tags: `Final Report`

STEPN Audit

April 4, 2022
by **Verilog Audit**

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:

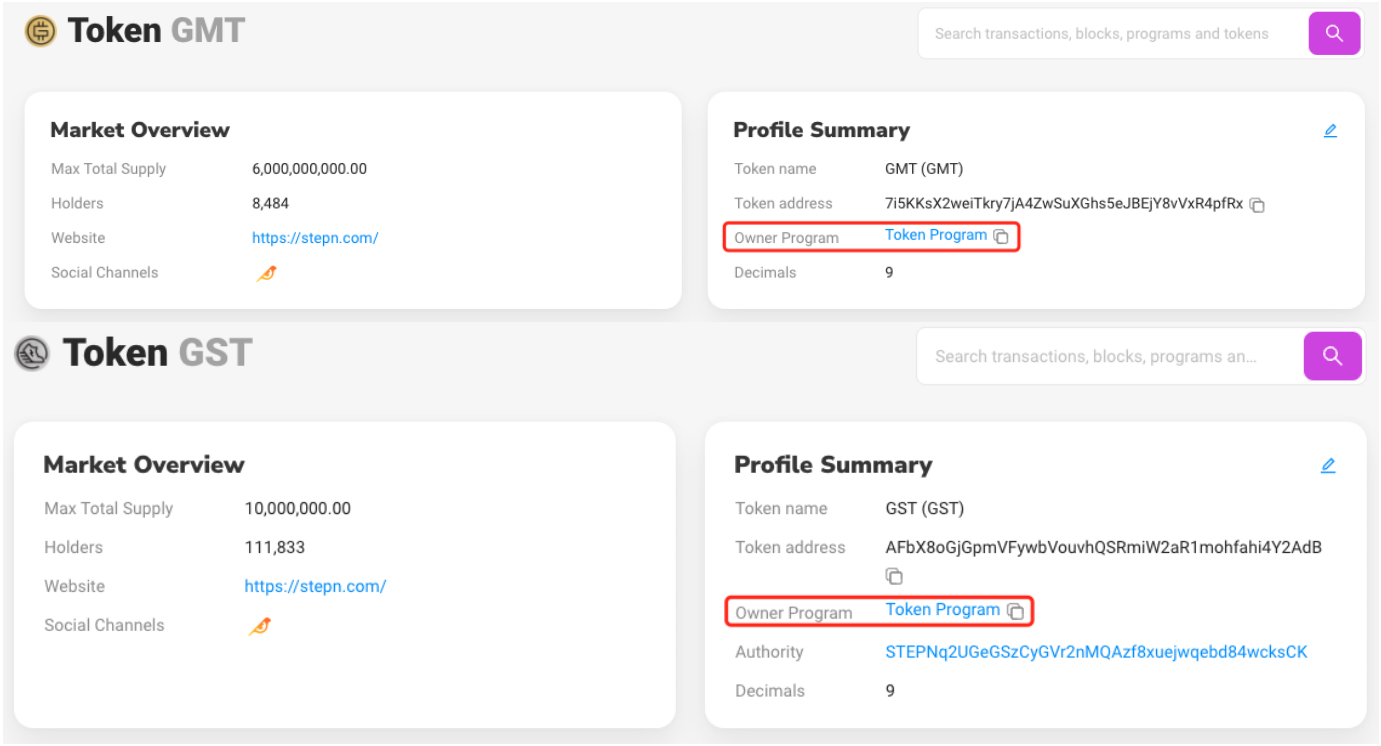
Network	Token	Token Address
Solana	GMT	7i5KKsX2weiTkry7jA4ZwSuXGhs5eJBEjY8vVxR4pfRx (https://solscan.io/token/7i5KKsX2weiTkry7jA4ZwSuXGhs5eJBEjY8vVxR4pfRx)
Solana	GST	AFbX8oGjGpmVFywbVouvhQSRmiW2aR1mohfahi4Y2AdB (https://solscan.io/token/AFbX8oGjGpmVFywbVouvhQSRmiW2aR1mohfahi4Y2AdB)
BNB	GMT	0x3019BF2a2eF8040C242C9a4c5c4BD4C81678b2A1 (https://bscscan.com/address/0x3019BF2a2eF8040C242C9a4c5c4BD4C81678b2A1)
BNB	GST	n/a (coming soon)

Deployment on Solana

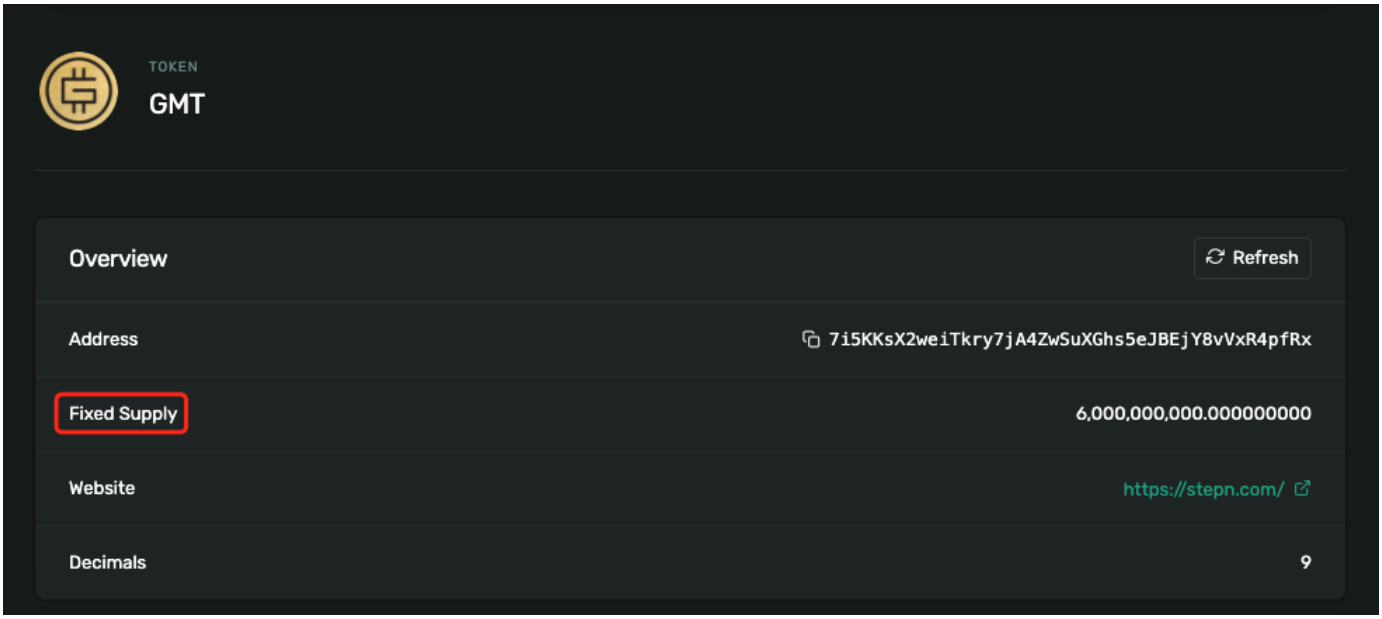
STEPN dev team used Solana Token Program to create both **GMT** & **GST** tokens. Token Program defines a common implementation for Fungible and Non Fungible tokens.

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:



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.



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).

Below is the summary of GMT token on BNB Chain:

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

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

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**

```
function setBaseURI(string memory buri) public onlyOwner {
    require(bytes(buri).length > 0, "wrong base uri");
    _buri = buri;
}

function mint(address to, uint256 tokenId) public onlyOwner {
    _safeMint(to, tokenId);
}
```

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

```
contract BEP20TokenImplementation is Context, IBEP20, Initializable {
    using SafeMath for uint256;

    mapping (address => uint256) private _balances;
    mapping (address => mapping (address => uint256)) private _allowances;
    uint256 private _totalSupply;
    string private _name;
    string private _symbol;
    uint8 private _decimals;

    address private _owner;
    event OwnershipTransferred(address indexed previousOwner, address indexed n

    bool private _mintable;

    constructor() public {
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(_owner == _msgSender(), "Ownable: caller is not the owner");
        _;
    }

    /**
     * @dev sets initials supply and the owner
     */
    function initialize(string memory name, string memory symbol, uint8 decimal
        _owner = owner;
        _name = name;
        _symbol = symbol;
        _decimals = decimals;
        _mintable = mintable;
        _mint(owner, amount);
    }

    /**
    * @dev Leaves the contract without owner. It will not be possible to call
    * `onlyOwner` functions anymore. Can only be called by the current owner.
    *
    * NOTE: Renouncing ownership will leave the contract without an owner,
    * thereby removing any functionality that is only available to the owner.
    */
    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
```

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

```
159
160         /**
161          * @dev Atomically increases the allowance granted to `spender` by the call
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 retu
173             _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(a
174             return true;
175         }
176
177         /**
178          * @dev Atomically decreases the allowance granted to `spender` by the call
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
192             _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(s
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 Burn `amount` tokens and decreasing the total supply.
```

```
     * @dev Burn `amount` tokens and decreasing the total supply.
     */
    function burn(uint256 amount) public returns (bool) {
        _burn(_msgSender(), amount);
        return true;
    }

    /**
     * @dev Moves tokens `amount` from `sender` to `recipient`.
     *
     * This is internal function is equivalent to {transfer}, and can be used t
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
     * - `sender` cannot be the zero address.
     * - `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     */
    function _transfer(address sender, address recipient, uint256 amount) inter
        require(sender != address(0), "BEP20: transfer from the zero address");
        require(recipient != address(0), "BEP20: transfer to the zero address")

        _balances[sender] = _balances[sender].sub(amount, "BEP20: transfer amou
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }

    /** @dev Creates `amount` tokens and assigns them to `account`, increasing
     * the total supply.
     *
     * Emits a {Transfer} event with `from` set to the zero address.
     *
     * Requirements
     *
     * - `to` cannot be the zero address.
     */
    function _mint(address account, uint256 amount) internal {
        require(account != address(0), "BEP20: mint to the zero address");

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`, reducing the
     * total supply.
     *
     * Emits a {Transfer} event with `to` set to the zero address.
     *
     * Requirements
```

```
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
    function _burn(address account, uint256 amount) internal {
        require(account != address(0), "BEP20: burn from the zero address");

        _balances[account] = _balances[account].sub(amount, "BEP20: burn amount
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner`s token
     *
     * This is internal function is equivalent to `approve`, and can be used to
     * e.g. set automatic allowances for certain subsystems, etc.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     */
    function _approve(address owner, address spender, uint256 amount) internal
        require(owner != address(0), "BEP20: approve from the zero address");
        require(spender != address(0), "BEP20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`.`amount` is then deducted
     * from the caller's allowance.
     *
     * See {_burn} and {_approve}.
     */
    function _burnFrom(address account, uint256 amount) internal {
        _burn(account, amount);
        _approve(account, _msgSender(), _allowances[account][_msgSender()].sub(
    }
}
```

GST Token Contract

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

```
// contracts/GreenSatoshiToken.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

/**
 * Green Satoshi Token
 * @author STEPN
 */
contract GreenSatoshiToken is ERC20, ERC20Burnable, Ownable {
    constructor() ERC20("GreenSatoshiToken", "GST") {}

    function mint(address to, uint256 amount) public onlyOwner {
        _mint(to, amount);
    }

    function decimals() public view virtual override returns (uint8) {
        return 8;
    }
}

```

STEPNNFT nft contract

```sol
// contracts/STEPNNFT.sol
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "@openzeppelin/contracts/token/ERC721/ERC721.sol";
import "@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

/**
 * STEPN NFTs
 * @author STEPN
 */
contract STEPNNFT is ERC721Enumerable, Ownable {
    // base uri for nfts
    string private _buri;

    constructor() ERC721("STEPNNFT", "SNFT") {}

    function _baseURI() internal view override returns (string memory) {
        return _buri;
    }

    function setBaseURI(string memory buri) public onlyOwner {
        require(bytes(buri).length > 0, "wrong base uri");
        _buri = buri;
    }

    function mint(address to, uint256 tokenId) public onlyOwner {
        _safeMint(to, tokenId);
    }

    function burn(uint256 tokenId) public virtual {
        require(
            _isApprovedOrOwner(_msgSender(), tokenId),
            "burn caller is not owner nor approved"
        );
        _burn(tokenId);
    }
}
```

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.