HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Soul Society
Date: 15 Nov, 2023

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Soul Society
Approved By	Viktor Lavrenenko | SC Auditor at Hacken OÜ David Camps Novi | SC Audits Lead at Hacken OÜ Paul Fomichov | SC Audits Approver at Hacken OÜ
Tags	ERC20, ERC721, SoulBound
Platform	EVM
Language	Solidity
Methodology	Link
Website	Website
Changelog	06.10.2023 - Initial Review 31.10.2023 - Second Review 10.11.2023 - Third Review 15.11.2023 - Fourth Review

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

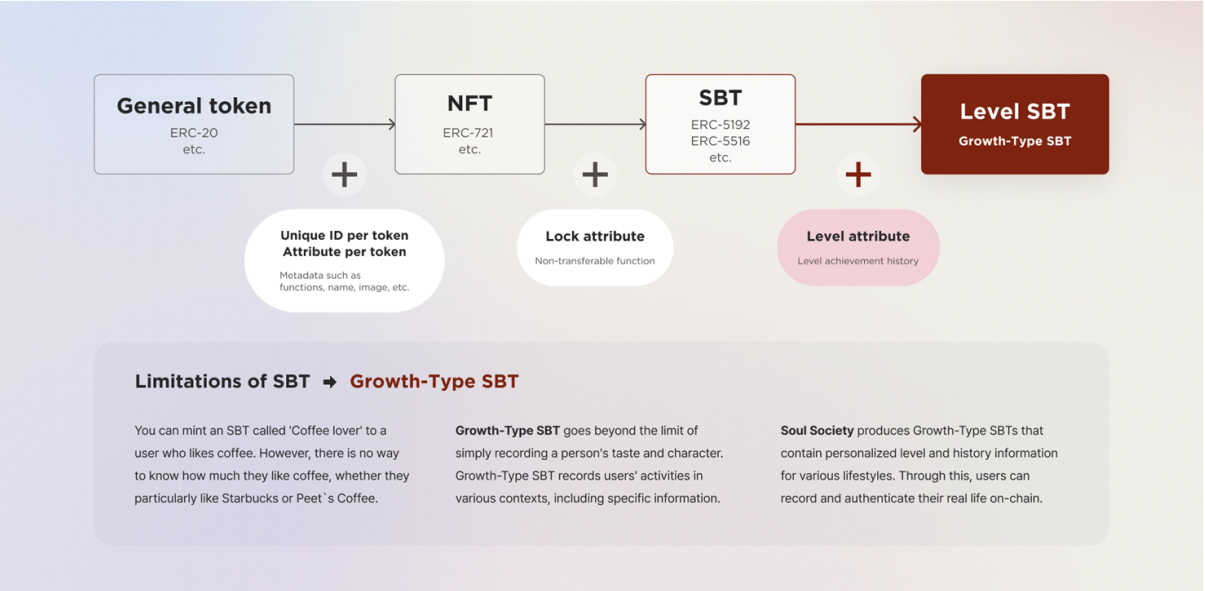
Table of contents

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Introduction

Hacken OÜ (Consultant) was contracted by Soul Society (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

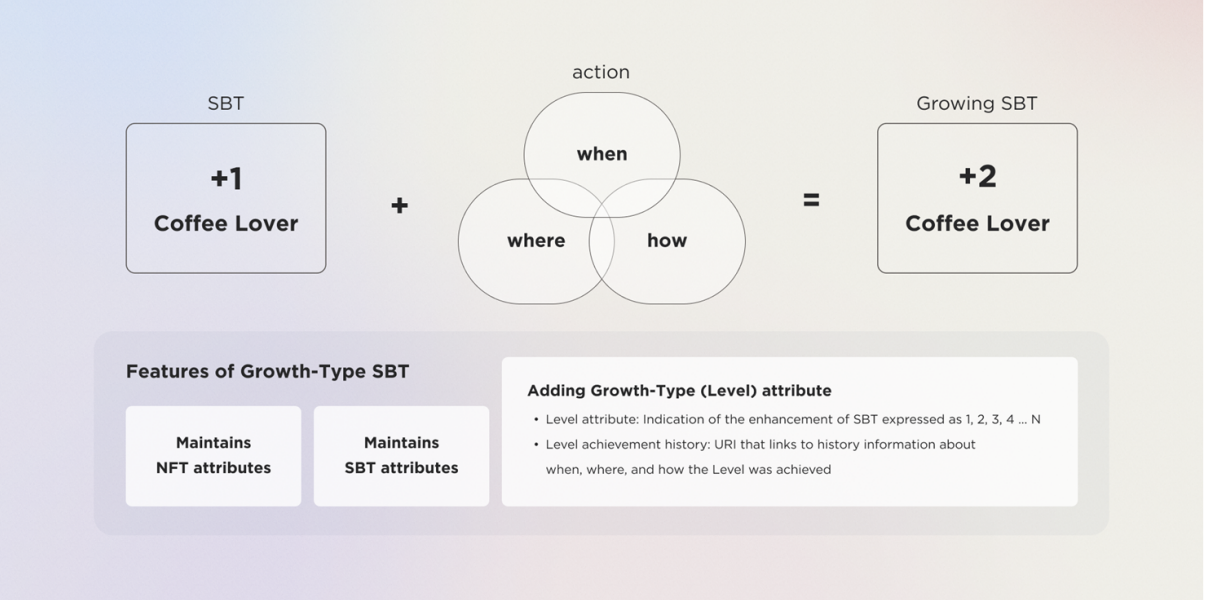
System Overview



SOUL_SOCIETY is a Growth-Type Soul-Bound Tokens (SBTs) protocol:

- Users can engage in several activities and acquire SBTs that define their identities. A user can own several SBTs.
- SBTs minted through Soul Society can be viewed by anyone and be applied to third-party services.

Growth-Type (level) feature of SBTs:



- Growth-Type Attribute: indication of the Level of engagement (1,2,3,...,N).
- Growth-Type Achievement History: URI that links to history information about when, where and how the Level was achieved.

The project presents the following contracts:

- HONToken- ERC20 token contract.
- SoulSocietySBT - custom SBT contract.

Privileged roles

- SoulSocietySBT contract Owner - can mint, burn and grow SBTs.
- HONToken contract Owner - can mint and burn HON Tokens as well as transfer Ownership.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are provided:
  - The purpose of the contracts is described.
  - The project's features are provided.
  - Business logic is included.
  - Use Cases are described.
- The technical description is complete:
  - Environment configuration is provided.

Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured.
- Best practices are followed.

Test coverage

Code coverage of the project is **0%** (branch coverage).

- For a project with less than 250 LOC (Lines of Code) the test coverage is not mandatory, and it is not accounted for in the final score.

Security score

As a result of the audit, the code contains no issues. The security score is **10** out of **10**.

All found issues are displayed in the "Findings" section.

Summary

According to the assessment, the Customer's smart contract has the following score: **10**. The system users should acknowledge all the risks summed up in the risks section of the report.

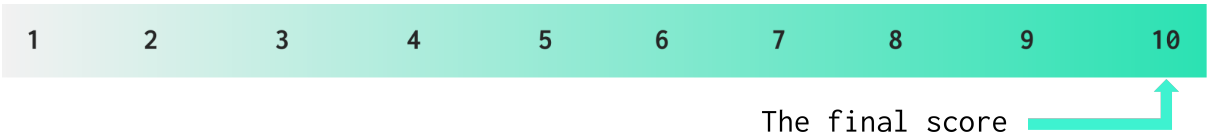


Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
06 October 2023	4	2	3	0
31 October 2023	0	1	0	0
10 November 2023	0	0	0	0
15 November 2023	0	0	0	0

Risks

No risks have been identified during the audit review.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Description	Status	Related Issues
Default Visibility	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed	
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed	
Outdated Compiler Version	It is recommended to use a recent version of the Solidity compiler.	Passed	
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed	
Unchecked Call Return Value	The return value of a message call should be checked.	Not Relevant	
Access Control & Authorization	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed	
SELFDESTRUCT Instruction	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant	
Check-Effect-Interaction	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed	
Assert Violation	Properly functioning code should never reach a failing assert statement.	Passed	
Deprecated Solidity Functions	Deprecated built-in functions should never be used.	Passed	
Delegatecall to Untrusted Callee	Delegatecalls should only be allowed to trusted addresses.	Not Relevant	
DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless required.	Passed	

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.	Passed	
Authorization through tx.origin	tx.origin should not be used for authorization.	Not Relevant	
Block values as a proxy for time	Block numbers should not be used for time calculations.	Not Relevant	
Signature Unique Id	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Not Relevant	
Shadowing State Variable	State variables should not be shadowed.	Passed	
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant	
Incorrect Inheritance Order	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed	
Calls Only to Trusted Addresses	All external calls should be performed only to trusted addresses.	Passed	
Presence of Unused Variables	The code should not contain unused variables if this is not justified by design.	Passed	
EIP Standards Violation	EIP standards should not be violated.	Passed	
Assets Integrity	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed	
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Passed	
Data Consistency	Smart contract data should be consistent all over the data flow.	Passed	

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Flashloan Attack	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used. Contracts shouldn't rely on values that can be changed in the same transaction.	Not Relevant	
Token Supply Manipulation	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed	
Gas Limit and Loops	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Not Relevant	
Style Guide Violation	Style guides and best practices should be followed.	Passed	
Requirements Compliance	The code should be compliant with the requirements provided by the Customer.	Passed	
Environment Consistency	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed	
Secure Oracles Usage	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant	
Tests Coverage	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed	
Stable Imports	The code should not reference draft contracts, which may be changed in the future.	Passed	

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Findings

Critical

No critical severity issues were found.

High

H01. Requirements Violation: The Supply of HON Tokens Is Not Limited

Impact	High
Likelihood	Medium

There is a mismatch between the documentation and the implementation.

The documentation states that the maxTotal supply is expected to be 1,000,000,000. However, the implementation lacks the necessary functionality to limit the supply to the previously mentioned number. As a consequence, the owner will be able to mint as many HON tokens as they want.

Path: ./contracts/HONToken.sol: *mint()*.

Recommendation: Fix the mismatch between the code and the requirements. It can be achieved by implementing the limit check for the in the *mint()* function.

Found in: fe50b45

Status: Fixed (Revised commit: 30a1f36)

Resolution: A check was introduced into *mint()* to make sure the maximum supply is not surpassed.

H02. Token Burn Does Not Follow ERC721 Standard And Leads To Inconsistencies

Impact	Medium
Likelihood	High

SoulBound Tokens are burned by calling *burn()* and triggering *_setGrowthToZero()*. These calls will reset *_tokenGrowths[tokenId_]=0*:

```
function burn(address to_, uint256 tokenId_) public onlyOwner  {
       _setGrowthToZero(to_, tokenId_);
}

function _setGrowthToZero(address to_, uint tokenId_) private onlyOwner {

       _requireMintedOf(to_, tokenId_);
```

```
        _tokenGrowths[tokenId_] = 0;

        emit Burn(to_, tokenId_);
}
```

In ERC721, there is an update of the *_balances* and *_owners* since the burn will remove the token from its current owner. The amount of tokens from the user will be decreased by one, and the new owner will be set to *address(0)*:

```
function _burn(uint256 tokenId) internal {
        address previousOwner = _update(address(0), tokenId, address(0));
        ...
}

function _update(address to, uint256 tokenId, address auth) internal virtual
returns (address) {
        ...
        if (from != address(0)) {
            ...
            unchecked {
                _balances[from] -= 1;
            }
        }
        ...
        _owners[tokenId] = to;

        ...
    }
```

As a consequence, the function *_exists()* will not work correctly because it will not revert for burned tokens:

```
function _exists(uint256 tokenId) internal view virtual returns (bool) {
        return _ownerOf(tokenId) != address(0);
    }
```

Additionally, when a user has all of their tokens burned, it will still remain as an owner because even though it actually does not own any token due to: 1) their *_ownedTokens* was not updated and 2) *_totalUsers* is not decreased.

A call to *_exists()* should be added at the beginning of the function, to maintain a consistency. Otherwise, *_balances* and *_totalUsers* will be incorrectly updated.

Finally, decreasing *_tokenGrowths* functionality is not related to regular token burning since SBTs will still appear as if they are owned by users and such users will still be able to use them. Therefore, we recommend keeping the *_setGrowthToZero()* in a separate function from *burn()*.

Path: ./v2/contracts/SoulSocietySBT.sol: *burn()*, *_setGrowthToZero()*.

Recommendation: Several changes are required in order to fix this issue:

1. Update the variables *_balances*, *_owners* and *_ownedTokens* when a token is burned.
2. Add a check that, if the user has no tokens in *_ownedTokens*, the *_totalUsers* decreases by *1*.
3. Add a call to *_exists()* at the beginning of the function.
4. Create a separate function to set the growth of the tokens to zero, using *_setGrowthToZero()*.

References: ERC721.

Found in: fe50b45

Status: Fixed (Revised commit: 30a1f36)

Resolution: The burn functionality was removed. Instead, the function *reset()* was introduced, which has the aim to set the growth of the SBT to 0, but to burn tokens.

H03. Too Highly Permissive Role Allows Owner To Burn Tokens From Users Without Their Consent Or Previous Notice

Impact	Medium
Likelihood	High

SoulBound Tokens can be burned by the Owner without notice:

```
function burn(address to_, uint256 tokenId_) public onlyOwner  {
       _setGrowthToZero(to_, tokenId_);
}
```

The Owner should not be able to access users' assets without their notice or consent.

Path: ./v2/contracts/SoulSocietySBT.sol: *burn()*.

Recommendation: Let users trigger the *burn()* function or add a user interaction check of some kind.

Found in: fe50b45

Status: Fixed (Revised commit: 30a1f36)

Resolution: The burn functionality was removed. Instead, the function reset() was introduced, which has the aim to set the growth of the SBT to 0. Said reset() functionality includes a check that ensures the users requested such change in their SBT level.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Medium

M01. Missing Safety Check for Non-EOA Receivers of Tokens Can Lead to Locked Tokens

Impact	Medium
Likelihood	Medium

The *_safeMint()* function lacks a necessary safety check that validates a recipient contract's ability to receive and handle ERC-721 tokens. Without this safeguard, tokens can inadvertently be sent to an incompatible contract, causing them, and any assets they hold, to become irretrievable.

```
function _safeMint(address to_, uint256 tokenType_) internal virtual onlyOwner
returns(uint256) {

        uint256 tokenId = _totalCount + 1;


        if (to_ == address(0)) {
            revert SoulSocietySBTInvalidReceiver(address(0));
        }

        if (_exists(tokenId)) {
            revert SoulSocietySBTExistToken(tokenId);
        }

        // if to is false , to address is new user
        if(!_existsOwner(to_)) {
            _totalUser += 1;
        }

        unchecked {
        // Will not overflow unless all 2**256 token ids are minted to the same
owner.
        // Given that tokens are minted one by one, it is impossible in practice
that
        // this ever happens. Might change if we allow batch minting.
        // The ERC fails to describe this case.
            _balances[to_] += 1;
            _totalCount += 1;

        }

        _owners[tokenId] = to_;
        _tokenTypes[tokenId] = tokenType_;
        _tokenGrowths[tokenId] = 1;
        _userProtects[to_] = false;
        _ownedTokens[to_][_balances[to_]-1] = tokenId; // index from 0

        emit Mint(address(0), to_, tokenId, tokenType_);
```

```
        return tokenId;
    }
```

The following *_checkOnERC721Received()* check should be added to the function *_safeMint()*.

```
function _safeMint(address to, uint256 tokenId, bytes memory data) internal virtual {
        _mint(to, tokenId);
        require(
            _checkOnERC721Received(address(0), to, tokenId, data),
            "ERC721: transfer to non ERC721Receiver implementer"
        );
    }
```

Paths: ./v2/contracts/SoulSocietySBT.sol: _safeMint().

Recommendation: Implement the *_safeMint()* function with the previously mentioned functionality to ensure that the recipient is equipped to handle ERC-721 tokens, thus mitigating the risk that NFTs could become frozen.

Found in: fe50b45

Status: Fixed (Revised commit: 13f46a1)

Resolution: The call *_checkOnERC712Received()* implemented into *_safeMint()*, following the ERC721 standard.

M02. Missing User Approval For Growth Level Update Results In Highly Centralized Growth System

Impact	**High**
Likelihood	**Low**

The owner of the SoulBound token contract can increase or decrease the growth level of the tokens without previous on-chain user request.

This means the owner has the potential to manipulate the growth level of the SBTs. Given the protocol intends to use the growth level as a key feature of the SBTs, a user request system should be introduced.

Path: ./v2/contracts/SoulSocietySBT.sol: *growUp()*, *_setGrowthToZero()*

Recommendation: Implement an on-chain user request system (e.g. a mapping user => tokenId => bool) to be checked in *growUp()* and *_setGrowthToZero()*.

Found in: fe50b45

Status: Fixed (Revised commit: 30a1f36)

Resolution: The function *growUp()* and the function *reset()* include a check on *_getApprovalGrowth()*, which makes sure the changes in the growth are approved by the token holders.

Low

L01. Floating Pragma

Impact	Low
Likelihood	Low

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Paths:
./contracts/*.sol

Recommendation: Lock the pragma version in all contracts as *0.8.19* instead of *^0.8.19*.

References: SWC-103.

Found in: fe50b45

Status: Fixed (Revised commit: 30a1f36)

Resolution: The pragma version was fixed to 0.8.20.

L02. Missing Events for Critical Value Updates

Impact	Low
Likelihood	Low

Events should be emitted after sensitive changes take place, to facilitate tracking and notify off-chain clients following the contract's activity.

Paths: ./v2/contracts/SoulSocietySBT.sol: *constructor()*, *setTokenURI()*.

Recommendation: Consider emitting events in previously mentioned functions.

Found in: fe50b45

Status: Fixed (Revised commit: 30a1f36)

Resolution: The events were added.

L03. Missing URI Length Check

Impact	Low
Likelihood	Low

The function *tokenURI()* lacks the necessary check that would ensure that the empty URI string is not included in the final encoded URI.

```
function tokenURI(uint256 tokenId_) external view virtual  returns (string memory) {

        // check minted
        _requireMinted(tokenId_);

        // check protected status
        _isProtectedTokenId(tokenId_);

        uint256 tokenType = _tokenTypes[tokenId_];

        return string(abi.encodePacked(_uri,tokenId_.toString(),"?tokenType=",
tokenType.toString()));
        // return string.concat(_uri, tokenId_.toString());
    }
```

The actual behavior might lead to unexpected issues if the incorrect URI will be used instead.

The following piece of code can be added to mitigate the previously mentioned issue.

```
return bytes(_uri).length > 0 ? string(abi.encodePacked(_uri,
tokenId_.toString())) : "";
```

Paths: ./v2/contracts/SoulSocietySBT.sol: *tokenURI().*

Recommendation: It is recommended to implement the aforementioned check to avoid the potential problems.

Found in: fe50b45

Status: Fixed (Revised commit: 30a1f36)

Resolution: The empty URI check was implemented.

L04. Inefficient Checks in setProtected() Result In Inefficient Code

Impact	Low
Likelihood	Medium

The *setProtected()* function lacks the check to ensure that the *to_* address has already minted some SBT tokens.

```
function setProtected(address to_, bool isProtected_) public  returns (bool) {
```

```
        if (msg.sender != to_) {
            revert SoulSocietySBTInvalidOwner(to_);
        }

        _userProtects[to_] = isProtected_;

        return getProtected(to_);
    }
```

As a consequence, the protection will be enabled for addresses which are not participants of the system.

Additionally, the check if(msg.sender!=to_) is unnecessary. It is recommended to use msg.sender instead of to_ in order to enforce the same behavior with a simpler code:

```
function setProtected(bool isProtected_) public  returns (bool) {

        if (!_balanceOf(msg.sender) > 0){
             revert SoulSocietySBTExistToken(msg.sender);
        }

        _userProtects[msg.sender] = isProtected_;

        return getProtected(msg.sender);
    }
```

Paths: ./v2/contracts/SoulSocietySBT.sol: *setProtected().*

Recommendation: Add a call to *_balanceOf()* and use *msg.sender* instead of *to_*.

Found in: fe50b45

Status: Fixed (Revised commit: 30a1f36)

Resolution: The function was updated as recommended.

Informational

I01. Redundant Initialization Is Not Gas Efficient

The state variables *_totalUser* and *_totalCount* are set to *0* during deployment, spending more Gas than necessary:

```
// The number of users who own SBT.
   uint256 private _totalUser = 0;

   // Total number of SBT issued
   uint256 private _totalCount = 0;
```

Variables of type *uint256* are initialized as *0* by default, making it unnecessary to set them to *0*:

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

```
// The number of users who own SBT.
   uint256 private _totalUser;

   // Total number of SBT issued
   uint256 private _totalCount;
```

Similarly, the state variable *_userProtects[to_]* is redundantly set to false in *_safeMint()*.

Path: ./v2/contracts/SoulSocietySBT.sol: *_safeMint()*.

Recommendation: Do not initialize the state variables *_totalUser* and *_totalCount* and *_userProtects*.

Found in: fe50b45

Status: Fixed (Revised commit: 30a1f36)

Resolution: The redundant initialization was removed.

I02. Function tokenURI Is Not Gas Efficient

The call *_requireMinted(tokenId_)* is redundant, since it performs a check included in *_isProtectedTokenId()*:

```
function _requireMinted(uint256 tokenId) internal view virtual {
       if (!_exists(tokenId)) {
           revert SoulSocietySBTNonexistentToken(tokenId);
       }
   }
```

```
function _isProtectedTokenId(uint256 tokenId_) internal view {
       if (!_exists(tokenId_)) {
           revert SoulSocietySBTNonexistentToken(tokenId_);
       }
       _isProtected(_owners[tokenId_]);
   }
```

Therefore, the *tokenURI()* can be simplified by removing the call to *_requireMinted()* and the function *_requireMinted()* can be removed from the contract since it is not used anywhere else.

```
function tokenURI(uint256 tokenId_) external view virtual  returns (string memory)
{
       _isProtectedTokenId(tokenId_);
       uint256 tokenType = _tokenTypes[tokenId_];
       return string(abi.encodePacked(_uri,tokenId_.toString(),"?tokenType=",
tokenType.toString()));
}
```

Additionally, the memory variable *tokenType* is declared in the function *tokenURI()* by reading from storage once. This variable will be used later once.

Instead, the storage variable *_tokenTypes[tokenId]* can be returned directly, without any need to declare the memory variable tokenType:

```
function tokenURI(uint256 tokenId_) external view virtual  returns (string memory)
{
        _isProtectedTokenId(tokenId_);
        return string(abi.encodePacked(_uri,tokenId_.toString(),"?tokenType=",
_tokenTypes[tokenId_].toString()));
}
```

Path: ./v2/contracts/SoulSocietySBT.sol: *tokenURI()*.

Recommendation: Remove the call to *_requireMinted()* and the function *requireMinted()*. Delete the variable tokenType and use *_tokenTypes[tokenId]* instead.

Found in: fe50b45

Status: Fixed (Revised commit: 30a1f36)

Resolution: The redundant code was removed.

I03. Disabled Solidity Optimizer

Disabled Solidity optimizer increases the overall Gas cost.

Path: ./contracts/*.config

Recommendation: Enable the Solidity compiler optimizer to minimize the size of the code and the cost of execution via inline operations, deployments costs, and function call costs.

Found in: fe50b45

Status: Fixed (Revised commit: 30a1f36)

Resolution: The code is deployed via Remix IDE, in which the optimizer was enabled.

I04. State Variables Can Be Constant

Compared to regular state variables, the Gas costs of constant and immutable variables are much lower. The following variables are set only once during deployment time, hence they can be set constant to save not only storage space, but the Gas Costs as well.

```
    // token Name
    string private _name;

    // token Symbol
```

```
    string private _symbol;
```

Path: ./v2/contracts/SoulSocietySBT.sol: *_name*, *_symbol*.

Recommendation: Use constant keywords on state variables to decrease Gas costs.

Found in: fe50b45

Status: Fixed (Revised commit: 30a1f36)

Resolution: The variables were set as constants.

I05. Redundant onlyOwner Requirements Are Not Gas Efficient

The internal functions *_safeMint()*, *_setGrowthToZero()* and *_growUp()* are only called by *mint()*, *burn()* and *growUp()* respectively.

However, all these six functions use an *onlyOwner* modifier:

```
   function mint(address to_, uint256 tokenType_) public virtual onlyOwner
returns(uint256) {
   function _safeMint(address to_, uint256 tokenType_) internal virtual onlyOwner
returns(uint256) {

   function burn(address to_, uint256 tokenId_) public onlyOwner  {
   function _setGrowthToZero(address to_, uint tokenId_) private onlyOwner {

   function growUp(address to_, uint256 tokenId_) public  onlyOwner
returns(uint256) {
   function _growUp(address to_, uint256 tokenId_) internal onlyOwner
returns(uint256) {
```

It is not necessary to call *onlyOwner* in the *internal* functions, since that check will be executed in the *public* functions that call them. Thus, there is an unnecessary expense of Gas.

Path: ./v2/contracts/SoulSocietySBT.sol: *_safeMint()*, *_setGrowthToZero()*, *_growUp()*.

Recommendation: Remove the *onlyOwner* modifier from the *internal* functions *_safeMint()*, *_setGrowthToZero()* and *_growUp()*.

Found in: fe50b45

Status: Fixed (Revised commit: 30a1f36)

Resolution: The onlyOwner modifier was removed from the reported functions.

I06. Lack Of Clear Code In _growUp() Function

In *_growUp*, the increase of *_tokenGrowths* is mixed with the declaration of *tokenGrowth*

```
        uint256 tokenGrowth = _tokenGrowths[tokenId_] += 1;
```

It is recommended to separate the increase of *_tokenGrowths* from the declaration of *tokenGrowth* in order to make the code cleaner and avoid any confusion:

```
function _growUp(address to_, uint256 tokenId_) internal onlyOwner
returns(uint256) {


        _requireMintedOf(to_, tokenId_);

        _tokenGrowths[tokenId_] += 1

        uint256 tokenGrowth = _tokenGrowths[tokenId_];

        emit GrowUp(to_, tokenId_, tokenGrowth);

        return tokenGrowth;
    }
```

Path: ./v2/contracts/SoulSocietySBT.sol: *_growUp().*

Recommendation: Separate the increase of *_tokenGrowths* from the declaration of *tokenGrowth*.

Found in: fe50b45

Status: Fixed (Revised commit: 30a1f36)

Resolution: The code was updated as recommended.

I07. Style Guide Violation: Order Of Layout

Contract readability and code quality are influenced significantly by adherence to established style guidelines. In *Solidity* programming, there exist certain norms for code arrangement and ordering. These guidelines help to maintain a consistent structure across different contracts, libraries, or interfaces, making it easier for developers and auditors to understand and interact with the code.

The suggested order of elements within each *contract*, *library*, or *interface* is as follows:

- Type declarations
- State variables
- Events
- Modifiers
- Functions

Functions should be ordered and grouped by their *visibility* as follows:

- Constructor
- Receive function (if exists)
- Fallback function (if exists)
- External functions
- Public functions
- Internal functions
- Private functions

Within each grouping, *view* and *pure* functions should be placed at the end.

Path: ./v2/contracts/SoulSocietySBT.sol

Recommendation: It is recommended to change the order of layout to fit the Official Style Guide.

References: Solidity Style Guide

Found in: fe50b45

Status: Fixed (Revised commit: 30a1f36)

Resolution: The layout was updated to comply with the Solidity Style Guide.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	**Critical**	**High**	**Medium**
Medium Likelihood	**High**	**Medium**	**Low**
Low Likelihood	**Medium**	**Low**	**Low**

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

High: High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/SoulSocietyDev/soulsociety-sbt-contract/tree/master/
Commit	fe50b45
Whitepaper	Link
Requirements	Link
Technical Requirements	README
Contracts	File: hon/contracts/HonContract.sol SHA3: 10e0e1a68f2781de55c5b424d0cfb52fc08748713b1f25d022ac957b16bea0a8 File: v2/contracts/SoulSocietySBT.sol SHA3: 4ac9a642185524e665283109d6d4e3405051eb3be066a649b361cd506ff0e6e4 File: v2/contracts/interfaces/ISoulSocietyEnumerableSBT.sol SHA3: 999c2e3e59f9c6556485aee4ba4d189c26881b97a561b558ea97e2b270d1cbeb File: v2/contracts/interfaces/ISoulSocietySBT.sol SHA3: ebd22b75ec2dd5af51818289b0504929104910809990 4e5a4423512e6d68d63a File: v2/contracts/interfaces/ISoulSocietySBTErrors.sol SHA3: 52a55b050926c734e740408f557f429c082c544cfa7cd20faf1c61c1280cd233 File: v2/contracts/interfaces/ISoulSocietySBTMetadata.sol SHA3: c51ad1618a40477207fbc4ea76802d3465f05c4ab9a096a4f041c90420bdd940

Second review scope

Repository	https://github.com/SoulSocietyDev/soulsociety-sbt-contract/tree/master/
Commit	30a1f36
Whitepaper	Link
Requirements	Link
Technical Requirements	README
Contracts	File: hon/contracts/HonContract.sol SHA3: 06b8459db582b9b8a2ef02fa170b5e78a7ce08cb827e62be4cebfef3fd8074cc File: v2/contracts/SoulSocietySBT.sol SHA3: 52daaaf09ec4d28b2f7ba5c0d05610872417010a8351bf8e5155d06fef2e5101 File: v2/contracts/interfaces/ISoulSocietyEnumerableSBT.sol

	SHA3: 94dfd1fd605d393abb431935ba66109faf06d563428e98ad5ef9afd095c20a95 File: v2/contracts/interfaces/ISoulSocietySBT.sol SHA3: e06668309b6b53cf3b0b10f821be3ef4c7b2b11dc7fc11df2e645fd23d339df9 File: v2/contracts/interfaces/ISoulSocietySBTErrors.sol SHA3: 9d963e089c4ef0c70bf24581d9f5b4e6804028d07a6bdb1e99df09380796b3ac File: v2/contracts/interfaces/ISoulSocietySBTMetadata.sol SHA3: 6c643affd372db919f5e924d991ca456bff6b2cbcd593923ab546d40f03320fe

Third review scope

Repository	https://github.com/SoulSocietyDev/soulsociety-sbt-contract/tree/master/
Commit	13f46a1
Whitepaper	Link
Requirements	Link
Technical Requirements	README
Contracts	File: hon/contracts/HonContract.sol SHA3: 06b8459db582b9b8a2ef02fa170b5e78a7ce08cb827e62be4cebfef3fd8074cc File: v2/contracts/SoulSocietySBT.sol SHA3: 27a63f26d0a750ecf1ee06e6adde8fe3517db0ee40deaa019c772d3f0b6786fe File: v2/contracts/interfaces/ISoulSocietyEnumerableSBT.sol SHA3: 94dfd1fd605d393abb431935ba66109faf06d563428e98ad5ef9afd095c20a95 File: v2/contracts/interfaces/ISoulSocietySBT.sol SHA3: e06668309b6b53cf3b0b10f821be3ef4c7b2b11dc7fc11df2e645fd23d339df9 File: v2/contracts/interfaces/ISoulSocietySBTErrors.sol SHA3: 9d963e089c4ef0c70bf24581d9f5b4e6804028d07a6bdb1e99df09380796b3ac File: v2/contracts/interfaces/ISoulSocietySBTMetadata.sol SHA3: 6c643affd372db919f5e924d991ca456bff6b2cbcd593923ab546d40f03320fe

Fourth review scope

Repository	https://github.com/SoulSocietyDev/soulsociety-sbt-contract/tree/7684979dab221d52ad4020313269b8cb93136cca
Commit	76d55f5
Whitepaper	Link
Requirements	Link
Technical Requirements	README

Contracts	File: ./hon-contract/src/HonContract.sol SHA3: 5516d042addbe4aa3be1cedc28a40a5014c68e710b09e56a2725bf376dfc47f2 File: ./sbt-contract/src/SoulSocietySBT.sol SHA3: 27e4ca2fff68c209d60a9ce556981aaac96b4eede3eff0375a8ad34143f3cefe File: ./sbt-contract/src/interfaces/ISoulSocietyEnumerableSBT.sol SHA3: ad281fab5bcc9a73ad5139424ab50939673a815f6c6e0e1c5580d8255a90775e File: ./sbt-contract/src/interfaces/ISoulSocietySBT.sol SHA3: e06668309b6b53cf3b0b10f821be3ef4c7b2b11dc7fc11df2e645fd23d339df9 File: ./sbt-contract/src/interfaces/ISoulSocietySBTErrors.sol SHA3: 9d963e089c4ef0c70bf24581d9f5b4e6804028d07a6bdb1e99df09380796b3ac File: ./sbt-contract/src/interfaces/ISoulSocietySBTMetadata.sol SHA3: 0f5b28f4c92f57baf6c2e1adc89458ac18012a290a9347870a1a5b02569ad935