Sovi Finance

Security Assessment

January 10th, 2021

For :
Sovi Finance

By :

Zach Zhou @ CertiK
jun.zhou@certik.org

Rudolph Wang @ CertiK
shaozhong.wang@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has indeed completed a round of auditing with the intention to increase the quality of the company/product’s IT infrastructure and or source code.

Overview

Project Summary

Project Name	sovi-contract
Description	A Defi project，user can gain rewards by staking and recruiting
Platform	Ethereum; Solidity
Codebase	GitHub Repository
Commit	2cdca3d37e3aae70d930617dfef3ffa6554dfe40

Audit Summary

Delivery Date	Jan. 10, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	Jan. 6, 2021 - Jan. 10, 2021

Vulnerability Summary

Total Issues	14
Total Critical	0
Total Major	0
Total Minor	6
Total Informational	8

Executive Summary

This report has been prepared for **Sovi** protocol to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Dynamic Analysis, Static Analysis, and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

Additionally, this audit is based on a premise that all parameters passed in from outside are legal, and the implementation of external contract corresponding to the address is safe. Furthermore, this protocol has several unimplemented functions. The system should only be used after these functions to be implemented safely.

Github link had been changed from https://github.com/sovietfinance/sovi-contract to https://github.com/Sovi-Finance/sovi-contract. The audited commit is 2cdca3d37e3aae70d930617dfef3ffa6554dfe40 and the files included in the scope were as below：

File in Scope

ID	Contract	SHA-256 Checksum
BP	IBadgePool.sol	74198294438ba0ff87f17cc2ca3aabcfa125c8fe8b7ca2c045411097549b93de
SP	SoviProtocol.sol	4d8ceb080a31d4792de36ab46796548b3f52d118836c26608e1e2381d28963ce
ST	SoviToken.sol	0c87c43a8100d629aae305515361b6e926ae2de7f33df6f0b5518723b0df8a17
IR	IReferral.sol	71eb298c5530e9d18e62db88a61d7bc341eac5f148e517b302618d2c02918288

Documentation

The sources of truth regarding the operation of the contracts in scope were lackluster and are something we advise to be enriched to aid in the legibility of the codebase as well as project. To help aid our understanding of each contract's functionality we referred to in-line comments and naming conventions.

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

Review Notes

Certain optimization steps that we pinpointed in the source code mostly referred to coding standards and inefficiencies, however 6 minor vulnerabilities were identified during our audit that solely concerns the specification.

Certain discrepancies between the expected specification and the implementation of it were identified and were relayed to the team, however they pose no type of vulnerability and concern an optional code path that was unaccounted for.

The project has adequate doumentation and specification outside of the source files, and the code comment coverage is good.

Recommendations

Overall, the codebase of the contracts should be refactored to assimilate the findings of this report, enforce linters and / or coding styles as well as correct any spelling errors and mistakes that appear throughout the code to achieve a high standard of code quality and security.

Findings

ID	Title	Type	Severity	Resolved
BP-01	Several functions are not implemented	Optimization	Minor	
SP-01	Old Compiler Version Declaration	Optimization	Informational	✓
SP-02	Proper Usage of “public” and “external” type	Gas Optimization	Informational	✓
SP-03	State variables that could be declared constant	Gas Optimization	Informational	✓
SP-04	Function of DevAddr	Optimization	Informational	✓
SP-05	Constant value that could be declared as a variable	Coding Style	Informational	✓
SP-06	Incorrect condition	Logical Issue	Minor	✓
SP-07	Emit an event for updatePool	Optimization	Informational	✓
SP-08	Logic loophole of calculation model of block rewards	Logical Issue	Minor	
SP-09	Unused calculation result	Logical Issue	Minor	✓
SP-10	Missing zero address validation	Optimization	Informational	✓
SP-11	Incosistency of Initial reward per block	Inconsistency	Minor	
SP-12	Reentrancy Issue	Logical Issue	Minor	✓
SP-13	Missing check for specific pool	Optimization	Informational	✓

BP-01: Several functions are not implemented

Type	Severity	Location
Optimization	Minor	IBadgePool.sol, IReferral.sol

Description:

`getYieldAddition()`, `migrate()`, `getReferrals()`, `getInvitees()` are not implemented in all the contracts we audit this time.

Recommendation:

Consider implementing these functions.

Alleviation:

No Alleviation.

(**Sovi Finance - Response**) We will implement these functions in our second phase of the plan.

(**Certik - Response**) The system should only be used after these functions to be implemented safely.

SP-01: Old Compiler Version Declaration

Type	Severity	Location
Optimization	Informational	SoviProtocol.sol L1, SoviToken.sol L1, IReferral.sol L1, interface IBadgePool.sol L1

Description:

`solc` frequently releases new compiler versions. Using an old version prevents access to new Solidity security checks.

Recommendation:

Deploy with any of the following Solidity versions:

- 0.6.8,
- 0.6.10 - 0.6.11. Use a simple pragma version that allows any of these versions. Consider using the latest version of Solidity for testing.

Alleviation:

The team heeded our advice and locked the version of their contracts at version 0.6.8, ensuring that compiler-related bugs can easily be narrowed down should they occur.

The recommendations were applied in commit 62f3e9e7e91f8a000b6d07eabe300a40b3671e73.

SP-02: Proper Usage of "public" and "external" type

Type	Severity	Location
Gas Optimization	Informational	SoviProtocol.sol L127,L146,L156,L371,L395,L424,L443, SoviToken.sol L10

Description:

"public" functions that are never called by the contract could be declared "external" . When the inputs are arrays "external" functions are more efficient than "public" functions.
Examples
Functions like : `add()` , `set()` , `setBadgePool()` , `deposit()` , `withdraw()` , `emergencyWithdraw()` , `dev()` , `mint()`

Recommendation:

Consider using the "external" attribute for functions never called from the contract.

Alleviation:

The team heeded our advice and used the “external” attribute for functions never called from the contract.
The recommendations were applied in commit 0b35badae0303620e6bf67516e52b9a02e0b4d14.

SP-03: State variables that could be declared constant

Type	Severity	Location
Gas Optimization	Informational	SoviProtocol.sol L78,L80,L81,L82,L83,L84,L86,L87,L88

Description:

Constant state variables should be declared constant to save gas.

Recommendation:

Consider changing it as following example:

```
    uint256 public constant TEN = 10;
```

Alleviation:

The team heeded our advice and declared all variables which won't be changed in the contract as constant variables. The recommendations were applied in commit 0b35badae0303620e6bf67516e52b9a02e0b4d14.

SP-04: Function of DevAddr

Type	Severity	Location
Optimization	Informational	SoviProtocol.sol

Description:

We noticed that some pool rewards will be minted to the DevAddr and DevAddr can be changed by previous dev.

Recommendation:

Using multisig to avoid malicous operation.

Alleviation:

No alleviation.

(**Sovi Finance - Response**) The devAddr receives constant 10% of the liquidity mining production, and inherited from Sushi code, this could be modified yes, we will be applying with multi-sig address for the team to co-manage the fund for good use.

(**Certik - Response**) Could you please give us your muli-sig address?

(**Sovi Finance - Response**) ethereum: 0x18fa7efB817bC97AF4E274BF640eCbE6c1948660 so far there is no multi-sig solution on heco, therefore we will use the deploy account to receive instead.

(**Certik - Response**) Could you please give a detail introduction of your multisig design? Do you have a MultiSigWalletWithTimelock and what is its address?

(**Sovi Finance - Response**) Multi-signature uses gnosis safe. The team includes technology and operations. Only when multiple parties complete the signature will they use the assets.

SP-05: Constant value that could be declared as a variable

Type	Severity	Location
Coding Style	Informational	SoviProtocol.sol L78,L181,L183,L241,L420

Description:

`1e18` is used too many times in the contract.

Recommendation:

Consider declaring it as a constant variable like below:

```
    uint256 public constant decimal = 1e18;
    ......
    uint256 public REBATE_HSOV_BASE = uint256(1000).mul(decimal);
```

Alleviation:

The team heeded our advice and declared a constant variable to store `1e18` which will be used in the subsequent codes. The recommendations were applied in commit 0b35badae0303620e6bf67516e52b9a02e0b4d14.

SP-06: Incorrect condition

Type	Severity	Location
Logical Issue	Minor	SoviProtocol.sol L200

Description:

The `if` condition in the loop of `selectInvitees` should be used to judge the length of invitees of `u` but not `_addr` since the recursion is used to traverse invitees of `u` .

```
    if (hSOV.getInvitees(_addr).length > 0) {
        (uint256 _count, uint256 _amount) = selectInvitees(_pid, u);
        _total_count = _total_count.add(_count);
        _total_amount = _total_amount.add(_amount);
    }
```

Recommendation:

Consider changing `if` condition like below:

```
    if (hSOV.getInvitees(u).length > 0) {
        (uint256 _count, uint256 _amount) = selectInvitees(_pid, u);
        _total_count = _total_count.add(_count);
        _total_amount = _total_amount.add(_amount);
    }
```

Alleviation:

The team heeded our advice and changed the code.
The recommendations were applied in commit 0b35badae0303620e6bf67516e52b9a02e0b4d14.

SP-07: Emit an event for updatePool

Type	Severity	Location
Optimization	Informational	SoviProtocol.sol

Description:

`updatePool()` will update reward variables of the given pool to be up-to-date. These variables are sensitive .

Recommendation:

Emit an event for these sensitive parameter changes.

Alleviation:

The team heeded our advice and added a new event which will be emitted when calling `updatePool()` .
The recommendations were applied in commit 0b35badae0303620e6bf67516e52b9a02e0b4d14.

SP-08: Logic loophole of calculation model of block rewards

Type	Severity	Location
Logical Issue	Minor	SoviProtocol.sol L236

Description:

`updatePool()` will calculate block reward by multiplying `rewardPerBlock` and the difference of `block.number` and `pool.lastRewardBlock`. But `rewardPerBlock` will be changed based on different block interval. Current logic may result in a wrong calculation result.

For example:

If `pool.lastRewardBlock` is 93045, `block.number` is 400000, `rewardPerBlock` is 5. From block number 93045 to 400000, there are three different interval(93045~232609, 232610~372175, 372176~400000) with different `rewardPerBlock` (3.5000, 2.4500, 1.7150). `reductionPercent` is used to calculate `rewardPerBlock`. Based on current logic, if `updatePool()` is called in the third interval but not the first two, `rewardPerBlock` will be set to 5*70% (3.5000) directly. `rewardPerBlock` did not changed in the first two interval since `updatePool()` was not called during that period. Using this value to multiply the whole block interval is not correct(blockReward = (400000-93045) *3.5000). The right calculation is as below:

From 93045 to 232609, reward1 = (232609-93045)*3.5000;

From 232610 to 372175, reward2 = (372175-232610)*2.4500;

From 372176 to 400000, reward3 = (400000-372176)*1.7150;

blockReward = reward1+reward2+reward3

Recommendation:

Consider defining a function to return `rewardPerBlock` of each block interval. Then calculate block reward by calculate the sum of each child interval result.

Alleviation:

No alleviation.

(**Sovi Finance - Response**) We will avoid this problem in business, make sure to harvest every cycle.

SP-09: Unused calculation result

Type	Severity	Location
Logical Issue	Minor	SoviProtocol.sol L269,L271

Description:

The results of `_badgePoolReward.add(levelReward)` and `_thisReward.add(levelReward)` are not used. Based on the logic of `setRebateReward`, both of the results shoud be assigned to related variables.

```
    for (uint256 idx = 0; idx < level; idx ++) {
        if (userInfo[_pid][refs[idx]].extraRebateRatio == 0) {
            _badgePoolReward.add(levelReward);
        } else {
            _thisReward.add(levelReward);
            userInfo[_pid][refs[idx]].pendingRebate = userInfo[_pid]
    [refs[idx]].pendingRebate.add(levelReward);
        }
    }
```

Recommendation:

Consider changing codes like below:

```
    for (uint256 idx = 0; idx < level; idx ++) {
        if (userInfo[_pid][refs[idx]].extraRebateRatio == 0) {
            _badgePoolReward = _badgePoolReward.add(levelReward);
        } else {
            _thisReward = _thisReward.add(levelReward);
            userInfo[_pid][refs[idx]].pendingRebate = userInfo[_pid]
    [refs[idx]].pendingRebate.add(levelReward);
        }
    }
```

Alleviation:

The team heeded our advice and assigned calculation result to specific variables.
The recommendations were applied in commit 0b35badae0303620e6bf67516e52b9a02e0b4d14.

SP-10: Missing zero address validation

Type	Severity	Location
Optimization	Informational	SoviProtocol.sol L95,L96,L97,L99,L100,L127,L169,L188,L311,L324

Description:

Detect missing zero address validation.

Recommendation:

Consider adding validation for address like below:

```
    constructor(
            SoviToken _SOVI,
            IReferral _hSOV,
            address _devAddr,
            uint256 _startBlock,
            address usdtContract,
            address uniPoolAddress
        ) public {
        require(_SOVI != address(0), "SoviProtocol: SOVI is the zero address");
        require(_hSOV != address(0), "SoviProtocol: hSOV is the zero address");
        require(_devAddr != address(0), "SoviProtocol: devAddr is the zero address");
        require(usdtContract != address(0), "SoviProtocol: usdtContract is the zero address");
        require(uniPoolAddress != address(0), "SoviProtocol: uniPoolAddress is the zero
    address");
        ......
        SOVI = _SOVI;
        devAddr = _devAddr;
        ......
        hSOV = _hSOV;
        USDT_CONTRACT = ERC20(usdtContract);
        SOVI_POOL_ADDRESS = address(uniPoolAddress);
    }
```

Alleviation:

The team heeded our advice and added validation for zero address.
The recommendations were applied in commit 0b35badae0303620e6bf67516e52b9a02e0b4d14.

SP-11: Incosistency of Initial reward per block

Type	Severity	Location
Inconsistency	Minor	SoviProtocol.sol L107

Description:

Intial reward per block in the contract is set to 10, but white paper declared this value as 5.

Recommendation:

Consider changing `rewardPerBlock` of the contract to 5 or udpating white paper. Please ensure consistency on both sides.

Alleviation:

No alleviation.

(**Sovi Finance - Response**) Adjustments are in progress and will be adjusted to be consistent before going launch.

SP-12: Reentrancy Issue

Type	Severity	Location
Logical Issue	Minor	SoviProtocol.sol L424~430

Description:

`emergencyWithdraw()` that could be called repeatedly, before the first invocation of the function was finished. This may cause the different invocations of the function to interact in destructive ways.

Since the user's balance is not set to 0, the second (and later) invocations will still succeed, and will withdraw the balance over and over again.

```
    function emergencyWithdraw(uint256 _pid) public {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        pool.lpToken.safeTransfer(address(msg.sender), user.amount);
        emit EmergencyWithdraw(msg.sender, _pid, user.amount);
        delete userInfo[_pid][msg.sender];
    }
```

Recommendation:

Consider declaring a temporary variable to store `user.amount` . Clear `user.amount` before `pool.lpToken.safeTransfer()` . Sample is as below:

```
    function emergencyWithdraw(uint256 _pid) public {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        uint256 amount = user.amount;
        user.amount = 0;
        pool.lpToken.safeTransfer(address(msg.sender), amount);
        emit EmergencyWithdraw(msg.sender, _pid, user.amount);
        delete userInfo[_pid][msg.sender];
    }
```

Alleviation:

The team heeded our advice and changed the code.
The recommendations were applied in commit 0b35badae0303620e6bf67516e52b9a02e0b4d14.

SP-13: Missing check for specific pool

Type	Severity	Location
Optimization	Informational	SoviProtocol.sol L127,L146,L169,L188,L218,L311,L345,L357,L371,L395,L424

Description:

It is possible to add duplicate `lpToken` to pools by `add()` .

It is possible that `_pid` is not exists.

Recommendation:

Consider declaring a map that used to record if `lpToken` had been added.

For example:

```
    mapping(address => bool) public existed;
    ......
    function add(uint256 _allocPoint, IERC20 _lpToken, bool _withUpdate, bool _enableRebate)
    public onlyOwner {
            require(!existed[_lpToken], "SoviProtocol: lpToken had been added!");
            //Minor check lp existed
            if (_withUpdate) {
                massUpdatePools();

            ......
    }
```

Consider adding validation for `_pid` like below:

```
    function set(uint256 _pid, uint256 _allocPoint, bool _enableRebate, bool _withUpdate) public
    onlyOwner {
            require(_pid < poolInfo.length && _pid > 0, "SoviProtocol: _pid is not exist");
            if (_withUpdate) {
                massUpdatePools();
            }
            totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
            poolInfo[_pid].allocPoint = _allocPoint;
            poolInfo[_pid].enableRebate = _enableRebate;
    }
```

Alleviation:

The team heeded our advice and added validation for pools.
The recommendations were applied in commit a5b81033609e09787b7cc9425f440804a73ca2ab.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an instorage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete` .

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.

Icons explanation

 : Issue resolved.

 : Issue not resolved / Acknowledged. The team will be fixing the issues in the own timeframe.

 : Issue partially resolved. Not all instances of an issue was resolved.