verichains

SECURITY AUDIT OF

SINGSING SMART CONTRACTS

SingSing

Public Report

Apr 19, 2022

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.

EXECUTIVE SUMMARY

This Security Audit Report prepared by Verichains Lab on Apr 19, 2022. We would like to thank the SingSing for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the SingSing Smart Contracts. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some vulnerable issues in the smart contract code, along with some recommendations. SingSing team has resolved and updated most of the issues following our recommendations.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About SingSing Smart Contracts

SingSing is a social singing platform, connecting SUPERFANS with their favorite SINGERS to build a new music economy together on blockchain.

SuperPass NFT is the core asset of SingSing, it's also the password to enter the SingSing world. The platform has a native utility token called SING TOKEN. SING is an essential part of the SingSing platform and they are working on establishing key mechanics that make it intrinsically tied to SingSing platform and its own values.

1.2. Audit scope

This audit focused on identifying security flaws in code and the design of SingSing Smart Contracts. It was conducted on commit 5a7bad9e3ef39a575ced412ea6a6ed4acae14508 from git repository *https://github.com/phamsonha/SING-BEP20/* and commit 0c818da4d4cf8050e61368a65d4a005510d9a94f from git repository *https://github.com/phamsonha/SingPass/*.

The latest version of the following files were made available in the course of the review:

SHA256 Sum	File
0328386b0d193024f908a2e4e88f5b83a80945818ce6b0a0a2920b0ca1e7bfca	**./singpass/contracts/PassBreeding.sol**
6b6a0aaf46dbee6b4022594dbeb52602a6e60fb0e286a9e0f4b661bcf0fd007a	**./singpass/contracts/IGeneScience.sol**
5913c4671ee8efb7750765bf1e995fb7866f150f31338f3e2a6a856e9a3a6c2a	**./singpass/contracts/PassGeneScience.sol**
719c603d51b8a5230fd3776ac6d08015452b65ed2417bf0c5dc6e0132c092657	**./singpass/contracts/PassMinting.sol**
0228323153498965173a9a8c7fa57e25dedd4c079b2344b272f2d30347071497	**./singpass/contracts/IVerichainsNetRandomService.sol**
fcb174b239839229a309de295db219f17952204566d6ff0b5c87d024522bc9af	**./singpass/contracts/PassBase.sol**
89dfdc198270837ed2ccc5ff7c528e192f3f881e760da4814d20e07aa564e611	**./singpass/contracts/PassCore.sol**
81343582f8a9425552b0e861f22b1661d7a717039f155aea40cad1c8ad691ab8	**./singpass/contracts/PassAccessControl.sol**

b2c743ca2b96e50494d634c88ceb33e117dca43bc9f1082726e4c3ec0a6f52b3	**./singpass/contracts/PassNFT.sol**
59f707d2c805461154a32800e0f7cf4a845e8b9ebeed1452c157f45de375bad6	**./bep20/contracts/TokenVesting.sol**
a681e9f0721ecea5984e07155601d0094518cacf41c51086bd1efdc2811f41df	**./bep20/contracts/Utils/Verifier.sol**
24f910d9f07502c7f5ad357c1270c162fb7e9056eebab75c27c2ce71b6020d04	**./bep20/contracts/SingSingToken.sol**

1.3. Audit methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Please note that security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a 100% secure smart contract. However, auditing allows discovering vulnerabilities that were unobserved, overlooked during development and areas where additional security measures are necessary.

2. AUDIT RESULT

2.1. Overview

The SingSing Smart Contracts was written in Solidity language, with the required version to be ^0.8.0. The source code was written based on OpenZeppelin's library and CryptoKitties NFT game.

There are two main parts in the audit scope as shown in the below section:

2.1.1. ERC20 token and vesting contract

SING token is the main ERC20 token in SingSing ecosystem. The SingSingToken contract extends ERC20, ERC20Burnable, Pausable, and Ownable. With Ownable, by default, the token owner is also the contract deployer, but he can transfer ownership to another address at any time. When the token contract is initialized, all the tokens (max supply amount) will be minted and transferred to the contract owner wallet. Furthermore, he can pause/unpause contract using Pausable contract, users can only transfer tokens when contract is not paused.

The below table lists some properties of the audited SingSingToken contract (as of the report writing time).

PROPERTY	VALUE
Name	SingSing Token
Symbol	SING
Decimals	18
Max Supply	2,400,000,000 ($x10^{18}$)

Table 2. SING token properties

The logic for the token vesting contract is defined in TokenVesting.sol, this contract implements a vesting mechanism to lock and release tokens according to a configured schedule defined by the contract operator. The token distribution process can be summarized as below:

- Before the start time, all tokens are locked in the vesting contract.
- Once the start time is reached, a TGE amount will be unlocked.
- When the cliff time is reached, all the remaining tokens will be released at the end of each cliff period (calculated linearly from the start time).

Note: The contract owner can withdraw tokens in the vesting contract in case of an emergency situation. However, he must wait for 7 days after submitting the withdrawal request.

2.1.2. SuperPass NFT contracts

Superpass NFT is the core asset of SingSing, which is defined in the PassCore contract. SuperPasses are divided into three classes, which is Bronze, Silver, and Gold. Two superpasses can be mixed together to create new superpass via the breeding feature, their classes are mixed randomly using Verichains' on-chain RNG service for random seeds generation.

2.2. Findings

During the audit process, the audit team found some vulnerabilities in the given version of SingSing Smart Contracts.

SingSing fixed the code, according to Verichains's draft report, in commit 959d8eba383a07bf550237d8816ea4137c73b1dd (SingPass) and 7e75cc59f63b50632bda7947c93ce7503b8fb31b (SingBep).

2.2.1. PassBreeding.sol - Missing contract call blocking CRITICAL

In the PassBreeding contract, the giveBirth function doesn't have any contract blocking mechanism. So, if this function is called from a smart contract, users can easily revert the whole transaction in case they do not get their desired superpass.

```
function giveBirth(uint256 _matronId, bytes32 _revealHash)
    public
    whenNotPaused
    returns(uint256)
{
    uint256 random = reveal(_matronId, _revealHash);
    // Grab a reference to the matron in storage.
    SuperPass storage matron = superpasses[_matronId];
    // ...
}
```

RECOMMENDATION

We must add a contract blocking mechanism to this function, a simple but effective way to do this is to check if msg.sender == tx.origin.

UPDATES

- *Apr 12, 2022*: This issue has been acknowledged and fixed by SingSing team in commit 959d8eba383a07bf550237d8816ea4137c73b1dd.

2.2.2. PassBreeding.sol - User can control randomness result in reveal function HIGH

The reveal function is used to generate the random number which will be used for generating the gene and class for the newly created superpass. However, if users delay the call to the giveBirth function (which calls the reveal function), they can control the random result which their revealHash since the blockHash in this case would be a zero-filled bytes32 string.

```
function reveal(uint256 matronId, bytes32 revealHash) private returns(uin…
  t256) {
    //make sure it hasn't been revealed yet and set it to revealed
    require(commits[msg.sender][matronId].revealed==false,"CommitReveal::…
  reveal: Already revealed");
    commits[msg.sender][matronId].revealed=true;
    //require that they can produce the committed hash
    require(getHash(revealHash)==commits[msg.sender][matronId].commit,"Co…
  mmitReveal::reveal: Revealed hash does not match commit");
    //require that the block number is greater than the original block
    require(uint64(block.number)>commits[msg.sender][matronId].block,"Com…
  mitReveal::reveal: Reveal and commit happened on the same block");
    //get the hash of the block that happened after they committed
    bytes32 blockHash;
    if (uint64(block.number)<=commits[msg.sender][matronId].block + 250)
        blockHash = blockhash(commits[msg.sender][matronId].block);
    else blockHash = blockhash(uint64(block.number)); // ZERO BLOCKHASH
    //hash that with their reveal that so miner shouldn't know
    uint256 random = uint256 (keccak256(abi.encodePacked(blockHash, revea…
  lHash))); // USER CAN CONTROL RANDOM
    emit RevealHash(msg.sender, revealHash, random);
    return random;
}
```

RECOMMENDATION

The safest way to generate random numbers at this moment is using ChainLink VRF. Check out their documentation for more information https://docs.chain.link/docs/chainlink-vrf/.

UPDATES

- *Apr 12, 2022*: This issue has been acknowledged and fixed by SingSing team in commit 959d8eba383a07bf550237d8816ea4137c73b1dd.

2.2.3. TokenVesting.sol - Vesting contract may not have enough tokens MEDIUM

When adding a new beneficiary using addBeneficiary function, the contract owner doesn't transfer a corresponding amount of tokens. So, we cannot ensure that this vesting contract has enough tokens to release.

```
function addBeneficiary(address _beneficiary, uint256 _tgeUnlockAmount, u…
  int256 _vestingAmount)
    public
    onlyOwner
{
    require(
        _beneficiary != address(0),
        "The beneficiary's address cannot be 0"
    );
    require(_vestingAmount > 0, "Shares amount has to be greater than 0")…
  ;

    if (shares[_beneficiary] == 0) {
        beneficiaries.push(_beneficiary);
    }

    shares[_beneficiary] = shares[_beneficiary].add(_vestingAmount);
    tgeUnlock[_beneficiary] = tgeUnlock[_beneficiary].add(_tgeUnlockAmoun…
  t);
}
```

RECOMMENDATION

The corresponding amount of tokens should be transferred to the vesting contract when a new beneficiary is added.

UPDATES

- *Apr 12, 2022*: This issue has been acknowledged by SingSing team.

2.2.4. PassBase.sol - Breeding fees must be updated when maxBreedTimes is updated MEDIUM

In the PassBase contract, the default value for maxBreedTimes is 3 for now so that the setupBreedingFees function will be called for 3 times. However, when updating the maxBreedTimes variable, the breeding fees are not updated accordingly.

```
// PassBase.sol
function setMaxBreedTimes (uint16 breedTimes) public onlyCOO {
    maxBreedTimes = breedTimes;
}

// PassCore.sol
constructor()  {
    // ...

    //init setup breeding fee
    setupBreedingFees (0, 10 * 10 ** 18, 1000 * 10 ** 18);  //10 * 10^18 …
  in wei
    setupBreedingFees (1, 20 * 10 ** 18, 3000 * 10 ** 18);  //20 * 10^18 …
  in wei
    setupBreedingFees (2, 30 * 10 ** 18, 10000 * 10 ** 18); //30 * 10^18 …
  in wei

    //init fee master wallet
    feeMasterWallet = msg.sender;
}
```

RECOMMENDATION

When updating the maxBreedTimes variable, we must update the breeding fees accordingly.

UPDATES

- *Apr 12, 2022*: This issue has been acknowledged and fixed by SingSing team in commit 959d8eba383a07bf550237d8816ea4137c73b1dd.

2.2.5. PassNFT.sol - transfer function can only be used by token owner MEDIUM

The transfer function allows approved user calling it. However, if the calling user is not the owner of input token, they still cannot transfer it using the transfer function.

```
modifier onlyApproved(uint256 _superpassId) {
    require(
        isSuperPassOwner(_superpassId) ||
        isApproved(_superpassId) ||
        isApprovedOperatorOf(_superpassId),
        "sender not superpass owner OR approved"
    );
    _;
```

```
}

function transfer(address _to, uint256 _tokenId)
    external override
    onlyApproved(_tokenId)
    notZeroAddress(_to)
{
    // require(_to != address(this), "transfer to contract address");

    _transfer(msg.sender, _to, _tokenId);
}
```

RECOMMENDATION

The transfer function should restrict callers to token owners only.

UPDATES

- *Apr 12, 2022*: This issue has been acknowledged and fixed by SingSing team in commit 959d8eba383a07bf550237d8816ea4137c73b1dd.

2.2.6. PassBreeding.sol - Autobirth daemon cannot call giveBirth without _revealHash MEDIUM

The breedWithAuto function of the PassBreeding contract is used to call giveBirth automatically after breeding. However, the autobirth daemon may not have the corresponding _revealHash of the input _dataHash so it cannot call the giveBirth function.

```
function breedWithAuto(uint256 _matronId, uint256 _sireId, bytes32 _dataH…
  ash)
    public
    payable
    whenNotPaused
{
    // Check for payment
    require(msg.value >= autoBirthFee);

    // Call through the normal breeding flow
    breedWith(_matronId, _sireId, _dataHash);

    // Emit an AutoBirth message so the autobirth daemon knows when and f…
  or what pass to call
    // giveBirth().
```

```
    SuperPass storage matron = superpasses[_matronId];
    emit AutoBirth(_matronId, matron.cooldownEndTime);
}
```

RECOMMENDATION

There must be some APIs for user to supply the _revealHash data to the backend server. And with the supplied _revealHash, the autobirth daemon can call the giveBirth function.

UPDATES

- *Apr 12, 2022*: This issue has been acknowledged and fixed by SingSing team in commit 959d8eba383a07bf550237d8816ea4137c73b1dd.

2.2.7. PassBase.sol - classTokenCount may exceed classTokenLimit LOW

In the _selectClass function, when the selectedClass is Bronze, the classTokenCount may exceed the classTokenLimit.

```
function _selectClass(
    uint8 _matronClass,
    uint8 _sireClass,
    uint256 _random
)
    internal view
    returns (uint8 selectedClass, uint256 rand)
{
    // ...
    for (uint8 i=0; i<10; i++) {
        rand =  uint256(keccak256(abi.encode(_random, block.timestamp, i+…
  1)))%99; // 0~99
        selectedClass = arr[rand];
        //check if class is Bronze then break loop becase we don't need t…
  o limit Bronze
        if (selectedClass == 0)
            break;
        //check limit, if class is not Bonze and if token count by class …
  is smaller than limit then break loop
        if (selectedClass > 0 && classTokenCount[selectedClass] < classTo…
  kenLimit[selectedClass]) { // POSSIBLE OF ERRORS
            break;
        }
    }
```

```
    return (selectedClass, rand);
}
```

RECOMMENDATION

The classTokenLimit should be applied for all three classes (Bronze, Silver, and Gold). In this case, the contract admins should increase the class limit instead of returning the Bronze class.

UPDATES

- *Apr 12, 2022*: This issue has been acknowledged by SingSing team.

2.2.8. PassNFT.sol - Missing IERC721Enumerable and IERC721Metadata interface LOW

In the supportsInterface function of the PassNFT contract, the returning supported interfaces should include IERC721Enumerable and IERC721Metadata.

```
function supportsInterface(bytes4 _interfaceId)
    external
    view
    returns (bool)
{
    return (_interfaceId == _INTERFACE_ID_ERC165 ||
        _interfaceId == _INTERFACE_ID_ERC721);
}
```

RECOMMENDATION

We should add two missing interfaces to the supportsInterface function.

UPDATES

- *Apr 12, 2022*: This issue has been acknowledged and fixed by SingSing team in commit 959d8eba383a07bf550237d8816ea4137c73b1dd.

2.2.9. SingSingToken.sol - Pausable logic must be implement in _beforeTokenTransfer hook LOW

The SingSingToken contract extends the Pausable contract which may be used to pause token transfers. However, the implementation is not correct.

```
contract SingSingToken is ERC20, ERC20Burnable, Pausable, Ownable {
    using SafeMath for uint256;
    uint256 private totalTokens;
```

```
    constructor() ERC20("SingSing Token", "SING") {
        totalTokens = 2400000000 * 10**uint256(decimals());
        _mint(owner(), totalTokens); // total supply fixed at 2.4 billion…
   tokens
    }

    function pause() public whenNotPaused {
        _pause();
    }

    function unpause() public whenPaused {
        _unpause();
    }
}
```

RECOMMENDATION

The whenNotPaused modifier must be added to the token transfer functions or the hook _beforeTokenTransfer to get it works.

UPDATES

- *Apr 12, 2022*: This issue has been acknowledged and fixed by SingSing team in commit 7e75cc59f63b50632bda7947c93ce7503b8fb31b.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Apr 19, 2022*	Public Report	Verichains Lab

Table 3. Report versions history