

# Security Audit Report



## Coffer Network Coffer Smart Contracts

Initial Report // January 10, 2025  
Final Report // January 23, 2025

### Team Members

Ahmad Jawid Jamiulahmadi // Senior Security Auditor  
Mukesh Jaiswal // Senior Security Auditor



# Table of Contents

<u>1.0 Scope</u>	3
<u>1.1 Technical Scope</u>	
<u>2.0 Executive Summary</u>	4
<u>2.1 Schedule</u>	
<u>2.2 Overview</u>	
<u>2.3 Threat Model</u>	
<u>2.4 Secure Implementation</u>	
<u>2.5 Tests</u>	
<u>2.6 Project Documentation</u>	
<u>3.0 Key Findings Table</u>	5
<u>4.0 Findings</u>	6
<u>4.1 increaseAllowance Function Decreases the Allowance</u>	
High	Fixed
<u>4.2 Event Emits an Incorrect Value</u>	
Low	Fixed
<u>4.3 Duplicate Events Can be Emitted in wipeFrozenAddress Function</u>	
Low	Fixed
<u>4.4 Important Protocol Privileged Addresses Can be Frozen</u>	
Low	Not Fixed
<u>4.5 Incorrect Use of Access Role in wipeFrozenAddress Function</u>	
Low	Fixed
<u>4.6 Address With Zero Balance Can be Frozen</u>	
None	Fixed
<u>4.7 Duplicate Functions</u>	
None	Not Fixed
<u>4.8 No Constructor Implemented in Smart Contracts</u>	
None	Fixed
<u>4.9 Implement Custom Errors to Save Gas</u>	
None	Not Fixed
<u>4.10 Privileged Addresses Can Be Set to Existing Values</u>	
None	Fixed
<u>4.11 Unlock Pragma Version</u>	
None	Fixed
<u>5.0 Appendix A</u>	15
<u>5.1 Severity Rating Definitions</u>	
<u>6.0 Appendix B</u>	16
<u>6.1 Thesis Defense Disclaimer</u>	



# About Thesis Defense

Defense is the security auditing arm of Thesis, Inc., the venture studio behind tBTC, Fold, Mezo, Acre, Tahoe, Etcher, and Embody. At Defense, we fight for the integrity and empowerment of the individual by strengthening the security of emerging technologies to promote a decentralized future and user freedom. Defense is the leading Bitcoin applied cryptography and security auditing firm. Our team of security auditors have carried out hundreds of security audits for decentralized systems across a number of ecosystems including Bitcoin, Ethereum + EVMs, Stacks, Cosmos SDK, NEAR and more. We offer our services within a variety of technologies including smart contracts, bridges, cryptography, node implementations, wallets and browser extensions, and dApps.

Defense will employ the Defense Audit Approach and Audit Process to the in scope service. In the event that certain processes and methodologies are not applicable to the in scope services, we will indicate as such in individual audit or design review SOWs. In addition, Thesis Defense provides clear guidance on successful Security Audit Preparation.

## Section 1.0 Scope

### Technical Scope

- **Repository:** <https://github.com/coffer-network/coffer-smart-account/compare/defense-audit>
- **Audit Commit:** 47ca7cd1af3497b68836c6937a441ff64b305be7
- **Verification Commit:** b34df60c3a4cfc37e62aebda240ff8c1cc3e4ee9
- **Files in Scope:**
  - Interface/ICofferErc20.sol
  - Interface/ICofferManageErc20.sol
  - Interface/IOwnerable.sol
  - Interface/IFreezable.sol
  - Interface/IPausable.sol
  - CofferErc20.sol
  - CofferManageErc20.sol
  - Freezable.sol
  - Ownerable.sol
  - Pausable.sol



# Section 2.0

## Executive Summary

### Schedule

This security audit was conducted from January 7, 2025 to January 10, 2025 by 2 security auditors for a total of 1 person-week.

### Overview

The Coffe Network smart contract is an ERC-20 token that tracks the Bitcoin balance of users stored on the Bitcoin Network. This tracked Bitcoin generates yield over time, and the accrued yield is distributed to the end users.

### Threat Model

As part of our threat model, we conducted a thorough review to confirm that the token contract adheres to the ERC-20 token standard. We also verified that any customizations or modifications introduced to the standard did not introduce security vulnerabilities or compromise the integrity of the token. In addition, we examined the contract for compliance with general best practices for Solidity smart contract development, ensuring that the code follows recommended security patterns and minimizes potential risks.

### Secure Implementation

In our code review we found that security has been taken into consideration by adhering to the ERC-20 token standard and implementing appropriate access control. However, we identified issues whereby the `increaseAllowance` function fails to correctly increase the allowance ([Issue 1](#)). Certain events are not emitted properly ([Issue 2](#), [Issue 3](#)). Additionally, the contracts allow freezing of important protocol privileged addresses ([Issue 4](#)). Furthermore, an incorrect access modifier in the `wipeFrozenAddress` function grants the Pauser role more privileges than intended, beyond just pausing and unpausing the smart contract ([Issue 5](#)).

We also identified areas of improvement in terms of adhering to Solidity best practices ([Issue 7](#), [Issue 8](#), [Issue 9](#), [Issue 11](#)).

### Tests

There are not tests implemented for the smart contract. We recommend implementing comprehensive tests which help identify implementation errors that could lead to security vulnerabilities.

### Project Documentation

There was no external project documentation available for this review, but the code comments provided were generally sufficient to understand the intended behavior of the code. However, one of the comments for a function was inaccurate.



# Section 3.0

## Key Findings Table

Issues	Severity	Status
ISSUE #1 <code>increaseAllowance</code> Function Decreases the Allowance	High	Fixed
ISSUE #2 Event Emits an Incorrect Value	Low	Fixed
ISSUE #3 Duplicate Events Can be Emitted in <code>wipeFrozenAddress</code> Function	Low	Fixed
ISSUE #4 Important Protocol Privileged Addresses Can be Frozen	Low	Not Fixed
ISSUE #5 Incorrect Use of Access Role in <code>wipeFrozenAddress</code> Function	Low	Fixed
ISSUE #6 Address With Zero Balance Can be Frozen	None	Fixed
ISSUE #7 Duplicate Functions	None	Not Fixed
ISSUE #8 No Constructor Implemented in Smart Contracts	None	Fixed
ISSUE #9 Implement Custom Errors to Save Gas	None	Not Fixed
ISSUE #10 Privileged Addresses Can Be Set to Existing Values	None	Fixed
ISSUE #11 Unlock Pragma Version	None	Fixed

Severity definitions can be found in [Appendix A](#)



# Section 4.0

## Findings

We describe the security issues identified during the security audit, along with their potential impact. We also note areas for improvement and optimizations in accordance with best practices. This includes recommendations to mitigate or remediate the issues we identify, in addition to their status before and after the fix verification.

ISSUE#1

### increaseAllowance Function Decreases the Allowance

^ High

✓ Fixed

#### Location

[contracts/core/CofferErc20.sol#L143](#)

#### Description

The `increaseAllowance` function in the `CofferErc20` smart contract mistakenly uses the `-` operator instead of the `+` operator when calculating the new allowance, causing the allowance to decrease instead of increase.

#### Impact

The spender cannot successfully transfer the intended token amount from the owner's account because the necessary allowance has not been properly set.

#### Recommendation

We recommend replacing the `-` operator with the `+` operator in the referenced calculation to ensure the allowance is correctly increased.



## Event Emits an Incorrect Value

✓ Low
✓ Fixed

### Location

[contracts/core/Pausable.sol#L39](#)

[contracts/core/Freezable.sol#L31](#)

[contracts/core/Ownerable.sol#L35](#)

[contracts/core/CofferManageErc20.sol#L73](#)

### Description

The referenced events are triggered whenever a critical access role is updated, such as when a new pauser is accepted. For example, the following function emits the `PauserTransferred` event.

```
function acceptPauser() external {
    require(pendingPauser_ == msg.sender, "cannot accept pauser");
    pauser = pendingPauser_ :
    emit PauserTransferred(pauser, pendingPauser_);
    pendingPauser_ = address(0);
}
```

However, before the event is emitted, the value of `pauser` is set to `pendingPauser_`, causing the event to have two identical values. This issue arises because the function fails to retain the value of the previous pauser when updating to the new one.

### Impact

It creates challenges in effectively tracking critical role updates off-chain.

### Recommendation

We recommend emitting the event before the update of the role (e.g., `pauser` value) or storing the old role (e.g., `pauser` value) value in a separate variable and then emitting the event as written in the two code snippets below for the pauser role. Emitting the event before updating the `pauser` :

```
function acceptPauser() external {
    require(pendingPauser_ == msg.sender, "cannot accept pauser");
    emit PauserTransferred(pauser, pendingPauser_);
    pauser = pendingPauser_ :
    pendingPauser_ = address(0);
}
```

Storing the old `pauser` value in a separate variable and then emitting the event:

```
function acceptPauser() external {
    require(pendingPauser_ == msg.sender, "cannot accept pauser");
    address old owner = pauser;
    pauser = pendingPauser_ :
    emit PauserTransferred(old owner, pauser);
    pendingPauser_ = address(0);
}
```



ISSUE#3

## Duplicate Events Can be Emitted in `wipeFrozenAddress` Function

✓ Low

☑ Fixed

### Location

[contracts/core/CofferManageErc20.sol#L132](#)

### Description

The `wipeFrozenAddress` function reduces the balance of a frozen address to zero and emits three events: `FrozenAccountWiped`, `SupplyDecreased`, and `Transfer`. However, since the function does not verify whether the frozen address has a non-zero balance, it can be called repeatedly, leading to the emission of duplicate events.

```
function wipeFrozenAddress(address addr) public onlyPauser whenNotPaused {
    require(frozen[addr], "address is not frozen");
    uint256 balance = balances[addr];
    balances[addr] = 0;
    totalSupply_ = totalSupply_ - balance;
    emit FrozenAccountWiped(addr);
    emit SupplyDecreased(addr, balance);
    emit Transfer(addr, address(0), balance);
}
```

### Impact

This can lead to external entities misinterpreting the events, as they may receive duplicate information, resulting in unintended behavior or faulty processing.

### Recommendation

We recommend adding the following check at the start of the `wipeFrozenAddress` function.

```
require(balances[addr] > 0, "error message");
```





ISSUE#4

## Important Protocol Privileged Addresses Can be Frozen

✓ Low

✗ Not Fixed

### Location

[contracts/core/Freezable.sol#L45](#)

### Description

The `freeze` function in the `Freezable` smart contract is responsible for freezing an address, but it lacks a check to prevent freezing privileged addresses such as the `supplyController`. If the `supplyController` address is frozen and the `wipeFrozenAddress` function is called before it is unfrozen, the balance of the `supplyController` could be reduced to zero.

```
function freeze(address addr) public onlyFreezer {
    require(!frozen[addr], "address already frozen");
    frozen[addr] = true;
    AccountFrozen(_addr);
}
```

### Impact

Freezing privileged addresses, such as the `supplyController`, prevents them from transferring funds. Additionally, the `wipeFrozenAddress` function can set their balances to zero.

### Recommendation

We recommend implementing a check to prevent the freezing of privileged addresses.

### Verification Status

The Coffey team stated that they intend to address this issue in the future.



ISSUE#5

## Incorrect Use of Access Role in wipeFrozenAddress Function

✓ Low

✓ Fixed

### Location

[contracts/core/CofferManageErc20.sol#L132](#)

### Description

The `wipeFrozenAddress` function uses the `onlyPauser` access modifier, which is intended to allow only pausing and unpausing of the contract. However, in this case, it is also used to reduce the frozen address balance to zero.

```
function wipeFrozenAddress(address addr) public onlyPauser whenNotPaused {
    require(frozen[addr], "address is not frozen");
    uint256 balance = balances[addr];
    balances[addr] = 0;
    totalSupply = totalSupply - balance;
    emit FrozenAccountWiped(addr);
    emit SupplyDecreased(addr, balance);
    emit Transfer(addr, address(0), _balance);
}
```

### Impact

Using the `onlyPauser` modifier in the `wipeFrozenAddress` function expands the Pauser's privileges beyond contract pausing, which could lead to security risks, such as the accidental or malicious wiping of balances. It also violates the principle of least privilege, centralizes control, and can cause confusion in the contract's behavior.

### Recommendation

We suggest restricting this function to either `Freezer` or `Owner`.

ISSUE#6

## Address With Zero Balance Can be Frozen

∨ None

✓ Fixed

### Location

[contracts/core/Freezable.sol#L45](#)

### Description

The `freeze` function in the `Freezable` smart contract can freeze an address with zero balance having no effect on the management of blacklisted addresses balances.

### Impact

None.



## Recommendation

We recommend adding a check to prevent freezing an address with zero balance.

ISSUE#7

## Duplicate Functions

None

Not Fixed

## Location

[contracts/core/CofferManageErc20.sol#L140C1-L150](#)

[contracts/core/Pausable.sol#L29-L33](#)

[contracts/core/Freezable.sol#L21-L25](#)

## Description

Currently, there are two ways in which privileged roles Pauser and Freezer can be updated. The first method allows these roles to assign new roles themselves, while the second method allows the Owner to assign the values of `pendingPauser_`, `pendingFreezer_`. This creates redundancy in the process, leading to duplicate functionality.

## Impact

None.

## Recommendation

We recommend moving the transfer pauser and transfer freezer functionality to the `CofferERC20` smart contract and merging them with the `setPauser` and `setFreezer` functions respectively to prevent duplicate code. Additionally, we recommend using a modifier that verifies the Pauser or Owner role when setting the `pendingPauser_`, and a modifier that verifies Freezer or Owner role when setting the `pendingFreezer_`.

## Verification Status

The Coffer team stated that they intend to address this issue in the future.



ISSUE#8

## No Constructor Implemented in Smart Contracts

### Location

[contracts/core/Freezable.sol#L7](#)

[contracts/core/Ownerable.sol](#)

[contracts/core/Pausable.sol](#)

### Description

Contracts `Pausable`, `Freezable`, and `Ownerable` do not have constructors to initialize the contracts' state variables `pauser`, `freezer`, and `owner`, respectively, rather they are set in the `CofferErc20` smart contract constructor instead. It is considered best practice to initialize a smart contract's state variables in its own constructor. Ideally, state variables should be initialized in the parent contract's constructor, especially if they are essential for the contract's logic. This ensures that the state is set consistently for all contracts that inherit from the parent contract.

**Inheritance Chain Complexity:** If the state variable is only set in the child contract's constructor, it can create an unclear initialization order. **Reliance on Child Constructor:** If the child contract fails to correctly initialize the state variable, the state variable will remain uninitialized or incorrectly set, which can break the contract's logic.

### Impact

None.

### Recommendation

We recommend creating a constructor for each of the aforementioned contracts and setting the state variables in their respective constructor with a default value or via input parameters.

ISSUE#9

## Implement Custom Errors to Save Gas

### Location

Non-exhaustive:

[contracts/core/CofferManageErc20.sol#L143](#)

[contracts/core/CofferManageErc20.sol#L133](#)

[contracts/core/CofferManageErc20.sol#L58](#)

### Description

The above-referenced statements use `require` and `revert` string messages for error handling. However, using a `revert` with a custom error, instead of a string error message, optimizes gas costs.



## Impact

None.

## Recommendation

We recommend defining and using custom errors as described in the [Solidity Documentation](#).

ISSUE#10

## Privileged Addresses Can Be Set to Existing Values

None

Fixed

## Location

[contracts/core/Ownerable.sol#L26](#)

[contracts/core/Pausable.sol#L29](#)

[contracts/core/Freezable.sol#L21](#)

## Description

Functions `transferOwner`, `transferPauser`, and `transferFreezer` can transfer a privilege to the same address performing unnecessary and misleading action.

## Impact

None.

## Recommendation

We recommend preventing setting a privilege to the same address by adding a check in the aforementioned functions.

ISSUE#11

## Unlock Pragma Version

None

Fixed

## Location

[contracts/core](#)

## Description

When using the pragma directive in Solidity, it is essential to specify the exact version of the Solidity compiler that your smart contract is compatible with. This practice, known as locking the pragma, ensures that your contract is compiled and executed as intended, avoiding potential issues caused by compiler version differences.

## Impact

None.



## Recommendation

We recommend specifying the most recent, exact version of the Solidity compiler.



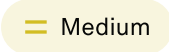
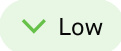



# Section 5.0

## Appendix A

### Severity Rating Definitions

At Defense by Thesis, we utilize the [Immunefi Vulnerability Severity Classification System - v2.3](#).

Severity	Definition
 Critical	<ul style="list-style-type: none"><li>• Manipulation of governance voting result deviating from voted outcome and resulting in a direct change from intended effect of original results</li><li>• Direct theft of any user funds, whether at-rest or in-motion, other than unclaimed yield</li><li>• Direct theft of any user NFTs, whether at-rest or in-motion, other than unclaimed royalties</li><li>• Permanent freezing of funds</li><li>• Permanent freezing of NFTs</li><li>• Unauthorized minting of NFTs</li><li>• Predictable or manipulable RNG that results in abuse of the principal or NFT</li><li>• Unintended alteration of what the NFT represents (e.g. token URI, payload, artistic content)</li><li>• Protocol insolvency</li></ul>
 High	<ul style="list-style-type: none"><li>• Theft of unclaimed yield</li><li>• Theft of unclaimed royalties</li><li>• Permanent freezing of unclaimed yield</li><li>• Permanent freezing of unclaimed royalties</li><li>• Temporary freezing of funds</li><li>• Temporary freezing NFTs</li></ul>
 Medium	<ul style="list-style-type: none"><li>• Smart contract unable to operate due to lack of token funds</li><li>• Enabling/disabling notifications</li><li>• Griefing (e.g. no profit motive for an attacker, but damage to the users or the protocol)</li><li>• Theft of gas</li><li>• Unbounded gas consumption</li></ul>
 Low	<ul style="list-style-type: none"><li>• Contract fails to deliver promised returns, but doesn't lose value</li></ul>
 None	<ul style="list-style-type: none"><li>• We make note of issues of no severity that reflect best practice recommendations or opportunities for optimization, including, but not limited to, gas optimization, the divergence from standard coding practices, code readability issues, the incorrect use of dependencies, insufficient test coverage, or the absence of documentation or code comments.</li></ul>



# Section 6.0

## Appendix B

### Thesis Defense Disclaimer

Defense conducts its security audits and other services provided based on agreed-upon and specific scopes of work (SOWs) with our Customers. The analysis provided in our reports is based solely on the information available and the state of the systems at the time of review. While Thesis Defense strives to provide thorough and accurate analysis, our reports do not constitute a guarantee of the project's security and should not be interpreted as assurances of error-free or risk-free project operations. It is imperative to acknowledge that all technological evaluations are inherently subject to risks and uncertainties due to the emergent nature of cryptographic technologies.

Our reports are not intended to be utilized as financial, investment, legal, tax, or regulatory advice, nor should they be perceived as an endorsement of any particular technology or project. No third party should rely on these reports for the purpose of making investment decisions or consider them as a guarantee of project security.

Links to external websites and references to third-party information within our reports are provided solely for the user's convenience. Thesis Defense does not control, endorse, or assume responsibility for the content or privacy practices of any linked external sites. Users should exercise caution and independently verify any information obtained from third-party sources.

The contents of our reports, including methodologies, data analysis, and conclusions, are the proprietary intellectual property of Thesis Defense and are provided exclusively for the specified use of our Customers. Unauthorized disclosure, reproduction, or distribution of this material is strictly prohibited unless explicitly authorized by Thesis Defense. Thesis Defense does not assume any obligation to update the information contained within our reports post-publication, nor do we owe a duty to any third party by virtue of making these analyses available.

