Solav Token

Smart Contract Security Audit

Prepared by ShellBoxes

Nov 28th, 2023 - Nov 29th, 2023

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	Solav
Version	1.0
Classification	Confidential

Scope

Contract Name	Contract Address
ERC20	0xfc5e4ed56153b57aa8ef769eba3e79e58e19be93

Re-Audit

Contract Name	Contract Address
ERC20	0xfc5e4ed56153b57aa8ef769eba3e79e58e19be93

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction

Solav engaged ShellBoxes to conduct a security assessment on the Solav Token beginning on Nov 28th, 2023 and ending Nov 29th, 2023. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Solav

SECURING ART'S VALUE IN THE BLOCKCHAIN ERA SOLAV, short for "Solution of Art Value," is dedicated to revolutionizing the art investment experience by leveraging blockchain technology for secure and transparent art authentication, ownership verification, and tracking.

Issuer	Solav
Website	`https://www.solav.io`
Type	Solidity Smart Contracts
Documentation	Solav Docs
Audit Method	Whitebox

1.2 Approach & Methodology

ShellBoxes used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.2.1 Risk Methodology

Vulnerabilities or bugs identified by ShellBoxes are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

Impact			
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low
	High	Medium	Low

Likelihood

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the Solav Token implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include , 1 medium-severity, 1 low-severity, 1 informational-severity vulnerabilities.

Vulnerabilities	Severity	Status
SHB.1. Centralized Allocation of Entire Token Supply to Deployer Address	MEDIUM	Fixed
SHB.2. Floating Pragma Directive	LOW	Acknowledged
SHB.3. Lack of Allowance Check in transferFrom Function	INFORMATIONAL	Acknowledged

3 Finding Details

SHB.1 Centralized Allocation of Entire Token Supply to Deployer Address

- Severity : MEDIUM
- Status : Fixed
- Likelihood : 3
- Impact : 1

Description:

The constructor of the ERC20 token contract allocates the entire token supply to the deployer's address, resulting in a centralized distribution model. This concentration of tokens in a single address introduces potential security risks and lacks a diversified distribution strategy.

Files Affected:

SHB.1.1: ERC20.sol

```
    constructor(string memory name_, string memory symbol_, uint256 totalSupply_) {
        _name = name_;
        _symbol = symbol_;
        _totalSupply = totalSupply_;
        _balances[_msgSender()] = totalSupply_;
        emit Transfer(address(0), _msgSender(), totalSupply_);
    }
```

Recommendation:

To address this issue, it is advisable to formulate a comprehensive tokenomic plan that includes diverse distribution strategies for team members, investors, community contributors, and strategic partners. Additionally, distributing portions of the token supply to multiple addresses mitigates the concentration risk associated with a single deployer's address, fostering decentralization and enhancing the overall resilience of the token ecosystem.

Updates

The Solav team has addressed the identified issue by executing a comprehensive token distribution according to the allocated plan. The updated list of token holders on Solav Token Holders - Etherscan attests to the successful implementation of their decentralized distribution strategy. This initiative, in alignment with the outlined Solav Ecosystem Tokenomics, serves as conclusive proof that the tokens are no longer concentrated solely in the deployer's address

SHB.2 Floating Pragma Directive

- Severity : LOW
- Status : Acknowledged
- Likelihood : 1
- Impact : 2

Description:

The contract employs a floating Solidity pragma of 0.8.0, indicating compatibility with any compiler version from 0.8.0 (inclusive) up to, but not including, version 0.9.0. This flexibility may introduce unexpected behavior if compiled with a newer compiler version that includes breaking changes.

Files Affected:

SHB.2.1: ERC20.sol

```
pragma solidity ^0.8.0;
```

Recommendation:

Consider specifying a fixed Solidity compiler version without the caret (^) to ensure consistent behavior across different compiler versions. We recommend evaluating and using a newer stable version of the compiler to benefit from the latest features, bug fixes, and optimizations.

Updates

The Solav team has acknowledged this issue and is committed to addressing it in a subsequent update.

SHB.3 Lack of Allowance Check in transferFrom Function

- Severity : INFORMATIONAL
- Status : Acknowledged
- Likelihood : 1
- Impact : 0

Description:

The transferFrom function in the ERC20 token contract lacks a crucial check on the spender's allowance before initiating a _transfer. The absence of this allowance verification creates a potential vulnerability, as the transaction may revert with an arithmetic underflow or overflow error if the spender's allowance is insufficient for the specified amount. This error message may not provide clear insights to users, leading to potential confusion and hindering the overall user experience.

Files Affected:

SHB.3.1: ERC20.sol

```
    function transferFrom(address sender, address recipient, uint256
       ↪ amount) public override returns (bool) {
        _transfer(sender, recipient, amount);
```

```
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()]
        ↪ - amount);
        return true;
    }
```

Recommendation:

Consider integrating an allowance check in the transferFrom function of the token contract to bolster security and mitigate potential arithmetic underflow or overflow errors. Additionally, implementing a custom error, such as ERC20InsufficientAllowance, can provide users with a more informative and context-specific error message, enhancing overall clarity in case of insufficient allowances.

Updates

The Solav team has acknowledged this issue, stating that they plan to address it in the future.

4 Best Practices

BP.1 Remove Unused Ownable Contract

Description:

The ERC20 token contract inherits the Ownable contract, which includes the onlyOwner modifier. Given that this modifier is not applied to any of the token's functions, the entire Ownable contract is unused. To optimize the codebase, reduce unnecessary complexity, and enhance gas efficiency, it is recommended to remove the entire unused Ownable contract.

Files Affected:

BP.1.1: ERC20.sol

```
contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address
        ↪ indexed newOwner);

    constructor() {
        _owner = _msgSender();
        emit OwnershipTransferred(address(0), _owner);
    }

    function owner() public view returns (address) {
        return _owner;
    }

    modifier onlyOwner() {
        require(_owner == _msgSender(), "Ownable: caller is not the owner
            ↪ ");
        _;
```

```
    }
}
```

Status - Acknowledged

BP.2 Utilize OpenZeppelin ERC20 Standard Contract

Description:

The token contract currently redundantly hardcodes its own ERC20 implementation, which is already well-defined in the OpenZeppelin ERC20 contract. To optimize code structure and adhere to best practices, it is recommended to import and inherit directly from the OpenZeppelin ERC20 contract. This approach eliminates unnecessary code duplication, ensures consistency with industry standards, and facilitates easier maintenance and understanding.

Files Affected:

BP.2.1: ERC20.sol

```
interface IERC20 {
    event Transfer(address indexed from, address indexed to, uint256
        ↪ value);
    event Approval(address indexed owner, address indexed spender,
        ↪ uint256 value);

    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, uint256 amount) external
        ↪ returns (bool);
    function allowance(address owner, address spender) external view
        ↪ returns (uint256);
    function approve(address spender, uint256 amount) external returns (
        ↪ bool);
```

```
    function transferFrom(address sender, address recipient, uint256
        ↪ amount) external returns (bool);
}
```

BP.2.2: ERC20.sol

```
contract ERC20 is Context, IERC20, Ownable {
```

Status - Acknowledged

BP.3 Rename Contract for Clarity

Description:

The current contract is named ERC20 a standard token name in the Ethereum ecosystem. To enhance clarity, avoid potential conflicts, and adhere to best practices, it is recommended to rename the contract to a specific name dedicated to the project or token. This practice ensures a unique and descriptive identifier, reducing the likelihood of naming clashes and providing a clear representation of the contract's purpose within the broader context of the project.

Files Affected:

BP.3.1: ERC20.sol

```
contract ERC20 is Context, IERC20, Ownable {
```

Status - Acknowledged

5 Conclusion

In this audit, we examined the design and implementation of Solav Token contract and discovered several issues of varying severity. Solav team addressed 1 issues raised in the initial report and implemented the necessary fixes, while classifying the rest as a risk with low-probability of occurrence. Shellboxes' auditors advised Solav Team to maintain a high level of vigilance and to keep those findings in mind in order to avoid any future complications.

6 Scope Files

6.1 Audit

Files	MD5 Hash
contracts/ERC20.sol	0fb319d42f29d1df9ca3d6d507f02bd9

6.2 Re-Audit

Files	MD5 Hash
contracts/ERC20.sol	0fb319d42f29d1df9ca3d6d507f02bd9

7 Disclaimer

Shellboxes reports should not be construed as ”endorsements” or ”disapprovals” of particular teams or projects. These reports do not reflect the economics or value of any ”product” or ”asset” produced by any team or project that engages Shellboxes to do a security evaluation, nor should they be regarded as such. Shellboxes Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the examined technology, nor do they provide any indication of the technology's proprietors, business model, business or legal compliance. Shellboxes Reports should not be used in any way to decide whether to invest in or take part in a certain project. These reports don't offer any kind of investing advice and shouldn't be used that way. Shellboxes Reports are the result of a thorough auditing process designed to assist our clients in improving the quality of their code while lowering the significant risk posed by blockchain technology. According to Shellboxes, each business and person is in charge of their own due diligence and ongoing security. Shellboxes does not guarantee the security or functionality of the technology we agree to research; instead, our purpose is to assist in limiting the attack vectors and the high degree of variation associated with using new and evolving technologies.

SHELLBOXES

For a Contract Audit, contact us at contact@shellboxes.com