SMART CONTRACT AUDIT REPORT

for

Banking Circle Stablecoin (ERC20)

Prepared By: Xiaomi Huang

PeckShield

June 18, 2024

Document Properties

Client	Banking Circle
Title	Smart Contract Audit Report
Target	Banking Circle
Version	1.0
Author	Xuxian Jiang
Auditors	Jason Shen, Xuxian Jiang
Reviewed by	Xiaomi Huang
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author	Description
1.0	June 18, 2024	Xuxian Jiang	Final Release

Contact

For more information about this document and its contents, please contact PeckShield Inc.

Name	Xiaomi Huang
Phone	+86 183 5897 7782
Email	contact@peckshield.com

Contents

1 Introduction 4
- 1.1 About Banking Circle . . . 4
- 1.2 About PeckShield . . . 5
- 1.3 Methodology . . . 5
- 1.4 Disclaimer . . . 6

2 Findings 8
- 2.1 Summary . . . 8
- 2.2 Key Findings . . . 9

3 ERC20 Compliance Checks 10

4 Detailed Results 13
- 4.1 Trust Issue of Admin Keys . . . 13

5 Conclusion 15

References 16

1 Introduction

Given the opportunity to review the design document and related source code of the Banking Circle token contract, we outline in the report our systematic method to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistency between smart contract code and the documentation, and provide additional suggestions or recommendations for improvement. Our results show that the given version of the smart contract exhibits no ERC20 compliance issues or security concerns. This document outlines our audit results.

1.1 About Banking Circle

Banking Circle S.A ("Banking Circle") which has been granted authorisation as a credit institution under the supervision of the Luxembourg Commission de surveillance du secteur financier ("CSSF") is the issuer of the stablecoin. Headquartered in Luxembourg, Banking Circle also has branches in the UK, Germany and Denmark. Banking Circle is a fully licensed next generation payments bank that is designed to meet the global banking and payments needs of payments businesses, banks and marketplaces. Banking Circle's solutions are powering the payments propositions of more than 250 regulated businesses, financial institutions and marketplaces, enabling them to gain the geographic reach and access to the markets in which their customers want to trade. This specific audit focuses on its stablecoin token contract, an ERC20-compliant token with freeze and gas-less transaction capability. The basic information of the audited contract is as follows:

Table 1.1: Basic Information of Banking Circle Token Contract

Item	Description
Name	Banking Circle
Website	https://www.bankingcircle.com/
Type	Ethereum ERC20 Token Contract
Platform	Solidity
Audit Method	Whitebox
Audit Completion Date	June 18, 2024

In the following, we show the Git repository of reviewed files and the commit hash value used in this audit.

- https://bankingcirclepublic@dev.azure.com/bankingcirclepublic/bankingcircle.git (e85ca86)

1.2 About PeckShield

PeckShield Inc. [4] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystem by offering top-notch, industry-leading services and products (including the service of smart contract auditing). We are reachable at Telegram (https://t.me/peckshield), Twitter (http://twitter.com/peckshield), or Email (contact@peckshield.com).

1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [3]:

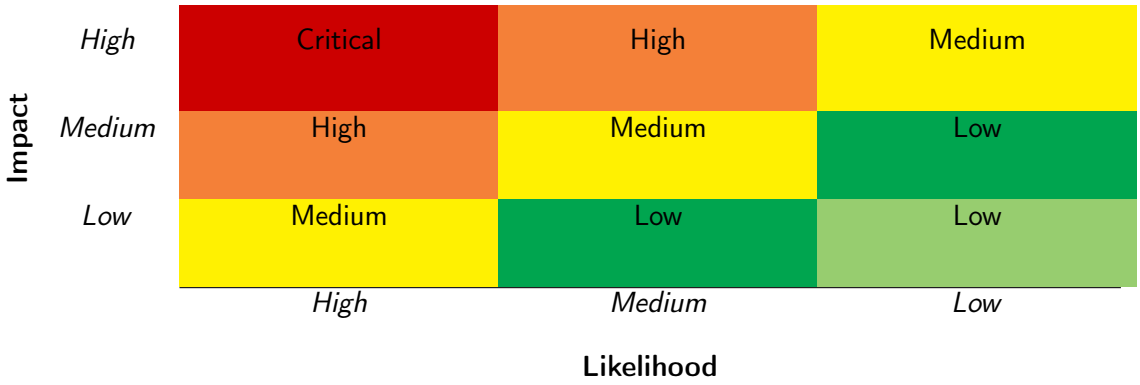
- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact measures the technical loss and business damage of a successful attack;
- Severity demonstrates the overall criticality of the risk;

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact and can be classified into four categories accordingly, i.e., *Critical*, *High*, *Medium*, *Low* shown in Table 1.2.

We perform the audit according to the following procedures:

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.
- ERC20 Compliance Checks: We then manually check whether the implementation logic of the audited smart contract(s) follows the standard ERC20 specification and other best practices.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

Table 1.2: Vulnerability Severity Classification



To evaluate the risk, we go through a list of check items and each would be labeled with a severity category. For one check item, if our tool does not identify any issue, the contract is considered safe regarding the check item. For any discovered issue, we might further deploy contracts on our private testnet and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.3.

1.4 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 of 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. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Last but not least, this security audit should not be used as investment advice.

Table 1.3: The Full List of Check Items

Category	Check Item
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead of Transfer
	Costly Loop
	(Unsafe) Use of Untrusted Libraries
	(Unsafe) Use of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
	Approve / TransferFrom Race Condition
ERC20 Compliance Checks	Compliance Checks (Section 3)
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

2 | Findings

2.1 Summary

Here is a summary of our findings after analyzing the Banking Circle token contract. During the first phase of our audit, we study the smart contract source code and run our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by our tool. We further manually review business logics, examine system operations, and place ERC20-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	# of Findings	
Critical	0	
High	0	
Medium	0	
Low	1	■
Informational	0	
Total	1	

Moreover, we explicitly evaluate whether the given contracts follow the standard ERC20 specification and other known best practices, and validate its compatibility with other similar ERC20 tokens and current DeFi protocols. The detailed ERC20 compliance checks are reported in Section 3. After that, we examine a few identified issues of varying severities that need to be brought up and paid more attention to. (The findings are categorized in the above table.) Additional information can be found in the next subsection, and the detailed discussions are in Section 4.

2.2 Key Findings

Overall, no ERC20 compliance issue was found and our detailed checklist can be found in Section 3. While there is no critical or high severity issue, the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 1 low-severity vulnerability.

Table 2.1: Key Banking Circle Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Low	Trust Issue Of Admin Keys	Security Features	Mitigated

Besides recommending specific countermeasures to mitigate the above issue(s), we also emphasize that it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms need to kick in at the very moment when the contracts are being deployed in mainnet. Please refer to Section 3 for our detailed compliance checks and Section 4 for elaboration of reported issues.

3 | ERC20 Compliance Checks

The ERC20 specification defines a list of API functions (and relevant events) that each token contract is expected to implement (and emit). The failure to meet these requirements means the token contract cannot be considered to be ERC20 -compliant. Naturally, as the first step of our audit, we examine the list of API functions defined by the ERC20 specification and validate whether there exist any inconsistency or incompatibility in the implementation or the inherent business logic of the audited contract(s).

Table 3.1: Basic View-Only Functions Defined in The ERC20 Specification

Item	Description	Status
name()	Is declared as a public view function	✓
	Returns a string, for example "Tether USD"	✓
symbol()	Is declared as a public view function	✓
	Returns the symbol by which the token contract should be known, for example "USDT". It is usually 3 or 4 characters in length	✓
decimals()	Is declared as a public view function	✓
	Returns decimals, which refers to how divisible a token can be, from 0 (not at all divisible) to 18 (pretty much continuous) and even higher if required	✓
totalSupply()	Is declared as a public view function	✓
	Returns the number of total supplied tokens, including the total minted tokens (minus the total burned tokens) ever since the deployment	✓
balanceOf()	Is declared as a public view function	✓
	Anyone can query any address' balance, as all data on the blockchain is public	✓
allowance()	Is declared as a public view function	✓
	Returns the amount which the spender is still allowed to withdraw from the owner	✓

Our analysis shows that there is no ERC20 inconsistency or incompatibility issue found in the audited Banking Circle token contract. In the surrounding two tables, we outline the respective list of basic view-only functions (Table 3.1) and key state-changing functions (Table 3.2) according to the widely-adopted ERC20 specification.

Table 3.2: Key State-Changing Functions Defined in The ERC20 Specification

Item	Description	Status
transfer()	Is declared as a public function	✓
	Returns a boolean value which accurately reflects the token transfer status	✓
	Reverts if the caller does not have enough tokens to spend	✓
	Allows zero amount transfers	✓
	Emits Transfer() event when tokens are transferred successfully (include 0 amount transfers)	✓
	Reverts while transferring to zero address	✓
transferFrom()	Is declared as a public function	✓
	Returns a boolean value which accurately reflects the token transfer status	✓
	Reverts if the spender does not have enough token allowances to spend	✓
	Updates the spender's token allowances when tokens are transferred successfully	✓
	Reverts if the from address does not have enough tokens to spend	✓
	Allows zero amount transfers	✓
	Emits Transfer() event when tokens are transferred successfully (include 0 amount transfers)	✓
	Reverts while transferring from zero address	✓
	Reverts while transferring to zero address	✓
approve()	Is declared as a public function	✓
	Returns a boolean value which accurately reflects the token approval status	✓
	Emits Approval() event when tokens are approved successfully	✓
	Reverts while approving to zero address	✓
Transfer() event	Is emitted when tokens are transferred, including zero value transfers	✓
	Is emitted with the from address set to $address(0x0)$ when new tokens are generated	✓
Approval() event	Is emitted on any successful call to approve()	✓

In addition, we perform a further examination on certain features that are permitted by the ERC20 specification or even further extended in follow-up refinements and enhancements, but not required for implementation. These features are generally helpful, but may also impact or bring certain incompatibility with current DeFi protocols. Therefore, we consider it is important to highlight them as well. This list is shown in Table 3.3.

Table 3.3: Additional Opt-in Features Examined in Our Audit

Feature	Description	Opt-in
Deflationary	Part of the tokens are burned or transferred as fee while on transfer()/transferFrom() calls	—
Rebasing	The balanceOf() function returns a re-based balance instead of the actual stored amount of tokens owned by the specific address	—
Pausable	The token contract allows the owner or privileged users to pause the token transfers and other operations	✓
Upgradable	The token contract allows for future upgrades	✓
Whitelistable	The token contract allows the owner or privileged users to whitelist a specific address such that only token transfers and other operations related to that address are allowed	✓
Mintable	The token contract allows the owner or privileged users to mint tokens to a specific address	✓
Burnable	The token contract allows the owner or privileged users to burn tokens of a specific address	✓

4 Detailed Results

4.1 Trust Issue of Admin Keys

- ID: PVE-001
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `Stablecoin`
- Category: Security Features [2]
- CWE subcategory: CWE-287 [1]

Description

In the audited token contract, there exists a privileged `owner` account that plays important roles in governing and regulating the contract-wide operations. In the following, we examine this privileged account and the related privileged accesses in current contract. In particular, the privileged functions in the `Stablecoin` contract allows for the `owner` to mint additional tokens into circulation and freeze chosen accounts.

```
54      function mint(address account, uint256 amount) external onlyOwner returns (bool) {
55          _mint(account, amount);
56          emit Mint(_msgSender(), account, amount);
57          return true;
58      }
59      ...
60      function freeze(address account) external onlyOwner {
61          frozen[account] = true;
62          emit Freeze(_msgSender(), account);
63      }
64      ...
65      function unfreeze(address account) external onlyOwner {
66          delete frozen[account];
67          emit Unfreeze(_msgSender(), account);
68      }
69      ...
70      function pause() external onlyOwner {
71          _pause();
72      }
```

```
73      ...
74      function unpause() external onlyOwner {
75          _unpause();
76      }
77      ...
78      function setTrustForwarder(address forwarder_) external onlyOwner {
79          super._setTrustForwarder(forwarder_);
80      }
```

Listing 4.1: Example Privileged Operations in `Stablecoin`

We understand the need of the privileged functions for proper contract operations, but at the same time the extra power to these privileged accounts may also be a counter-party risk to the contract users. Therefore, we list this concern as an issue here from the audit perspective and highly recommend making these privileges explicit or raising necessary awareness among protocol users.

Recommendation Make the list of extra privileges granted to these privileged accounts explicit to the token users.

Status This issue has been resolved as the team confirms the use of a multi-sig account to manage the admin key.

5 Conclusion

In this security audit, we have examined the Banking Circle token contract design and implementation. During our audit, we first checked all respects related to the compatibility of the ERC20 specification and other known ERC20 pitfalls/vulnerabilities and found no issue in these areas. We then proceeded to examine other areas such as coding practices and business logics. Overall, no issue was found in these areas, and the current deployment follows the best practice. Meanwhile, as disclaimed in Section 1.4, we appreciate any constructive feedbacks or suggestions about our findings, procedures, audit scope, etc.

References

[1] MITRE. CWE-287: Improper Authentication. https://cwe.mitre.org/data/definitions/287.html.

[2] MITRE. CWE CATEGORY: 7PK - Security Features. https://cwe.mitre.org/data/definitions/254.html.

[3] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.

[4] PeckShield. PeckShield Inc. https://www.peckshield.com.