SALUS SECURITY

MAR 2024

CODE SECURITY ASSESSMENT

POLYHEDRA

Overview

Project Summary

- Name: Polyhedra - StandardERC20Token
- Platform: EVM-compatible Chains
- Language: Solidity
- Repository: https://github.com/zkBridge-integration/standard-erc20-token
- Audit Scope: See Appendix - 1

Project Dashboard

Application Summary

Name	Polyhedra - StandardERC20Token
Version	v2
Type	Solidity
Date	Mar 04 2024
Logs	Mar 01 2024; Mar 04 2024

Vulnerability Summary

Total High-Severity issues	0
Total Medium-Severity issues	0
Total Low-Severity issues	1
Total informational issues	1
Total	2

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low impact in view of the client's business circumstances.
Informational	The issue does not pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Centralization risk with initial token distribution	Low	Centralization	Acknowledged
2	Use of the magic number	Informational	Code Quality	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Centralization risk with initial token distribution	
Severity: Low	Category: Centralization
Target: - src/Token.sol	

Description

When the contract is deployed, $TOKEN is sent to one account. This account then has full control over the token distribution. If it is an EOA account, any compromise of its private key could drastically affect the project – for example, attackers could dump the price of $TOKEN on the DEX if they gain access to the private key.

Recommendation

It is recommended to transfer tokens to a multi-sig account and promote transparency by providing a breakdown of the intended initial token distribution in a public location.

Status

This issue has been acknowledged by the team. The team has stated that the usage scenario of this token contract is entirely managed by smart contracts, not by EOAs, preventing private key compromise issues.

2.3 Informational Findings

2. Use of the magic number

Severity: Informational	Category: Code Quality

Target:
- src/Token.sol

Description

There is a literal value being used. To improve the code’s readability and facilitate refactoring, consider defining a constant for every magic number, giving it a clear and self-explanatory name.

src/Token.sol:L9

```
_mint(msg.sender, 1000000000 * 10 ** decimals());
```

Recommendation

Consider defining a constant variable INITIAL_SUPPLY for the magic number 1000000000.

Status

The team has resolved this issue in commit afcba4b.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit 357677e:

File	SHA-1 hash
Token.sol	f0845548c0e71e07393e529e38891b9b4f8dec6a