QuillAudits

Audit Report
October, 2023

For

CREATE PROTOCOL

Table of Content

Table of Content

Executive Summary

Project Name	Create Protocol
Project URL	*https://www.createprotocol.org/*
Overview	The Create Protocol's token is an independent token acting as a native multi-utility token in the Create ecosystem. It can be used to pay for services, access features, and participate in governance. The primary role of the token revolves around ecosystem.
Audit Scope	*https://goerli.etherscan.io/address/0xD69a339285Ea8B02E7175756a700fe9B7ca7db63#code*
Contracts in Scope	CreateToken
Commit Hash	NA
Language	Solidity
Blockchain	Ethereum
Method	Manual Analysis, Functional Testing, Automated Testing
Review 1	25th September 2023 - 26th September 2023
Updated Code Received	4th October 2023
Review 2	4th October 2023 - 5th October 2023
Fixed In	*https://goerli.etherscan.io/address/0x217099c8db711033f1b1fc426b03a6e0aaf3c89f#writeProxyContract*

Number of Security Issues per Severity

8
Issues Found

🟥 High	🟨 Medium
🟩 Low	🟪 Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	0	1	2
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	3	2

Checked Vulnerabilities

- ✓ Access Management
- ✓ Arbitrary write to storage for
- ✓ Centralization of control
- ✓ Ether theft
- ✓ Improper or missing events
- ✓ Logical issues and flaws
- ✓ Arithmetic Correctness
- ✓ Race conditions/front running
- ✓ SWC Registry
- ✓ Re-entrancy
- ✓ Timestamp Dependence
- ✓ Gas Limit and Loops
- ✓ Exception Disorder
- ✓ Gasless Send
- ✓ Use of tx.origin
- ✓ Malicious libraries
- ✓ Compiler version not fixed
- ✓ Address hardcoded
- ✓ Divide before multiply
- ✓ Integer overflow/underflow
- ✓ ERC's conformance
- ✓ Dangerous strict equalities
- ✓ Tautology or contradiction
- ✓ Return values of low-level calls
- ✓ Missing Zero Address Validation
- ✓ Private modifier
- ✓ Revert/require functions
- ✓ Multiple Sends
- ✓ Using suicide
- ✓ Using delegatecall
- ✓ Upgradeable safety
- ✓ Using throw

Checked Vulnerabilities

- ✓ Using inline assembly
- ✓ Style guide violation
- ✓ Unsafe type inference
- ✓ Implicit visibility level

Techniques and Methods

Throughout the audit of smart contracts, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments, match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods, and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analyzed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

A static Analysis of Smart Contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual Analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behavior of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Solhint, Mythril, Slither, Solidity Statistic Analysis.

Types of Severity

Every issue in this report has been assigned to a severity level. There are four levels of severity, and each of them has been explained below.

High Severity Issues

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium Severity Issues

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low Severity Issues

Low-level severity issues can cause minor impact and are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are four severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

A. Contract - CreateProtocol

High Severity Issues

No issues were found.

Medium Severity Issues

No issues were found.

Low Severity Issues

A.1: Centralization Issue

Description

The contract owner is responsible for minting more tokens after the initial minting at deployment, and can as well pause and unpause the contract. When a pause is made on the contract, all token holders would be unable to make transfers out of the address due to the halt in the contract. This shows that there is a centralization of power on the end of the contract owner if this address is owned by a person.

Remediation

Use a multisig wallet address which users trust the signatories to the account.

Status

Resolved

A.2: Ownership Transfer should be a Two-way Process

Description

Due to the importance of the contract owner to be responsible for mint, pause, and unpause of the contract, it is important to stress the need for the use of a two-way process on the transfer of ownership. When the current owner invokes the transferOwnership function, passing the parameter of the new address, this sets the new address immediately supposing it is not an address zero, hence would revert. But the issue arises when the address passed was that of a wrong address, this would not be redeemable anymore.

A.2: Ownership Transfer should be a Two-way Process

Remediation

Use the Openzeppeling Ownable2StepUpgradable to remedy the issue of instantaneous transfer to the wrong address. This way, the assigned address would claim ownership first, before the completion of ownership transfer.

Status

Resolved

A.3: Upgradeable contract is missing a `__gap[50]` storage variable to allow for new storage variables in later versions

Description

While some contracts may not currently be sub-classed, adding the variable now protects against forgetting to add it in the future.

Remediation

For a description of this storage variable see **this**.

Status

Resolved

Reference

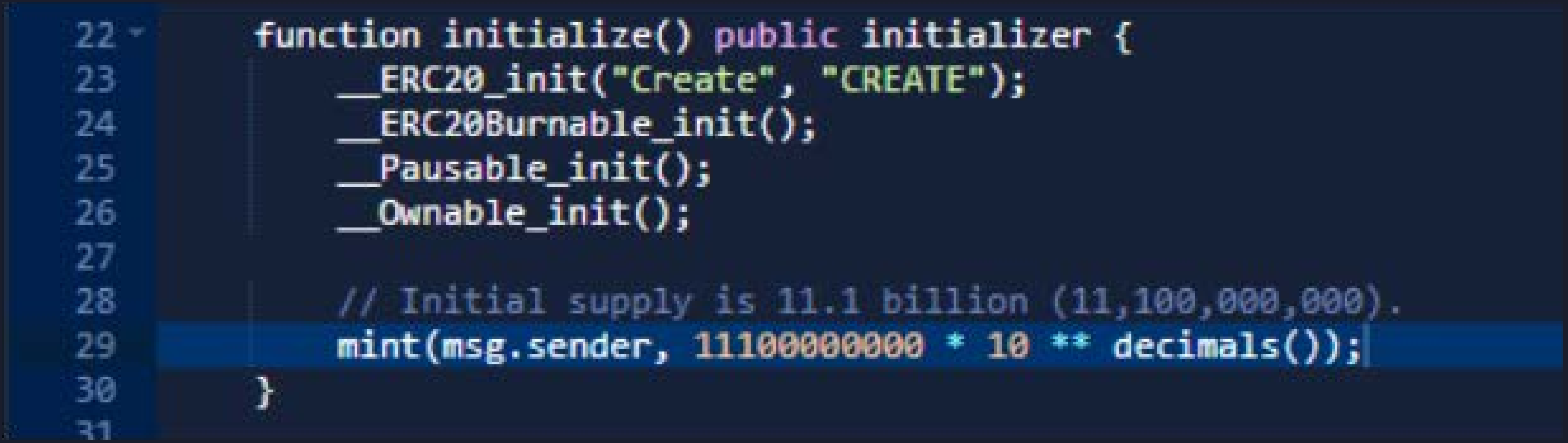
https://docs.openzeppelin.com/contracts/4.x/upgradeable#storage_gaps

A.4: Missing Management Mechanism

Line

29

Function - initialize



```
22      function initialize() public initializer {
23          __ERC20_init("Create", "CREATE");
24          __ERC20Burnable_init();
25          __Pausable_init();
26          __Ownable_init();
27
28          // Initial supply is 11.1 billion (11,100,000,000).
29          mint(msg.sender, 11100000000 * 10 ** decimals());
30      }
31
```

Description

The deployer of the CreateToken smart contract is responsible for the distribution of the CREATE tokens as the entire token supply is assigned to msg.sender (i.e. the Ethereum address who called the initialize() function during deployment).
If this particular account's private key is lost after the deployment, it will become impossible to allocate the CREATE tokens to any other users. Similarly, if this private key is leaked, malicious actors can control the entire supply.

Remediation

Consider implementing additional roles to manage access control rules for the CREATE token contract. Refer to this **contract** for an example implementation.

Status

Acknowledged

Reference

https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/access/AccessControlUpgradeable.sol

Informational Issues

A.5: Floating Solidity Version (pragma solidity ^0.8.7)

Description

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Remediation

Here all the in-scope contracts have an unlocked pragma, it is recommended to lock the same. Moreover, we strongly suggest not to use experimental Solidity features (e.g., pragma experimental ABIEncoderV2) or third-party unaudited libraries. If necessary, refactor the current code base to only use stable features.

Status

Acknowledged

A.6: Public Functions could be Declared as External in Order to Save Gas

Description

Whenever functions are not called within the contract, it is recommended that these functions use the external visibility in order to save gas. This way, the mint function should be made external as well, and the mint function called at the initializer function could be replaced by the _mint internal function from ERC20Upgradable.

Remediation

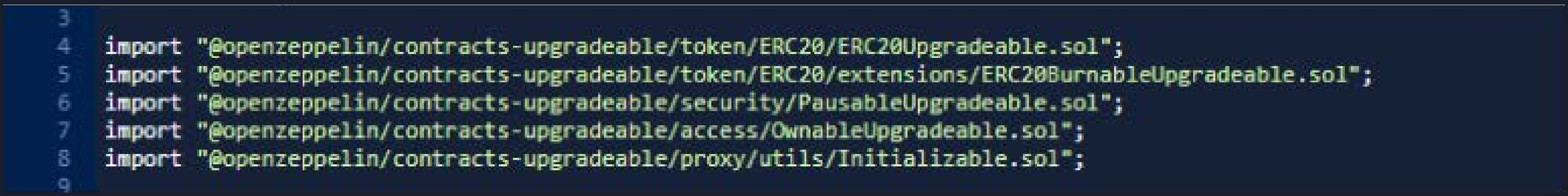
Use the external visibility for functions made public in order to save gas.

Status

Resolved

A.7: Non-usage of specific imports (Gas)

Proof of Concept



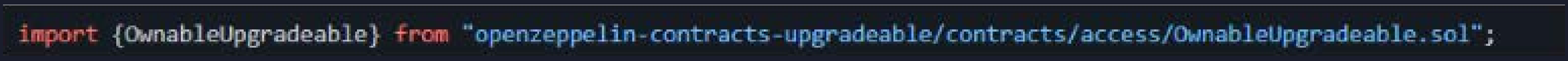
```
import "@openzeppelin/contracts-upgradeable/token/ERC20/ERC20Upgradeable.sol";
import "@openzeppelin/contracts-upgradeable/token/ERC20/extensions/ERC20BurnableUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/security/PausableUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
```

Description

The current form of relative path import is not recommended for use because it can unpredictably pollute the namespace.
Instead, the Solidity docs recommend specifying imported symbols explicitly.
https://docs.soliditylang.org/en/v0.8.15/layout-of-source-files.html#importing-other-source-files

Remediation

A good example-



```
import {OwnableUpgradeable} from "openzeppelin-contracts-upgradeable/contracts/access/OwnableUpgradeable.sol";
```

Status

Resolved

A.8: No comments in Codebase

Description

Although the contract has few functions, there is no proper code comment in the contract that would ease the process of comprehension for non-techincal users engaging with the contract. Good codebase possesses the attributes of a proper code comment.

Remediation

It is recommended to use the Natspec code comment format because of its ability to unveil functions underlying to non-technical users.

Status

Acknowledged

Reference

https://docs.soliditylang.org/en/latest/natspec-format.html

Functional Tests

Some of the tests performed are mentioned below:

- ✓ Should get the name of the token
- ✓ Should get the symbol of the token
- ✓ Should get the decimal of the token
- ✓ Should get the total supply of the token when deployed
- ✓ Should get balance of the owner when contract is deployed
- ✓ Should transfer tokens to other address
- ✓ Should approve another account to spend token
- ✓ Should confirm that token holders can burn the token
- ✓ Should confirm the ability to transfer when contract is paused
- ✓ Should transfer ownership to an address
- ✓ Should renounce ownership and see effect on pause regulations

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of Create Protocol. We performed our audit according to the procedure described above.

Disclaimer

QuillAudits Smart contract security audit provides services to help identify and mitigate potential security risks in Create Protocol smart contracts. However, it is important to understand that no security audit can guarantee complete protection against all possible security threats. QuillAudits audit reports are based on the information provided to us at the time of the audit, and we cannot guarantee the accuracy or completeness of this information. Additionally, the security landscape is constantly evolving, and new security threats may emerge after the audit has been completed.

Therefore, it is recommended that multiple audits and bug bounty programs be conducted to ensure the ongoing security of Create Protocol smart contracts. One audit is not enough to guarantee complete protection against all possible security threats. It is important to implement proper risk management strategies and stay vigilant in monitoring your smart contracts for potential security risks.

QuillAudits cannot be held liable for any security breaches or losses that may occur subsequent to and despite using our audit services. It is the responsibility of the Create Protocol to implement the recommendations provided in our audit reports and to take appropriate steps to mitigate potential security risks.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

850+
Audits Completed

$30B
Secured

$30B
Lines of Code Audited

Follow Our Journey

Audit Report
October, 2023

For

CREATE PROTOCOL

QuillAudits

Canada, India, Singapore, UAE, UK

www.quillaudits.com

audits@quillhash.com