Membrane Finance

# EUROe Concordium Contract Review

*Version: 2.0*

**December, 2023**

# Contents

**Introduction** **2**
Disclaimer . . . 2
Document Structure . . . 2
Overview . . . 2

**Security Assessment Summary** **3**
Findings Summary . . . 3

**Detailed Findings** **4**

**Summary of Findings** **5**
Missing Contract Schema In Init Function . . . 6
Miscellaneous General Comments . . . 7

**A Vulnerability Severity Classification** **9**

# Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the EUROe smart contract on the Concordium blockchain. The review focused solely on the security aspects of the Rust implementation of the solution, though general recommendations and informational comments are also provided.

## Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review, regarding the underlying business model or the individuals involved in the project.

## Document Structure

The first section provides an overview of the functionality of the EUROe smart contract on the Concordium blockchain within the scope of the security review.

A summary followed by a detailed review of the discovered vulnerabilities is then given which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Findings which do not have direct security implications (but are potentially of interest) are marked as *informational*.

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the EUROe smart contract on the Concordium blockchain.

## Overview

EUROe is an EU regulated, full-reserve euro stablecoin that brings fiat euro liquidity onchain.

The EUROe Concordium smart contract is a fully CIS-2 and CIS-3 compliant token implementing the core functionality, such as minting and burning and additional functionality for pausing, blocklisting and access controls.

# Security Assessment Summary

This review was conducted on the files hosted on the euroe-stablecoin-concordium repository and were assessed at commit f57df39. Retesting activities targeted commit 8c26692.

*Note: external libraries and dependencies were excluded from the scope of this assessment.*

The manual code review section of the report is focused on identifying any and all issues/vulnerabilities associated with the business logic implementation of the solution. This includes their internal interactions, intended functionality and correct implementation in Rust.

Additionally, the manual review process focused on all known Rust anti-patterns and attack vectors. These include, but are not limited to, the following vectors: error handling and wrapping, panicking macros, arithmetic errors, UTF-8 strings handling, index out of bounds and resource exhaustion.

To support this review, the testing team used the following automated testing tools:

- cargo audit: `https://crates.io/crates/cargo-audit`
- cargo deny: `https://github.com/EmbarkStudios/cargo-deny`
- cargo tarpaulin: `https://crates.io/crates/cargo-tarpaulin`
- cargo geiger: `https://github.com/rust-secure-code/cargo-geiger`
- clippy: `https://github.com/rust-lang/rust-clippy`

Output for these automated tools is available upon request.

## Findings Summary

The testing team identified a total of 2 issues during this assessment. Categorised by their severity:

- Informational: 2 issues.

# Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the EUROe smart contract on the Concordium blockchain.

Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labelled as “informational”.

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team.
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

# Summary of Findings

| ID | Description | Severity | Status |
|---|---|---|---|
| MEC-01 | Missing Contract Schema In Init Function | Informational | **Closed** |
| MEC-02 | Miscellaneous General Comments | Informational | **Resolved** |

| MEC-01 | Missing Contract Schema In Init Function |
|---|---|
| Asset | `src/lib.rs` |
| Status | **Closed:** See Recommendations |
| Rating | Informational |

## Description

The init function `contract_init()` is missing the contract schema in the `init` derive macro.

The contract schema is a description of how to represent bytes in a more structured representation and it is used by external tools when displaying the return value of a receive function and for specifying parameters using a structured representation, such as JSON.

On line [**523**] of `src/lib.rs`, the `event` attribute specifying the contract schema is missing.

```
522 /// Initialize contract instance with no token types.
    #[init(contract = "euroe_stablecoin")] // @SigP: Missing event attribute
524 fn contract_init<S: HasStateApi>(
        ctx: &impl HasInitContext,
526     state_builder: &mut StateBuilder<S>,
    ) -> InitResult<State<S>> {
528     // Construct the initial contract state.
        let invoker: Address = Address::Account(ctx.init_origin());
530     Ok(State::empty(state_builder, invoker))
    }
```

## Recommendations

Use the `SchemaType` of the `Event` enum in the `event` attribute of the `init` derive macro to specify the contract schema.

## Recommendations

The issue has been acknowledged and they have provided the following comments.

*No change is implemented as this is a one-time function call.*

| MEC-02 | Miscellaneous General Comments |
|---|---|
| Asset | `src/*` |
| Status | **Resolved:** See Resolution |
| Rating | Informational |

## Description

This section details miscellaneous findings discovered by the testing team that do not have direct security implications:

1. **Migrate contracts for `concordium-std` 8.1.**
   The new version of the standard library reduces the need for generics and traits in the init and receive methods as described in the documentation.
2. **Unused `State.token` field.**
   The `State.token` field is not used and can be removed to reduce contract size.
3. **Unnecessary use of `ContractTokenId`.**
   Use of `ContractTokenId` can be removed from `State.token_balance` and `AddressState.balances` since it does not provide any functionality, reducing contract size and usage costs.
4. **Blocklisting of the smart contract address.**
   Blocklist the smart contract address to prevent any tokens from being transferred by mistake.
5. **Unnecesary `mutable` attribute.**
   The `contract_view_message_hash()` function does not modify the state and does not need the `mutable` attribute.

## Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

## Resolution

The development team have acknowledged these findings, addressing them where appropriate as follows:

1. **Migrate contracts for `concordium-std` 8.1.**
   The issue has been acknowledged and they have provided the following comments.
   *In line with discussions with relevant core Concordium developers we have decided not to migrate the contracts.*
2. **Unused `State.token` field.**
   This issue has been fixed in commit 8c26692.

3. **Unnecessary use of `ContractTokenId`.**
   The issue has been acknowledged and they have provided the following comments.
   *No change is implemented due to added complexity versus reference implementations provided by Concordium*.
4. **Blocklisting of the smart contract address.**
   The issue has been acknowledged and they have provided the following comments.
   *Membrane Finance Oy will blocklist the issuer after deployment*.
5. **Unnecesary `mutable` attribute.**
   This issue has been fixed in commit 8c26692.

# Appendix A Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

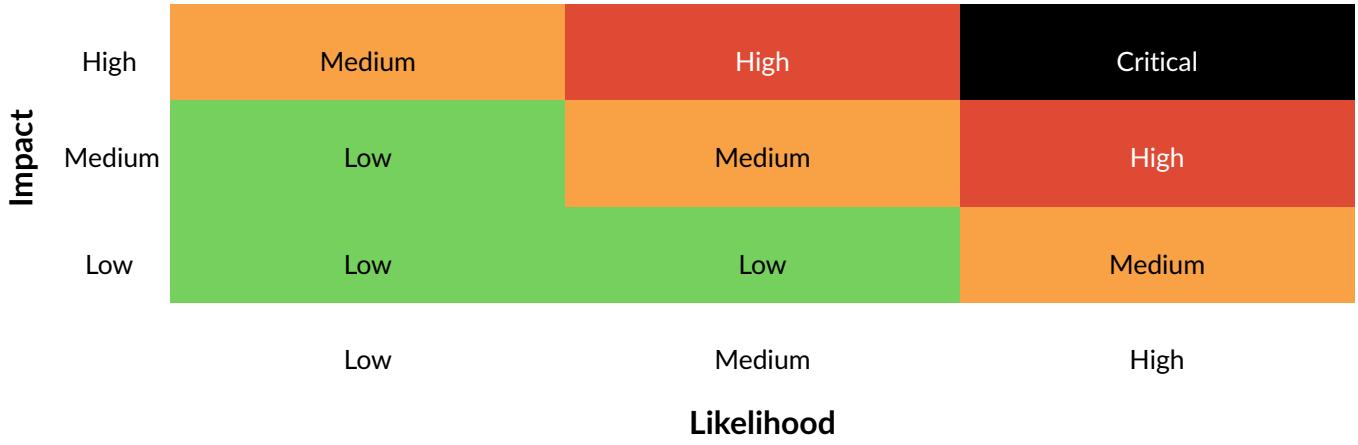


Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.