Fine penetration tests for fine websites

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Review-Report BOB modified USDC bridge library 04.2024

Cure53, Dr.-Ing. M. Heiderich, Dr. A. Pirker, Dr. N. Kobeissi

Index

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Introduction

> *"BOB (Build on Bitcoin) is the first Bitcoin L2 with full EVM compatibility & native Bitcoin support empowering everyone to build and innovate on Bitcoin. BOB (Build on Bitcoin) enables DeFi and innovation across all fields of Bitcoin use cases & experimentation. Whatever you're building on Bitcoin, BOB is your swiss-army-knife for all things build on Bitcoin."*
>
> From https://www.gobob.xyz/

This report, assigned the unique identifier *BOB-02-WP1*, presents the results of a Cure53 cryptography review and source code audit against the modified USDC bridges library.

Stakeholders from Distributed Crafts Ltd. contacted Cure53 in April 2024 to discuss the aims and expected outcomes of the project. Once the scope and budget had been finalized, the review was scheduled for CW16 of the same month. Three senior pentesters from Cure53's talent pool were selected to complete the assignment, based on their proficiency and experience handling components of this nature.

To aid the white-box technical analysis, Cure53 was provided with sources and other necessary materials. One distinct Work Package (WP) was created for efficiency reasons, defined by the following headings:

- **WP1**: Cryptography reviews & code audits against modified USDC bridge library

Please note that the second work package (WP2) tracked within this project covered a different area of code and is documented in a separate report.

A number of essential preparations were completed in April 2024, namely in CW15, to encourage a seamless working environment. Throughout the assessment, communication channels remained open via a dedicated Telegram channel shared by the development team and Cure53. All relevant personnel from both parties joined the channel and engaged in the collaborative process when required. The scope definition was clearly mapped out and the test team was suitably equipped to conduct the initiatives. Cure53 provided regular status updates on the testing progress and associated findings, though live reporting was deemed unnecessary.

As for the findings, a total of two were identified after a comprehensive review of the WP1 scope items. To break these down, one was categorized as a security vulnerability and the other was a lower risk, a common vulnerability.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

The provided source code proved resilient to a plethora of typical breach schemes, which confirms the development team's effectiveness in minimizing the attack surface and mitigating vulnerabilities within the assessed components. However, several of the findings represent valuable hardening recommendations.

A number of vulnerabilities were encountered that should be addressed and resolved at the earliest possible convenience for the internal team. The sole *High* rated vulnerability, which pertains to a potential USDC bridge takeover scenario (see BOB-02-001), should be prioritized for remediation. To caveat this, the development team’s proactive actions to resolve some of the vulnerabilities during the active testing phase is praiseworthy.

Moving forward, the report presents a selection of key chapters for ease of reference. Firstly, the *Scope* clarifies the test setup and available materials. Next, the *Identified Vulnerabilities* and *Miscellaneous Issues* comprise all observed findings in ticket format. The tickets provide supporting information such as a technical rundown, Proof-of-Concept (PoC), affected code examples, and remediation advice.

To finalize the document, the *Conclusions* section summarizes Cure53’s opinion of the scope’s security performance by taking a closer look at the coverage and discoveries.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Scope

- **Cryptography reviews & code audits against BOB Solidity SCs & USDC bridge library**
  - **WP1:** Cryptography reviews & code audits against modified USDC bridge library
    - **Sources:**
      - https://github.com/bob-collective/optimism/pull/1
    - **Commits:**
      - https://github.com/bob-collective/optimism/pull/1/commits/a80a28610962d361cf1c8b67c3f513d0ffb1f792
      - https://github.com/bob-collective/optimism/pull/1/commits/12996ea145327eab87684376e79e845108cf6a67
      - https://github.com/bob-collective/optimism/pull/1/commits/2ba5514db1a8cf68bedd6de34000498824f28246
  - **Test-supporting material was shared with Cure53**
  - **All relevant sources were shared with Cure53**

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Identified Vulnerabilities

The following section lists all vulnerabilities and implementation issues identified during the testing period. Notably, findings are cited in chronological order rather than by degree of impact, with the severity rank offered in brackets following the title heading for each vulnerability. Furthermore, all tickets are given a unique identifier (e.g., *BOB-02-001*) to facilitate any future follow-up correspondence.

BOB-02-001 WP1: Front-running allows USDC bridge takeover on init *(High)*

Fix note: *The issue was mitigated by the customer during the assessment by disabling initializers for all USDC bridges when creating them directly without a proxy. The fix was verified by Cure53 and the problem no longer exists.*

The deployment of a new L1 or L2 USDC bridge involves two steps, creation and initialization. The bridge constructors are empty, while both contracts implement a function entitled *initialize*. Firstly, the bridge owner must create the corresponding bridge by creating a new contract instance via its constructor. Secondly, the bridge owner must call the *initialize* function of the respective bridge to claim ownership. The *initialize* function also sets the addresses of the respective tokens, the other bridge, and the utilized messenger. Here, testing determined that the process of bridge initialization (and also therefore ownership claiming) is vulnerable to front-running attacks.

This constitutes an immediate security risk for all USDC bridge contracts. In the event that a victim creates a new USDC bridge contract instance, an attacker could attempt to front-run the invocation of the *initialize* function and take over the newly created bridge instance. Since bridges originate from the USDC bridge feature, the adversary could utilize the contract instances to mount Denial-of-Service (DoS) attacks or cause further, potentially even financial, damage to victim users.

The internal team clarified that all bridge contracts will be inspected and the deployments in question will not be used on the UI in the event of front-running. Recurring attempts before deployment success without front-running will result in a waste of gas for the victim operator. Moreover, one could utilize a hijacked USDC bridge to instigate severe damage.

The excerpts below demonstrate that bridge construction procedures are completely decoupled from their initialization. The constructors fail to transfer the contract ownership and the *initialize* functions correspond to public functions without access control, which renders them invokable by anyone. Ultimately, the respective *initialize* functions invoke the internal *__UsdcBridge_init* function of the *UsdcBridge* contract, which sets all fields and transfers the ownership.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Affected file #1:
packages/contracts-bedrock/src/L1/L1UsdcBridge.sol

Affected code #1:
```
constructor() UsdcBridge() { }
[...]
function initialize(
    CrossDomainMessenger _messenger,
    UsdcBridge _otherBridge,
    address _l1Usdc,
    address _l2Usdc,
    address _owner
)
    public
    initializer
{
    __UsdcBridge_init({
        _messenger: _messenger,
        _otherBridge: _otherBridge,
        _l1Usdc: _l1Usdc,
        _l2Usdc: _l2Usdc,
        _owner: _owner
    });
}
```

Affected file #2:
packages/contracts-bedrock/src/L2/L2UsdcBridge.sol

Affected code #2:
```
constructor() UsdcBridge() { }
[...]
function initialize(UsdcBridge _otherBridge, address _l1Usdc, address
_l2Usdc, address _owner) public initializer {
    __UsdcBridge_init({
        _messenger:
CrossDomainMessenger(Predeploys.L2_CROSS_DOMAIN_MESSENGER),
        _otherBridge: _otherBridge,
        _l1Usdc: _l1Usdc,
        _l2Usdc: _l2Usdc,
        _owner: _owner
    });
}
```

Affected file #3:
packages/contracts-bedrock/src/universal/UsdcBridge.sol

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Affected code #3:

```
function __UsdcBridge_init(
    CrossDomainMessenger _messenger,
    UsdcBridge _otherBridge,
    address _l1Usdc,
    address _l2Usdc,
    address _owner
)
    internal
    onlyInitializing
{
    messenger = _messenger;
    otherBridge = _otherBridge;
    l1Usdc = _l1Usdc;
    l2Usdc = _l2Usdc;
    _transferOwnership(_owner);
}
```

To mitigate this issue, Cure53 strongly recommends atomically creating and initializing the concrete USDC bridges via the constructor of L1 and L2 USDC bridges. When used through a transparent proxy, the development team could consider other solutions such as *upgradeToAndCall*[1], for example. In this case, one must restrict the ability to invoke the respective *initialize* functions when creating and using L1 and L2 USDC bridges directly.

[1] https://github.com/eth[...]imism/optimism/blob/a12[...]d2/packages/[...]/contracts/universal/Proxy.sol#L98

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Miscellaneous Issues

This section covers any and all noteworthy findings that did not incur an exploit but may assist an attacker in successfully achieving malicious objectives in the future. Most of these results are vulnerable code snippets that did not provide an easy method by which to be called. Conclusively, while a vulnerability is present, an exploit may not always be possible.

BOB-02-002 WP1: Potential loss of ownership via 0-address owner *(Low)*

Fix note: *The issue was mitigated by the customer during the assessment and fix-verified by Cure53.*

The pull request for the integration of USDC bridges contains a USDC manager contract. This contract enables an allow-listed address (that can be set by the owner of the USDC manager contract instance) to alter the administrator of a USDC proxy, transfer the ownership of a token proxy, and remove the minter role from a master minter address. The contract participates in the management process of USDC tokens. However, the audit team found that the contract fails to check for the 0 address when transferring the USDC roles.

In Solidity, developers often use the 0 address to indicate the special case of a non-existent address. An attacker that corresponds to the allow-listed takeover origin of a USDC manager could intentionally provide the 0 address when transferring USDC roles. Since it remains unclear how the referenced contracts handle 0 addresses, this could potentially render the USDC token proxy inaccessible.

The excerpt below underscores that the *owner* address parameter of the *transferUSDCRoles* function is not checked for the 0 address.

Affected file:
packages/contracts-bedrock/src/L2/UsdcManager.sol

Affected code:

```
function transferUSDCRoles(address owner) external {
    require(msg.sender == whitelistedTakeoverOrigin, "Unauthorized
transfer");
    // Change proxy admin
    IUsdcProxy(tokenProxyAddress).changeAdmin(owner);
    // remove minter
    IMasterMinter(masterMinterAddress).removeMinter();
    // Transfer implementation owner
    IUsdcImpl(tokenProxyAddress).transferOwnership(owner);
}
```

To mitigate this issue, Cure53 advises checking the *owner* parameter of the *transferUSDCRoles* function for the 0 address using a *require* statement.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

Conclusions

This Q2 2024 audit consisted of two work packages. The one covered in this report, WP1, focused on evaluating the security of the pull request for the modified USDC bridges library. The WP1 pull request was publicly available, and the customer provided the URL prior to the assessment. Since this engagement was a source code audit only, no additional resources such as infrastructure or testing environment were provided.

The external and in-house teams remained in contact via a specifically established Telegram channel, which hosted open questions and allowed the testers to relay progress updates. The cross-team communication was generally excellent and assistance was provided whenever requested.

The auditors achieved an adequate level of coverage within the allotted time frame. In context, the source code of both work packages was compositionally moderate. The smart contracts in scope are written in Solidity. On a positive note, the respective code bases were well organized at the time of inspection.

The smart contracts were reviewed for common vulnerabilities that affect Solidity specifically:

- The first area of concern was reentrancy issues. Cure53 found that the smart contracts perform external calls almost exclusively after performing all state-changing operations to the contract itself, which tends to rule out reentrancy flaws by default. Despite strenuous efforts in this area, the test team was unable to discover any connected problems.

- Next, the team scoured the source code for issues related to front-running, another prominent vulnerability class for smart contracts. Here, testing determined that both the L1 and L2 USDC bridges were vulnerable to takeovers via front-running upon initialization, as documented in ticket BOB-02-001. However, the development team pushed a fix that disables the *initialize* function of the respective bridges upon construction, which effectively renders these functions inaccessible when creating an L1 or L2 USDC bridge without a proxy.

- Oftentimes, Solidity contracts suffer from arithmetic errors due to loss of precision, resulting from an incorrect order of arithmetic operations. Furthermore, former versions of Solidity neglect to check for overflow and underflow situations. Nevertheless, the assessors verified that the smart contracts exhibit negligible attack surface with regards to these circumstances.

Dr.-Ing. Mario Heiderich, Cure53
Wilmersdorfer Str. 106
D 10629 Berlin
cure53.de · mario@cure53.de

- The team also stringently investigated the visibilities and modifiers of the contract functions. However, the conclusion was made that the smart contracts do not expose any mechanisms that would otherwise widen the attack surface, contributing to the robust overall impression.

- Another focus aspect was the likelihood of DoS situations and griefing attacks, which could be attempted by threat actors in order to disrupt or modify the behavior of smart contracts. The team explored this vulnerability angle in depth, discovering several points of contention.

- The absence of parameter validations in smart contracts also correlates with the aforementioned griefing attack vectors, since parameters deviating from expected or tolerated values may render the usage of a contract unfavorable or even unacceptable. Here, the observation was made that some of the contracts fail to check parameters for 0 addresses, which could potentially lead to a loss of ownership or funds, as reported in ticket BOB-02-002.

- Elsewhere, the audit team searched for missing authorization checks and attacks leading to impersonations. Here, it was positively concluded that the contracts successfully nullify these compromise strategies. Cure53 also sought to pinpoint any logical flaws such as the assignment of absolute approval values, for instance, though no associated behaviors were noted.

In summary, Cure53 can confirm that the provided source code exhibits satisfactory security proficiency under the current configuration. Many of the identified issues correspond to hardening recommendations that will provide defense-in-depth and further protect assets against malicious actors outside of the current threat model.

The development team has successfully minimized the exposed attack surface and negated most vulnerability classes that could plausibly affect the characteristics in-scope for this audit. Lastly, the in-house team’s diligence toward addressing some of the pressing concerns soon after detection is commendable and corroborates the argument that their framework is progressing in an upward trajectory from a security viewpoint.

Cure53 would like to thank Gregory Hill, Sander Bosma, and Dominik Harz from the Distributed Crafts Ltd. team for their excellent project coordination, support, and assistance, both before and during this assignment.