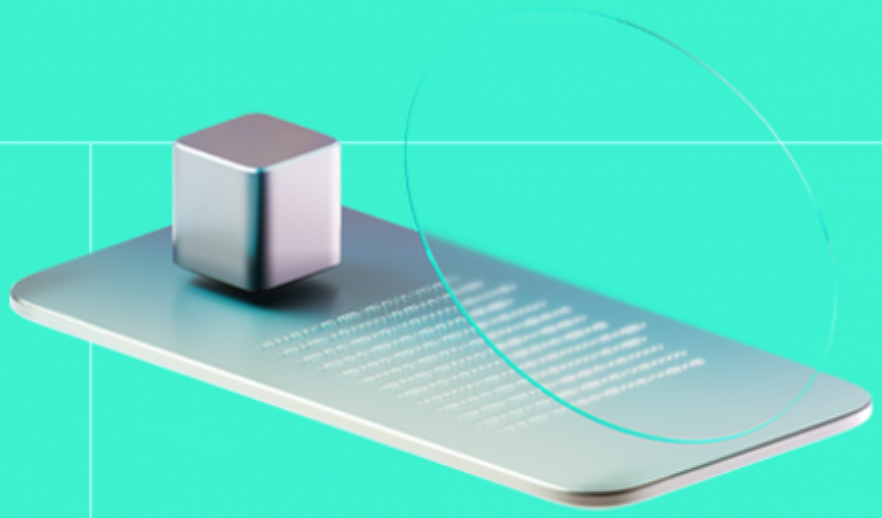
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: Zero1 Labs

Date: 21/03/2024

We express our gratitude to the Zero1 Labs team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

The DEAI smart contract is designed for the issuance and management of the "Zero1 Token" (DEAI), a digital asset on the Ethereum blockchain. Leveraging OpenZeppelin's upgradeable contracts, DEAI provides a scalable and secure framework for token minting, burning, and rescue operations, ensuring long-term operational flexibility and safety

Platform: EVM

Language: Solidity

Tags: ERC-20 Upgradable

Timeline: 19/03/2024 - 21/03/2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://gitlab.com/hacken-audit-contracts/zero1-token-deai
Commit	3da461f

Audit Summary

10/10	10/10	100%	10/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 10/10

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

1	1	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	1
High	0
Medium	0
Low	0

Vulnerability	Status
F-2024-1600 - Invalid type for argument in function call	Fixed

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 Zero1 Labs
Audited By	Giovanni Franchi
Approved By	Yves Toiser
Website	https://z1labs.ai/
Changelog	21/03/2024 - Preliminary Report && 21/03/2024 - Final Report

Table of Contents

System Overview

Zero1 Labs is an upgradable ERC-20 composed by the following contracts:

DEAI — Upgradable ERC-20 token that mints all initial supply to the default admin. Additional minting is not allowed. The contract enables burning functionalities restricted to a burning role and capability to rescue tokens accidentally sent to the contract.

It has the following attributes:

- Name: Zero1 Token
- Symbol: DEAI
- Decimals: 18
- Total supply: 1 billion.

TokensRescuer - Abstract utility contract that enables rescuing tokens accidentally sent to the contract.

CheckerZeroAddr - Abstract utility contract that serves the purpose of offering a modifier for zero addresses checks.

Privileged roles

- The default admin of the contract can arbitrarily mint to itself the total supply of tokens. This operation can only occur once.
- The default admin of the contract can arbitrarily assign and revoke burning roles.
- The burning role can arbitrarily burn tokens from its own balance.

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.
- Technical description is provided.

Code quality

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

- The code presents robust NatSpecs.
- The code adheres to best practises.

Test coverage

Code coverage of the project is **100%** (branch coverage).

- Test coverage is not required below 250 lines of code.

Security score

Upon auditing, the code was found to contain **1** critical, **0** high, **0** medium, and **0** 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 **10**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- Upgradeable contracts typically rely on an upgrade mechanism controlled by one or more addresses, introducing centralization and potentially creating single points of failure or trust.

Findings

Vulnerability Details

F-2024-1600 - Invalid type for argument in function call - Critical

Description:

The contract includes external functions intended for rescuing ERC20 and native tokens (**rescueERC20Token** and **rescueNativeToken**). These functions are designed to invoke internal utility functions (**_rescueERC20Token** and **_rescueNativeToken**) to perform the actual token rescue operations. However, there is a critical mismatch in the order of arguments passed from the external functions to their internal counterparts.

When the external **rescueERC20Token** function attempts to call the internal **_rescueERC20Token** function, it passes the arguments (**token**, **amount**, **receiver**) in an order that does not match the expected order of the internal function. Similarly, the **rescueNativeToken** function exhibits the same issue with its respective internal function call.

This discrepancy leads to a type mismatch error, preventing the contract from compiling successfully. As a result, the entire contract functionality is rendered inoperative.

Assets:

- zero1-token-deai/contracts/DEAI.sol [https://gitlab.com/hacken-audit-contracts/zero1-token-deai/-/blob/main/contracts/DEAI.sol?ref_type=heads]

Status: Fixed

Classification

Severity: Critical

Impact:

Likelihood [1-5]: 5

Impact [1-5]: 5

Exploitability [0-2]: 0

Complexity [0-2]: 0

Final Score: 5.0 (Critical)

Hacken Calculator Version: 0.6

Recommendations

Recommendation: Ensure that the order of arguments in the calls from external to internal functions matches precisely, aligning with the parameters' types and purposes. This adjustment will resolve the compilation error and restore the intended functionality of these critical operations.

Correcting this issue requires modifying the external functions to match the internal functions' expected argument order. No changes are needed for the internal functions themselves.

Remediation (commit: 1e5edfe8): Right order of arguments passed to the internal functions has been restored leading to the code being able to compile and work as intended.

Evidences

Compilation Failure

Files:

```
TypeError: Invalid type for argument in function call. Invalid implicit conversion from uint256 to address requested.
  --> contracts/DEAI.sol:56:34:
   |
56 |         _rescueERC20Token(token, amount, receiver);
   |                                  ^^^^^^

TypeError: Invalid type for argument in function call. Invalid implicit conversion from address to uint256 requested.
  --> contracts/DEAI.sol:56:42:
   |
56 |         _rescueERC20Token(token, amount, receiver);
   |                                          ^^^^^^^^

TypeError: Invalid type for argument in function call. Invalid implicit conversion from uint256 to address requested.
  --> contracts/DEAI.sol:64:28:
   |
64 |         _rescueNativeToken(amount, receiver);
   |                            ^^^^^^

TypeError: Invalid type for argument in function call. Invalid implicit conversion from address to uint256 requested.
  --> contracts/DEAI.sol:64:36:
   |
64 |         _rescueNativeToken(amount, receiver);
   |                                    ^^^^^^^^

Error HH600: Compilation failed
```

Observation Details

F-2024-1604 - Absence of Initialization Lock in Upgradeable Contract - Info

Description: The implementation contract for the upgradeable system lacks the invocation of **_disableInitializers()** within its constructor. This function is crucial for preventing the direct initialization of the implementation contract once deployed. While this does not pose an immediate security risk due to the nature of upgradeable contract deployment and interaction through a proxy, it represents a best practice deviation.

Assets:

- zero1-token-deai/contracts/DEAI.sol [https://gitlab.com/hacken-audit-contracts/zero1-token-deai/-/blob/main/contracts/DEAI.sol?ref_type=heads]

Status: Fixed

Recommendations

Recommendation: To align with best practices and further secure the contract architecture, it is recommended to invoke _disableInitializers() in the implementation contract's constructor. This change ensures that the contract is immediately locked against direct initialization post-deployment, reinforcing its role as part of an upgradeable system and preventing any potential misuse.

```
constructor() {
_disableInitializers();
}
```

Remediation (commit: 1e5edfe8): The new version of the contract abides to upgradability best practices having introduced `_disableInitializers()` within its constructor.

External References:

- Disable Initializer

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 **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

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://gitlab.com/hacken-audit-contracts/zero1-token-deai
Commit	3da461f07af1d3939c188c7f6dc0b331f5faf6c8
Whitepaper	N/A
Requirements	./contracts/DEAI doc.pdf
Technical Requirements	./contracts/DEAI doc.pdf

Remediation Scope Details

Repository	https://gitlab.com/hacken-audit-contracts/zero1-token-deai
Commit	1e5edfe84c553632aacc6e2c7dc079353eeefc9f
Whitepaper	N/A
Requirements	./contracts/DEAI doc.pdf
Technical Requirements	./contracts/DEAI doc.pdf

Contracts in Scope

./contracts/DEAI.sol

./contracts/extensions/CheckerZeroAddr.sol

./contracts/extensions/TokensRescuer.sol

./contracts/interfaces/ITokensRescuer.sol