Security Audit Report

Structured ERC20 Converter, ETH Wrapper, MaxBTC ERC20 for Eureka Bridge

v1.0

November 25, 2025

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security GmbH has been engaged to perform a security audit of Structured ERC20 Converter, ETH Wrapper, MaxBTC ERC20 for Eureka Bridge.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/structured-org/maxbtc-erc20
Label	Paths referencing this target are prefixed below with `maxbtc-erc20:`
Commit	`087473280977859fd578886fccd97fbaba4a1690`
Scope	All contracts were in scope.
Fixes verified at commit	`729c66e2273f0d8ce2b2a7cc3845c580cdf4c298` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Repository	https://github.com/structured-org/jumpgates-fork
Label	Paths referencing this target are prefixed below with `jumpgates-fork:`
Commit	`26bfa8c292a62b3284ab3b317f8805c1003d4a16`
Scope	The scope is restricted to the `contracts/EthWrapper.sol` contract
Fixes verified at commit	`09394fe55f9655ea881f3a9ccd8f3bd7f6c9afb2` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Repository	https://github.com/structured-org/erc20-converter
Label	Paths referencing this target are prefixed below with `erc20-converter:`
Commit	`6e83ca998fbdd2b5984a0c539e2802276c919c5e`
Scope	All contracts were in scope.
Fixes verified at commit	`0306fbca98d006d5ca7d70e3295bd309ea4e451f` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

The scope of the audit consists of three contracts:

- The maxBTC ERC20 contract is an upgradable token implementation where minting and burning are controlled externally by the Eureka bridge to represent maxBTC across Neutron and Cosmos.

- The ETH wrapper contract is a minimal WETH-like component that receives native ETH and issues ERC20 output, enabling automated bridging workflows that require ERC20 inputs without requiring end users to perform manual wrapping.

- The converter contract implements a fixed-rate exchange mechanism between two ERC20 tokens, allowing holders of a temporary or legacy token to redeem it for a replacement token based on a predefined conversion ratio.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending**, **Acknowledged**, **Partially Resolved**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Low-Medium	-
Code readability and clarity	Medium-High	Contracts are well-documented and self-explanatory.
Level of documentation	Medium	-
Test coverage	Medium-High	The test coverage is: • `maxbtc-erc20: 70.59%` • `jumpgates-fork: 100%` • `erc20-converter: 61.90%`

Summary of Findings

No	Description	Severity	Status
1	The `MaxBTCERC20` contract allows for renouncing its ownership	Minor	Resolved
2	Centralization concerns	Minor	Acknowledged
3	Missing minimum output parameter enabling slippage exploitation	Minor	Resolved
4	Missing validation allowing zero exchange rate and identical asset pair configuration	Minor	Resolved
5	Access control flaw in `initializer` causing ownership and initialization order inconsistency	Minor	Resolved
6	No event emission for critical configuration updates	Informational	Resolved
7	`IWETH` interface `transfer` signature mismatches `WETH9` contract	Informational	Resolved
8	Ownership initialization is performed in the child instead of the base contract initializer	Informational	Resolved
9	Superfluous `recoverEther` function because the contract cannot receive ETH	Informational	Acknowledged
10	Missing zero address validations	Informational	Resolved
11	Contracts should implement a two-step ownership transfer	Informational	Resolved
12	Configuration updates allow asset mismatch	Informational	Resolved
13	Miscellaneous comments	Informational	Partially Resolved

Detailed Findings

1. The `MaxBTCERC20` contract allows for renouncing its ownership

Severity: Minor

The `MaxBTCERC20` contract inherits from the `OwnableUpgradeable` contract but does not override the `renounceOwnership` function.

As a result, the owner can call `renounceOwnership`, which sets the owner to `address(0)`. Since upgrade authorization relies on the `onlyOwner` modifier, this action irreversibly bricks the UUPS upgrade path and any owner-gated admin functionality.

Recommendation

We recommend disallowing renouncing the contract ownership by overriding the `renounceOwnership` function and reverting with an error message on invocation.

Status: Resolved

2. Centralization concerns

Severity: Minor

The `ERC20Converter` contract has centralization risks that give the contract owner complete control over user funds:

- The lack of upper bound validation on the exchange rate numerator allows the owner to set extremely high values, enabling drainage of the entire `assetOut` reserve in a single transaction.
- The owner can set `exchangeRateNumerator` to zero, causing permanent denial of service as all conversions will return zero output and revert with `ZeroAmountOut` error.
- The owner can enable MEV sandwich attacks through instant rate changes.

Recommendation

We recommend implementing governance mechanisms or multi-signature controls to mitigate the risks.

Status: Acknowledged

3. Missing minimum output parameter enabling slippage exploitation

Severity: Minor

In `erc20-converter:src/ERC20Converter.sol:106-118`, the `convert` function allows users to exchange `assetIn` for `assetOut` based on the mutable `exchangeRate` configuration.

However, it lacks a `minAmountOut` parameter that would allow users to specify the minimum acceptable output.

This omission introduces a slippage and rate-manipulation risk, a compromised owner could modify the `exchangeRate` immediately before a transaction is mined.

Recommendation

We recommend adding a `minAmountOut` parameter to the `convert` function and enforcing a condition that reverts the transaction if the computed `amountOut` falls below this threshold.

Status: Resolved

4. Missing validation allowing zero exchange rate and identical asset pair configuration

Severity: Minor

In `erc20-converter:src/ERC20Converter.sol:70-87`, the `_updateConfig` function updates the converter's configuration without validating critical parameters.

Specifically, it does not verify that `exchangeRate_` is greater than zero, nor that `assetIn_` and `assetOut_` are distinct tokens.

As a result, the contract could be configured with an invalid or nonsensical conversion rate, or with identical assets, effectively breaking conversion logic. This could lead to incorrect token exchange calculations and failed transfers.

Recommendation

We recommend enforcing strict validation checks in `_updateConfig`.

Status: Resolved

5. Access control flaw in `initializer` causing ownership and initialization order inconsistency

Severity: Minor

In `erc20-converter:src/ERC20Converter.sol:70-87`, the `initialize` function calls `updateConfig`, which is restricted by the `onlyOwner` modifier.

During proxy initialization, the `msg.sender` is typically the deployer, not the designated `owner_`.

As a result, `updateConfig` will revert unless `owner_` is set to the deployer address at initialization time.

This creates a flawed initialization sequence and may lead teams to temporarily assign ownership to the deployer to complete initialization.

Doing so introduces a security risk, as the deployer retains control over the contract until ownership is manually transferred, creating a potential attack window or operational misconfiguration.

Recommendation

We recommend refactoring the initialization logic to ensure that configuration updates occur before ownership transfer, or alternatively, to perform the initial setup through the internal function that bypasses the `onlyOwner` restriction during deployment.

Status: Resolved

6. No event emission for critical configuration updates

Severity: Informational

In `erc20-converter:src/ERC20Converter.sol`, the `_updateConfig` function modifies critical contract parameters (tokens and exchange rate) without emitting an event.

This makes it difficult for users and monitoring systems to track when and how the contract configuration changes, reducing transparency and auditability.

Recommendation

We recommend adding an event declaration and emitting it after successful configuration.

Status: Resolved

7. `IWETH` interface `transfer` signature mismatches `WETH9` contract

Severity: Informational

The `IWETH` interface in `jumpgates-fork:contracts/EthWrapper.sol:4-7` declares `transfer(address,uint)` with no return value, while the canonical `WETH9` contract on mainnet implements the ERC-20 signature `transfer(address,uint256) returns (bool)`.

Solidity will ignore the extra return value, so the call does not revert due to the mismatch. However, this hides the success indicator (which is always `true` for WETH) and diverges from the actual ABI.

Recommendation

We recommend updating the `IWETH.transfer` to match the actual `WETH9` signature.

Status: Resolved

8. Ownership initialization is performed in the child instead of the base contract initializer

Severity: Informational

The `ERC20Converter` contract calls `__Ownable_init(owner_)` inside its initializer function in `erc20-converter:src/ERC20Converter.sol:36`, although `AssetRecoverer` is the base contract that directly inherits from the `OwnableUpgradeable` contract.

This splits responsibility for ownership initialization across inheritance layers and may lead to mis-initialization if other inheriting contracts reuse `AssetRecoverer` without replicating the child-level owner initialization.

Recommendation

We recommend adding an initializer function named `__AssetRecoverer_init(address owner_)` to the `AssetRecoverer` contract that calls `__Ownable_init(owner_)`.

Furthermore, the `ERC20Converter.initialize` function (and any other inheritors) should call `__AssetRecoverer_init(owner_)` instead of the direct `__Ownable_init(owner_)` call.

Status: Resolved

9. Superfluous `recoverEther` function because the contract cannot receive ETH

Severity: Informational

The `AssetRecoverer` base contract exposes a `recoverEther` function, defined in `erc20-converter:src/AssetRecoverer.sol:50-62`, to sweep ETH balances, but neither the `AssetRecoverer` nor its child `ERC20Converter` contract can receive ETH because they define no `receive/fallback` or other `payable` functions.

As a result, these contracts cannot accumulate ETH.

Recommendation

We recommend clarifying the intended use case and either allowing receiving ETH or removing the `recoverEther` function.

Status: Acknowledged

10. Missing zero address validations

Severity: Informational

Multiple contracts lack validation for zero addresses during initialization:

- In the `ERC20Converter` contract, the `_updateConfig` function does not validate that `assetIn_` and `assetOut_` parameters are non-zero addresses. The function immediately attempts to call `decimals` on these addresses without checking if they are valid. If either parameter is set to `address(0)`, the contract will revert when trying to read decimals, making the converter unusable. This affects both the initialize and `updateConfig` functions.
- In the `MaxBTCERC20` contract, the `initialize` function directly stores the `ics20_` address without validation. The contract would fail authorization checks since the zero address cannot be a valid caller.
- In the `EthWrapper` contract, the `constructor` directly stores the `jumpgate_` address without validation. The contract would fail authorization checks since the zero address cannot be a valid caller.

Recommendation

We recommend adding zero address validation in the abovementioned instances.

Status: Resolved

11. Contracts should implement a two-step ownership transfer

Severity: Informational

The contracts within the scope of this audit allow the current owner to execute a one-step ownership transfer. While this is common practice, it presents a risk for the ownership of the contract to become lost if the owner transfers ownership to an incorrect address. A two-step ownership transfer will allow the current owner to propose a new owner, and then the account that is proposed as the new owner may call a function that will allow them to claim ownership and actually execute the config update.

Recommendation

We recommend implementing a two-step ownership transfer. The flow can be as follows:

1. The current owner proposes a new owner address that is validated and lowercased.
2. The new owner account claims ownership, which applies the configuration changes.

Status: Resolved

12. Configuration updates allow asset mismatch

Severity: Informational

In `erc20-converter:src/ERC20Converter.sol:62-67`, the `updateConfig` function allows the contract owner to modify the `assetIn` and `assetOut` parameters even after initialization.

This introduces a potential risk of asset mismatch or manipulation, as changing these critical configuration parameters post-deployment could lead to unexpected token conversions.

A compromised or negligent owner could reconfigure the converter to map incompatible or malicious ERC-20 tokens, leading to fund misdirection or denial of service during conversions.

Recommendation

We recommend enforcing immutability on the `assetIn` and `assetOut` parameters once the contract has been initialized.

Status: Resolved

13. Miscellaneous comments

Severity: Informational

The codebase contains several maintenance and clarity issues:

- Misleading parameter name in `burn(address mintAddress, uint256 amount)` function in `maxbtc-erc20:src/MaxBTCERC20.sol:69`, `mintAddress` implies minting, not burning.
- Hardcoded WETH address in `jumpgates-fork:contracts/EthWrapper.sol:10` is only valid on Ethereum mainnet, causing misconfiguration on other networks.
- The contract unnecessarily enforces `decimalsIn == decimalsOut` in `erc20-converter:src/ERC20Converter.sol:77`, even though `exchangeRateNumerator/exchangeRateDenominator` can encode cross-decimal scaling.

Recommendation

We recommend:

- Rename `mintAddress` to `burnAddress` and align the interface to avoid confusion and potential misuse.
- Ensure that the `EthWrapper` contract is only ever used on the Ethereum mainnet.
- Remove the equality check, keep `exchangeRateDenominator = 10 ** decimalsIn`, and require or document that `exchangeRateNumerator` accounts for conversion into `decimalsOut`.

Status: Partially Resolved