Bonsai3 ERC20
Security Audit Report

PREPARED FOR:

Bonsai3 Crypto

ARCADIA CONTACT INFO
Email: audits@arcadiamgroup.com

Telegram: https://t.me/thearcadiagroup

Revision history

Date	Reason	Commit
11/16/2023	Initial Audit Scope	#9e89e2eb1be0667cebc7c7b673225759faab18af
11/21/2023	Review Of Remediations	#a24eb2670ab30b669ba246cc8c3d439efd728d45

Table of Contents

Executive Summary

Introduction

Bonsai3 engaged Arcadia to perform a security audit of their main smart contracts version 2. Our review of their codebase occurred on the commit hash #9e89e2eb1be0667cebc7c7b673225759faab18af

Review Team

1. Tuan “Anhnt” Nguyen - Security Researcher and Engineer
2. Joel Farris - Project Manager

Project Background

Bonsai3 offers a pioneering no-code toolkit designed by developers, aiming to be the WordPress equivalent for Web3. It empowers teams to deploy projects swiftly, affordably, and securely at any scale. The platform ensures community ownership and decentralized asset management, featuring unique tool sets for token creation, pre-sales, smart contract management, and market-making strategies. The roadmap encompasses phases focused on mechanics, community-driven features, UI/UX enhancements, and a comprehensive suite of developer tools, promising a robust evolution in line with Web3's future.

Bonsai3 SEED is a customized version of OpenZeppelin's ERC20, altering the **_transfer** function logic to include a tax on purchases and sales made through an Automated Market Maker (AMM).

Coverage

For this audit, we performed research, test coverage, investigation, and review of Bonsai3’s main contract followed by issue reporting, along with mitigation and remediation instructions as outlined in this report. The following code repositories, files, and/or libraries are considered in scope for the review.

Contracts	Lines	nLines	nSLOC	Comment Lines	Complex. Score
seed-token/Seed.sol	357	338	260	22	185

Methodology

Arcadia completed this security review using various methods, primarily consisting of dynamic and static analysis. This process included a line-by-line analysis of the in-scope contracts, optimization analysis, analysis of key functionalities and limiters, and reference against intended functionality.

The followings are the steps we have performed while auditing the smart contracts:

- Investigating the project and its technical architecture overview through its documentation
- Understanding the overview of the smart contracts, the functions of the contracts, the inheritance, and how the contracts interface with each others thanks to the graph created by Solidity Visual Developer
- Manual smart contract audit:
    - Review the code to find any issue that could be exploited by known attacks listed by Consensys
    - Identifying which existing projects the smart contracts are built upon and what are the known vulnerabilities and remediations to the existing projects
    - Line-by-line manual review of the code to find any algorithmic and arithmetic related vulnerabilities compared to what should be done based on the project's documentation
    - Find any potential code that could be refactored to save gas
    - Run through the unit-tests and test-coverage if exists
- Static Analysis:
    - Scanning for vulnerabilities in the smart contracts using Static Code Analysis Software
    - Making a static analysis of the smart contracts using Slither
- Additional review: a follow-up review is done when the smart contracts have any new update. The follow-up is done by reviewing all changes compared to the audited commit revision and its impact to the existing source code and found issues.

Summary

There were **6** issues found, **1** of which were deemed to be ‘critical’, and **1** of which were rated as ‘high’. At the end of these issues were found throughout the review of a rapidly changing codebase and not a final static point in time.

Severity Rating	Number of Original Occurrences	Number of Remaining Occurrences
CRITICAL	1	0
HIGH	1	0
MEDIUM	0	0
LOW	0	0
INFORMATIONAL	4	2

Findings in Manual Audit

(BONE-1) Potential execution issue with splitting fee transactions.

Status

Resolved

Risk Level

Severity: Critical, likelihood: High.

Code Segment

```
function swapBack() private {
    uint256 contractBalance = balanceOf(address(this));
    uint256 totalTokensToSwap = tokensForBuyback + tokensForMktg;
    bool success;

    if (contractBalance == 0 || totalTokensToSwap == 0) {
        return;
    }

    if (contractBalance > swapTokensAtAmount * 20) {
        contractBalance = swapTokensAtAmount * 20;
    }

    swapTokensForEth(contractBalance);

    uint256 ethBalance = address(this).balance;

    uint256 ethForBuyBack = ethBalance.mul(tokensForBuyback).div(
        totalTokensToSwap
    );
    uint256 ethForMtkg = ethBalance.mul(tokensForMktg).div(
        totalTokensToSwap
    );

    tokensForBuyback = 0;
```

```
    tokensForMktg = 0;

    (success, ) = address(mtkgWallet).call{value: ethForMtkg}("");
    (success, ) = address(buyBackWallet).call{value: ethForBuyBack}("");
}
```

Description

Following the execution of **swapTokensForEth(contractBalance)** another **_transfer** function is called, leading to an increment in **tokensForBuyback** and **tokensForTreasury**. This scenario results in **ethForBuyBack + ethForMtkg > ethBalance** thus leading to a failure in one or both sending ether transactions.

Recommendation

Utilize the **buyBuybackFee**, **sellBuybackFee**, **buyTreasuryFee** and **sellTreasuryFee** variables to distribute the ether balance.

(BONE-2) Contract code heavily relies on ***tx.origin*** rather than ***msg.sender*** for authorization.

Status

Resolved

Risk Level

Severity: High, likelihood: High.

Description

The entire contract code heavily relies on tx.origin rather than msg.sender for authorization, potentially exposing vulnerabilities when using call (sending ETH or calling other contracts).

Besides the issue with authorization, there is a chance that tx.origin will be removed from the Ethereum protocol in the future, so code that uses tx.origin won't be compatible with future releases Vitalik: 'Do NOT assume that tx.origin will continue to be usable or meaningful.'

It's also worth mentioning that by using tx.origin you're limiting interoperability between contracts because the contract that uses tx.origin cannot be used by another contract as a contract can't be the tx.origin (ie proxies ..)

Recommendation

Using **`msg.sender`** provides better security by referring to the direct caller of a function, which helps prevent certain types of attacks or unauthorized access compared to using **`tx.origin`**.

(BONE-3) Use SafeMath with solidity 0.8.10.

Status

Resolved

Risk Level

Severity: Informational, likelihood: High

Description

The code continues to utilize Safemath and employs SafeMath syntax (mul/div/add). However, with Solidity 0.8.x, the use of SafeMath becomes redundant due to built-in overflow/underflow checks. This redundancy increases code size and gas costs.

Recommendation

Remove SafeMath and use built-in operations of solidity.

(BONE-4) Absence of test code.

Status

Acknowledged

Reason

The team has acknowledged the problem but has decided not to address it.

Risk Level

Severity: Informational

Description

The absence of any test code poses a significant risk for potential issues in the future.

Recommendation

To ensure comprehensive coverage, it is strongly recommended to cover at least 90% of the code.

(BONE-5) Gas Costs Surge 2-4x Normal Rates.

Status

Acknowledged

Risk Level

Severity: Informational, likelihood: High

Description

The project's customized logic within the _transfer function triggers a 2-3x surge in gas fees for typical ERC20 transfers, averaging around 159,026 gas. This inconvenience might impact users using products of the Bonsai3 ecosystem.

(BONE-6) Low code quality.

Status

Resolved

Risk Level

Severity: Informational, likelihood: High

Description

There's significant room for code improvement, including eliminating intermediate variables, shortening functions, and removing redundant or unused code/logic. Some issues can be prevented by implementing straightforward tests.

Disclaimer

While best efforts and precautions have been taken in the preparation of this document, The Arcadia Group and the Authors assume no responsibility for errors, omissions, or damages resulting from the use of the provided information. Additionally, Arcadia would like to emphasize that the use of Arcadia's services does not guarantee the security of a smart contract or set of smart contracts and does not guarantee against attacks. One audit on its own is not enough for a project to be considered secure; that categorization can only be earned through extensive peer review and battle testing over an extended period.