InterFi Network

@interfinetwork

SMART CONTRACT AUDIT OF SANTA FE

SANTA FE

AUDITED ON SEPTEMBER 01, 2021

USING INTERFI AUDITING ARCHITECTURE

Summary

Audit:

Auditing Firm	InterFi Network
Architecture	InterFi Auditing Architecture
Smart Contract Audit Approved By	Chris | Blockchain Specialist at InterFi
Project Overview Approved BY	Albert | Project Specialist at InterFi (Not Applicable)
Platform	Solidity
Audit Check (Mandatory)	Vulnerability Check, Source Code Review, Functional Test
Project Check (Optional)	Website Review, Socials Review, Token Review (Not Applicable)
Consultation Request Date	August 30, 2021
Assessment Date	September 01, 2021

Risk profile:

InterFi team has performed a line-by-line manual analysis and automated review of the smart contract. The smart contract was analyzed mainly for common smart contract vulnerabilities, exploits, and manipulation hacks. According to the smart contract audit, **SantaFactoryV2** smart contract source code has **Low Risk Severity.**

For the detailed understanding of risk severity, source code vulnerability, and functional test, kindly refer to the audit.

Table of contents

Project overview

InterFi was consulted by Santa Fe on August 30, 2021 to conduct a smart contract security audit for one of their source codes.

Public information

Santa Fe is a decentralized NFT staking and lending protocol powered by Citizen Finance.

Citizen Finance is a multi-chain ecosystem powering the next generation NFT-based utilities in gaming, art and decentralized finance. With the aim to accelerate the world's transition into NFTs and immersive technologies.

Information	**Santa Fe Application**
Blockchain	Binance Smart Chain / Ethereum Chain
Language	Solidity
Contract	https://github.com/citizenfi/santafeapp/blob/master/src/contracts/nft/SantaFactoryV2.sol
Website	https://dapp.santafeapp.io/
Twitter	https://twitter.com/SantaFeapp
Telegram	https://t.me/citizenfinance
GitHub	https://github.com/citizenfi/santafeapp

Audit scope and methodology

The scope of this report is to audit the one of the smart contract codes of Santa Fe Application.

The source code in its entirety can be viewed here:

https://github.com/citizenfi/santafeapp/blob/master/src/contracts/nft/SantaFactoryV2.sol

InterFi has scanned the contract and reviewed the project for common vulnerabilities, exploits, hacks, and back-doors. Below is the list of commonly known smart contract vulnerabilities, exploits, and hacks:

Category	
Smart Contract Vulnerabilities	❖ Re-entrancy (RE) ❖ Unhandled Exceptions (UE) ❖ Transaction Order Dependency (TO) ❖ Integer Overflow (IO) ❖ Unrestricted Action (UA)
Source Code Review	❖ Gas Limit and Loops ❖ Deployment Consistency ❖ Repository Consistency ❖ Data Consistency ❖ Code Typo Error ❖ Token Supply Manipulation
Functional Assessment	❖ Access Control and Authorization ❖ Operations Trail and Event Generation ❖ Assets Manipulation ❖ Liquidity Access

InterFi methodology

The aim of this report is to analyze the smart contract and identify the vulnerabilities and the hacks in the smart contract. Mentioned are the steps used by InterFi to assess the smart contract:

1. Code review that includes the following
   - ❖ Review of the specifications, sources, and instructions provided to InterFi to make sure we understand the size, scope, and functionality of the smart contract.
   - ❖ Manual review of code, which is the process of reading source code line-byline to identify potential vulnerabilities.
2. Testing and automated analysis that includes the following
   - ❖ Test coverage analysis, which is the process of determining whether the test cases are covering the code and how much code is exercised when we run those test cases.
   - ❖ Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts

Automated 3P frameworks used to assess the vulnerabilities

- ❖ Slither
- ❖ MythX
- ❖ Uniswap V2
- ❖ Open Zeppelin
- ❖ Solidity Code Complier

General risk factors

Smart contracts are generally designed to manipulate and hold funds denominated in Ether. This makes them very tempting attack targets, as a successful attack may allow the attacker to directly steal funds from the contract. Below are the typical risk levels of a smart contract:

Vulnerable: A contract is vulnerable if it has been flagged by a static analysis tool as such. As we will see later, this means that some contracts may be vulnerable because of a false-positive.

Exploitable: A contract is exploitable if it is vulnerable and the vulnerability could be exploited by an external attacker. For example, if the "vulnerability" flagged by a tool is in a function which requires to own the contract, it would be vulnerable but not exploitable.

Exploited: A contract is exploited if it received a transaction on Ethereum's main network which triggered one of its vulnerabilities. Therefore, a contract can be vulnerable or even exploitable without having been exploited.

Risk severity	Meaning
! Critical	This level vulnerabilities could be exploited easily, and can lead to asset loss, data loss, asset manipulation, or data manipulation. They should be fixed right away.
! High	This level vulnerabilities are hard to exploit but very important to fix, they carry an elevated risk of smart contract manipulation, which can lead to critical risk severity
! Medium	This level vulnerabilities are should be fixed, as they carry an inherent risk of future exploits, and hacks which may or may not impact the smart contract execution.
! Low	This level vulnerabilities can be ignored. They are code style violations, and informational statements in the code. They may not affect the smart contract execution

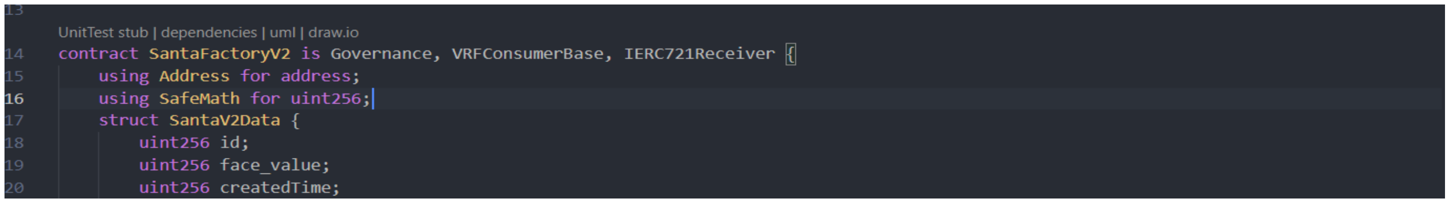
Audit Overview

Contract Calls

No	Query	Tested	Verdict
1	"@openzeppelin/contracts/math/SafeMath.sol"	Yes	No Threat
2	"@openzeppelin/contracts/token/ERC721/IERC721.sol"	Yes	No Threat
3	"@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol"	Yes	No Threat
4	"@openzeppelin/contracts/utils/Address.sol"	Yes	No Threat
5	"@openzeppelin/contracts/token/ERC20/ERC20.sol"	Yes	No Threat
6	"@chainlink/contracts/src/v0.7/VRFConsumerBase.sol"	Yes	No Threat
7	"../library/Governance.sol"	Yes	No Threat
8	"../interface/INftAsset.sol"	Yes	No Threat
9	"./SantaV2.sol"	Yes	No Threat

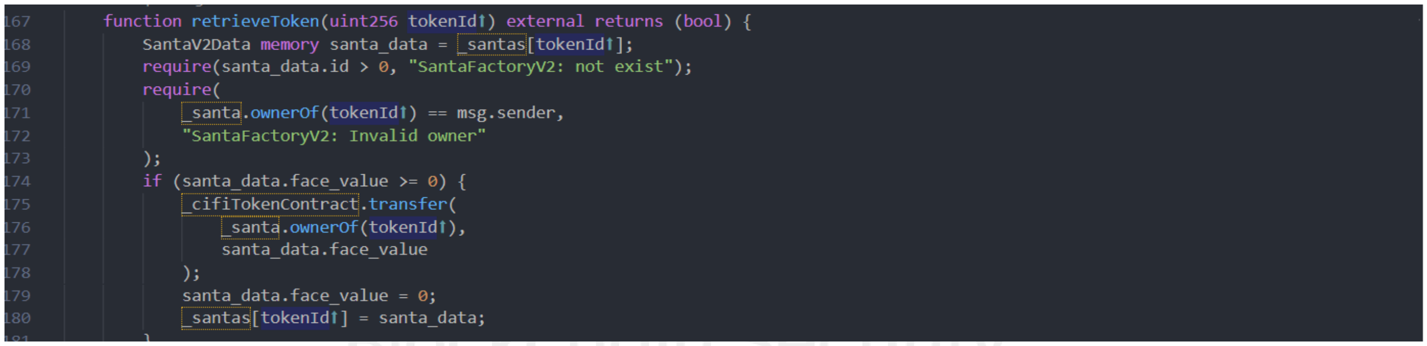
Points To Note

1. The smart contract utilizes the **SafeMath to prevent Integer Overflow.**



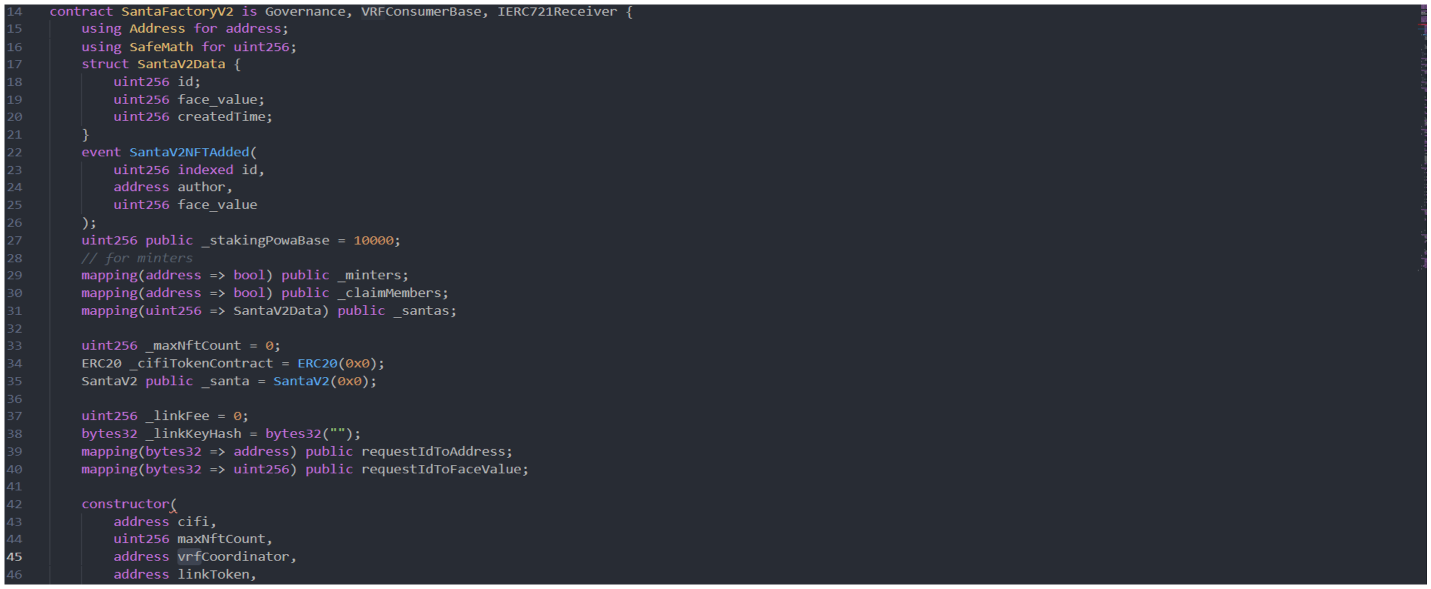
```
UnitTest stub | dependencies | uml | draw.io
contract SantaFactoryV2 is Governance, VRFConsumerBase, IERC721Receiver {
    using Address for address;
    using SafeMath for uint256;
    struct SantaV2Data {
        uint256 id;
        uint256 face_value;
        uint256 createdTime;
```

2. The smart contract enables users to lock the specific tokens to mint NFT as a face value. These NFTs are added as users' assets. **These locked tokens can only be retrieved by the NFT asset owner.**



```
    function retrieveToken(uint256 tokenId) external returns (bool) {
        SantaV2Data memory santa_data = _santas[tokenId];
        require(santa_data.id > 0, "SantaFactoryV2: not exist");
        require(
            _santa.ownerOf(tokenId) == msg.sender,
            "SantaFactoryV2: Invalid owner"
        );
        if (santa_data.face_value >= 0) {
            _cifiTokenContract.transfer(
                _santa.ownerOf(tokenId),
                santa_data.face_value
            );
            santa_data.face_value = 0;
            _santas[tokenId] = santa_data;
```

3. Santa Fe uses Chainlink VRF "@chainlink/contracts/src/v0.7/VRFConsumerBase.sol". It is used to randomly allocate tokenid when the users mint NFT. **Santa Fe utilizes VRF in this way to ensure transparency, and build trust.**

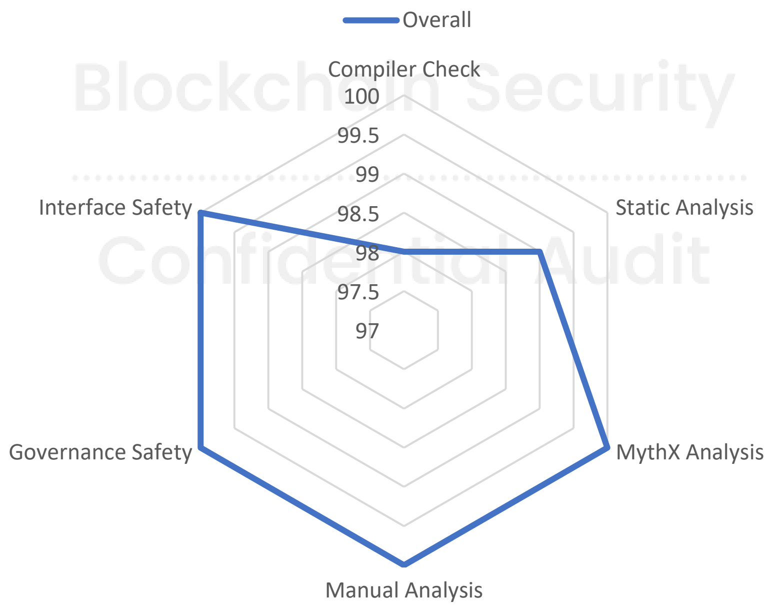


```
contract SantaFactoryV2 is Governance, VRFConsumerBase, IERC721Receiver {
    using Address for address;
    using SafeMath for uint256;
    struct SantaV2Data {
        uint256 id;
        uint256 face_value;
        uint256 createdTime;
    }
    event SantaV2NFTAdded(
        uint256 indexed id,
        address author,
        uint256 face_value
    );
    uint256 public _stakingPowaBase = 10000;
    // for minters
    mapping(address => bool) public _minters;
    mapping(address => bool) public _claimMembers;
    mapping(uint256 => SantaV2Data) public _santas;

    uint256 _maxNftCount = 0;
    ERC20 _cifiTokenContract = ERC20(0x0);
    SantaV2 public _santa = SantaV2(0x0);

    uint256 _linkFee = 0;
    bytes32 _linkKeyHash = bytes32("");
    mapping(bytes32 => address) public requestIdToAddress;
    mapping(bytes32 => uint256) public requestIdToFaceValue;

    constructor(
        address cifi,
        uint256 maxNftCount,
        address vrfCoordinator,
        address linkToken,
```

Vulnerability	Status
Compiler errors	**! Low**
Re-entrancy. Race conditions and cross function race conditions (RE)	**Passed**
Possible delays in data delivery	**Passed**
Gas optimization	**Passed**
Integer Underflow and overflow	**Passed**
Oracle Calls	**Passed**
Call stack depth attack	**Passed**
Parity Multisig Bug	**Passed**
Tx ordering dependency (TO)	**Passed**
DOS with revert and block gas limit	**Passed**
Private user data leaks	**Passed**
Malicious event log	**Passed**
Safe open zeppelin contract implementation and usage	**Passed**
The impact of exchange rate on the logic	**Passed**
Functions that are not used (dead-code)	**Passed**
Typographical Errors	**Passed**
Signature Malleability	**Passed**
Floating Pragma	**Passed**
Scoping and declarations	**Passed**

Risk Severity	Status
! Critical	None critical severity issues identified
! High	None high severity issues identified
! Medium	None medium severity issues identified
! Low	(1) Low severity issues identified Expected identifier, got 'LParen' solc [42,16] **(No Functional Impact)** Outdated compiler **(No Functional Impact)**

Assessment



Conclusion

InterFi team has performed a line-by-line manual analysis and automated review of the smart contract. The smart contract was analyzed mainly for common smart contract vulnerabilities, exploits, and manipulation hacks.

SantaFactoryV2.sol smart contract code has LOW RISK SEVERITY.

SantaFactoryV2.sol smart contract code has PASSED the InterFi's ECHELON-1 standard smart contract audit.

Auditor's Notes:

- **The smart contract enables users to lock the specific tokens to mint NFT as a face value**
- **These locked tokens can only be retrieved by the NFT asset owner. No other party has can access these locked tokens.**
- **The smart contract utilizes the SafeMath to prevent Integer Overflow.**
- **Santa Fe utilizes Chainlink VRF to randomly allocate tokenid. It ensures transparency.**
- **Only the source code in the scope has been audited by InterFi. No other source codes developed/deployed by Citizen Finance have been checked for this particular audit.**

Disclaimer

InterFi Network provides contract auditing and project verification services for blockchain projects. The purpose of the audit is to analyse the on-chain smart contract source code, and to provide basic overview of the project. **This report should not be transmitted, disclosed, referred to, or relied upon by any person for any purposes without InterFi's prior written consent.**

InterFi provides the easy-to-understand assessment of the project, and the smart contract (otherwise known as the source code). The audit makes no statements or warranties on the security of the code. It also cannot be considered as an enough assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have used all the data at our disposal to provide the transparent analysis, 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. **Be aware that smart contracts deployed on a blockchain aren't resistant from external vulnerability, or a hack. Therefore, InterFi does not guarantee the explicit security of the audited smart contract.**

The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

This report should not be considered as an endorsement or disapproval of any project or team. The information provided on this report does not constitute investment advice, financial advice, trading advice, or any other sort of advice and you should not treat any of the report's content as such. Do conduct your own due diligence and consult your financial advisor before making any investment decisions.

About InterFi Network

InterFi Network provides intelligent blockchain solutions. InterFi is developing an ecosystem that is seamless and responsive. Some of our services: Blockchain Security, Token Launchpad, NFT Marketplace, etc. **InterFi's mission is to interconnect multiple services like Blockchain Security, DeFi, Gaming, and Marketplace under one ecosystem that is seamless, multi-chain compatible, scalable, secure, fast, responsive, and easy-to-use.**

InterFi is built by a decentralized team of UI experts, contributors, engineers, and enthusiasts from all over the world. Our team currently consists of 6+ core team members, and 10+ casual contributors. **InterFi provides manual, static, and automatic smart contract analysis, to ensure that project is checked against known attacks and potential vulnerabilities.**

For more information, visit https://interfi.network

To book an audit, message https://t.me/interfiaudits