EtaBridge Smart Contracts
Security Review

Cantina Managed review by:

WindHustler, Security Researcher
Ladboy233, Security Researcher

June 19, 2025

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**

3 Findings **4**
- 3.1 High Risk . . . 4
  - 3.1.1 Lack of liquidity management in cross-chain bridge leads to indefinite lockup of user funds . . . 4
- 3.2 Medium Risk . . . 4
  - 3.2.1 ERC20 tokens missing return values are not supported . . . 4
  - 3.2.2 Funds can be lost if the ERC20 token does not have the same decimal in source chain and destination chain . . . 5
- 3.3 Low Risk . . . 6
  - 3.3.1 Admin privileges can be abused . . . 6
  - 3.3.2 Fee-on-transfer tokens are not supported . . . 6
- 3.4 Gas Optimization . . . 7
  - 3.4.1 Consider assign a variable to `IERC20(supportedTokens[symbol]` to save gas . . . 7
- 3.5 Informational . . . 7
  - 3.5.1 Emit events in function `updateFee` . . . 7
  - 3.5.2 Redundant Ownable Inheritance . . . 7

1 Introduction

1.1 About Cantina

Cantina is a security services marketplace that connects top security researchers and solutions with clients. Learn more at cantina.xyz

1.2 Disclaimer

Cantina Managed provides a detailed evaluation of the security posture of the code at a particular moment based on the information available at the time of the review. While Cantina Managed endeavors to identify and disclose all potential security issues, it cannot guarantee that every vulnerability will be detected or that the code will be entirely secure against all possible attacks. The assessment is conducted based on the specific commit and version of the code provided. Any subsequent modifications to the code may introduce new vulnerabilities that were absent during the initial review. Therefore, any changes made to the code require a new security review to ensure that the code remains secure. Please be advised that the Cantina Managed security review is not a replacement for continuous security measures such as penetration testing, vulnerability scanning, and regular code reviews.

1.3 Risk assessment

Severity	Description
Critical	*Must* fix as soon as possible (if already deployed).
High	Leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
Medium	Global losses <10% or losses to only a subset of users, but still unacceptable.
Low	Losses will be annoying but bearable. Applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.
Gas Optimization	Suggestions around gas saving practices.
Informational	Suggestions around best practices or readability.

1.3.1 Severity Classification

The severity of security issues found during the security review is categorized based on the above table. Critical findings have a high likelihood of being exploited and must be addressed immediately. High findings are almost certain to occur, easy to perform, or not easy but highly incentivized thus must be fixed as soon as possible.

Medium findings are conditionally possible or incentivized but are still relatively likely to occur and should be addressed. Low findings a rare combination of circumstances to exploit, or offer little to no incentive to exploit but are recommended to be addressed.

Lastly, some findings might represent objective improvements that should be addressed but do not impact the project's overall security (Gas and Informational findings).

2 Security Review Summary

EtaBridge is trustless cross-chain liquidity bridge, designed to simplify and streamline token transfers across multiple chains.

From Jun 2nd to Jun 3rd the Cantina team conducted a review of etabridge-smart-contracts on commit hash ad37920b. The team identified a total of **8** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	1	0	1
Medium Risk	2	2	0
Low Risk	2	1	1
Gas Optimizations	1	1	0
Informational	2	2	0
Total	**8**	**6**	**2**

3 Findings

3.1 High Risk

3.1.1 Lack of liquidity management in cross-chain bridge leads to indefinite lockup of user funds

Severity: High Risk

Context: EtaBridge.sol#L74-L86

Description: The `EtaBridge` contract implements a cross-chain token bridge without proper liquidity management mechanisms. This design flaw allows for scenarios where users' tokens can be indefinitely locked due to insufficient liquidity on the destination chain.

Proof of Concept:

1. Bridge is deployed on Base, Arbitrum, and Ethereum with 500 USDC liquidity on each chain.
2. User A initiates a bridge of 100 USDC from Base to Ethereum.
3. Before User A's transaction is processed, an attacker frontruns by bridging 500 USDC from Arbitrum to Ethereum.
4. When User A's transaction arrives on Ethereum, there is insufficient liquidity to fulfill the transfer.
5. User A's tokens remain locked until someone deposits additional USDC on Ethereum.
6. Even if the deposit occurs, the attacker can shift that liquidity.

Recommendation: Fixing the issue requires tracking inflows and outflows of liquidity from/to each chain and periodically rebalancing the liquidity. Major architectural changes are necessary to accommodate these changes.

EtaBridge: If a message cannot be processed due to insufficient liquidity on the destination chain, the smart contract will revert with an error. In this case, the LayerZero message will not be considered delivered and can be manually retried later, once sufficient liquidity is available on the destination chain (see LayerZero docs on Retry Message).

Our project encourages users to check the available liquidity on the destination chain to avoid delivery delays. Maintaining a high level of liquidity on the bridge minimizes the likelihood of such issues.

In addition, we are actively developing an automatic rebalancing mechanism - one of the key milestones of our project - which will help maintain healthy liquidity levels across all endpoints.

Cantina Managed: Acknowledged.

3.2 Medium Risk

3.2.1 ERC20 tokens missing return values are not supported

Severity: Medium Risk

Context: EtaBridge.sol#L63

Description: The `EtaBridge` contract assumes all ERC20 tokens follow the standard interface that returns a boolean from `transfer()` and `transferFrom()` functions. However, some tokens like USDT on Ethereum mainnet implement `transfer` and `transferFrom` functions without returning a boolean value. When the contract tries to check the return value with `require()`, the transaction will revert.

Proof of Concept:

1. Deploy `EtaBridge` contract with USDT as a supported token.
2. User attempts to bridge USDT tokens:

```
// In EtaBridge::bridgeTokens
require(IERC20(supportedTokens[symbol]).transferFrom(msg.sender, address(this), amount), "Token transfer
↪  failed");
```

3. The transaction reverts because USDT's `transferFrom()` doesn't return a boolean.

The worst case scenario is if the token has a standard interface on the sending chain and non-standard interface on the receiving chain since it would lead to token loss.

Recommendation: Use OpenZeppelin's `SafeERC20` library:

```
+ import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";

  contract EtaBridge is Ownable, OApp, ReentrancyGuard {
+     using SafeERC20 for IERC20;
```

Replace all `transfer`/`transferFrom` call with `safeTransfer` and `safeTransferFrom` calls.

EtaBridge: Fixed in commit 372380a7.

Cantina Managed: Fix verified.

3.2.2 Funds can be lost if the ERC20 token does not have the same decimal in source chain and destination chain

Severity: Medium Risk

Context: EtaBridge.sol#L54-L72

Description: The assumption is the user send token by locking X amount of tokekn in source chain and the receiver receives X amount of token in destination chian. However, fund can be lost if the ERC20 token does not have the same decimal in source chain and destination chain.

For example:

- USDT on mainnet has 6 decimals.
- USDT on BSC has 18 decimals.

Then user can bridge 100 USDT (`100 * 10^6`) from Ethereum and then receive a dust amount of USDT in Binance smart chain.

Recommendation: Validating the token in source chain and destination chain has the same decimal offchain, or if the protocol intends to support token with different decimals across chains, consider make some changes, the payload can encode the normalized amount in `_lzSend`:

```
uint256 scale = 1e18;
uint256 normalizedAmount = (amount * scale) / 10**token.decimals();
```

and then in `_lzReceive`, normalize the `amount` using local token decimals.

EtaBridge: Fixed in commit b45e08bc.

Cantina Managed: Due to the fee, some dust is lost if you transfer from a chain with 18 decimals to a chain with 6 decimals. This happens since you're recording amountAfterFee after deducting the fee, and this can still contain dust amounts that can't be represented with 6 decimals. You can check this reference from LayerZero that describes the process for OFTs.

EtaBridge: When user transfers `3,333...` coins from chain with 18 decimals to chain with 6 decimals token:

```
amount = 3_333_333_333_333_333_333
transferredAmount = 3_333_333_333_333_333_333
normalizedAmount = 3_333_333_333_333_333_333
fee (0.3%) = 9_999_999_999_999_999
amountAfterFee = 3_323_333_333_333_333_334
```

Will be delivered on chain with 6 decimals: `3_323_333` The dust (`333_333_333_334`) can't be represented in 6 decimals token, so in `lzReceive` user will receive `3_323_333` tokens. Even if we imagine fee is 0%, we transferring `3_333_333_333_333_333_333`, and user receive `3_333_333`, part, which can't be represented - is dropped.

If your take that we should provide to `_lzSend` `3_323_333_000_000_000_000` - then it requires high complication of logic (to store each destination chain token decimals on the source chain) and don't allow user to transfer not round enough numbers. Which doesn't make sense as for me.

Cantina Managed: Your example is accurate, in the LayerZero's OFT implementation, you could only send `3_323_333_000_000_000_000` amount, i.e. the last 12 digits should be equal to 0 since the 6 decimals can't represent them. You're also right for the example where fee is 0% and the user is transferring `3_333_333_333_333_333_333`.

This does require some changes to how the fee is taken, so the final amount that gets transferred doesn't contain any dust in case the tokens have different decimals on different chains.

EtaBridge: I think we are good to move with current implementation.

Cantina Managed: Fixed.

3.3 Low Risk

3.3.1 Admin privileges can be abused

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description:

1. The admin of the `EtaBridge` contract can not only withdraw the fee, but also withdraw all ERC20 tokens that are held in the contract, potentially stealing user funds.
2. The admin can freely change the `supportedTokens[_symbol]` to any arbitrary token, i.e. user bridges USDC from Base $\rightarrow$ Arbitrum but receives some dummy token on Arbitrum.
3. The `EtaBridge::removeSupportedToken` function allows the owner to remove support for a token by deleting its mapping entry. However, this can lead to permanent fund loss if there are in-flight cross-chain messages for the removed token. When a user bridges tokens from Chain A to Chain B, the message is sent through LayerZero. If the owner removes support for the token on Chain B before the message arrives, the `_lzReceive` function will revert due to the `require(supportedTokens[symbol] != address(0))` check. This results in the user's tokens being permanently locked in the bridge contract on Chain B.

Recommendation: Make sure to exercise admin privileges with caution and notify users in advance of any critical changes to the system, i.e,. removing a support for a specific token.

EtaBridge: Admin of the contract have extended privileges by design of the application. The reasons behind this decision are:

1. Admin should have possibility to recover stuck funds.
2. Admin should have possibility to introduce/remove chains and tokens.
3. Admin should have possibility to inject/remove liquidity from particular pool/chain.

Since source of liquidity is internal - project risking it's own funds in this case. Also, contract ownership will be changed to multisig upon reaching specific amount of liquidity, which will significantly reduce risk of privilege abuse.

Cantina Managed: Acknowledged.

3.3.2 Fee-on-transfer tokens are not supported

Severity: Low Risk

Context: EtaBridge.sol#L54-L72

Description: The assumption is the user send token by locking X amount of tokekn in source chain and the receiver receives X amount of token in destination chain.

```
IERC20(supportedTokens[symbol]).transferFrom(msg.sender, address(this), amount)
```

If the ERC20 token charge fees, the smart contract received amount is less than the `amount`. Then if the token charges 1% transfer fee, the user locks 99% token `amount` in source chain but may receive 100% `amount` in destination chain.

Recommendation: If the protocol intends to support fee-on-transfer token, consider makes the change:

```
IERC20 token = supportedTokens[symbol];

uint256 balanceBefore = token.balanceOf(address(this));

token.transferFrom(msg.sender, address(this), amount);

uint256 balanceAfter = tokek.balanceOf(address(this));

uint256 amount = balanceAfter - balanceBefore;
```

EtaBridge: Fixed in commit bdff273c.

Cantina Managed: Fix verified.

3.4 Gas Optimization

3.4.1 Consider assign a variable to `IERC20(supportedTokens[symbol]` to save gas

Severity: Gas Optimization

Context: EtaBridge.sol#L62-L71

Description: The code queries `supportedTokens[symbol]` a few times and does not assign variable to `supportedTokens[symbol]`, which is not gas efficient.

Recommendation: Consider assign a variable to `IERC20(supportedTokens[symbol` to save gas in function `_lzSend` and `_lzReceive`.

```
  IERC20 token = supportedTokens[symbol];
+ token.transferFrom(msg.sender, address(this), amount)
  // ...
+ emit TokensBridged(receipt.guid, msg.sender,  token, amountAfterFee, receiver, fee, targetChainId);
```

EtaBridge: Fixed in commit 2d31d271.

Cantina Managed: Fix verified.

3.5 Informational

3.5.1 Emit events in function `updateFee`

Severity: Informational

Context: EtaBridge.sol#L33-L36

Description: The `updateFee` update fees but does not emit a event.

Recommendation: Consider emit a event for fee change offchain tracking in function `updateFee`.

EtaBridge: Fixed in commit 02a3dc78.

Cantina Managed: Fix verified.

3.5.2 Redundant Ownable Inheritance

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: The `EtaBridge` contract inherits from both `Ownable` and `OApp`, where `OApp` already inherits from `Ownable`.

Recommendation:

1. Remove the redundant `Ownable` inheritance:

```
- contract EtaBridge is Ownable, OApp, ReentrancyGuard {
+ contract EtaBridge is OApp, ReentrancyGuard {
```

EtaBridge: Fixed in commit 15fa673d.

Cantina Managed: Fix verified.