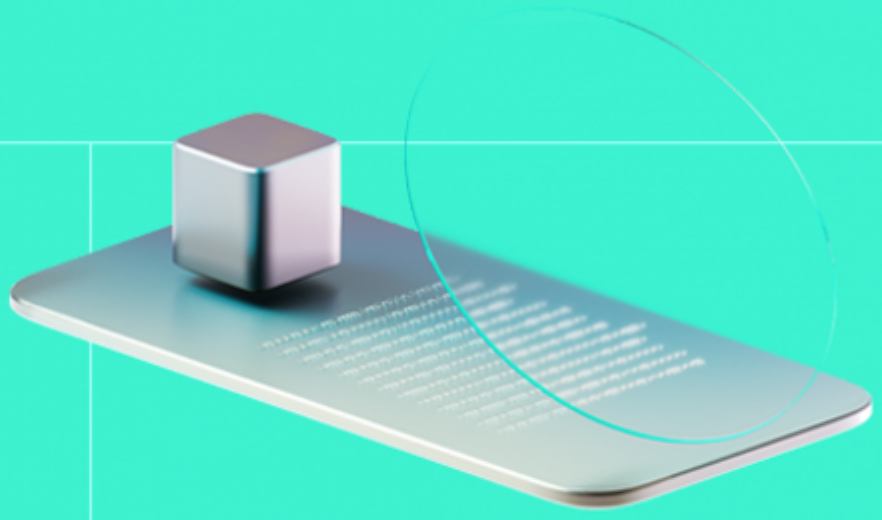
HACKEN

Smart Contract Code Review And Security Analysis Report

Customer: Cadaico GmbH

Date: 15/05/2024



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

CADAICO's technology stack blends Artificial Intelligence and Blockchain, anchored by the versatile AI Model customizable for specific needs. The integrated blockchain guarantees secure transactions.

Platform: EVM

Language: Solidity

Tags: ERC20

Timeline: 02/05/2024 - 03/05/2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/CADAICO/contracts
Commit	83cb55e

Audit Summary

10/10	7/10	100%	8/10
Security score	Code quality score	Test coverage	Documentation quality score

Total 9.2/10

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

4	4	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	0
Medium	1
Low	3

Vulnerability	Status
F-2024-1966 - Two step ownership transfer mechanism can be used instead of regular Ownable	Fixed
F-2024-1967 - The owner of the contract can renounce the ownership, without the possibility of recovery	Fixed
F-2024-1972 - Difference in naming between ERC20Permit and ERC20	Fixed
F-2024-2026 - Minted amount of tokens can be higher than MAX_TOTAL_SUPPLY	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 Cadaico GmbH
Audited By	Bohdan Pukhno, Philipp Eder
Approved By	Yves Toiser
Website	https://www.cadai.co/
Changelog	06/05/2024 - Preliminary Report & 15/05/2024 - Final Report

Table of Contents

System Overview **6**

Privileged Roles 6

Executive Summary **7**

Documentation Quality 7

Code Quality 7

Test Coverage 7

Security Score 7

Summary 7

Risks **8**

Findings **9**

Vulnerability Details 9

Observation Details 17

Disclaimers 26

Appendix 1. Severity Definitions **27**

Appendix 2. Scope **28**

System Overview

CADAICO — simple ERC-20 token that mints 1 million tokens of initial supply to a deployer. Additional minting is allowed only by the owner. Burning is allowed. The token has permit functionality.

It has the following attributes:

- Name: CADAICO
- Symbol: wCADAI
- Decimals: 18
- Total supply: 100 Million tokens.

Privileged roles

- The owner of the contract can mint tokens to any address. The amount of mintable tokens is limited to 100 million

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 **8** out of **10**.

- Functional requirements are provided.
- Technical description is not provided.
  - Detailed NatSpecs are provided.

Code quality

The total Code quality score is **7** out of **10**.

- The code follows Solidity Style Guide.
- The development environment is not provided.

Test coverage

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

- Tests are not mandatory for projects with less than 250 LOC.

Security score

Upon auditing, the code was found to contain **0** critical, **0** high, **1** medium, and **3** low severity issues. Out of these, **4** issues have been addressed and resolved, 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 **9.2**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- **Centralization risk:** 10 Million tokens will be minted upon deployment and the sole privilege to mint the remaining tokens up to the maximum supply lies with the owner.

Findings

Vulnerability Details

F-2024-1972 - Difference in naming between ERC20Permit and ERC20 - Medium

Description:

In the `CADAICO` contract, the `name` that is inserted into the `ERC20` token `constructor` is `"CADAICO"`, but the `name` that is inserted into the `ERC20Permit` `constructor` is `"Cadaico"`.

`ERC20Permit` passes the value of this `name` to `EIP712` for message signatures, where the hashed `name` will be used to build a domain separator, and then will be used in the function `_hashTypedDataV4` to compare signatures.

Differences in the case of letters can negatively affect correct operation. According to the ASCII table lowercase letters have a different code than uppercase letters and when comparing the words `"Cadaico"` and `"CADAICO"`, the hashed output will not match.

Assets:

- wCADAI.sol [https://github.com/CADAICO/contracts]

Status: Fixed

Classification

Impact: 2/5

Likelihood: 4/5

Exploitability: Independent

Complexity: Simple

Likelihood [1-5]: 4
Impact [1-5]: 2
Exploitability [0-2]: 0
Complexity [0-2]: 0
Final Score: 3.0 (Medium)
Hacken Calculator Version: 0.6

Severity: Medium

Recommendations

Remediation: Use the same `name` parameter that is passed to the corresponding `constructor`.

This will help to avoid incorrect signatures of permits shared to users.

Resolution:

Remediation(revised commit: 160dfa0): The client has fixed this issue by using the same `name` parameter in both constructors' calls ERC20 and ERC20Permit.

F-2024-1966 - Two step ownership transfer mechanism can be used instead of regular Ownable - Low

Description:

The `CADAICO` contract currently uses a simple `Ownable` pattern, where ownership can be transferred in a single transaction. While this is straightforward and easy to understand, it can potentially lead to issues if the new owner's address is input incorrectly, as ownership would be irreversibly transferred to an incorrect (and potentially inaccessible) address.

A more secure pattern is the two-step ownership transfer, also known as "claimable" ownership. In this pattern, the current owner initiates the transfer, but the new owner must also call a function to claim ownership. This provides an additional layer of security, as it ensures that the new owner's address is correct and that the new owner is ready to take over the contract.

Assets:

- wCADAI.sol [https://github.com/CADAICO/contracts]

Status: Fixed

Classification

Impact: 4/5

Likelihood: 2/5

Exploitability: Dependent

Complexity: Simple

Likelihood [1-5]: 2

Impact [1-5]: 4

Exploitability [0-2]: 2

Complexity [0-2]: 0

Final Score: 2.1 (Low)

Hacken Calculator Version: 0.6

Severity: Low

Recommendations

Remediation:

To improve the security of your contract's ownership management, it is recommended to utilize the `Ownable2Step` contract from ***OpenZeppelin***. This contract implements a two-step process for transferring ownership, requiring the new owner to actively confirm acceptance through a contract call. This mechanism helps prevent the unintended transfer of ownership to an incorrect address, thereby enhancing overall security with an additional verification step.

Resolution: **Remediation(revised commit: 160dfa0):** the `Ownable2Step` contract from ***OpenZeppelin*** implemented in the `CADAICO` contract.

F-2024-1967 - The owner of the contract can renounce the ownership, without the possibility of recovery - Low

Description:

The `CADAICO` contract currently uses the `Ownable` library for authorization control, which includes the `renounceOwnership()` function. This function allows the current owner to permanently renounce ownership, making the contract ownerless.

The contract also includes a variable `mint` function, which is restricted to the owner:

```
function mint(address to, uint256 amount) public onlyOwner
```

This function is *critical* for the operation of the contract as it allows to mint a new tokens. If ownership is renounced, this function becomes inaccessible, effectively making the contract unable to mint a new tokens.
Renouncing the ownership is not mentioned in the documentation as an intended functionality.

Assets:

- wCADAI.sol [https://github.com/CADAICO/contracts]

Status: Fixed

Classification

Impact: 3/5

Likelihood: 2/5

Exploitability: Dependent

Complexity: Simple

Likelihood [1-5]: 2

Impact [1-5]: 3

Exploitability [0-2]: 2

Complexity [0-2]: 0

Final Score: 1.8 (Low)

Hacken Calculator Version: 0.6

Severity: Low

Recommendations

Remediation:

It is advised to override the `renounceOwnership()` function to reject any transaction. This adjustment will ensure that the contract owner cannot inadvertently or deliberately make the contract ownerless and, therefore unusable.

Resolution: **Remediation(revised commit: 160dfa0):** The client has fixed this issue by overriding the `renounceOwnership()` function to reject any transaction.

F-2024-2026 - Minted amount of tokens can be higher than MAX_TOTAL_SUPPLY - Low

Description:

The description states that `MAX_TOTAL_SUPPLY` of the token is `100 million` tokens. And the upper limit is calculated with the value of `totalSupply()` function in the `mint()` function:

```
function mint(address to, uint256 amount) public onlyOwner {
require(totalSupply() + amount <= MAX_TOTAL_SUPPLY, "Minting would exceed ma
x total supply");
_mint(to, amount);
}
```

Since this contract inherits from ERC20Burnable by calling `burn()` or `burnFrom()` functions it will decrease the value of the `totalSupply()` function's output

```
function _burn(address account, uint256 value) internal {
if (account == address(0)) {
revert ERC20InvalidSender(address(0));
}
// Here is the call to the _update() function
_update(account, address(0), value);
}

function _update(address from, address to, uint256 value) internal virtual {
// Rest of the code
...
// Check if the `to` is equal to address(0) means that it is a burn
if (to == address(0)) {
unchecked {
// _totalSupply decreasement
_totalSupply -= value;
}
// Rest of the code
...
}
```

If a `totalSupply()` value of `100 million` is reached and any user decides to burn their tokens, the `totalSupply()` value will be decreased and the possibility to mint new tokens arises. The `totalSupply()` value will be `100 million` again while the total amount of tokens minted will be higher.

Assets:

- wCADAI.sol [https://github.com/CADAICO/contracts]

Status: Fixed

Classification

Impact: 2/5

Likelihood: 4/5

Exploitability: Semi-Dependent

Complexity: Medium

Likelihood [1-5]: 4
Impact [1-5]: 2
Exploitability [0-2]: 2

Complexity [0-2]: 0
Final Score: 2.1 (Low)

Severity: Low

Recommendations

Remediation: The output of `totalSupply()` is unsuitable as a criteria within the `mint()` function, instead a new variable to keep track of the amount of tokens minted can be created and used as the criteria within the `mint()` function.

Resolution: **Remediation(revided commit: 160dfa0):** The client has fixed this issue by adding a new variable `totalMinted` that stores all the amount of tokens that minted.

Observation Details

F-2024-1965 - State variables could be declared as immutable - Info

Description: The variable `MAX_TOTAL_SUPPLY` is set in the constructor and never updated, therefore it can be marked as immutable to enhance gas efficiency.

Assets:

- wCADAI.sol [https://github.com/CADAICO/contracts]

Status: Fixed

Recommendations

Remediation: Declare the variable `MAX_TOTAL_SUPPLY` as immutable to save Gas.

Resolution: **Remediation(revised commit: 160dfa0):** The client declared the variable `MAX_TOTAL_SUPPLY` as immutable by adding the `immutable` modifier into this variable.

F-2024-1968 - Floating Pragma - Info

Description: Contracts should be deployed with the same compiler version and flags that have been tested thoroughly. Locking the Pragma helps 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.

Assets:

- wCADAI.sol [https://github.com/CADAICO/contracts]

Status: Fixed

Recommendations

Remediation: Use a fixed version of the compiler (^ symbol should be removed from Pragma).

Resolution: **Remediation(revised commit: 160dfa0):** The client used a fixed version of the compiler (^ symbol removed from Pragma).

F-2024-1969 - Custom Errors in Solidity to enhance gas efficiency - Info

Description: Starting from `Solidity version 0.8.4`, the language introduced a feature known as "custom errors". These custom errors provide a way for developers to define more descriptive and semantically meaningful error conditions without relying on string messages. Prior to this version, developers often used the `require` statement with string error messages to handle specific conditions or validations. However, every unique string used as a revert reason consumes gas, making transactions more expensive.
`if` statement along with `revert` and `custom errors`, on the other hand, are identified by their name and the types of their parameters only, and they do not have the overhead of string storage. This means that, when using custom errors instead of `require` statements with string messages, the gas consumption can be significantly reduced, leading to more gas-efficient contracts.

Assets:

- wCADAI.sol [https://github.com/CADAICO/contracts]

Status: Mitigated

Recommendations

Remediation: It is recommended to use custom errors instead of revert strings to reduce gas costs, especially during contract deployment. Custom errors can be defined using the error keyword and can include dynamic information.

Resolution: **Remediation(revised commit: 160dfa0):** The client fixed previous implementation, but during the remediation process, the client implemented `renounceOwnership` function, which has string message instead of custom error.

F-2024-1971 - Large numeric literals should use underscores for readability - Info

Description:

From a security perspective, the use of underscores in large numeric literals in smart contracts is a practice that significantly enhances readability and maintainability, which are key factors in ensuring the security and reliability of smart contracts.

- **Prevents Misinterpretation**: Large numbers without any separators can be difficult to read and easy to misinterpret. For example, it's easier to correctly read **1_000_000_000** than **1000000000**. This clarity is crucial in smart contracts where a single digit can dramatically change the contract's behavior.
- **Reduces Human Error**: Readability is directly linked to the likelihood of human error. Developers are less likely to make mistakes in interpreting or modifying code that is clear and easy to understand.

Assets:

- wCADAI.sol [https://github.com/CADAICO/contracts]

Status: Fixed

Recommendations

Remediation:

Implement Underscores in Large Numeric Literals: It is strongly recommended to enhance the readability of large numeric literals by incorporating underscores (_) as separators. This practice aligns with the latest best practices in smart contract development and significantly aids in reducing human error.

- Current format:

```
MAX_TOTAL_SUPPLY = 100000000 * 10 ** decimals(); // Defines the max total su
pply.
_mint(msg.sender, 1000000 * 10 ** decimals()); // Mints initial supply.
```

- Recommended format:

```
MAX_TOTAL_SUPPLY = 100_000_000 * 10 ** decimals(); // Defines the max total
supply.
_mint(msg.sender, 1_000_000 * 10 ** decimals()); // Mints initial supply.
```

Resolution:

Remediation(revised commit: 160dfa0): The client fixed this observation by changing implemented numbers to the recommended format.

F-2024-1974 - Misleading Token naming - Info

Description: It is stated in the NatSpecs that this token is a wrapped version of the CADAI token, but its `name` does not mention it.

Assets:

- wCADAI.sol [https://github.com/CADAICO/contracts]

Status: Accepted

Recommendations

Remediation: Change the token `name` to inform users that it is a wrapped token.

Resolution: **Remediation(revised commit: 160dfa0):** The client accepted this observation.

F-2024-2023 - Functions not used internally can be marked as external - Info

Description: The function `mint` is currently set to the `public` visibility but it never called internally.

```
function mint(address to, uint256 amount) public onlyOwner {}
```

If other functions do not call `mint` function, this function should be marked as `external` to reduce gas costs when calling it.

Assets:

- wCADAI.sol [https://github.com/CADAICO/contracts]

Status: Fixed

Recommendations

Remediation: Change the `mint` function visibility from `public` to `external`.

Resolution: **Remediation(revised commit: 160dfa0):** The client changed the `mint` function visibility from `public` to `external`.

F-2024-2025 - Solidity version 0.8.20 might not work on all chains due to `PUSH0` - Info

Description: The Solidity version 0.8.20 employs the recently introduced `PUSH0` opcode in the Shanghai EVM. This opcode might not be universally supported across all blockchain networks and Layer 2 solutions. Thus, as a result, it might be not possible to deploy solution with version higher or equal than 0.8.20 on some blockchains.

Assets:

- wCADAI.sol [https://github.com/CADAICO/contracts]

Status: Accepted

Recommendations

Remediation: It is recommended to verify whether solution can be deployed on particular blockchain with the Solidity version 0.8.20 >=. Whenever such deployment is not possible due to lack of `PUSH0` opcode support and lowering the Solidity version is a must, it is strongly advised to review all feature changes and bugfixes in [Solidity releases](https://soliditylang.org/blog/category/releases/). Some changes may have impact on current implementation and may impose a necessity of maintaining another version of solution.

Resolution: **Remediation(revised commit: 160dfa0):** The client accepted this observation.

F-2024-2194 - Constants should be defined rather than using magic numbers - Info

Description: The amount of tokens to be minted to the deployer is set manually in the constructor without any context.

Assets:

- wCADAI.sol [https://github.com/CADAICO/contracts]

Status: Fixed

Recommendations

Remediation: Define the amount of tokens to be minted to the deployer as a constant, rather than initializing it in the constructor.

Resolution: **Remediation(revised commit: 160dfa0):** The client added the `initialMint` constant variable.

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/CADAICO/contracts
Commit	83cb55e
Whitepaper	Whitepaper.pdf
Requirements	Whitepaper.pdf
Technical Requirements	N/A

Contracts in Scope

./contracts/wCADAI.sol