AUDIT REPORT

CCTP Money

CCTP Contract

Prepared by SCV-Security

On 8th March 2024

Table of Contents

Introduction

SCV has been engaged by CCTP Money to conduct a comprehensive security review with the goal of identifying potential security threats and vulnerabilities within the codebase. The purpose of this audit is to evaluate the security posture of the codebase and provide actionable recommendations to mitigate any identified risks. This report presents an overview of the findings from our security audit, outlining areas of concern and proposing effective measures to enhance the codebase's security.

Scope Functionality

The CCTP bridge contract facilitates 1-click USDC transfers across multiple chains, including CCTP-enabled (e.g, Ethereum, Avalanche) chains to Cosmos app chains (e.g, Osmosis, Juno). The protocol integrates with Circle's `TokenMessenger` contract and Noble's `TokenMessengerWithMetadata` contract to handle token burnings and issue a mint request in the destination chain.

Only the `src/TokenMessengerWithMetadataWrapper.sol` contract is in the scope of the audit. The main entry points are `depositForBurn`, `depositForBurnPermit`, `depositForBurnIBC`, and `depositForBurnIBCPermit`. The last two entry points utilize IBC to perform cross-chain transfer in the Cosmos ecosystem, which will work with a future version of Noble.

Submitted Codebase

TokenMessengerWithMetadataWrapper.sol	
Repository	https://github.com/cctp-money/cctp-money-bridge-contracts/
Commit	`f12023459c4f923deae00d0652f8642e8bb070c7`
Branch	`master`

Methodologies

SCV performs a combination of automated and manual security testing based on the scope of testing. The testing performed is based on the extensive experience and knowledge of the auditor to provide the greatest coverage and value to CCTP Money. Testing includes, but is not limited to, the following:

- Understanding the application and its functionality purpose.
- Deploying SCV in-house tooling to automate dependency analysis and static code review.
- Analyse each line of the code base and inspect application security perimeter.
- Review underlying infrastructure technologies and supply chain security posture.

Code Criteria

This section provides an evaluation of specific criteria aspects as described below:

- **Documentation:** Evaluating the presence and comprehensiveness of publicly available or provided explanatory information, diagram flowcharts, comments, and supporting documents to enhance code understanding.
- **Coverage:** Evaluating whether the code adequately addresses all necessary cases and scenarios, ensuring that the intended functionality or requirements are sufficiently covered.
- **Readability:** Assessing how easily the code can be understood and maintained, considering factors such as code structure, naming conventions, and overall organisation.
- **Complexity:** Evaluating the complexity of the code, including factors such as, number of lines, conditional statements, and nested structures.

The status of each criteria is categorised as either **SUFFICIENT** or **NOT-SUFFICIENT** based on the audit assessment. This categorisation provides insights to identify areas that may require further attention and improvement.

Criteria	Status	Notes
Documentation	**SUFFICIENT**	Documentation is available at https://developers.circle.com/stablecoins/docs/cctp-getting-started and https://docs.cctp.money/.
Coverage	**SUFFICIENT**	`forge coverage` reports the following output: ● Lines: 73.33% (33/45) ● Statements: 73.68% (42/57) ● Branches: 80.00% (8/10) ● Functions: 58.33% (7/12)
Readability	**SUFFICIENT**	N/A
Complexity	**SUFFICIENT**	N/A

Findings Summary

Summary Title	Risk Impact	Status
`replaceDepositForBurn` cannot be called to recover stuck funds	MODERATE	ACKNOWLEDGED
Events are not emitted for important executions	INFO	ACKNOWLEDGED
Redundant `updateOwner` function	INFO	ACKNOWLEDGED
General code recommendations	INFO	ACKNOWLEDGED

Audit Observations

The audit observations section is intended to present potential findings that are related to the underlying design of the protocol and would require underlying design changes to remediate that may change the overall functioning of the protocol. SCV asks that the client formulate responses to add context to validate or invalidate the following concerns.

1. Infinite token allowance approved

The allowance mechanism in ERC20 contracts allows the spender to withdraw the approved amount from the owner. If the allowance amount is `type(uint256).max`, it means the approved amount is unlimited.

The `TokenMessengerWithMetadataWrapper` contract approves unlimited allowance to the `TokenMessenger` and `TokenMessengerWithMetadata` contracts. While this reduces gas consumption (don't need to call the `approve` function every time) and increases user experience, this presents a potential security risk as the `TokenMessenger` and `TokenMessengerWithMetadata` contracts can withdraw all USDC from the `TokenMessengerWithMetadataWrapper` contract.

If the `TokenMessenger` or `TokenMessengerWithMetadata` contract is compromised or contains an entry point to consume other users' approval, the USDC in the `TokenMessengerWithMetadataWrapper` contract can be stolen, which are the fees for the protocol.

Potential Remediation:

Consider modifying the implementation to approve the required amount instead of infinite allowance approval.

Findings Technical Details

1. `replaceDepositForBurn` cannot be called to recover stuck funds

RISK IMPACT: MODERATE	**STATUS:** ACKNOWLEDGED

Revision Notes

Team acknowledges the intention of explicitly trusting Circle's contracts and prefers a clear separation of concerns between theirs contracts and Circle's.

Description

When users want to bridge their USDC to another chain, they are required to provide a `mintRecipient` (address on the destination chain) to receive the funds. If the address cannot receive funds (e.g., due to an invalid address or blacklisted receiver), the mint request will fail. To recover the funds, Circle's `TokenMessenger` contract exposes a `replaceDepositForBurn` function that allows users to update the receiver address and re-issue the mint request.

However, the `replaceDepositForBurn` function only allows the `TokenMessengerWithMetadataWrapper` contract to call it. When interacting with the `TokenMessenger` and `TokenMessengerWithMetadata` contracts, the sender is set to the `TokenMessengerWithMetadataWrapper` contract instead of the actual user:

- `lib/evm-cctp-contracts/src/TokenMessenger.sol:459`
- `lib/cctp-contracts/src/TokenMessengerWithMetadata.sol:74`

As a result, users will be unable to recover their funds due to `lib/evm-cctp-contracts/src/TokenMessenger.sol:261-266`, causing a loss of funds scenario.

Recommendation

Consider implementing an entry point for users to call the `replaceDepositForBurn` function in the `TokenMessenger` contract.

2. Events are not emitted for important executions

RISK IMPACT: INFORMATIONAL	STATUS: ACKNOWLEDGED

Revision Notes

Team advises that it will be implemented in future versions.

Description

Throughout the codebase, there are several instances of important executions that lack events being emitted.

Relevant events and attributes should be emitted for off-chain listeners to record and index the configured parameters. These instances include:

- The `constructor` function should emit the relevant parameters when the contract is initialized.
- The `updateTokenMessengerWithMetadata` function should emit the new `TokenMessengerWithMetadata` contract address.
- The `setFee` function should emit new fee values for the destination domain.
- The `updateCollector` function should emit the new fee collector address.
- The `updateFeeUpdater` function should emit the new fee updater address.

Recommendation

Consider emitting relevant attributes or events where specified.

3. Redundant `updateOwner` function

RISK IMPACT: INFORMATIONAL	**STATUS:** ACKNOWLEDGED

Revision Notes

Team advises that it will be implemented in future versions.

Description

The `updateOwner` function in `src/TokenMessengerWithMetadataWrapper.sol:288` is redundant as the `transferOwnership` function in `lib/solmate/src/auth/Owned.sol:39` already covers its purpose.

Recommendation

Consider removing the `updateOwner` function and instead use the `transferOwnership` function to update ownership.

4. General code recommendations

RISK IMPACT: INFORMATIONAL	STATUS: ACKNOWLEDGED

Revision Notes

Team advises that it will be implemented in future versions.

Description

In `src/TokenMessengerWithMetadataWrapper.sol:114`, `183`, `250`, it is generally recommended to use `safeTransferFrom` instead of `transferFrom` for ERC20 token transfers due to security and better handling of failure cases.

Additionally, it is recommended to use `call` instead of `transfer` in line `306` due to the `transfer` function's hard dependency on gas costs. Relying on fixed gas amounts becomes risky because the gas costs associated with various operations can change over time.

Recommendation

Consider updating the code to adhere to the best practices.

Document Control

Version	Date	Notes
-	19th February 2024	Security audit commencement date.
0.1	22nd February 2024	Initial report with identified findings delivered.
0.5	-	Fixes remediations implemented and reviewed.
1.0	8th March 2024	Audit completed, final report delivered.

Appendices

A. Appendix - Risk assessment methodology

SCV-Security employs a risk assessment methodology to evaluate vulnerabilities and identified issues. This approach involves the analysis of both the LIKELIHOOD of a security incident occurring and the potential IMPACT if such an incident were to happen. For each vulnerability, SCV-Security calculates a risk level on a scale of 5 to 1, where 5 denotes the highest likelihood or impact. Consequently, an overall risk level is derived from combining these two factors, resulting in a value from 10 to 1, with 10 signifying the most elevated level of security risk

Risk Level	Range
CRITICAL	10
SEVERE	From 9 to 8
MODERATE	From 7 to 6
LOW	From 5 to 4
INFORMATIONAL	From 3 to 1

LIKELIHOOD and **IMPACT** would be individually assessed based on the below:

Rate	LIKELIHOOD	IMPACT
5	**Extremely Likely**	Could result in severe and irreparable consequences.
4	**Likely**	May lead to substantial impact or loss.
3	**Possible**	Could cause partial impact or loss on a wide scale.
2	**Unlikely**	Might cause temporary disruptions or losses.
1	**Rare**	Could have minimal or negligible impact.

B. Appendix - Report Disclaimer

This report should not be regarded as an "endorsement" or "disapproval" of any specific project or team. These reports do not indicate the economics or value of any "product" or "asset" created by a team or project that engages SCV-Security for a security review. The audit report does not make any statements or warranties about the code's utility, safety, suitability of the business model, regulatory compliance of the business model, or any other claims regarding the fitness of the implementation for its purpose or its bug-free status. The audit documentation is intended for discussion purposes only. The content of this audit report is provided "as is," without representations and warranties of any kind, and SCV-Security disclaims any liability for damages arising from or in connection with this audit report. Copyright of this report remains with SCV-Security.

THANK YOU FOR CHOOSING

SCV SECURITY

🌐 scv.services

✉ contact@scv.services