allow an unlimited amount of NFT sneakers to be minted by our users.

Minor

none ;)

Informational

none ;)

Reference Code

In this section, we listed the deployed contract on-chain for your reference.

GMT BNB Chain Implementation Code

```

1  contract BEP20TokenImplementation is Context, IBEP20, Initializable {
2      using SafeMath for uint256;
3
4      mapping (address => uint256) private _balances;
5      mapping (address => mapping (address => uint256)) private _allowances;
6      uint256 private _totalSupply;
7      string private _name;
8      string private _symbol;
9      uint8 private _decimals;
10
11     address private _owner;
12     event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);
13
14     bool private _mintable;
15
16     constructor() public {
17     }
18
19     /**
20      * @dev Throws if called by any account other than the owner.
21      */
22     modifier onlyOwner() {
23         require(_owner == _msgSender(), "Ownable: caller is not the owner");
24         _;
25     }
26
27     /**
28      * @dev sets initials supply and the owner
29      */
30     function initialize(string memory name, string memory symbol, uint8 decimals,
31         address owner,
32         bool mintable) public {
33         _owner = owner;
34         _name = name;
35         _symbol = symbol;
36         _decimals = decimals;
37         _mintable = mintable;
38         _mint(owner, totalSupply());
39     }
40
41     /**
42      * @dev Leaves the contract without owner. It will not be possible to call
43      * `onlyOwner` functions anymore. Can only be called by the current owner.
44      *
45      * NOTE: Renouncing ownership will leave the contract without an owner,
46      * thereby removing any functionality that is only available to the owner.
47      */
48     function renounceOwnership() public onlyOwner {
49         emit OwnershipTransferred(_owner, address(0));
50         _owner = address(0);
51     }
52
53     /**
54      * @dev Transfers ownership of the contract to a new account (`newOwner`).
55      *
56      * NOTE: Renouncing ownership will leave the contract without an owner,
57      * thereby removing any functionality that is only available to the owner.
58      */
59     function transferOwnership(address newOwner) public onlyOwner {
60         require(newOwner != address(0), "Ownable: new owner is the zero address");
61         emit OwnershipTransferred(_owner, newOwner);
62         _owner = newOwner;
63     }
64
65     /**
66      * @dev Creates `amount` tokens and assigns them to `owner`, increasing
67      * total supply by `amount`. If `amount` is 0, it decreases total supply by
68      * `amount`. Can only be called by the owner.
69      */
70     function mint(address owner, uint256 amount) public onlyOwner {
71         require(amount > 0, "Ownable: mint amount must be greater than 0");
72         _totalSupply = _totalSupply + amount;
73         _mint(owner, amount);
74     }
75
76     /**
77      * @dev Burns `amount` tokens, decreasing the total supply by `amount`.
78      * If `amount` is 0, it increases total supply by `amount`. Can only be
79      * called by the owner.
80      */
81     function burn(uint256 amount) public onlyOwner {
82         require(amount > 0, "Ownable: burn amount must be greater than 0");
83         _totalSupply = _totalSupply - amount;
84         _burn(owner, amount);
85     }
86
87     /**
88      * @dev Burns `amount` tokens from `owner`, decreasing the total supply
89      * by `amount`. If `amount` is 0, it increases total supply by `amount`.
90      * Can only be called by the owner.
91      */
92     function burnFrom(address owner, uint256 amount) public onlyOwner {
93         require(owner != address(0), "Ownable: burn from address must not be the zero address");
94         require(amount > 0, "Ownable: burn from amount must be greater than 0");
95         _totalSupply = _totalSupply - amount;
96         _burn(owner, amount);
97     }
98
99     /**
100     * @dev Approves `amount` tokens to be spent by `spender` on behalf of
101     * `owner`, increasing `allowance` by `amount`. Can only be called by the
102     * owner.
103     */
104     function approve(address spender, uint256 amount) public onlyOwner {
105         _approve(owner, spender, amount);
106     }
107
108     /**
109     * @dev Approves `amount` tokens to be spent by `spender` on behalf of
110     * `owner`, decreasing `allowance` by `amount`. Can only be called by the
111     * owner.
112     */
113     function disapprove(address spender, uint256 amount) public onlyOwner {
114         _approve(owner, spender, -amount);
115     }
116
117     /**
118     * @dev Approves `amount` tokens to be spent by `spender` on behalf of
119     * `owner`, setting `allowance` to the given `value`. Can only be called
120     * by the owner.
121     */
122     function setApprovalForAll(address spender, bool value) public onlyOwner {
123         _approve(owner, spender, value ? type(uint256).max : 0);
124     }
125
126     /**
127     * @dev Returns the amount of tokens approved by the owner to be spent
128     * by `spender`.
129     */
130     function allowance(address owner, address spender) public view returns (uint256) {
131         return _allowances[owner][spender];
132     }
133
134     /**
135     * @dev Returns the total supply of tokens in existence.
136     */
137     function totalSupply() public view returns (uint256) {
138         return _totalSupply;
139     }
140
141     /**
142     * @dev Returns the balance of `owner` in terms of tokens.
143     */
144     function balanceOf(address owner) public view returns (uint256) {
145         return _balances[owner];
146     }
147
148     /**
149     * @dev Returns the allowance of `spender` in terms of tokens that is
150     * approved by `owner`.
151     */
152     function allowanceOf(address owner, address spender) public view returns (uint256) {
153         return _allowances[owner][spender];
154     }
155
156     /**
157     * @dev Returns the name of the token.
158     */
159     function name() public view returns (string) {
160         return _name;
161     }
162
163     /**
164     * @dev Returns the symbol of the token.
165     */
166     function symbol() public view returns (string) {
167         return _symbol;
168     }
169
170     /**
171     * @dev Returns the decimals of the token.
172     */
173     function decimals() public view returns (uint8) {
174         return _decimals;
175     }
176
177     /**
178     * @dev Returns whether or not the contract is mintable.
179     */
180     function mintable() public view returns (bool) {
181         return _mintable;
182     }
183
184     /**
185     * @dev Returns the owner of the contract.
186     */
187     function owner() public view returns (address) {
188         return _owner;
189     }
190
191     /**
192     * @dev Returns the previous owner of the contract.
193     */
194     function previousOwner() public view returns (address) {
195         return _previousOwner;
196     }
197
198     /**
199     * @dev Returns the new owner of the contract.
200     */
201     function newOwner() public view returns (address) {
202         return _newOwner;
203     }
204
205     /**
206     * @dev Returns the previous allowance of `owner` to `spender`.
207     */
208     function previousAllowance(address owner, address spender) public view returns (uint256) {
209         return _previousAllowance[owner][spender];
210     }
211
212     /**
213     * @dev Returns the new allowance of `owner` to `spender`.
214     */
215     function newAllowance(address owner, address spender) public view returns (uint256) {
216         return _newAllowance[owner][spender];
217     }
218
219     /**
220     * @dev Returns the previous balance of `owner`.
221     */
222     function previousBalanceOf(address owner) public view returns (uint256) {
223         return _previousBalance[owner];
224     }
225
226     /**
227     * @dev Returns the new balance of `owner`.
228     */
229     function newBalanceOf(address owner) public view returns (uint256) {
230         return _newBalance[owner];
231     }
232
233     /**
234     * @dev Returns the previous total supply.
235     */
236     function previousTotalSupply() public view returns (uint256) {
237         return _previousTotalSupply;
238     }
239
240     /**
241     * @dev Returns the new total supply.
242     */
243     function newTotalSupply() public view returns (uint256) {
244         return _newTotalSupply;
245     }
246
247     /**
248     * @dev Returns the previous allowance of `owner` to `spender`.
249     */
250     function previousAllowanceOf(address owner, address spender) public view returns (uint256) {
251         return _previousAllowanceOf[owner][spender];
252     }
253
254     /**
255     * @dev Returns the new allowance of `owner` to `spender`.
256     */
257     function newAllowanceOf(address owner, address spender) public view returns (uint256) {
258         return _newAllowanceOf[owner][spender];
259     }
260
261     /**
262     * @dev Returns the previous balance of `owner`.
263     */
264     function previousBalanceOf(address owner) public view returns (uint256) {
265         return _previousBalanceOf[owner];
266     }
267
268     /**
269     * @dev Returns the new balance of `owner`.
270     */
271     function newBalanceOf(address owner) public view returns (uint256) {
272         return _newBalanceOf[owner];
273     }
274
275     /**
276     * @dev Returns the previous total supply.
277     */
278     function previousTotalSupply() public view returns (uint256) {
279         return _previousTotalSupplyOf;
280     }
281
282     /**
283     * @dev Returns the new total supply.
284     */
285     function newTotalSupply() public view returns (uint256) {
286         return _newTotalSupplyOf;
287     }
288
289     /**
290     * @dev Returns the previous allowance of `owner` to `spender`.
291     */
292     function previousAllowanceOf(address owner, address spender) public view returns (uint256) {
293         return _previousAllowanceOfOf[owner][spender];
294     }
295
296     /**
297     * @dev Returns the new allowance of `owner` to `spender`.
298     */
299     function newAllowanceOf(address owner, address spender) public view returns (uint256) {
300         return _newAllowanceOfOf[owner][spender];
301     }
302
303     /**
304     * @dev Returns the previous balance of `owner`.
305     */
306     function previousBalanceOf(address owner) public view returns (uint256) {
307         return _previousBalanceOfOf[owner];
308     }
309
310     /**
311     * @dev Returns the new balance of `owner`.
312     */
313     function newBalanceOf(address owner) public view returns (uint256) {
314         return _newBalanceOfOf[owner];
315     }
316
317     /**
318     * @dev Returns the previous total supply.
319     */
320     function previousTotalSupply() public view returns (uint256) {
321         return _previousTotalSupplyOfOf;
322     }
323
324     /**
325     * @dev Returns the new total supply.
326     */
327     function newTotalSupply() public view returns (uint256) {
328         return _newTotalSupplyOfOf;
329     }
330
331     /**
332     * @dev Returns the previous allowance of `owner` to `spender`.
333     */
334     function previousAllowanceOf(address owner, address spender) public view returns (uint256) {
335         return _previousAllowanceOfOfOf[owner][spender];
336     }
337
338     /**
339     * @dev Returns the new allowance of `owner` to `spender`.
340     */
341     function newAllowanceOf(address owner, address spender) public view returns (uint256) {
342         return _newAllowanceOfOfOf[owner][spender];
343     }
344
345     /**
346     * @dev Returns the previous balance of `owner`.
347     */
348     function previousBalanceOf(address owner) public view returns (uint256) {
349         return _previousBalanceOfOfOf[owner];
350     }
351
352     /**
353     * @dev Returns the new balance of `owner`.
354     */
355     function newBalanceOf(address owner) public view returns (uint256) {
356         return _newBalanceOfOfOf[owner];
357     }
358
359     /**
360     * @dev Returns the previous total supply.
361     */
362     function previousTotalSupply() public view returns (uint256) {
363         return _previousTotalSupplyOfOfOf;
364     }
365
366     /**
367     * @dev Returns the new total supply.
368     */
369     function newTotalSupply() public view returns (uint256) {
370         return _newTotalSupplyOfOfOf;
371     }
372
373     /**
374     * @dev Returns the previous allowance of `owner` to `spender`.
375     */
376     function previousAllowanceOf(address owner, address spender) public view returns (uint256) {
377         return _previousAllowanceOfOfOfOf[owner][spender];
378     }
379
380     /**
381     * @dev Returns the new allowance of `owner` to `spender`.
382     */
383     function newAllowanceOf(address owner, address spender) public view returns (uint256) {
384         return _newAllowanceOfOfOfOf[owner][spender];
385     }
386
387     /**
388     * @dev Returns the previous balance of `owner`.
389     */
390     function previousBalanceOf(address owner) public view returns (uint256) {
391         return _previousBalanceOfOfOfOf[owner];
392     }
393
394     /**
395     * @dev Returns the new balance of `owner`.
396     */
397     function newBalanceOf(address owner) public view returns (uint256) {
398         return _newBalanceOfOfOfOf[owner];
399     }
400
401     /**
402     * @dev Returns the previous total supply.
403     */
404     function previousTotalSupply() public view returns (uint256) {
405         return _previousTotalSupplyOfOfOfOf;
406     }
407
408     /**
409     * @dev Returns the new total supply.
410     */
411     function newTotalSupply() public view returns (uint256) {
412         return _newTotalSupplyOfOfOfOf;
413     }
414
415     /**
416     * @dev Returns the previous allowance of `owner` to `spender`.
417     */
418     function previousAllowanceOf(address owner, address spender) public view returns (uint256) {
419         return _previousAllowanceOfOfOfOfOf[owner][spender];
420     }
421
422     /**
423     * @dev Returns the new allowance of `owner` to `spender`.
424     */
425     function newAllowanceOf(address owner, address spender) public view returns (uint256) {
426         return _newAllowanceOfOfOfOfOf[owner][spender];
427     }
428
429     /**
430     * @dev Returns the previous balance of `owner`.
431     */
432     function previousBalanceOf(address owner) public view returns (uint256) {
433         return _previousBalanceOfOfOfOfOf[owner];
434     }
435
436     /**
437     * @dev Returns the new balance of `owner`.
438     */
439     function newBalanceOf(address owner) public view returns (uint256) {
440         return _newBalanceOfOfOfOfOf[owner];
441     }
442
443     /**
444     * @dev Returns the previous total supply.
445     */
446     function previousTotalSupply() public view returns (uint256) {
447         return _previousTotalSupplyOfOfOfOfOf;
448     }
449
450     /**
451     * @dev Returns the new total supply.
452     */
453     function newTotalSupply() public view returns (uint256) {
454         return _newTotalSupplyOfOfOfOfOf;
455     }
456
457     /**
458     * @dev Returns the previous allowance of `owner` to `spender`.
459     */
460     function previousAllowanceOf(address owner, address spender) public view returns (uint256) {
461         return _previousAllowanceOfOfOfOfOfOf[owner][spender];
462     }
463
464     /**
465     * @dev Returns the new allowance of `owner` to `spender`.
466     */
467     function newAllowanceOf(address owner, address spender) public view returns (uint256) {
468         return _newAllowanceOfOfOfOfOfOf[owner][spender];
469     }
470
471     /**
472     * @dev Returns the previous balance of `owner`.
473     */
474     function previousBalanceOf(address owner) public view returns (uint256) {
475         return _previousBalanceOfOfOfOfOfOf[owner];
476     }
477
478     /**
479     * @dev Returns the new balance of `owner`.
480     */
481     function newBalanceOf(address owner) public view returns (uint256) {
482         return _newBalanceOfOfOfOfOfOf[owner];
483     }
484
485     /**
486     * @dev Returns the previous total supply.
487     */
488     function previousTotalSupply() public view returns (uint256) {
489         return _previousTotalSupplyOfOfOfOfOfOf;
490     }
491
492     /**
493     * @dev Returns the new total supply.
494     */
495     function newTotalSupply() public view returns (uint256) {
496         return _newTotalSupplyOfOfOfOfOfOf;
497     }
498
499     /**
500     * @dev Returns the previous allowance of `owner` to `spender`.
501     */
502     function previousAllowanceOf(address owner, address spender) public view returns (uint256) {
503         return _previousAllowanceOfOfOfOfOfOfOf[owner][spender];
504     }
505
506     /**
507     * @dev Returns the new allowance of `owner` to `spender`.
508     */
509     function newAllowanceOf(address owner, address spender) public view returns (uint256) {
510         return _newAllowanceOfOfOfOfOfOfOf[owner][spender];
511     }
512
513     /**
514     * @dev Returns the previous balance of `owner`.
515     */
516     function previousBalanceOf(address owner) public view returns (uint256) {
517         return _previousBalanceOfOfOfOfOfOfOf[owner];
518     }
519
520     /**
521     * @dev Returns the new balance of `owner`.
522     */
523     function newBalanceOf(address owner) public view returns (uint256) {
524         return _newBalanceOfOfOfOfOfOfOf[owner];
525     }
526
527     /**
528     * @dev Returns the previous total supply.
529     */
530     function previousTotalSupply() public view returns (uint256) {
531         return _previousTotalSupplyOfOfOfOfOfOfOf;
532     }
533
534     /**
535     * @dev Returns the new total supply.
536     */
537     function newTotalSupply() public view returns (uint256) {
538         return _newTotalSupplyOfOfOfOfOfOfOf;
539     }
540
541     /**
542     * @dev Returns the previous allowance of `owner` to `spender`.
543     */
544     function previousAllowanceOf(address owner, address spender) public view returns (uint256) {
545         return _previousAllowanceOfOfOfOfOfOfOfOf[owner][spender];
546     }
547
548     /**
549     * @dev Returns the new allowance of `owner` to `spender`.
550     */
551     function newAllowanceOf(address owner, address spender) public view returns (uint256) {
552         return _newAllowanceOfOfOfOfOfOfOfOf[owner][spender];
553     }
554
555     /**
556     * @dev Returns the previous balance of `owner`.
557     */
558     function previousBalanceOf(address owner) public view returns (uint256) {
559         return _previousBalanceOfOfOfOfOfOfOfOf[owner];
560     }
561
562     /**
563     * @dev Returns the new balance of `owner`.
564     */
565     function newBalanceOf(address owner) public view returns (uint256) {
566         return _newBalanceOfOfOfOfOfOfOfOf[owner];
567     }
568
569     /**
570     * @dev Returns the previous total supply.
571     */
572     function previousTotalSupply() public view returns (uint256) {
573         return _previousTotalSupplyOfOfOfOfOfOfOfOf;
574     }
575
576     /**
577     * @dev Returns the new total supply.
578     */
579     function newTotalSupply() public view returns (uint256) {
580         return _newTotalSupplyOfOfOfOfOfOfOfOf;
581     }
582
583     /**
584     * @dev Returns the previous allowance of `owner` to `spender`.
585     */
586     function previousAllowanceOf(address owner, address spender) public view returns (uint256) {
587         return _previousAllowanceOfOfOfOfOfOfOfOfOf[owner][spender];
588     }
589
590     /**
591     * @dev Returns the new allowance of `owner` to `spender`.
592     */
593     function newAllowanceOf(address owner, address spender) public view returns (uint256) {
594         return _newAllowanceOfOfOfOfOfOfOfOfOf[owner][spender];
595     }
596
597     /**
598     * @dev Returns the previous balance of `owner`.
599     */
600     function previousBalanceOf(address owner) public view returns (uint256) {
601         return _previousBalanceOfOfOfOfOfOfOfOfOf[owner];
602     }
603
604     /**
605     * @dev Returns the new balance of `owner`.
606     */
607     function newBalanceOf(address owner) public view returns (uint256) {
608         return _newBalanceOfOfOfOfOfOfOfOfOf[owner];
609     }
610
611     /**
612     * @dev Returns the previous total supply.
613     */
614     function previousTotalSupply() public view returns (uint256) {
615         return _previousTotalSupplyOfOfOfOfOfOfOfOfOf;
616     }
617
618     /**
619     * @dev Returns the new total supply.
620     */
621     function newTotalSupply() public view returns (uint256) {
622         return _newTotalSupplyOfOfOfOfOfOfOfOfOf;
623     }
624
625     /**
626     * @dev Returns the previous allowance of `owner` to `spender`.
627     */
628     function previousAllowanceOf(address owner, address spender) public view returns (uint256) {
629         return _previousAllowanceOfOfOfOfOfOfOfOfOfOf[owner][spender];
630     }
631
632     /**
633     * @dev Returns the new allowance of `owner` to `spender`.
634     */
635     function newAllowanceOf(address owner, address spender) public view returns (uint256) {
636         return _newAllowanceOfOfOfOfOfOfOfOfOfOf[owner][spender];
637     }
638
639     /**
640     * @dev Returns the previous balance of `owner`.
641     */
642     function previousBalanceOf(address owner) public view returns (uint256) {
643         return _previousBalanceOfOfOfOfOfOfOfOfOfOf[owner];
644     }
645
646     /**
647     * @dev Returns the new balance of `owner`.
648     */
649     function newBalanceOf(address owner) public view returns (uint256) {
650         return _newBalanceOfOfOfOfOfOfOfOfOfOf[owner];
651     }
652
653     /**
654     * @dev Returns the previous total supply.
655     */
656     function previousTotalSupply() public view returns (uint256) {
657         return _previousTotalSupplyOfOfOfOfOfOfOfOfOfOf;
658     }
659
660     /**
661     * @dev Returns the new total supply.
662     */
663     function newTotalSupply() public view returns (uint256) {
664         return _newTotalSupplyOfOfOfOfOfOfOfOfOfOf;
665     }
666
667     /**
668     * @dev Returns the previous allowance of `owner` to `spender`.
669     */
670     function previousAllowanceOf(address owner, address spender) public view returns (uint256) {
671         return _previousAllowanceOfOfOfOfOfOfOfOfOfOfOf[owner][spender];
672     }
673
674     /**
675     * @dev Returns the new allowance of `owner` to `spender`.
676     */
677     function newAllowanceOf(address owner, address spender) public view returns (uint256) {
678         return _newAllowanceOfOfOfOfOfOfOfOfOfOfOf[owner][spender];
679     }
680
681     /**
682     * @dev Returns the previous balance of `owner`.
683     */
684     function previousBalanceOf(address owner) public view returns (uint256) {
685         return _previousBalanceOfOfOfOfOfOfOfOfOfOfOf[owner];
686     }
687
688     /**
689     * @dev Returns the new balance of `owner`.
690     */
691     function newBalanceOf(address owner) public view returns (uint256) {
692         return _newBalanceOfOfOfOfOfOfOfOfOfOfOf[owner];
693     }
694
695     /**
696     * @dev Returns the previous total supply.
697     */
698     function previousTotalSupply() public view returns (uint256) {
699         return _previousTotalSupplyOfOfOfOfOfOfOfOfOfOfOf;
700     }
701
702     /**
703     * @dev Returns the new total supply.
704     */
705     function newTotalSupply() public view returns (uint256) {
706         return _newTotalSupplyOfOfOfOfOfOfOfOfOfOfOf;
707     }
708
709     /**
710     * @dev Returns the previous allowance of `owner` to `spender`.
711     */
712     function previousAllowanceOf(address owner, address spender) public view returns (uint256) {
713         return _previousAllowanceOfOfOfOfOfOfOfOfOfOfOfOf[owner][spender];
714     }
715
716     /**
717     * @dev Returns the new allowance of `owner` to `spender`.
718     */
719     function newAllowanceOf(address owner, address spender) public view returns (uint256) {
720         return _newAllowanceOfOfOfOfOfOfOfOfOfOfOfOf[owner][spender];
721     }
722
723     /**
724     * @dev Returns the previous balance of `owner`.
725     */
726     function previousBalanceOf(address owner) public view returns (uint256) {
727         return _previousBalanceOfOfOfOfOfOfOfOfOfOfOfOf[owner];
728     }
729
730     /**
731     * @dev Returns the new balance of `owner`.
732     */
733     function newBalanceOf(address owner) public view returns (uint256) {
734         return _newBalanceOfOfOfOfOfOfOfOfOfOfOfOf[owner];
735     }
736
737     /**
738     * @dev Returns the previous total supply.
739     */
740     function previousTotalSupply() public view returns (uint256) {
741         return _previousTotalSupplyOfOfOfOfOfOfOfOfOfOfOfOf;
742     }
743
744     /**
745     * @dev Returns the new total supply.
746     */
747     function newTotalSupply() public view returns (uint256) {
748         return _newTotalSupplyOfOfOfOfOfOfOfOfOfOfOfOf;
749     }
750
751     /**
752     * @dev Returns the previous allowance of `owner` to `spender`.
753     */
754     function previousAllowanceOf(address owner, address spender) public view returns (uint256) {
755         return _previousAllowanceOfOfOfOfOfOfOfOfOfOfOfOfOf[owner][spender];
756     }
757
758     /**
759     * @dev Returns the new allowance of `owner` to `spender`.
760     */
761     function newAllowanceOf(address owner, address spender) public view returns (uint256) {
762         return _newAllowanceOfOfOfOfOfOfOfOfOfOfOfOfOf[owner][spender];
763     }
764
765     /**
766     * @dev Returns the previous balance of `owner`.
767     */
768     function previousBalanceOf(address owner) public view returns (uint256) {
769         return _previousBalanceOfOfOfOfOfOfOfOfOfOfOfOfOf[owner];
770     }
771
772     /**
773     * @dev Returns the new balance of `owner`.
774     */
775     function newBalanceOf(address owner) public view returns (uint256) {
776         return _newBalanceOfOfOfOfOfOfOfOfOfOfOfOfOf[owner];
777     }
778
779     /**
780     * @dev Returns the previous total supply.
781     */
782     function previousTotalSupply() public view returns (uint256) {
783         return _previousTotalSupplyOfOfOfOfOfOfOfOfOfOfOfOfOf;
784     }
785
786     /**
787     * @dev Returns the new total supply.
788     */
789     function newTotalSupply() public view returns (uint256) {
790         return _newTotalSupplyOfOfOfOfOfOfOfOfOfOfOfOfOf;
791     }
792
793     /**
794     * @dev Returns the previous allowance of `owner` to `spender`.
795     */
796     function previousAllowanceOf(address owner, address spender) public view returns (uint256) {
797         return _previousAllowanceOfOfOfOfOfOfOfOfOfOfOfOfOfOf[owner][spender];
798     }
799
800     /**
801     * @dev Returns the new allowance of `owner` to `spender`.
802     */
803     function newAllowanceOf(address owner, address spender) public view returns (uint256) {
804         return _newAllowanceOfOfOfOfOfOfOfOfOfOfOfOfOfOf[owner][spender];
805     }
806
807     /**
808     * @dev Returns the previous balance of `owner`.
809     */
810     function previousBalanceOf(address owner) public view returns (uint256) {
811         return _previousBalanceOfOfOfOfOfOfOfOfOfOfOfOfOfOf[owner];
812     }
813
814     /**
815     * @dev Returns the new balance of `owner`.
816     */
817     function newBalanceOf(address owner) public view returns (uint256) {
818         return _newBalanceOfOfOfOfOfOfOfOfOfOfOfOfOfOf[owner];
819     }
820
821     /**
822     * @dev Returns the previous total supply.
823     */
824     function previousTotalSupply() public view returns (uint256) {
825         return _previousTotalSupplyOfOfOfOfOfOfOfOfOfOfOfOfOfOf;
826     }
827
828     /**
829     * @dev Returns the new total supply.
830     */
831     function newTotalSupply() public view returns (uint256) {
832         return _newTotalSupplyOfOfOfOfOfOfOfOfOfOfOfOfOfOf;
833     }
834
835     /**
836     * @dev Returns the previous allowance of `owner` to `spender`.
837     */
838     function previousAllowanceOf(address owner, address spender) public view returns (uint256) {
839         return _previousAllowanceOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf[owner][spender];
840     }
841
842     /**
843     * @dev Returns the new allowance of `owner` to `spender`.
844     */
845     function newAllowanceOf(address owner, address spender) public view returns (uint256) {
846         return _newAllowanceOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf[owner][spender];
847     }
848
849     /**
850     * @dev Returns the previous balance of `owner`.
851     */
852     function previousBalanceOf(address owner) public view returns (uint256) {
853         return _previousBalanceOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf[owner];
854     }
855
856     /**
857     * @dev Returns the new balance of `owner`.
858     */
859     function newBalanceOf(address owner) public view returns (uint256) {
860         return _newBalanceOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf[owner];
861     }
862
863     /**
864     * @dev Returns the previous total supply.
865     */
866     function previousTotalSupply() public view returns (uint256) {
867         return _previousTotalSupplyOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf;
868     }
869
870     /**
871     * @dev Returns the new total supply.
872     */
873     function newTotalSupply() public view returns (uint256) {
874         return _newTotalSupplyOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf;
875     }
876
877     /**
878     * @dev Returns the previous allowance of `owner` to `spender`.
879     */
880     function previousAllowanceOf(address owner, address spender) public view returns (uint256) {
881         return _previousAllowanceOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf[owner][spender];
882     }
883
884     /**
885     * @dev Returns the new allowance of `owner` to `spender`.
886     */
887     function newAllowanceOf(address owner, address spender) public view returns (uint256) {
888         return _newAllowanceOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf[owner][spender];
889     }
890
891     /**
892     * @dev Returns the previous balance of `owner`.
893     */
894     function previousBalanceOf(address owner) public view returns (uint256) {
895         return _previousBalanceOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf[owner];
896     }
897
898     /**
899     * @dev Returns the new balance of `owner`.
900     */
901     function newBalanceOf(address owner) public view returns (uint256) {
902         return _newBalanceOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf[owner];
903     }
904
905     /**
906     * @dev Returns the previous total supply.
907     */
908     function previousTotalSupply() public view returns (uint256) {
909         return _previousTotalSupplyOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf;
910     }
911
912     /**
913     * @dev Returns the new total supply.
914     */
915     function newTotalSupply() public view returns (uint256) {
916         return _newTotalSupplyOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf;
917     }
918
919     /**
920     * @dev Returns the previous allowance of `owner` to `spender`.
921     */
922     function previousAllowanceOf(address owner, address spender) public view returns (uint256) {
923         return _previousAllowanceOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf[owner][spender];
924     }
925
926     /**
927     * @dev Returns the new allowance of `owner` to `spender`.
928     */
929     function newAllowanceOf(address owner, address spender) public view returns (uint256) {
930         return _newAllowanceOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf[owner][spender];
931     }
932
933     /**
934     * @dev Returns the previous balance of `owner`.
935     */
936     function previousBalanceOf(address owner) public view returns (uint256) {
937         return _previousBalanceOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf[owner];
938     }
939
940     /**
941     * @dev Returns the new balance of `owner`.
942     */
943     function newBalanceOf(address owner) public view returns (uint256) {
944         return _newBalanceOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf[owner];
945     }
946
947     /**
948     * @dev Returns the previous total supply.
949     */
950     function previousTotalSupply() public view returns (uint256) {
951         return _previousTotalSupplyOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf;
952     }
953
954     /**
955     * @dev Returns the new total supply.
956     */
957     function newTotalSupply() public view returns (uint256) {
958         return _newTotalSupplyOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf;
959     }
960
961     /**
962     * @dev Returns the previous allowance of `owner` to `spender`.
963     */
964     function previousAllowanceOf(address owner, address spender) public view returns (uint256) {
965         return _previousAllowanceOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf[owner][spender];
966     }
967
968     /**
969     * @dev Returns the new allowance of `owner` to `spender`.
970     */
971     function newAllowanceOf(address owner, address spender) public view returns (uint256) {
972         return _newAllowanceOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf[owner][spender];
973     }
974
975     /**
976     * @dev Returns the previous balance of `owner`.
977     */
978     function previousBalanceOf(address owner) public view returns (uint256) {
979         return _previousBalanceOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf[owner];
980     }
981
982     /**
983     * @dev Returns the new balance of `owner`.
984     */
985     function newBalanceOf(address owner) public view returns (uint256) {
986         return _newBalanceOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf[owner];
987     }
988
989     /**
990     * @dev Returns the previous total supply.
991     */
992     function previousTotalSupply() public view returns (uint256) {
993         return _previousTotalSupplyOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf;
994     }
995
996     /**
997     * @dev Returns the new total supply.
998     */
999     function newTotalSupply() public view returns (uint256) {
1000         return _newTotalSupplyOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf;
1001     }
1002
1003     /**
1004     * @dev Returns the previous allowance of `owner` to `spender`.
1005     */
1006     function previousAllowanceOf(address owner, address spender) public view returns (uint256) {
1007         return _previousAllowanceOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf[owner][spender];
1008     }
1009
1010     /**
1011     * @dev Returns the new allowance of `owner` to `spender`.
1012     */
1013     function newAllowanceOf(address owner, address spender) public view returns (uint256) {
1014         return _newAllowanceOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf[owner][spender];
1015     }
1016
1017     /**
1018     * @dev Returns the previous balance of `owner`.
1019     */
1020     function previousBalanceOf(address owner) public view returns (uint256) {
1021         return _previousBalanceOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf[owner];
1022     }
1023
1024     /**
1025     * @dev Returns the new balance of `owner`.
1026     */
1027     function newBalanceOf(address owner) public view returns (uint256) {
1028         return _newBalanceOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf[owner];
1029     }
1030
1031     /**
1032     * @dev Returns the previous total supply.
1033     */
1034     function previousTotalSupply() public view returns (uint256) {
1035         return _previousTotalSupplyOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf;
1036     }
1037
1038     /**
1039     * @dev Returns the new total supply.
1040     */
1041     function newTotalSupply() public view returns (uint256) {
1042         return _newTotalSupplyOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOfOf;
1043     }
1044
1045     /**
1046     * @dev Returns the previous allowance of `owner` to `spender`.
1047     */
1048     function previousAllowanceOf(address owner, address spender) public view returns (uint256) {
1049         return _previousAllowanceOf
```



```

53     * Can only be called by the current owner.
54     */
55     function transferOwnership(address newOwner) public onlyOwner {
56         require(newOwner != address(0), "Ownable: new owner is the zero address");
57         emit OwnershipTransferred(_owner, newOwner);
58         _owner = newOwner;
59     }
60
61     /**
62     * @dev Returns if the token is mintable or not
63     */
64     function mintable() external view returns (bool) {
65         return _mintable;
66     }
67
68     /**
69     * @dev Returns the bep token owner.
70     */
71     function getOwner() external override view returns (address) {
72         return _owner;
73     }
74
75     /**
76     * @dev Returns the token decimals.
77     */
78     function decimals() external override view returns (uint8) {
79         return _decimals;
80     }
81
82     /**
83     * @dev Returns the token symbol.
84     */
85     function symbol() external override view returns (string memory) {
86         return _symbol;
87     }
88
89     /**
90     * @dev Returns the token name.
91     */
92     function name() external override view returns (string memory) {
93         return _name;
94     }
95
96     /**
97     * @dev See {BEP20-totalSupply}.
98     */
99     function totalSupply() external override view returns (uint256) {
100         return _totalSupply;
101     }
102
103     /**
104     * @dev See {BEP20-balanceOf}.
105     */

```

```

106     function balanceOf(address account) external override view returns (uint256)
107         return _balances[account];
108     }
109
110     /**
111     * @dev See {BEP20-transfer}.
112     *
113     * Requirements:
114     *
115     * - `recipient` cannot be the zero address.
116     * - the caller must have a balance of at least `amount`.
117     */
118     function transfer(address recipient, uint256 amount) external override returns (bool)
119         _transfer(_msgSender(), recipient, amount);
120     }
121
122     /**
123     * @dev See {BEP20-allowance}.
124     */
125     function allowance(address owner, address spender) external override view returns (uint256)
126         return _allowances[owner][spender];
127     }
128
129     /**
130     * @dev See {BEP20-approve}.
131     *
132     * Requirements:
133     *
134     * - `spender` cannot be the zero address.
135     */
136     function approve(address spender, uint256 amount) external override returns (bool)
137         _approve(_msgSender(), spender, amount);
138     }
139
140     /**
141     * @dev See {BEP20-transferFrom}.
142     *
143     * Emits an {Approval} event indicating the updated allowance. This is not
144     * required by the EIP. See the note at the beginning of {BEP20};
145     *
146     * Requirements:
147     *
148     * - `sender` and `recipient` cannot be the zero address.
149     * - `sender` must have a balance of at least `amount`.
150     * - the caller must have allowance for `sender`'s tokens of at least
151     *   `amount`.
152     */
153     function transferFrom(address sender, address recipient, uint256 amount) external override
154         _transfer(sender, recipient, amount);
155         _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount,
156         return true;
157     }
158
159

```

```

159
160 /**
161  * @dev Atomically increases the allowance granted to `spender` by the caller.
162  *
163  * This is an alternative to {approve} that can be used as a mitigation for
164  * problems described in {BEP20-approve}.
165  *
166  * Emits an {Approval} event indicating the updated allowance.
167  *
168  * Requirements:
169  *
170  * - `spender` cannot be the zero address.
171  */
172 function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
173     _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
174     return true;
175 }
176
177 /**
178  * @dev Atomically decreases the allowance granted to `spender` by the caller.
179  *
180  * This is an alternative to {approve} that can be used as a mitigation for
181  * problems described in {BEP20-approve}.
182  *
183  * Emits an {Approval} event indicating the updated allowance.
184  *
185  * Requirements:
186  *
187  * - `spender` cannot be the zero address.
188  * - `spender` must have allowance for the caller of at least
189  *   `subtractedValue`.
190  */
191 function decreaseAllowance(address spender, uint256 subtractedValue) public returns (bool) {
192     _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue));
193     return true;
194 }
195
196 /**
197  * @dev Creates `amount` tokens and assigns them to `msg.sender`, increasing
198  * the total supply.
199  *
200  * Requirements
201  *
202  * - `msg.sender` must be the token owner
203  * - `_mintable` must be true
204  */
205 function mint(uint256 amount) public onlyOwner returns (bool) {
206     require(_mintable, "this token is not mintable");
207     _mint(_msgSender(), amount);
208     return true;
209 }
210
211 /**
212  * @dev Burns `amount` tokens and decreasing the total supply

```

```

212     @dev Burn `amount` tokens and decreasing the total supply.
213 */
214 function burn(uint256 amount) public returns (bool) {
215     _burn(_msgSender(), amount);
216     return true;
217 }
218
219 /**
220  * @dev Moves tokens `amount` from `sender` to `recipient`.
221  *
222  * This is internal function is equivalent to {transfer}, and can be used to
223  * e.g. implement automatic token fees, slashing mechanisms, etc.
224  *
225  * Emits a {Transfer} event.
226  *
227  * Requirements:
228  *
229  * - `sender` cannot be the zero address.
230  * - `recipient` cannot be the zero address.
231  * - `sender` must have a balance of at least `amount`.
232  */
233 function _transfer(address sender, address recipient, uint256 amount) internal {
234     require(sender != address(0), "BEP20: transfer from the zero address");
235     require(recipient != address(0), "BEP20: transfer to the zero address");
236
237     _balances[sender] = _balances[sender].sub(amount, "BEP20: transfer amount exceeds balance");
238     _balances[recipient] = _balances[recipient].add(amount);
239     emit Transfer(sender, recipient, amount);
240 }
241
242 /** @dev Creates `amount` tokens and assigns them to `account`, increasing
243  * the total supply.
244  *
245  * Emits a {Transfer} event with `from` set to the zero address.
246  *
247  * Requirements
248  *
249  * - `to` cannot be the zero address.
250  */
251 function _mint(address account, uint256 amount) internal {
252     require(account != address(0), "BEP20: mint to the zero address");
253
254     _totalSupply = _totalSupply.add(amount);
255     _balances[account] = _balances[account].add(amount);
256     emit Transfer(address(0), account, amount);
257 }
258
259 /**
260  * @dev Destroys `amount` tokens from `account`, reducing the
261  * total supply.
262  *
263  * Emits a {Transfer} event with `to` set to the zero address.
264  *
265  * Requirements

```

```

265     requirements
266     *
267     * - `account` cannot be the zero address.
268     * - `account` must have at least `amount` tokens.
269     */
270     function _burn(address account, uint256 amount) internal {
271         require(account != address(0), "BEP20: burn from the zero address");
272
273         _balances[account] = _balances[account].sub(amount, "BEP20: burn amount");
274         _totalSupply = _totalSupply.sub(amount);
275         emit Transfer(account, address(0), amount);
276     }
277
278     /**
279     * @dev Sets `amount` as the allowance of `spender` over the `owner`'s tokens.
280     *
281     * This is internal function is equivalent to `approve`, and can be used to
282     * e.g. set automatic allowances for certain subsystems, etc.
283     *
284     * Emits an {Approval} event.
285     *
286     * Requirements:
287     *
288     * - `owner` cannot be the zero address.
289     * - `spender` cannot be the zero address.
290     */
291     function _approve(address owner, address spender, uint256 amount) internal {
292         require(owner != address(0), "BEP20: approve from the zero address");
293         require(spender != address(0), "BEP20: approve to the zero address");
294
295         _allowances[owner][spender] = amount;
296         emit Approval(owner, spender, amount);
297     }
298
299     /**
300     * @dev Destroys `amount` tokens from `account`. `amount` is then deducted
301     * from the caller's allowance.
302     *
303     * See {_burn} and {_approve}.
304     */
305     function _burnFrom(address account, uint256 amount) internal {
306         _burn(account, amount);
307         _approve(account, _msgSender(), _allowances[account][_msgSender()].sub(amount, "BEP20: burn from allowance"));
308     }
309 }

```

GST Token Contract

STEPN team implemented a solidity version of GST token, which uses Openzeppelin's ERC20 standard libraries:

```

1  // contracts/GreenSatoshiToken.sol
2  // SPDX-License-Identifier: MIT
3  pragma solidity ^0.8.0;
4
5  import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
6  import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol";
7  import "@openzeppelin/contracts/access/Ownable.sol";
8
9  /**
10   * Green Satoshi Token
11   * @author STEP_N
12   */
13  contract GreenSatoshiToken is ERC20, ERC20Burnable, Ownable {
14      constructor() ERC20("GreenSatoshiToken", "GST") {}
15
16      function mint(address to, uint256 amount) public onlyOwner {
17          _mint(to, amount);
18      }
19
20      function decimals() public view virtual override returns (uint8) {
21          return 8;
22      }
23  }
24

```

STEPNNFT nft contract

```

1  // contracts/STEPNNFT.sol
2  // SPDX-License-Identifier: MIT
3  pragma solidity ^0.8.0;
4
5  import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
6  import "@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol";
7  import "@openzeppelin/contracts/access/Ownable.sol";
8
9  /**
10   * STEPNNFTs
11   * @author STEPNNFT
12   */
13  contract STEPNNFT is ERC721Enumerable, Ownable {
14      // base uri for nfts
15      string private _baseURI;
16
17      constructor() ERC721("STEPNNFT", "SNFT") {}
18
19      function _baseURI() internal view override returns (string memory) {
20          return _baseURI;
21      }
22
23      function setBaseURI(string memory baseURI) public onlyOwner {
24          require(bytes(baseURI).length > 0, "wrong base uri");
25          _baseURI = baseURI;
26      }
27
28      function mint(address to, uint256 tokenId) public onlyOwner {
29          _safeMint(to, tokenId);
30      }
31
32      function burn(uint256 tokenId) public virtual {
33          require(
34              _isApprovedOrOwner(_msgSender(), tokenId),
35              "burn caller is not owner nor approved"
36          );
37          _burn(tokenId);
38      }
39  }

```

Disclaimer

Verilog receives compensation from one or more clients for performing the smart contract and auditing analysis contained in these reports. The report created is solely for Clients and published with their consent. As such, the scope of our audit is limited to a review of code and only the code we note as being within the scope of our audit detailed in this report. It is important to note that the Solidity code itself presents unique and unquantifiable risks since

the Solidity language itself remains under current development and is subject to unknown risks and flaws. Our sole goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies. Thus, Verilog in no way claims any guarantee of security or functionality of the technology we agree to analyze.

In addition, Verilog reports do not provide any indication of the technologies proprietors, business, business model or legal compliance. As such, reports do not provide investment advice and should not be used to make decisions about investment or involvement with any particular project. Verilog has the right to distribute the Report through other means, including via Verilog publications and other distributions. Verilog makes the reports available to parties other than the Clients (i.e., "third parties") – on its website in hopes that it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.