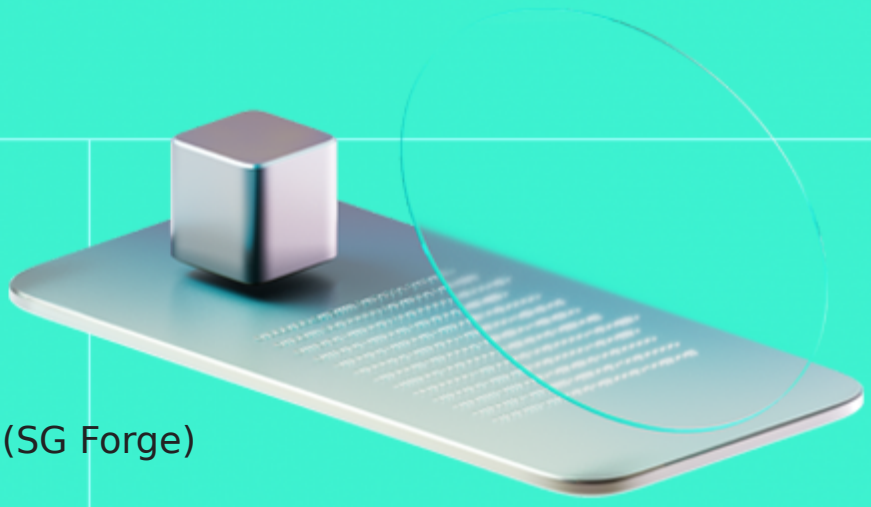
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer:

Societe Generale Forge (SG Forge)

Date: 12/06/2025

We express our gratitude to the Societe Generale Forge (SG Forge) team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

Platform: EVM

Language: Solidity

Tags: ERC20

Changelog: 06/06/2024 (2nd Review); 12/06/2025 (3rd Review)

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	Shared privately
Commit	1c3697243fa64e2d3b01ef516be53063bc1a7d10

Audit Summary

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

2	0	1	1
Total Findings	Resolved	Accepted	Mitigated

Findings by Severity

Severity	Count
Critical	0
High	0
Medium	1
Low	1

Vulnerability	Severity
F-2024-3285 - Use of constructor in upgradeable contract	Medium
F-2024-3306 - Asset wiping in wipeFrozenAddress function	Low

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 Societe Generale Forge (SG Forge)
Audited By	Carlo Parisi
Approved By	Przemyslaw Swiatowiec
Website	https://www.sgforge.com/product/coinvertible/
Changelog	24/05/2024 - Preliminary Report 06/06/2024 - 2nd Review Report 12/06/2025 - 3rd Review Report (Done By Oleksii Haponiuk)

Table of Contents

System Overview

SMART_COIN is an ERC20, with the following contracts:

- SmartCoin — ERC20 extended with:
  - UUPS upgrade mechanism
  - Operator Roles: registrar, operations, technical. These roles are introduced to manage the upgrade to new implementations and control transfers of tokens amongst these roles.
- EncodingUtils - a library that has the functionality to compute the hash of transfer requests
- AccessControlUpgradeable - an abstract contract that handles the access control for the SmartCoin contract.

Privileged roles

- Registrar operator:
  - Manages Whitelist of authorized users.
  - Validates/Rejects transfers to registrar and operations operators.
  - Names the operators for the new implementation.
  - Authorizes the upgrade to the next implementation.
  - Cannot be used as spender or destination of transferFrom().
  - Can retrieve tokens from any address to itself.
  - Can mint and burn SmartCoin tokens.
- Operations operator:
  - Cannot be used as spender or destination of transferFrom().
  - Transfers to operations must be validated by the registrar.
- Technical operator:
  - Launches a previously authorized (by registrar) implementation upgrade.

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project.

Documentation quality

- Functional requirements are mostly provided.
- Technical description is not provided.
- NatSpec is sufficient.

Code quality

- Gas consumption could be optimized.

Test coverage

Code coverage of the project is **95.45%** (branch coverage).

Security score

Upon auditing, the code was found to contain **0** critical, **0** high, **1** medium, and **1** low severity issues.

All identified issues are detailed in the “Findings” section of this report.

Risks

- SmartCoin is an ERC20 token that has centralized features, the tokens can be frozen or burned by the admins of the protocol.
- This audit covers the `SmartCoin.sol` contract, which is designed to be upgradeable. However, the audit does not cover the reliability of the first version (v1) of the contract, future versions, or potential mistakes that could be made by the admin during the upgrade process. This limitation could leave potential vulnerabilities undetected in the contract's lifecycle.

Findings

Vulnerability Details

F-2024-3285 - Use of constructor in upgradeable contract - Medium

Description: The `AccessControlUpgradeable.sol` contract uses a constructor to initialize the `registrar`, `operations`, and `technical` addresses. However, in the context of upgradeable contracts in Solidity, constructors should not be used. This is because the constructor code is only run when the contract is first created and will not be included in the deployed bytecode. As a result, when the contract is upgraded, the constructor will not be run again, and the state variables will not be re-initialized. Additionally, constructors in upgradeable contracts should call `_disableInitializers()` to prevent the initializer from being called more than once.

Assets:

- smartCoin/AccessControlUpgradeable.sol [N/A]

Status: Mitigated

Classification

Impact: 5/5

Likelihood: 2/5

Exploitability: Independent

Complexity: Simple

Likelihood [1-5]: 2

Impact [1-5]: 5

Exploitability [0-2]: 0

Complexity [0-2]: 0

Final Score: 3.5 (Medium)

Hacken Calculator Version: 0.6

Severity: Medium

Recommendations

Remediation: To resolve this issue, consider using an initializer function instead of a constructor. The initializer function can be run manually after the contract is deployed to set the initial state. Also, ensure that the `registrar`, `operations`, and `technical` variables are not declared as `immutable` since `immutable` variables can only be set in the constructor and cannot be changed later, which is not suitable for upgradeable contracts.

Resolution: The client has mitigated the inherent risk of sharing the same immutable value throughout all proxies that share that implementation contract. Team declared that this behaviour would not affect their contract upgrade processes.

F-2024-3306 - Asset wiping in wipeFrozenAddress function - Low

Description: The `wipeFrozenAddress` function in the `SmartCoin.sol` contract transfers the balance of a frozen address to the registrar and then burns the transferred amount. However, if there is a transaction pending approval or rejection when an address gets frozen, and the `wipeFrozenAddress` function is called, the assets involved in the pending transaction will not be wiped. This is because the `wipeFrozenAddress` function only considers the current balance of the frozen address, not the assets in pending transactions. This could result in the frozen address retaining assets if the pending transaction is rejected after the address has been wiped.

Assets:

- smartCoin/SmartCoin.sol [N/A]

Status: Accepted

Classification

Impact: 2/5

Likelihood: 3/5

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: To ensure all assets are wiped from a frozen address, the registrar should first reject any pending transactions from the

frozen address before calling the `wipeFrozenAddress` function. Alternatively, the `wipeFrozenAddress` function could be modified to automatically reject any pending transactions from the frozen address.

Observation Details

F-2024-3283 - Unnecessary gas consumption in wipeFrozenAddress function - Info

Description: The `wipeFrozenAddress` function in the `SmartCoin.sol` contract transfers the balance of a frozen address to the registrar, and then burns the transferred amount. This two-step process of transferring and then burning tokens results in unnecessary gas consumption, as each operation requires its own transaction and gas fees.

Assets:

- smartCoin/SmartCoin.sol [N/A]

Status: Fixed

Recommendations

Remediation: To optimize gas usage, consider implementing a direct burn function that can burn tokens from the frozen address in a single operation. This would eliminate the need for the intermediary transfer to the registrar, thereby reducing the overall gas cost.

F-2024-3284 - Redundant gas consumption in decreaseAllowance function - Info

Description: The `decreaseAllowance` function in the `SmartCoin.sol` contract includes the `forbiddenForRegistrar` and `forbiddenForOperations` modifiers. These modifiers check whether the `_spender` is the registrar or an operator. However, since the allowance cannot be approved or increased for the registrar and the operator, these checks are unnecessary and result in a waste of gas.

Assets:

- smartCoin/SmartCoin.sol [N/A]

Status: Fixed

Recommendations

Remediation: To optimize gas usage, consider removing the `forbiddenForRegistrar` and `forbiddenForOperations` modifiers from the `decreaseAllowance` function.

F-2024-3304 - Redundant gas consumption in pause and unpause functions - Info

Description: The `pause` and `unpause` functions in the `AccessControlUpgradeable.sol` contract include the `onlyWhenNotPaused` and `onlyWhenPaused` modifiers respectively. These modifiers check the current state of the contract before allowing the functions to proceed. However, since these functions are only callable by the `registrar` who should be aware of the contract's state, these checks are unnecessary and result in a waste of gas.

Assets:

- smartCoin/AccessControlUpgradeable.sol [N/A]

Status: Accepted

Recommendations

Remediation: To optimize gas usage, consider removing the `onlyWhenNotPaused` and `onlyWhenPaused` modifiers from the `pause` and `unpause` functions respectively.

F-2024-3305 - Incorrect comment in ISmartCoin.sol - Info

Description: The comment above the `ISmartCoin.sol` interface states that all transfers need to be validated by an operator (the registrar operator) before the tokens are actually transferred. However, this is not accurate. Only transfers to the operator or registrar need to be validated, not normal transfers.

```
/**
* @dev This interface is a slightly modified version of the IERC20 standa
* To comply with the financial regulations(KYC, AML, Sanctions&Embargos),
* before the tokens are actually transferred
*/
```

Assets:

- smartCoin/ISmartCoin.sol [N/A]

Status: Fixed

Recommendations

Remediation: To avoid confusion and potential misuse of the interface, the comment should be corrected to accurately reflect the functionality of the contract.

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	Shared privately
Commit	1c3697243fa64e2d3b01ef516be53063bc1a7d10
Whitepaper	-
Requirements	-
Technical Requirements	-

Contracts in Scope

libraries/EncodingUtils.sol

smartCoin/ISmartCoin.sol

smartCoin/IAccessControl.sol

smartCoin/SmartCoinDataLayout.sol

smartCoin/SmartCoin.sol

smartCoin/AccessControlUpgradeable.sol

smartCoin/AccessControlDataLayout.sol