Coinbase Bridge (part 3)

Security Review

Cantina Managed review by:

Rikard hjort, Lead Security Researcher

Sujith Somraaj, Lead Security Researcher

0xHuy0512, Security Researcher

December 4, 2025

Contents

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 level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

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 are 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

Base is a secure, low-cost, builder-friendly Ethereum L2 built to bring the next billion users onchain.

From Sep 7th to Sep 10th the Cantina team conducted a review of bridge on commit hash 39804577. The team identified a total of **4** issues:

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	2	2	0
Low Risk	0	0	0
Gas Optimizations	0	0	0
Informational	2	1	1
Total	**4**	**3**	**1**

2.1 Scope

The security review had the following components in scope for bridge on commit hash 39804577:

```
├── base
│   └── src
│       └── periphery
│           └── RelayerOrchestrator.sol
└── solana
    └── programs
        └── base_relayer
            └── src
                ├── constants.rs
                ├── instructions
                │   ├── config
                │   │   ├── mod.rs
                │   │   ├── set_eip1559_config.rs
                │   │   ├── set_gas_config.rs
                │   │   └── set_guardian.rs
                │   ├── initialize.rs
                │   ├── mod.rs
                │   └── pay_for_relay.rs
                ├── internal
                │   ├── eip_1559.rs
                │   ├── gas_config.rs
                │   ├── math.rs
                │   └── mod.rs
                ├── lib.rs
                └── state
                    ├── cfg.rs
                    ├── message_to_relay.rs
                    └── mod.rs
```

3 Findings

3.1 Medium Risk

3.1.1 Minimum base fee is not respected

Severity: Medium Risk

Context: eip_1559.rs#L67-L75, eip_1559.rs#L107-L110, bridge.rs#L99-L107, bridge.rs#L139-L144

Description: The Solana fee is based on an EIP-1559 calculation. There is a minimum base fee configured. However, this minimum is not respected. It is only used in one location, inside the following expression:

```
self.current_base_fee
    .checked_sub(base_fee_delta)
    .unwrap_or(self.config.minimum_base_fee)
```

Not that the checked subtraction will succeed with a `Some(X)` value as long as the result of `self.current_base_fee as i128 - base_fee_delta as i128 >= 0`. As long as that is true, the value passed to `unwrap_or()` will not be used.

Furthermore, in the case of many empty windows (with no gas usage), the base fee gets decayed through the following calculation:

```
//      base_fee_n+1 = base_fee_n - (base_fee_n / denom)
//                   = base_fee_n * (1 - 1 / denom)
//                   = base_fee_n * (denom - 1) / denom
// Thus:
//      base_fee_n = base_fee_0 * [(denom - 1) / denom]^n
```

If this drops the base fee below the minimum, it will not be adjusted up to the minimum.

Impact Explanation: The minimum base fee is there for a reason: to guarantee minimal incentives. If it is not respected, it can cost an unprepared relayer who expects it to be respected. Alternatively, it can lead to a user not paying enough gas to incentivize relaying, and their transaction being stuck.

It is also possible that if usage stays low that a saw-tooth pattern develops, where the base fee drops towards 0, only to go below zero and jump up to the minimum, only to decay towards zero again, making for a strange user experience.

Likelihood explanation: The likelihood depends on how much the relayers are used, but it is highly likely that at some point, the usage lulls enough to drop the price below the threshold.

Proof of Concept: In `solana/programs/base_relayer/src/internal/eip_1559.rs`:

- Test for `calc_base_fee()`:

```
#[test]
fn calc_base_fee_respects_minimum_base_fee() {
    // The calculation does not return a result respecting the base fee.
    let mut eip = new_eip();
    eip.config.minimum_base_fee = 90;
    eip.current_base_fee = 100;

    assert_eq!(eip.calc_base_fee(0), 90);
}
```

- Test for the decay calculation:

```
#[test]
fn calc_base_fee_respects_minimum_base_fee_refresh() {
    // When updating the base fee, the minimum is not respected.
    let mut eip = new_eip();
    eip.config.minimum_base_fee = 10;
    eip.current_base_fee = 100;

    eip.refresh_base_fee(eip.window_start_time + 100 *
    ↪  eip.config.window_duration_seconds as i64);
```

```
        assert_eq!(eip.current_base_fee, 10);
    }
```

Recommendation: In `calc_base_fee()`, use `unwrap_or_default()` to calculate the base rate update, disregarding the minimum:

```
self.current_base_fee
    .checked_sub(base_fee_delta)
    .unwrap_or_default()
```

This calculates, in essence:

$$\text{max}(0, \text{current\_base\_fee} - \text{base\_fee\_delta})$$

In `refresh_base_fee()`, adjust the fee to the minimum before updating and returning.

```
current_base_fee = current_base_fee.max(self.config.minimum_base_fee); // <-- Add this
↪  line

// Update state for new window
self.current_base_fee = current_base_fee;
self.current_window_gas_used = 0;
self.window_start_time = current_timestamp;

current_base_fee
```

Coinbase: Fixed in commit 6e87f4fb.

Cantina Managed: Fix verified.

3.1.2 Gas price calculation is incorrect due to misaligned time windows

Severity: Medium Risk

Context: initialize.rs#L47, eip_1559.rs#L31-L75, bridge.rs#L63-L107

Description: In `solana/programs/bridge/src/common/state/bridge.rs` and `solana/programs/base_relayer/src/internal/eip_1559.rs`, the `refresh_base_fee()` function is responsible for updating the `current_base_fee` based on network usage over a time window. The root cause of the issue is that `window_start_time` is updated with `self.window_start_time = current_timestamp;`, which sets the start of a new window to the current transaction's timestamp. This creates rolling time windows that vary in length depending on when transactions are processed, rather than fixed, aligned windows.

The calculation for `expired_windows_count` assumes fixed-duration windows. This discrepancy leads to incorrect gas price adjustments. For instance, if no transactions occur for a period of `2.5 * Eip1559.config.window_duration_seconds`, `expired_windows_count` will report 2 expired windows. The gas price will be adjusted as if two full, inactive windows have passed. The new window then starts at the current timestamp, making the elapsed period effectively a single, oversized window. This miscalculation results in a gas price that is lower than what would accurately reflect network activity, directly affecting the bridge by causing it to collect lower fees than expected.

Recommendation: To ensure the time windows are consistently aligned, `window_start_time` should be normalized to the beginning of the window in which the transaction falls.

```
- self.window_start_time = current_timestamp;
+ self.window_start_time += (expired_windows_count * self.config.window_duration_seconds)
↪  as i64;
```

Coinbase: Fixed in commit de7e2872.

Cantina Managed: Fix verified.

3.2 Informational

3.2.1 `validateAndRelay()` can be frontrun

Severity: Informational

Context: RelayerOrchestrator.sol#L54-L66

Description: As reported previously, `relayMessages()` can be frontrun -- realying any message in the array would cause the entire batch to fail, and this should be addressed.

Furthermore, it is possible to cause a `validateAndRelay()` call to fail by pre-registering messages. The call to `registerMessages()` can fail for two reasons:

1. If the bridge validator set has signed multiple batches with overlapping nonce numbers. Then someone could register a different batch, stopping the current batch from being validated. This should be considered a serious error among the validator set and is out of scope for this audit.
2. If the exact batch of message hashes has already been registered. This would update the nonce of the `BridgeValidator` causing `registerMessages()` to attempt to validate the hashes against the wrong nonce, causing a failure of signature verification.

Recommendation: Firstly, follow the recommendations in the previous front-running issue, so that trying to relay an already relayed message is idempotent and does not revert.

As for the failures of `registerMessages()`: The first case outlined above is considered unlikely and out of scope whereas the second case can easily come to pass, but with minimal disruption, since the exact batch would have already been validated.

To deal with the second case, the call to `registerMessages()` could be wrapped in a `try/catch` block, and a revert dealt with by e.g. emiting a message and carrying on.

Coinbase: Fixed in commit e4711cef.

Cantina Managed: The fixes guarantee that relaying will never revert, while validation may revert, which is acceptable.

3.2.2 Gas cost calculations are at risk of overflow

Severity: Informational

Context: gas_config.rs#L57-L58

Description: The gas cost calculations in `gas_config` are at risk of overflow for sizable parameters and do not use checked math to handle it gracefully.

Recommendation: Cast to `u128` or `u256` before calculating gas cost. Or else use checked math to report the overflow issue.

Coinbase: Acknowledged.

Cantina Managed: Acknowledged.