Crypto Unicorns Token

Security Audit Report

September 6, 2024

Contents

1 Introduction

1.1 About Crypto Unicorns Token

Crypto Unicorns is a new blockchain-based game centered around awesomely unique Unicorn NFTs which players can use in a fun farming simulation and in a variety of exciting battle loops. Crypto Unicorns Token ($CU) is the Unicorn multiverse's primary value token, which is an ERC-20 successor to the Rainbow Token ($RBW) on Polygon. $CU Tokens may only be minted via the LayerZero cross-chain bridge, which requires a permanent stake of $RBW tokens on Polygon, at a ratio of 10 to 1. $CU Tokens may be bridged from Arbitrum to the wrapped $CU tokens ($wCU) on the Xai gaming blockchain at a ratio of 1 to 1. This exchange is performed bidirectional by the native Arbitrum crosschain bridge.

1.2 Project Summary

The project basic information is listed in the table below.

Title	Description
Client	Crypto Unicorns
Website	https://www.cryptounicorns.fun
Project Name	Crypto Unicorns Token
Project Type	ERC20
Timeline	August 8, 2024 - September 6, 2024
Number of Auditors	2
Audit Platform	Hyacinth

1.3 Audit Scope

In the following, we show the repository, commit id and files that were reviewed in the audit:

- https://github.com/Laguna-Games/cu-public
- CommitID: efad2f9
- Files:
  - all the contracts under src/
  - all the contracts under lib/@lg-arb-bridge/
  - lib/@cu-tokens/src/libraries/LibERC20.sol
  - lib/@cu-tokens/src/facets/ERC20Facet.sol
  - lib/@cu-tokens/src/facets/ERC20InitializerFacet.sol
  - lib/@lagunagames/lg-diamond-template/src/libraries/LibContractOwner.sol
  - everything under lib/@lg-layerzero/src/
  - All the files used in all above files

2 | Overall Assessment

This report has been compiled to identify issues and vulnerabilities within Crypto Unicorns Token. Overall, no ERC20 compliance issues were found, and the detailed checklist can be found in Section 3. Additionally, no critical or high-severity issues were observed throughout this audit, including those identified using auxiliary tool techniques to supplement our thorough manual code review. Though the smart contracts are well-designed and engineered, the implementation can be further improved by resolving the 4 issues spanning various severity levels.

Severity	Count	Acknowledged	Won't Do	Addressed
Critical	-	-	-	-
High	-	-	-	-
Medium	1	1	-	-
Low	2	2	-	-
Informational	1	1	-	-
Total	4	4	-	-

3 | ERC20 Compliance Checks

ERC20 is a standard for fungible tokens, which defines a common set of rules that tokens must follow to ensure compatibility and interoperability with various platforms, wallets, and exchanges. In this section, we examine the implementation of the Crypto Unicorns Token from two perspectives: ERC20 Token Methods and ERC20 Events, and validate whether there are any inconsistencies or incompatibilities with the ERC20 specification.

3.1 ERC20 Token Methods

Method	Specification	Status
name	Returns the name of the token as a string, for example "MyToken"	☑
symbol	Returns the symbol of the token. E.g. "HIX"	☑
decimals	Returns the number of decimals the token uses - e.g. 8	☑
totalSupply	Returns the total token supply	☑
balanceOf	Returns the account balance of any address	☑
transfer	Returns a boolean value reflecting the token transfer status	☑
	Throws if the caller does not have enough balance	☑
	Transfers of 0 values must be treated as normal transfers	☑
	Fires the Transfer event	☑
transferFrom	Returns a boolean value reflecting the token transfer status	☑
	Throws if the spender does not have enough token allowances	☑
	Throws if the from address does not have enough balance	☑
	Transfers of 0 values must be treated as normal transfers	☑
	Fires the Transfer event	☑
approve	Returns a boolean value reflecting the token approval status	☑
	Fires the Approval event	☑
allowance	Returns the amount which the spender is still allowed to withdraw from the owner	☑

3.2 ERC20 Events

Event	Specification	Status
Transfer	MUST trigger when tokens are transferred, including zero value transfers	☑
	SHOULD trigger with the from address set to $0x0$ when tokens are created	☑
Approval	MUST trigger on any successful call to approve()	☑

In the two tables above, we have outlined the checklists of ERC20 methods and ERC20 events according to the ERC20 token standard, along with the examination result. Our analysis shows that the Crypto Unicorns Token has implemented all the required methods and events in accordance with the ERC20 specification. There is no ERC20 inconsistency or incompatibility issue found in the examination.

4 | Vulnerability Summary

4.1 Overview

In this section, we have listed all the identified issues throughout this audit. You can click on each item to jump directly to the detailed explanation page, or scroll down to view all the details.

4.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

Severity	Description
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and remediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

4.3 Vulnerability Details

4.3.1 [L-1] Suggested Prevention of Re-Initialization

Target	Category	IMPACT	LIKELIHOOD	STATUS
CUBridgeOFTFacet.sol	Coding Practices	Low	Low	Acknowledged

The `initLZOFTV2()` function in the `CUBridgeOFTFacet` contract is responsible for initializing the `sharedDecimals` and `lzEndpoint` parameters for $CU, as required by the Omnichain Fungible Token (OFT) Standard. During our review, we identified a potential re-initialization vulnerability within this function. The issue arises due to the absence of proper validation checks, which could allow the function to be called multiple times, thereby risking unintended re-initialization. Re-initialization can lead to unintended behavior, such as the resetting or modification of critical state variables, potentially compromising the integrity of the contract.

To elaborate, we show below the code snippet of the `initLZOFTV2()` function. If this function is re-triggered, the `sharedDecimals` and `lzEndpoint` parameters may be altered. Changing `sharedDecimals` will affect the decimal conversion rate (`ld2sdRate`), which could impact the amount of tokens to be minted on the destination chain. Altering `lzEndpoint` could lead to incorrect endpoint configuration, potentially disrupting current contract operations.

To prevent re-initialization, it is recommended to add a check to ensure the `initLZOFTV2()` function can only be executed once. This can be achieved by verifying that the contract is in an uninitialized state before allowing the initialization logic to proceed. Specifically, the following require statement should be added: `require(lzEndpoint == address(0)&& _lzEndpoint != address(0))`. This ensures that the `lzEndpoint` is only set when it has not been previously initialized, preventing subsequent calls from altering its value.

CUBridgeOFTFacet::initLZOFTV2()

```
11  function initLZOFTV2(
12      uint8 _sharedDecimals,
13      address _lzEndpoint
14  ) external virtual override {
15      LibContractOwner.enforceIsContractOwner();

17      uint8 _decimals = 18;
18      if (_sharedDecimals > _decimals) {
19          revert LibLZBridge.LZBridgeInvalidSharedDecimals(_sharedDecimals,
                _decimals);
20      }

22      ld2sdRate = 10**(_decimals - _sharedDecimals);
```

```
24      // From OFTCoreV2.sol
25      sharedDecimals = _sharedDecimals;

27      // From LzApp.sol
28      lzEndpoint = ILayerZeroEndpoint(_lzEndpoint);

30      // Since the LZ contracts use Ownable, set the owner
31      _transferOwnership(LibContractOwner.contractOwner());
32  }
```

Remediation Revisit the implementation of the `initLZOFTV2()` function and incorporate proper validation checks to mitigate the risk of re-initialization.

Response By Team This is a good point but low priority. We're good if we don't work on it.

4.3.2 [L-2] Revisited Validation of Function Arguments

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Coding Practices	Low	Low	Acknowledged

In the `CUArbBridgeSenderFacet` contract, the `setL1RouterAddress()` and `setL1CustomGatewayAddress()` functions are designed to set critical contract parameters, namely the `_l1RouterAddress` and the `_l1CustomGatewayAddress`. However, these functions currently lack proper validation checks on their inputs, making them vulnerable to the possibility of incorrect data being passed to the contract.

For example, the `setL1RouterAddress()` function allows the setting of the `_l1RouterAddress` parameter. Without proper validation, there is a risk that this address could be set to an invalid value, such as the zero address (address(0)), which could disrupt the contract's functionality.

Although these functions can only be triggered by the protocol owner, which significantly reduces the likelihood of the issue occurring, it is still recommended to add proper validation checks for their inputs. Specifically, it should be ensured that the addresses provided as inputs are valid and not set to the zero address. The following require statements should be added to the respective functions:

```
require(_l1RouterAddress != address(0)); require(_customGatewayAddress != address(0));
```

It is important to note that similar issues exist in other functions within the protocol, such as `CUArbBridgeReceiverFacet::setL2Gateway()`, `CUArbBridgeReceiverFacet::setL1TokenAddress()` and others. Implementing these validation checks across all relevant functions will help safeguard the contract's operations and enhance its overall security.

CUArbBridgeSenderFacet.sol

```
14  function setL1RouterAddress(address _l1RouterAddress) external override {
15      LibContractOwner.enforceIsContractOwner();

17      LibArbBridge.setL1RouterAddress(_l1RouterAddress);
18  }

20  function setL1CustomGatewayAddress(
21      address _customGatewayAddress
22  ) external override {
23      LibContractOwner.enforceIsContractOwner();

25      LibArbBridge.setL1CustomGatewayAddress(_customGatewayAddress);
26  }
```

Remediation Revisit the implementation of the above-mentioned functions to add proper validation checks for their inputs.

Response By Team This is a good point but low priority. We're good if we don't work on it.

4.3.3 [I-1] Meaningful Events for Key Operations

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Coding Practices	Low	N/A	Acknowledged

The `event` feature is vital for capturing runtime dynamics in a contract. Upon emission, `events` store transaction arguments in logs, supplying external analytics and reporting tools with crucial information. They play a pivotal role in scenarios like modifying system-wide parameters or handling token operations.

However, in our examination of protocol dynamics, we observed that certain privileged routines lack meaningful events to document their changes. We highlight the representative routines below.

CUArbBridgeReceiverFacet.sol

```
899    function setL2Gateway(address _l2Gateway) external override {
900      LibContractOwner.enforceIsContractOwner();

902      LibArbBridge.setL2Gateway(_l2Gateway);
903  }

905  function setL1TokenAddress(address _l1TokenAddress) external override {
906      LibContractOwner.enforceIsContractOwner();
```

```
908         LibArbBridge.setL1TokenAddress(_l1TokenAddress);
909     }
```

Remediation Ensure the proper emission of meaningful events containing accurate information to promptly reflect state changes.

Response By Team This is a good point but low priority. We're good if we don't work on it.

4.3.4 [M-1] Potential Risks Associated with Centralization

Target	Category	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	Medium	Low	Acknowledged

In the Crypto Unicorns protocol, the presence of a privileged `owner` accountintroduces risks of centralization, as it holds significant control and authority over critical operations governing the protocol. Our analysis of the onchain data reveals that the current `owner` is set to 0x4CBBE...Bb1f1, which is an externally owned account (EOA). In the following, we highlight the representative functions that are potentially affected by the privileges associated with this privileged account.

Examples of Privileged Operations

```
315 function diamondCut(FacetCut[] calldata _diamondCut, address _init, bytes
        calldata _calldata) external override {
316     (_init); // noop
317     (_calldata); // noop
318     LibDiamond.enforceIsContractOwner();
319     LibDiamond.diamondCut(_diamondCut);
320 }

322 function initLZOFTV2(
323   uint8 _sharedDecimals,
324   address _lzEndpoint
325 ) external virtual override {
326   LibContractOwner.enforceIsContractOwner();

328   uint8 _decimals = 18;
329   if (_sharedDecimals > _decimals) {
330       revert LibLZBridge.LZBridgeInvalidSharedDecimals(_sharedDecimals,
            _decimals);
331   }

333   ld2sdRate = 10**(_decimals - _sharedDecimals);

335   // From OFTCoreV2.sol
```

```
336     sharedDecimals = _sharedDecimals;
337
338     // From LzApp.sol
339     lzEndpoint = ILayerZeroEndpoint(_lzEndpoint);
340
341     // Since the LZ contracts use Ownable, set the owner
342     _transferOwnership(LibContractOwner.contractOwner());
343   }
344
345   function setL1RouterAddress(address _l1RouterAddress) external override {
346     LibContractOwner.enforceIsContractOwner();
347
348     LibArbBridge.setL1RouterAddress(_l1RouterAddress);
349   }
350
351   function setL1CustomGatewayAddress(
352     address _customGatewayAddress
353   ) external override {
354     LibContractOwner.enforceIsContractOwner();
355
356     LibArbBridge.setL1CustomGatewayAddress(_customGatewayAddress);
357   }
358
359   function registerTokenOnL2(
360     address l2CustomTokenAddress,
361     uint256 maxSubmissionCostForCustomBridge,
362     uint256 maxSubmissionCostForRouter,
363     uint256 maxGasForCustomBridge,
364     uint256 maxGasForRouter,
365     uint256 gasPriceBid,
366     uint256 valueForGateway,
367     uint256 valueForRouter,
368     address creditBackAddress
369   ) external override {
370     LibContractOwner.enforceIsContractOwner();
371     // we temporarily set `shouldRegisterGateway` to true for the callback in
          registerTokenToL2 to succeed
372     bool prev = LibArbBridge.getShouldRegisterGateway();
373     LibArbBridge.setShouldRegisterGateway(true);
374
375     IL1CustomGateway(LibArbBridge.getL1CustomGatewayAddress())
376         .registerTokenToL2(
377             l2CustomTokenAddress,
378             maxGasForCustomBridge,
379             gasPriceBid,
380             maxSubmissionCostForCustomBridge,
381             creditBackAddress,
382             valueForGateway
383         );
```

```
385     IL1GatewayRouter(LibArbBridge.getL1RouterAddress()).setGateway(
386         LibArbBridge.getL1CustomGatewayAddress(),
387         maxGasForRouter,
388         gasPriceBid,
389         maxSubmissionCostForRouter,
390         creditBackAddress,
391         valueForRouter
392     );
394     LibArbBridge.setShouldRegisterGateway(prev);
395 }
```

Remediation To mitigate the identified issue, it is recommended to introduce multi-sig mechanism to undertake the roles of the privileged accounts. Moreover, it is advisable to implement timelocks to govern all modifications to the privileged operations.

Response By Team We'll drop the ownership of the contract soon, once we detected that we don't have any medium/high/critical issues.

5 | Conclusion

Crypto Unicorns is a new blockchain-based game centered around awesomely unique Unicorn NFTs which players can use in a fun farming simulation and in a variety of exciting battle loops. Crypto Unicorns Token ($CU) is the Unicorn multiverse's primary value token, which is an ERC-20 successor to the Rainbow Token ($RBW) on Polygon. $CU Tokens may only be minted via the LayerZero cross-chain bridge, which requires a permanent stake of $RBW tokens on Polygon, at a ratio of 10 to 1. $CU Tokens may be bridged from Arbitrum to the wrapped $CU tokens ($wCU) on the Xai gaming blockchain at a ratio of 1 to 1. This exchange is performed bidirectional by the native Arbitrum crosschain bridge.

The current codebase is well-structured and neatly organized. No ERC20 compliance issues were found, and no critical or high-severity issues were observed throughout this audit. Those identified issues have been promptly confirmed and fixed.

6 Appendix

6.1 About AstraSec

AstraSec is a blockchain security company that serves to provide high-quality auditing services for blockchain-based protocols. With a team of blockchain specialists, AstraSec maintains a strong commitment to excellence and client satisfaction. The audit team members have extensive audit experience for various famous DeFi projects. AstraSec's comprehensive approach and deep blockchain understanding make it a trusted partner for the clients.

6.2 Disclaimer

The information provided in this audit report is for reference only and does not constitute any legal, financial, or investment advice. Any views, suggestions, or conclusions in the audit report are based on the limited information and conditions obtained during the audit process and may be subject to unknown risks and uncertainties. While we make every effort to ensure the accuracy and completeness of the audit report, we are not responsible for any errors or omissions in the report.

We recommend users to carefully consider the information in the audit report based on their own independent judgment and professional advice before making any decisions. We are not responsible for the consequences of the use of the audit report, including but not limited to any losses or damages resulting from reliance on the audit report.

This audit report is for reference only and should not be considered a substitute for legal documents or contracts.

6.3 Contact

Phone	+86 156 0639 2692
Email	contact@astrasec.ai
Twitter	https://twitter.com/AstraSecAI