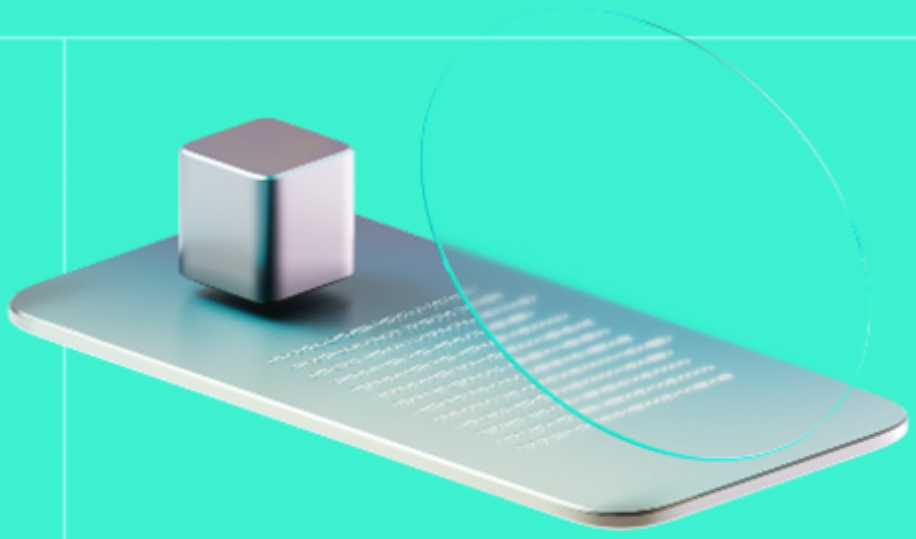
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Avive

Date: 18/12/2023

We thank Avive for allowing us to conduct a Smart Contract Security Assessment. This document outlines our methodology, limitations, and results of the security assessment.

Avivie is an ERC20 token.

Platform: Artbitrum

Language: Solidity

Tags: ERC20 Burnable, ERC20 Pausable

Timeline: 12.12.2023 - 14.12.2023

Methodology: https://hackenio.cc/sc_methodology

Last Review Scope

Repository	https://github.com/AviveWorld/Token/tree/main/
Commit	c1232b2933b42c7b8ab0e7902a763f3e2f73c147

Audit Summary

10/10	9/10	75%	10/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 9.8/10

The system users should acknowledge all the risks summed up in the risks section of the report

2	0	2	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	0
Medium	0
Low	2

Vulnerability	Status
F-2023-0120 - Ownership Irrevocability Vulnerability	Accepted
F-2023-0121 - Single-Step Ownership Transfer	Accepted

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Avive
Audited By	Vladyslav Khomenko, Eren Gonen
Approved By	Przemyslaw Swiatowiec
Website	https://www.avive.world/
Changelog	14/12/2023 - Preliminary Report

Table to Contents

System Overview

Avive is a ERC20 token with the following contract:

Avive — simple ERC-20 token with burnable functionality that mints all initial supply to a deployer. Additional minting is not allowed.

It has the following attributes:

- Name: Avive
- Symbol: Avive
- Decimals: 18
- Total supply: 10 billion.

Privileged roles

- The owner of the Avive contract can pause or unpause the token transfer for all users at any time.

Executive Summary

This report presents an in-depth analysis and scoring of the Customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are provided.
  - NatSpec's are good
- Technical description is provided.

Code quality

The total Code Quality score is **9** out of **10**.

- The development environment is configured.
- Missing zero address checks were identified.

Test coverage

Code coverage of the project is **75%** (branch coverage)

- Deployment and basic interactions are covered with tests.

Security score

Upon auditing, the code was found to contain **0** critical, **0** high, **0** medium, and **2** low severity issues, leading to a security score of **10** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the Customer's smart contract yields an overall score of **9.8**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- The owner of the **Avive** contract has the authority to pause or unpause token transfers for all users at any time, without prior notice. If the contract is paused, users will not be able to transfer their tokens.
- The supply is not hard-coded into the contract but rather set on deployment.

Findings

Vulnerability Details

F-2023-0120 - Ownership Irrevocability Vulnerability - Low

Description:

The smart contract under inspection inherits from the `Ownable` library, which provides basic authorization control functions. The contract in question allows the owner to pause/unpause token use. The contract simply retains the default `renounceOwnership` function and `transferOwnership` function from `Ownable` library.

Given this, once the owner renounces ownership using the `renounceOwnership` function, the contract becomes ownerless. As evidenced in the provided transaction logs, after the `renounceOwnership` function is called, attempts to call functions that require owner permissions fail with the error message: `OwnableUnauthorizedAccount`.

This state renders the contract's adjustable parameters immutable and potentially makes the contract useless for any future administrative changes that might be necessary.

Assets:

- Avive.sol [https://github.com/AviveWorld/Token/commits/main]
- Avive.sol [https://github.com/AviveWorld/Token/commits/hacken-initial-audit/]

Status: Accepted

Classification

Severity: Low

Impact: 2/5

Recommendations

Recommendation: To mitigate this vulnerability:

- Override the `renounceOwnership` function to revert transactions: By overriding this function to simply revert any transaction, it will become impossible for the contract owner to unintentionally (or intentionally) render the contract ownerless and thus immutable.

F-2023-0121 - Single-Step Ownership Transfer - Low

Description:

The current implementation of the **Avive** contract utilizes OpenZeppelin's **Ownable.sol**, which facilitates a single-step process for ownership transfer. This approach, while straightforward, does not include a verification step for the new owner address before finalizing the transfer. The absence of such a precautionary measure can lead to significant security and operational risks, particularly if an incorrect address is provided during the ownership transfer process. Mistakes or malicious activities could result in the permanent transfer of ownership to an unintended address, potentially leading to loss of control over the contract's administrative functionalities.

Security Risks:

The single-step ownership transfer process increases the risk of accidental or malicious transfers, as there is no opportunity to verify or cancel the transfer once initiated.

Operational Risks:

An incorrect transfer of ownership could result in administrative functions becoming inaccessible, potentially crippling the contract's operations and management.

Assets:

- Avive.sol [https://github.com/AviveWorld/Token/commits/hacken-initial-audit/]
- Avive.sol [https://github.com/AviveWorld/Token/commits/main]

Status: Accepted

Classification

Severity: Low

Impact: 2/5

Recommendations

Recommendation:

Implement the **Ownable2Step** extension or a similar mechanism that introduces a two-step process for ownership transfer. This process typically involves nominating a new owner and then requiring a separate confirmation step to finalize the transfer.

Observation Details

F-2023-0119 - Floating Pragma - Info

Description:

A **floating pragma** in Solidity refers to the practice of using a pragma statement that does not specify a fixed compiler version but instead allows the contract to be compiled with any compatible compiler version. This issue arises when pragma statements like pragma solidity ^0.8.0 are used without a specific version number, allowing the contract to be compiled with the latest available compiler version. This can lead to various compatibility and stability issues.

Version Compatibility: Using a floating pragma makes the contract susceptible to potential breaking changes or unexpected behavior introduced in newer compiler versions. Contracts that rely on specific compiler features or behaviors may break when compiled with a different version.

Interoperability Issues: Contracts compiled with different compiler versions may have compatibility issues when interacting with each other or with external services. This can hinder the interoperability of the contract within the Ethereum ecosystem.

The project uses floating pragma `^0.8.20`.

Path: contracts/Avive.sol

Assets:

- Avive.sol [https://github.com/AviveWorld/Token/commits/hacken-initial-audit/]
- Avive.sol [https://github.com/AviveWorld/Token/commits/main]

Status: Fixed

Recommendations

Recommendation:

To mitigate these risks, it is recommended to use a fixed pragma statement that specifies a known, well-tested compiler version. This helps ensure the stability, security, and predictability of the smart contract throughout its lifecycle.

F-2023-0122 - Missing Zero Address Validation - Info

Description:

In Solidity, the Ethereum address `0x0000000000000000000000000000000000000000` is known as the "**zero address**". This address has significance because it is the default value for uninitialized address variables and is often used to represent an invalid or non-existent address.

The "**Missing zero address control**" issue arises when a Solidity smart contract does not properly check or prevent interactions with the zero address, leading to unintended behavior.

For instance, consider a contract that includes a function to change its owner. This function is crucial, as it determines who has administrative access. However, if this function lacks proper validation checks, it might inadvertently permit the setting of the owner to the zero address. Consequently, the administrative functions will become unusable.

There `constructor()` function is not protected against use of **zero address**.

Assets:

- Avive.sol [https://github.com/AviveWorld/Token/commits/main]
- Avive.sol [https://github.com/AviveWorld/Token/commits/hacken-initial-audit/]

Status: Accepted

Recommendations

Recommendation:

Implement zero address validation for the given parameters. This can be achieved by adding require statements that ensure address parameters are not the zero address.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, 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.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details

Repository	https://github.com/AviveWorld/Token/tree/hacken-initial-audit
Commit	bf614ddacb85ca6e984cac3dc698da60609a37e3
Whitepaper	Not provided
Requirements	Confidential
Technical Requirements	https://github.com/AviveWorld/Token/blob/hacken-initial-audit/README.md

Scope Details

Repository	https://github.com/AviveWorld/Token
Commit	c1232b2933b42c7b8ab0e7902a763f3e2f73c147
Whitepaper	Not provided
Requirements	Confidential
Technical Requirements	https://github.com/AviveWorld/Token/blob/main/README.md

Contracts in Scope

./contracts/Avive.sol