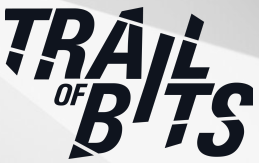

USDC Custom Gateway

Security Assessment (Summary Report)

August 29, 2024

Prepared for:
Offchain Labs
Offchain Labs

Prepared by: **Gustavo Grieco and Jaime Iglesias**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
497 Carroll St., Space 71, Seventh Floor
Brooklyn, NY 11215
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Mary O'Brien, Project Manager
mary.obrien@trailofbits.com

The following engineering director was associated with this project:

Josselin Feist, Engineering Director, Blockchain
josselin.feist@trailofbits.com

The following consultants were associated with this project:

Gustavo Grieco, Consultant
gustavo.grieco@trailofbits.com

Jaime Iglesias, Consultant
jaime.iglesias@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
July 11, 2024	Pre-project kickoff call
July 24, 2024	Delivery of report draft
July 24, 2024	Report readout meeting
August 29, 2024	Delivery of summary report

Project Targets

The engagement involved a review and testing of the targets listed below.

Bridged USDC custom gateway

Repository	https://github.com/OffchainLabs/token-bridge-contracts/pull/87
Version	d2c09d2
Type	Solidity
Platform	Ethereum

Upgrade ArbOS Governance Action contract

Repository	https://github.com/ArbitrumFoundation/governance/pull/297
Version	163d7fa
Type	Solidity
Platform	Ethereum

Executive Summary

Engagement Overview

Offchain Labs engaged Trail of Bits to review the security of the USDC custom gateway. This gateway complies with the bridged USDC standard. The contracts can be used by new Orbit chains that want to provide a USDC bridging solution while maintaining the ability to upgrade to native USDC at a later point. We also reviewed a small governance action to upgrade the ArbOS version.

A team of two consultants conducted the review from July 15 to July 23, 2024, for a total of two engineer-weeks of effort. With full access to source code and documentation, we performed a manual code review of the code under scope.

Observations and Impact

The focus of this audit was the USDC custom gateway. We reviewed the correctness of the gateway code deployment, the expected deposit and withdrawal workflows, and the migration workflow toward native token for any possible pitfalls, error-prone steps, or underspecified instructions. The USDC custom gateway uses the existing gateway contracts to implement its functionality, overriding a number of functions. We examined the relevant inherited code, but we have not audited all of the remaining contracts in that codebase.

During our review, we found a number of corner cases and unspecified/undocumented situations that could potentially result in funds stuck either before or after the USDC native migration.

Recommendations

As a general recommendation, the Offchain Labs team should review internal and external documentation for assumptions or requirements regarding the usage of the reference USDC implementation.

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Lack of proper scripts to deploy a USDC token in L2	Denial of Service	**Low**
2	USDC token bridge contract does not handle certain corner cases	Denial of Service	**Low**
3	Valid USDC in-flight operations should be clearly specified	Documentation	**Low**

Detailed Findings

1. Lack of proper scripts to deploy a USDC token in L2	
Severity: **Low**	Difficulty: **Medium**
Type: Denial of Service	Finding ID: TOB-USDC-BRIDGE-001
Target: `scripts`	

Description
The deployment of the USDC token itself requires a number of conditions that are not properly checked in the deployment scripts.

The Circle documentation states how the procedure of deployment should account for certain function calls:

```
FiatToken has a number of one-time use initialization functions (listed below) that
are not permissioned, and therefore should be called during contract deployment.
Due to the proxy pattern used by FiatToken, after (or while) specifying the current
implementation that the proxy points to, these initialization functions must be
called to set the values correctly on the proxy's storage. If not, then any caller
could invoke them in the future.
It is also recommended to call these functions directly on the implementation
contract itself, separately from the proxy, to disallow any outside caller invoking
them later. At the timing of writing, these initialization functions include:

* initialize
* initializeV2
* initializeV2_1
* initializeV2_2
```

Figure 3.1: Part of the documentation on the deployment of the FiatToken contract (USDC reference implementation)

However, it is unclear if the L2 USDC token deployment includes these recommendations. We have access to only the Sepolia test scripts for local deployment, and these perform only individual transactions to the proxy:

```
    const l2UsdcFiatToken = IFiatToken__factory.connect(
      l2UsdcProxyAddress,
      deployerL2Wallet
    )
    const masterMinterL2 = deployerL2Wallet
```

```
await (
  await l2UsdcFiatToken.initialize(
    'USDC token',
    'USDC.e',
    'USD',
    6,
    masterMinterL2.address,
    ethers.Wallet.createRandom().address,
    ethers.Wallet.createRandom().address,
    deployerL2Wallet.address
  )
).wait()
await (await l2UsdcFiatToken.initializeV2('USDC')).wait()
await (
  await l2UsdcFiatToken.initializeV2_1(ethers.Wallet.createRandom().address)
).wait()
await (await l2UsdcFiatToken.initializeV2_2([], 'USDC.e')).wait()
```

Figure 2.1: Part of the deployment local scripts

Additionally, the same calls to the implementation contract are missing.

Exploit Scenario

Alice deploys an L2 USDC token contract in her Arbitrum chain to support the custom token bridge but forgets to call the initialization functions, allowing any other account to perform these calls but with other parameters.

Recommendations

Short term, consider using governance actions and dedicated contracts to deploy and properly configure the L2 USDC token contract in a single transaction.

Long term, review the Circle documentation for additional steps, assumptions, or requirements regarding the usage of the reference USDC implementation.

2. USDC token bridge contract does not handle certain corner cases	
Severity: **Low**	Difficulty: Low
Type: Denial of Service	Finding ID: TOB-USDC-BRIDGE-002
Target: `L1ArbitrumGateway.sol`, `L2USDCGateway.sol`	

Description

The USDC token bridge does not handle special cases, such as when the token itself is paused or the destination address is denylisted.

The handling of USDC in new chains will be performed using a custom token bridge that contains specific code for deposits and withdrawals. However, certain interactions are not handled properly and can result in stuck funds during either the confirmation of deposit or withdraws. In particular, this can happen if:

- The USDC contract is paused
- The recipient of the deposit or the destination of the withdraw is denylisted

For both cases, code that checks for these corner cases is missing. For instance, in the case of deposits into L2, the final step involves minting USDC:

```
    function inboundEscrowTransfer(address _l2Address, address _dest, uint256
_amount)
        internal
        override
    {
        IFiatToken(_l2Address).mint(_dest, _amount);
    }
```

Figure 2.1: The `inboundEscrowTransfer` function from `L2ArbitrumGateway`

For withdrawals from L2, the final step involves transferring USDC from the bridge to the destination:

```
    function inboundEscrowTransfer(
        address _l1Token,
        address _dest,
        uint256 _amount
    ) internal virtual {
        // this method is virtual since different subclasses can handle escrow
differently
        IERC20(_l1Token).safeTransfer(_dest, _amount);
```

```
    }
```

Figure 2.2: The `inboundEscrowTransfer` function from `L1ArbitrumGateway`

Both steps will be blocked in the circumstances listed above with no possibility of the users of undoing their operations.

Exploit Scenario

Alice wants to transfer to Eve. Alice is not sure if Eve's address is denylisted, but she expects that if this the case, the operation will revert and she will be able to recover her funds, so Alice proceeds with the deposit. However, Alice's funds are stuck in the bridge since the final step of the deposit/withdrawal always reverts.

Recommendations

Short term, consider adding code to properly handle these corner cases, allowing users to recover their funds (e.g., by undoing the deposit in case the contract is paused or the destination is denylisted as part of the L2 deposit flow).

Long term, review the Circle documentation for additional steps, assumptions, or requirements regarding the usage of the reference USDC implementation. Thoroughly document this behavior so that users are aware of it.

3. Valid USDC in-flight operations are not clearly specified	
Severity: **Low**	Difficulty: **Medium**
Type: Documentation	Finding ID: TOB-USDC-BRIDGE-003
Target: Protocol level	

Description

The process for migrating to a native token requires a clear specification of the in-flight operations to avoid miscounting the total supply.

The USDC custom gateway documentation describes how to migrate from bridge USDC to a native token. In particular, it describes how to deal with unclaimed deposits:

```
There should be no in-flight deposits when minter role is revoked. If there are any,
they should be executed (can be done by anyone by claiming the failed retryable
ticket which does the USDC depositing).
…
chain owner reads the total supply of USDC on child chains. Then, he invokes
setBurnAmount(uint256) on the parent child gateway where the amount matches the
total supply
* in case there are unclaimed deposits, their total amount should be added to the
supply as those tokens shall eventually be minted by child chain gateway
```

Figure 3.1: Part of the documentation on the migration process to native

However, extensive documentation on what constitutes an in-flight operation is needed to avoid confusion when these are counted. In particular, the documentation should include a procedure to determine if a transaction is a potential deposit/withdrawal or not. A clear specification should essentially describe how to use events in certain contracts to detect transactions that are valid deposits/withdrawals. This specification should be used to match all previous deposits, even the ones that are still in the retryable ticket queue. (These deposits can be as old as the chain, since retryable tickets can be kept alive indefinitely if users provide values for them.)

Additionally, the documentation should also explain how to constrain the set of possible deposits to avoid the following scenarios:

- Withdrawals/deposits cannot be finalized (e.g., the destination address is denylisted).
- Transactions look like deposits/withdrawals, but will always revert.

Exploit Scenario

Eve creates a number of retryable tickets that seem to interact with the USDC custom bridge in L2. Alice is the chain owner and wants to start the procedure to convert the USDC to a native token. She miscounts Eve's transactions as valid deposits and burns an incorrect amount of tokens.

Recommendations

Short term, expand the documentation to provide a clear specification of in-flight operations.

Long term, review the internal documentation of each component to identify any gaps or underspecified sections.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Quality Findings

These findings are not issues, but represent opportunities to enhance the security of the codebase.

Documentation

- The USDC custom gateway can be deployed in Orbit chains where a custom token fee is used; however, it cannot be used if the USDC itself is the fee token. Add this behavior to the documentation to avoid confusion.

Smart Contracts

- Beware of isolated changes in code style. The new code uses custom errors while the old gateways use require statements; this can lead to errors due to differences in the way conditions are checked (e.g., negative vs. positive conditions).

```
function initialize(
    address _l1Counterpart,
    address _router,
    address _l1USDC,
    address _l2USDC,
    address _owner
) public {
    if (_l1USDC == address(0)) {
        revert L2USDCGateway_InvalidL1USDC();
    }
    if (_l2USDC == address(0)) {
        revert L2USDCGateway_InvalidL2USDC();
    }
    if (_owner == address(0)) {
        revert L2USDCGateway_InvalidOwner();
    }
```

Figure D.1: `initialize` function in L2USDCGateway.sol#L58-L73

```
function _initialize(address _l1Counterpart, address _router) internal override {
     TokenGateway._initialize(_l1Counterpart, _router);
     // L1 gateway must have a router
     require(_router != address(0), "BAD_ROUTER");
 }
```

Figure D.2: `initialize` function in L2ArbitrumGateway.sol#L72-L76