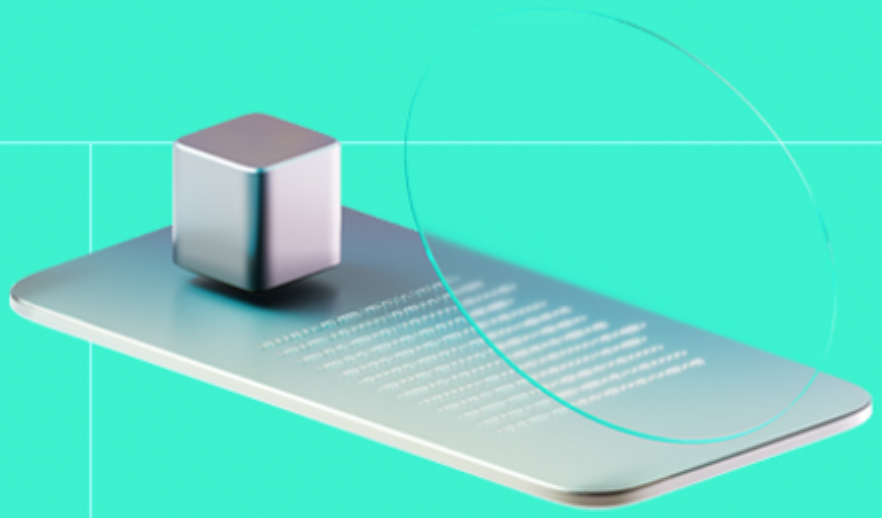
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: CREO

Date: 03/04/2024

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

Creo Engine is a private bridge solution.

Platform: Ethereum

Language: Solidity

Tags: ERC20, Bridge

Timeline: 11/03/2024 - 03/04/2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/creoenginecto/creobridge
Commit	c7d6502

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

0	0	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	0
Medium	0
Low	0

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 CREO
Audited By	Turgay Arda Usman
Approved By	Grzegorz Trawinski
Website	-
Changelog	15/03/2024 - Preliminary Report && 02/04/2024 Final Report

Table of Contents

System Overview

Creo is a private bridge with the following contracts:

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

It has the following attributes:

- Decimals: 18

BridgeAssist — a contract that creates incoming and outgoing transaction instances for the bridge. It also handles fees.

Privileged roles

- An admin can change the bridge parameters and withdraw funds from the bridge. Admins should be able to add more admins. The role can also be renounced and transferred to another wallet.
- There can be multiple relayers. A relayer can prove that the bridge operation has been initiated and funds have been locked on the sending chain. The role can be granted and revoked by admins.

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 follows best practices and style guides
- The development environment is configured.

Test coverage

Code coverage of the project is **100.00%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is adequate .
- Interactions by several users are tested thoroughly.

Security score

Upon auditing, the code was found to contain **0** 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

- The provided documentation states that the system will allow Multicall, which allows users to provide integration via ERC2771. ERC2711 contains a multicall issue which allows a malicious user to send transactions by using any role or user. Since the audit scope is a part of a bigger system this vulnerability can harm the system.
- The exchange rates used in the system are being determined by the manager, which creates the possibility of these rates being outdated. This can prevent users benefitting from the latest rates.

Findings

Vulnerability Details

Observation Details

F-2024-1467 - Missing Zero Address Violation - 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.

The constructor and `setLocked()` functions are lack of missing zero address validation

Assets:

- CreoEngine.sol [https://github.com/GotBit/creo-bridge-solana]

Status: Fixed

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.

Remediation (Commit: e9fc4eb): The zero address checks implemented as suggested.

F-2024-1468 - Checks Effects Interactions Pattern Violation - Info

Description: It was identified that `BridgeAssist.sol` contract has an instance of Checks-Effects-Interactions (CEI) pattern violation, where state variables are updated after the external calls to the token contract. As explained in Solidity Security Considerations, it is best practice to follow the CEI pattern while interacting with external contracts to avoid reentrancy-related issues. However, no reentrancy vulnerability was identified during the security assessment. The finding is reported as a deviation from leading security practices.

```
function send(
uint256 amount,
string memory toUser, // marked as memory to prevent "stack too deep
"
string calldata toChain
) external whenNotPaused {
...
_receiveTokens(msg.sender, amount);
...
if (currentFee != 0) _dispenseTokens(feeWallet, currentFee * exchang
eRate);
transactions[msg.sender].push(
Transaction({
fromUser: msg.sender,
toUser: toUser,
amount: amount / exchangeRate - currentFee, // @audit-ok this number
is actually different than the calculated amount (CHECK)
// No logic of the system relies on this timestamp,
// it's only needed for displaying on the frontend
timestamp: block.timestamp,
fromChain: CURRENT_CHAIN(),
toChain: toChain,
nonce: nonce++,
block: block.number
})
);
}
```

Assets:

- BridgeAssist.sol [https://github.com/GotBit/creo-bridge-solana]

Status: Fixed

Recommendations

Recommendation: It is recommended to follow the CEI pattern when interacting with external contracts.

Remediation (Commit: e9fc4eb): The function is re-implmeneted according to the Checks-Effects-Interaction pattern.

F-2024-1535 - EIP712 signatures are not single use and not time limitted - Info

Description: The BridgeAssist.sol contract uses signatures to verify transactions. However, it does not properly adopt the EIP712 standard for creating and verifying these signatures. The relevant code is as follows:

```
function _hashTransaction(FulfillTx memory transaction) private view
returns (bytes32){
return
_hashTypedDataV4(
keccak256(
abi.encode(
FULFILL_TX_TYPEHASH,
transaction.amount,
keccak256(abi.encodePacked(transaction.fromUser)),
transaction.toUser,
keccak256(abi.encodePacked(transaction.fromChain)),
transaction.nonce
)
)
);
}
```

However, it was identified that the signed message does not include deadline to prevent signature reuse. Thus, every observed on-chain signature can be replayed. Thus, the risk of signature replay is limited as firstly the relayer private key must be disclosed in prior. Alternatively, a signed message must be disclosed off-chain.

Assets:

- BridgeAssist.sol [https://github.com/GotBit/creo-bridge-solana]

Status: Mitigated

Recommendations

Recommendation: It is recommended to add both the deadline parameter to the signed message in the EIP712 implementation. The deadline parameter should be short enough to allow trigger the transaction and disallow using it after reasonable period. The nonce parameter should be a number tracked by the solution and incremented each time signature is correctly used.

Remediation (Mitigated): The platform informed us that the implementation currently does not need a deadline parameter

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://github.com/creoenginecto/creobridge
Commit	c7d6502
Whitepaper	-
Requirements	Provided as a File
Technical Requirements	Provided as a File

Contracts in Scope

BridgeAssist.sol

CreoEngine.sol