Sumati Token

AUDIT REPORT

Version 1.0.0

Serial No. 20220411000220016

Presented by Fairyproof

April 11, 2022

01. Introduction

This document includes the results of the audit performed by the Fairyproof team on the Sumati token.

Audit Start Time:

April 7, 2022

Audit End Time:

April 10, 2022

Token Standards:

ERC-20, ERC-721

Project Tokens' Names:

Sumati BOSO, SARIRA

Token Symbols:

BOSO, SARIRA

Token Precisions:

18, 1

Audited Source Files:

The source files audited include all the files with the extension "sol" as follows:

```
./
├── BOSO.sol
└── sumati.sol

directory: 1 file: 2  lines: 1280
```

The goal of this audit is to review Sumati's solidity implementation for its token issuance functions, study potential security vulnerabilities, its general design and architecture, and uncover bugs that could compromise the software in production.

We make observations on specific areas of the code that present concrete problems, as well as general observations that traverse the entire codebase horizontally, which could improve its quality as a whole.

This audit only applies to the specified code, software or any materials supplied by the Sumati team for specified versions. Whenever the code, software, materials, settings, environment etc is changed, the comments of this audit will no longer apply.

— Disclaimer

Note that as of the date of publishing, the contents of this report reflect the current understanding of known security patterns and state of the art regarding system security. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. If the audited source files are smart contract files, risks or issues introduced by using data feeds from off-chain sources are not extended by this review either.

Given the size of the project, the findings detailed here are not to be considered exhaustive, and further testing and audit is recommended after the issues covered are fixed.

To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

— Methodology

The above files' code was studied in detail in order to acquire a clear impression of how the its specifications were implemented. The codebase was then subject to deep analysis and scrutiny, resulting in a series of observations. The problems and their potential solutions are discussed in this document and, whenever possible, we identify common sources for such problems and comment on them as well.

The Fairyproof auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Fairyproof to make sure we understand the size, scope, and functionality of the project's source code.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Fairyproof describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run the test cases.

ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the source code to improve maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

— Structure of the document

This report contains a list of issues and comments on all the above source files. Each issue is assigned a severity level based on the potential impact of the issue and recommendations to fix it, if applicable. For ease of navigation, an index by topic and another by severity are both provided at the beginning of the report.

— Documentation

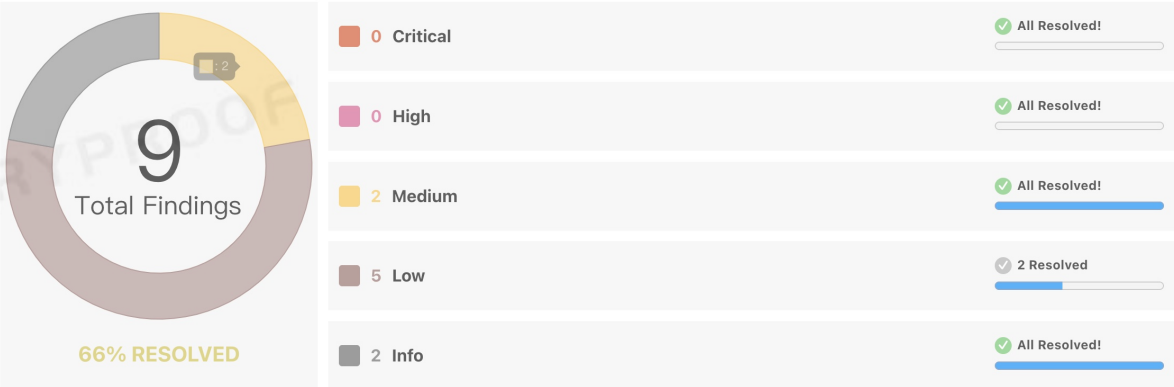
For this audit, we used the following sources of truth about how the token issuance functions should work:

Smart Contract Source Files

These were considered the specification, and when discrepancies arose with the actual code behavior, we consulted with the Sumati team or reported an issue.

— Comments from Auditor

Serial Number	Auditor	Audit Time	Resule
2022041100022016	Fairyproof Security Team	2022.04.07 - 2022.04.10	Low



Summary:

The Fairyproof security team used its auto analysis tools and manual work to audit the project. During the audit, 2 risks of medium-severity, 5 risks of low-severity and 2 risks of informational-severity were discovered. 2 risks of medium-severity, 2 risks of low-severity and 2 risks of informational-severity were fixed and 3 risks of low-severity were confirmed.

02. About Fairyproof

Fairyproof is a leading technology firm in the blockchain industry, providing consulting and security audits for organizations. Fairyproof has developed industry security standards for designing and deploying blockchain applications.

03. Major functions of audited code

The audited code mainly implements the following functions:

Issuance of ERC-20 Token

Token Name: Sumati BOSO

Token Symbol: BOSO

Token Precision: 18

Total Supply: 21,000,000,000

Mint/Burn: No additional mintage, no token burn

Transfer Pause/Freeze: Transfer cannot be paused or frozen.

Burn on Transaction: No

Issuance of ERC-721 Token

Token Name: SARIRA

Token Symbol: SARIRA

Total Supply: Uncapped

Mint/Burn: tokens can be minted additionally, no token burn

Transfer Pause/Freeze: Transfer cannot be paused or frozen.

Burn on Transaction: No

Misc: a callback function is defined and will be called when token transfer happens. When this token interacts with other applications, this callback function may introduce a re-entrancy risk.

Staking of ERC-721 Token

Users can stake the ERC-721 token to get rewards in the ERC-20 token.

04. Coverage of issues

The issues that the Fairyproof team covered when conducting the audit include but are not limited to the following ones:

- Re-entrancy Attack
- Replay Attack
- Reordering Attack
- Miner's Advantage
- Rollback Attack
- DDos Attack
- Transaction Ordering Attack
- Race Condition
- Access Control
- Integer Overflow/Underflow
- Timestamp Attack
- Gas Consumption
- Inappropriate Callback Function
- Function Visibility
- Implementation Vulnerability
- Uninitialized Storage Pointer
- Arithmetic Precision
- Tx.origin
- Fake Deposit
- Shadow Variable
- Parameter Check
- Design Vulnerability
- Token Issuance
- Admin Rights
- Inappropriate Proxy Design
- Inappropriate Use of Slots
- Asset Security
- Contract Upgrade/Migration
- Code Improvement
- Misc

05. Severity level reference

Every issue in this report was assigned a severity level from the following:

Critical severity issues need to be fixed as soon as possible.

High severity issues will probably bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Informational is not an issue or risk but a suggestion for code improvement.

06. Major areas that need attention

Based on the provided source code the Fairyproof team focused on the possible issues and risks related to the following functions or areas.

- Integer Overflow/Underflow

We checked all the code sections, which had arithmetic operations and might introduce integer overflow or underflow if no safe libraries were used. All of them used safe libraries.

We didn't find issues or risks in these functions or areas at the time of writing.

- Access Control

We checked each of the functions that could modify a state, especially those functions that could only be accessed by "owner".

We didn't find issues or risks in these functions or areas at the time of writing.

- Token Issuance

We checked whether or not the contract files could mint tokens at will.

We found an issue. In `Sumati.sol`, the admin could mint the ERC-721 token at will. Please refer to【FP-4】in "08. Issue descriptions" for more details.

- State Update

We checked some key state variables which should only be set at initialization.

We found two issues. In `Sumati.sol`, if the admin changed the `SIGNER` it would cause all the previous signatures invalid. Please refer to【FP-1】in "08. Issue descriptions" for more details. Additionally in `Sumati.sol` some addresses were not validated. Please refer to【FP-6】in "08. Issue descriptions" for more details.

- Asset Security

We checked whether or not all the functions that transfer assets were safely handled.

We didn't find issues or risks in these functions or areas at the time of writing.

- Code Improvement

We checked whether or not there was code that could be improved.

In both `Sumati.sol` and `BOSO.sol`, `SafeMath` was not used and could be removed. Please refer to【FP-8】in "08. Issue descriptions" for more details. In `Sumati.sol` the `redeem` function had some redundant `require` statements. Please refer to【FP-9】in "08. Issue descriptions" for more details.

- Design Vulnerabilities

We checked whether or not there were design vulnerabilities.

In the `Sumati.sol` contract file, the `mintBat` function had two conflict `require` statements. Please refer to【FP-5】in "08. Issue descriptions" for more details.

In the `Sumati.sol` contract file, the `redeem` function had issues in conditional checking. Please refer to【FP-7】in "08. Issue descriptions" for more details.

In `Sumati.sol` some transactions needed to be executed in a specific order. Please refer to【FP-2】in "08. Issue descriptions" for more details.

- Centralized Logic

We checked whether or not some implementation had too much centralized logic.

In `Sumati.sol` , the setting of the mint price of the ERC-721 token and the algorithm of the staking reward were handled by centralized logic. Please refer to【FP-3】in "08. Issue descriptions" for more details.

- Miscellaneous

Note: a callback function is defined in the ERC-721 token and will be called when token transfer happens. When the ERC-721 token interacts with other applications, this callback function may introduce a re-entrancy risk.

Furthermore, in order for a user to stake or withdraw the ERC-721 token by calling a function, the `onERC721Received` function in `IERC721Receiver` must be implemented otherwise the staked ERC-721 token wouldn't be withdrawn.

07. List of issues by severity

Index	Title	Issue/Risk	Severity	Status
FP-1	Inappropriate Change of SIGNER	Design Vulnerabilities	Low	Confirmed
FP-2	Inappropriate Reliance On Timestamps To Verify Signatures	Transaction Ordering Attack	Low	Confirmed
FP-3	Excessive Centralized Logic	Design Vulnerabilities	Low	Confirmed
FP-4	Minting of NFTs At Will	Token Issuance	Medium	✓ Fixed
FP-5	Conflict `require` Statements	Implementation Vulnerabilities	Medium	✓ Fixed
FP-6	Missing Zero-Address Check	Parameter Check	Low	✓ Fixed
FP-7	Incorrect Verification	Design Vulnerabilities	Low	✓ Fixed
FP-8	Unused Libraries	Code Improvement	Informational	✓ Fixed
FP-9	Redundant `require` Statements	Code Improvement	Informational	✓ Fixed

08. Issue descriptions

[FP-1] Inappropriate Change of SIGNER

Low

Confirmed

Issue/Risk: Design Vulnerabilities

Description:

In `Sumati.sol`, if the admin changed the `SIGNER` it would cause all the previous signatures invalid. Operations of `mint`, `redeem` and `bosoClaim` would be affected.

Recommendation:

Consider not changing `SIGNER`. If a change is needed, it must be publicly announced to users.

Update:

The Sumati team replied that allowing change of `SIGNER` was to prevent compromise of signatures. The team would change `SIGNER` with great care and publicly announce the change to users.

Status:

It has been confirmed by the Sumati team.

[FP-2] Inappropriate Reliance On Timestamps To Verify Signatures

Low

Confirmed

Issue/Risk: Transaction Ordering Attack

Description:

In `Sumati.sol`, the `mint`, `mintBat`, `redeem` and `bosoClaim` functions all used timestamps to verify whether or not a signature was already used. If a user used a more recent signature in an earlier transaction, it would cause an earlier signature invalid.

```
//mint
require(timeMint[to] < timestamp, "timestamp error");
//mintBat
require(timeBat < timestamp, "timestamp error");
//redeem
require(timeRedeem[to] < timestamp, "timestamp error");
//bosoClaim
require(timeClaim[to] < timestamp, "tokenId not stake");
```

Recommendation:

Consider using parameter hash values to verify signatures as follows:

```
mapping(bytes32 => bool) public mintHashes;
```

```
  function mint(uint256 tokenAmount, address to, uint256 timestamp, bytes memory
sig) public nonReentrant() {
      require(saleIsActive, "Sale must be active to mint ");
      // require(timeMint[to] < timestamp, "timestamp error");
      require(block.timestamp <= timestamp + 6 hours);
      // require((timestamp-timeMint[to]) < 20000000000000, "timestamp error2");
      bytes32 hash = _mintCommon(tokenAmount, to, timestamp, sig);
      require(!mintHashes[hash],"has minted");
      mintHashes[hash] = true;
  }

  function _mintCommon(uint256 tokenAmount, address to, uint256 timestamp, bytes
memory sig) internal returns(bytes32 ) {

      bytes32 hash = keccak256(abi.encode(tokenAmount, to, timestamp));
      require(
          hash.toEthSignedMessageHash().recover(sig) == SIGNER,
          "PxG: Invalid signature"
      );

      if (tokenAmount > 0) {
          assert(IERC20(tokenAddr).transferFrom(msg.sender, dev, tokenAmount));
      }

      _safeMint(to, start);
      nftMap[start] = tokenAmount;
      start = start + 1;
      // timeMint[to] = timestamp;
      return hash;
  }
```

Update:

The Sumati team replied that this is required by the business logic. If there are conflicts, users need to contact the Sumati team to get new signatures

Status:

It has been confirmed by the Sumati team.

[FP-3] Excessive Centralized Logic

Low Confirmed

Issue/Risk: Design Vulnerabilities

Description:

In `Sumati.sol` , the setting of the mint price of the ERC-721 token and the algorithm of the staking reward were handled by off-chain centralized logic.

Recommendation:

Consider open sourcing the off-chain logic and listing the parameters users need to check.

Update:

The Sumati team replied that this was required by the business logic. The team will open source the off-chain logic.

Status:

It has been confirmed by the Sumati team.

[FP-4] Minting of NFTs At Will

Medium ✓ Fixed

Issue/Risk: Token Issuance

Description:

In `Sumati.sol` every time when an NFT token was minted its mint price was set by the Sumati team. In addition the batch mint function `mintBat` which was controlled by the admin controlled the mint price as well. Therefore the Sumati team could mint NFTs at will at a zero price.

Recommendation:

Consider adding a constraint for the total number of NFTs the `mintBat` function could mint.

Update:

The Sumati team added a constraint for the mint price.

Status:

It has been fixed by the Sumati team.

[FP-5] Conflict `require` Statements

Medium ✓ Fixed

Issue/Risk: Implementation Vulnerabilities

Description:

In `Sumati.sol` the `mintBat` function had two conflict `require` statements. If the first `require` held true the second would hold false. In addition, SafeMath is no longer needed if the compiler is Solidity V 0.8 or above.

```
require(timeBat < timestamp, "timestamp error");
require(timeBat.sub(timestamp) < 100000000, "timestamp error");
```

Recommendation:

Consider changing the code as follows:

```
require(timeBat < timestamp, "timestamp error");
require((timestamp-timeBat) < 20000000000000, "timestamp error");
```

Status:

It has been fixed by the Sumati team.

[FP-6] Missing Zero-Address Check

Low ✓ Fixed

Issue/Risk: Parameter Check

Description:

In `Sumati.sol` the following addresses didn't have zero-address checks

```
    function setDev(address _addr) public onlyOwner {
        dev = _addr;
    }

    function setSign(address _addr) public onlyOwner {
        SIGNER = _addr;
    }

    function setStakeAddr(address _addr) public onlyOwner {
        stakeAddr = _addr;
    }

    function setBurnAddr(address _addr) public onlyOwner {
        burnAddr = _addr;
    }

    function setToken(address _addr) public onlyOwner {
        tokenAddr = _addr;
    }
```

Recommendation:

Consider changing the code as follows:

```
    function setDev(address _addr) public onlyOwner {
        require(_addr!=address(0),"addr error");
        dev = _addr;
    }

    function setSign(address _addr) public onlyOwner {
        require(_addr!=address(0),"addr error");
        SIGNER = _addr;
    }

    function setStakeAddr(address _addr) public onlyOwner {
        require(_addr!=address(0),"addr error");
        stakeAddr = _addr;
    }

    function setBurnAddr(address _addr) public onlyOwner {
        require(_addr!=address(0),"addr error");
        burnAddr = _addr;
    }

    function setToken(address _addr) public onlyOwner {
        require(_addr!=address(0),"addr error");
        tokenAddr = _addr;
    }
```

Update:

The Sumati team added some validation code.

Status:

It has been fixed by the Sumati team.

[FP-7] Incorrect Verification

Low ✓ Fixed

Issue/Risk: Design Vulnerabilities

Description:

In `Sumati.sol`, the `redeem` function didn't correctly verify `` `to `` and `msg.sender`.

```
_safeTransfer(address(this), msg.sender, tokenId, "");
```

Recommendation:

Consider verifying them correctly.

Update:

The Sumati team added `require(to == msg.sender, "to not sender");` to require `to` to be `msg.sender`.

Status:

It has been fixed by the Sumati team.

[FP-8] Unused Libraries

Informational ✓ Fixed

Issue/Risk: Code Improvement

Description:

In both `Sumati.sol` and `BOSO.sol`, `SafeMath` was unused.

Recommendation:

Consider removing this library and all the `using SafeMath for uint256;` statements

Update:

The Sumati team has removed the code

Status:

It has been fixed by the Sumati team.

[FP-9] Redundant `require` Statements

Informational ✓ Fixed

Issue/Risk: Code Improvement

Description:

In `Sumati.sol`, in the `redeem` function, the first `require` was redundant since the second `require` could ensure the first `require` held true.

```
require(stakeList[tokenId] != address(0), "tokenId not stake");
require(stakeList[tokenId] == msg.sender, "you not staker");
```

Recommendation:

Consider removing the first `require` and keeping the second `require` as follows:

```
require(stakeList[tokenId] == msg.sender, "you not staker");
```

Update:

The Sumati team has removed the code.

Status:

It has been fixed by the Sumati team.

09. Recommendations to enhance the overall security

We list some recommendations in this section. They are not mandatory but will enhance the overall security of the system if they are adopted.

- Consider transferring admin rights to multi-sig wallets or DAOs
- Consider using contract address + Chain ID + parameters to verify signatures to prevent re-play attacks that may take place in cross-contract or cross-chain transactions.

Appendix

Unit Test Result:

79.06% Statements 268/339

69.14% Branches 112/162

77.37% Functions 106/137

78.57% Lines 275/350

File		Statements		Branches		Functions		Lines	
BOSO.sol		100%	44/44	83.33%	10/12	95.65%	22/23	100%	45/45
sumati.sol		75.93%	224/295	68%	102/150	73.68%	84/114	75.41%	230/305

Audited Files' SHA-256 Values:

```
BOSO.sol: 0x877d38c1dd5d59b8e1d1270315715a020e43ad021f4e43fe69c234e25be1d344
sumati.sol: 0x52ea5733c9474dfb18761f4b88da19d41c211166da391dba2efe4a4b2d3e7f9f
```

FAIRYPROOF

https://medium.com/@FairyproofT

https://twitter.com/FairyproofT

https://www.linkedin.com/company/fairyproof-tech

https://t.me/Fairyproof_tech

Reddit: https://www.reddit.com/user/FairyproofTech