CANTINA

Coinbase Bridge

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 **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**
2.1 Scope . . . 3

3 Findings **6**
3.1 Medium Risk . . . 6
3.1.1 Unbounded message size could cause temporary fund lock due to Solana transaction limits . . . 6
3.2 Low Risk . . . 6
3.2.1 Bridge initialization can be frontrun . . . 6
3.2.2 Frontrunning can block message batch submissions . . . 6
3.2.3 Lack of zero-value checks on denominators can lead to denial of service . . . 8
3.2.4 Pausing does not stop message registration on Base . . . 8
3.2.5 Stale output root cannot be overridden . . . 8
3.3 Gas Optimization . . . 9
3.3.1 Unnecessary twin contract deployment for simple transfers . . . 9
3.4 Informational . . . 9
3.4.1 Remove unused struct, constants and functions . . . 9
3.4.2 Anchor discriminator magic constant . . . 9
3.4.3 `executeCall()` does not need to be `payable` . . . 10
3.4.4 Unused `payer` account in `RelayMessage` . . . 10
3.4.5 Unnecessary `Pda` struct increases code complexity . . . 11
3.4.6 Unnecessary signature length check . . . 11
3.4.7 Incorrect validation for signature recovery ID . . . 11
3.4.8 Misleading variable name for mint account . . . 12
3.4.9 Sol vault balance dropping below rent exemption can halt relaying message on Solana 12
3.4.10 Unnecessary mutable attribute for signer account . . . 12
3.4.11 `initialize_call_buffer()` reverts for buffer sizes over 10KB . . . 13
3.4.12 Inaccurate error documentation in `VerificationLib` . . . 13
3.4.13 Integer division truncation in `TokenLib` . . . 13
3.4.14 Inconsistent threshold configuration for validators . . . 14
3.4.15 Inconsistent event emission on the Solana program . . . 14
3.4.16 Inconsistent validator limit between contracts . . . 14
3.4.17 Unhelpful error message when trying to register a 0-length array . . . 15
3.4.18 Message execution flag could be set before CPI calls for enhanced safety . . . 15
3.4.19 In fix: overly permissive account encoding of Solana accounts . . . 16

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 Aug 21st to Sep 2nd the Cantina team conducted a review of bridge on commit hash c9b24df2. The team identified a total of **26** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	1	1	0
Low Risk	5	4	1
Gas Optimizations	1	1	0
Informational	19	17	2
Total	**26**	**23**	**3**

2.1 Scope

The security review had the following components in scope for bridge on commit hash c9b24df2:

```
├── base
│   └── src
│       ├── Bridge.sol
│       ├── BridgeValidator.sol
│       ├── CrossChainERC20.sol
│       ├── CrossChainERC20Factory.sol
│       ├── interfaces
│       │   ├── IBuilderCodes.sol
│       │   └── IPartner.sol
│       ├── libraries
│       │   ├── CallLib.sol
│       │   ├── MessageLib.sol
│       │   ├── MessageStorageLib.sol
│       │   ├── SVMBridgeLib.sol
│       │   ├── SVMLib.sol
│       │   ├── TokenLib.sol
│       │   └── VerificationLib.sol
│       ├── periphery
│       │   └── BridgeRewards.sol
│       └── Twin.sol
└── solana
    └── programs
        └── bridge
            ├── Cargo.toml
            ├── src
            │   ├── base_to_solana
            │   │   ├── constants.rs
            │   │   ├── instructions
            │   │   │   ├── mod.rs
            │   │   │   ├── prove_message.rs
            │   │   │   ├── register_output_root.rs
            │   │   │   ├── relay_message.rs
            │   │   │   └── token
            │   │   │       ├── finalize_sol_transfer.rs
            │   │   │       ├── finalize_spl_transfer.rs
            │   │   │       ├── finalize_wrapped_token_transfer.rs
            │   │   │       └── mod.rs
            │   │   │
            │   │   ├── internal
            │   │   │   ├── ix.rs
            │   │   │   ├── mmr.rs
            │   │   │   ├── mod.rs
            │   │   │   └── signatures.rs
            │   │   ├── mod.rs
            │   │   └── state
            │   │       ├── incoming_message.rs
            │   │       ├── mod.rs
            │   │       ├── output_root.rs
            │   │       └── partner_config.rs
            │   ├── common
            │   │   ├── constants.rs
            │   │   ├── instructions
            │   │   │   ├── config
            │   │   │   │   ├── base_oracle_signers.rs
            │   │   │   │   ├── buffer.rs
            │   │   │   │   ├── eip1559.rs
            │   │   │   │   ├── gas.rs
            │   │   │   │   ├── mod.rs
            │   │   │   │   ├── pause.rs
            │   │   │   │   └── protocol.rs
            │   │   │   ├── guardian.rs
            │   │   │   ├── initialize.rs
            │   │   │   └── mod.rs
            │   │   ├── internal
            │   │   │   ├── init_config.rs
            │   │   │   ├── math.rs
            │   │   │   ├── metadata.rs
            │   │   │   └── mod.rs
            │   │   ├── mod.rs
            │   │   └── state
            │   │       ├── bridge.rs
            │   │       └── mod.rs
            │   ├── lib.rs
            │   ├── solana_to_base
            │   │   ├── constants.rs
            │   │   ├── instructions
            │   │   │   ├── bridge_call.rs
            │   │   │   ├── bridge_sol.rs
            │   │   │   ├── bridge_spl.rs
            │   │   │   ├── bridge_wrapped_token.rs
            │   │   │   ├── buffered
            │   │   │   │   ├── append_to_call_buffer.rs
            │   │   │   │   ├── bridge_call.rs
            │   │   │   │   ├── bridge_sol.rs
            │   │   │   │   ├── bridge_spl.rs
            │   │   │   │   ├── bridge_wrapped_token.rs
            │   │   │   │   ├── close_call_buffer.rs
            │   │   │   │   ├── initialize_call_buffer.rs
            │   │   │   │   └── mod.rs
            │   │   │   ├── mod.rs
            │   │   │   └── wrap_token.rs
            │   │   ├── internal
            │   │   │   ├── bridge_call.rs
            │   │   │   ├── bridge_sol.rs
            │   │   │   ├── bridge_spl.rs
            │   │   │   ├── bridge_wrapped_token.rs
            │   │   │   └── mod.rs
            │   │   ├── mod.rs
            │   │   └── state
            │   │       ├── call_buffer.rs
            │   │       ├── mod.rs
            │   │       └── outgoing_message.rs
            │   └── test_utils
            │       └── mod.rs
            └── Xargo.toml
```

3 Findings

3.1 Medium Risk

3.1.1 Unbounded message size could cause temporary fund lock due to Solana transaction limits

Severity: Medium Risk

Context: MessageStorageLib.sol#L157, prove_message.rs#L55

Description: The Bridge contract allows users to send messages of arbitrary size from Base to Solana without enforcing size constraints that exist on the Solana side. This asymmetry creates an issue, allowing users to send oversized messages that become permanently unprocessable on the Solana network.

Solana enforces a 1232-byte transaction size limit, which will cause the `prove_message()` instruction to fail if the data parameter exceeds this threshold. However, the Base contract performs no equivalent validation before locking funds or burning tokens.

Likelihood explanation: This issue has a MEDIUM likelihood because the message payload for standard cross-chain bridging stays within the necessary limits. Only custom DApps, which need to send large data alongside token transfers, will be impacted.

For example, if the DApp needs to bridge and make a destination swap on Solana, then those transactions might go beyond 1232 bytes leading to funds being blocked.

Impact explanation: The core vulnerability is that users can create messages on Base that will never be processable on Solana, leading to temporary fund loss in some cases, making it a MEDIUM-impact issue, as through upgrading the program funds could be recovered.

Recommendation: Consider implementing a buffered message system for Base-to-Solana transfers, similar to the existing Solana-to-Base buffered call functionality, allowing large messages to be processed through chunking mechanisms.

Coinbase: Fixed in commit 7d870f4f.

Cantina Managed: Verified.

3.2 Low Risk

3.2.1 Bridge initialization can be frontrun

Severity: Low Risk

Context: initialize.rs#L8-L69

Description: There is no access control on the Bridge initialization. It is possible for a malicious actor to frontrun the initialization with their own parameters. If this goes undetected, user funds could be at risk.

See this thread by pcaversaccio for a recent widespread attack discovered on Ethereum exploiting insufficient frontrunning and initialization checks.

Recommendation: Restrict initialization to a privileged accounts.

Coinbase: Fixed in commit 4026e033.

Cantina Managed: Fix verified.

3.2.2 Frontrunning can block message batch submissions

Severity: Low Risk

Context: Bridge.sol#L335

Summary: Trying to re-relay an already successfully executed message reverts the entire transaction, even if it's a batch relay of many independent messages.

Finding Description: In `_validateAndRelay()` in `Bridge`, a check is performed that a message has not already been successfully relayed. If it has, the transaction reverts.

This becomes a problem because relaying messages is permissionless. An attacker that could observe mempool transactions could notice a batch of messages being relayed, and frontrun it, relaying a single message in the batch, which would cause the rest of the batch to fail to be relayed. Even without frontrunning/transaction ordering capabilities, they could sporadically perform bridge operations, observe when they have been pre-validated, and submit them, causing occasional batch failures.

Impact Explanation: If a batch relay fails for this reason, the relayer would have to re-submit the batch without the already executed transaction. This re-submission could be frontrun again in the same manner. This causes a loss of gas for the relayer and disrupts bridge operation, leading to longer bridging times.

Likelihood Explanation: The likelihood depends to a great deal on the chain. Base is fairly resistant to frontrunning due to being operated by trusted sequencers. A malicious sequencer, however, could in theory collude with an attacker and leak transactions, undetectably. Even without a reliable option to observer the mempool and frontrun, the attack can be carried out probabilistically, as described above.

Proof of Concept: A batch of messages `M` consists of messages that have been pre-validated. A relayer then calls `relayMessages(M)`. `M` may contain as many and as complex messages as the block gas limit permits.

An attacker picks a single message that they deem will succeed, `M[i]`. For maximum cost to the relayer, assume `M` is large and `i == M.length - 1`. The attacker then calls `relayMessages([M[i]])`.

```
function relayMessages(IncomingMessage[] calldata messages) external nonReentrant
↪  whenNotPaused {
    for (uint256 i; i < messages.length; i++) {
        _validateAndRelay(messages[i]);
    }
}
```

When the message succeeds, it is marked as successfully executed, in the `successes` mapping. When the relayer relays its batch, all messages up until `i-1` will be tried. When `M[i]` is validated and relayed, the highlighted line below will cause the whole transaction to revert, and no message in the batch except `M[i]` will have been successfully relayed yet.

```
function _validateAndRelay(IncomingMessage calldata message) private {
    bytes32 messageHash = getMessageHash(message);

    // Check that the message has not already been relayed.
    require(!successes[messageHash], MessageAlreadySuccessfullyRelayed()); // <--- THROWS
    ↪  ERROR!

    require(BridgeValidator(BRIDGE_VALIDATOR).validMessages(messageHash),
    ↪  InvalidMessage());

    try this.__relayMessage(message) {
        // Register the message as successfully relayed.
        delete failures[messageHash];
        successes[messageHash] = true;
        emit MessageSuccessfullyRelayed(messageHash);
    } catch {
        // Register the message as failed to relay.
        failures[messageHash] = true;
        emit FailedToRelayMessage(messageHash);
    }
}
```

Recommendation: Don't throw an error in case the message has already been relayed. Instead, return and move on to the next message. Optionally, emit an event. `Bridge.sol`:

```
- error MessageAlreadySuccessfullyRelayed();
+ event MessageAlreadySuccessfullyRelayed(bytes32 indexed messageHash);
```

```
-     require(!successes[messageHash], MessageAlreadySuccessfullyRelayed());
+     if(successes[messageHash]) {
+         emit MessageAlreadySuccessfullyRelayed(messageHash);
+         return ;
+ }
```

Coinbase: Fixed in commit e4711cef.

Cantina Managed: Fix verified.

3.2.3 Lack of zero-value checks on denominators can lead to denial of service

Severity: Low Risk

Context: bridge.rs#L51-L53, bridge.rs#L158-L159, bridge.rs#L167-L171

Description: The state variables `Bridge.Eip1559.Eip1559Config.denominator`, `Bridge.Eip1559.Eip1559Config.window_duration_seconds`, `Bridge.GasConfig.gas_cost_scaler_dp`, and `Bridge.ProtocolConfig.block_interval_requirement` can be set to zero. This can occur during contract initialization via `initialize()` or through their respective setter functions: `set_adjustment_denominator()`, `set_window_duration()`, `set_gas_cost_scaler_dp()`, and `set_block_interval_requirement()`. These functions lack input validation to prevent zero values. If any of these variables are set to zero and subsequently used in a division, the transaction will revert due to a division-by-zero error, causing a denial of service for features relying on these calculations.

Recommendation: Implement zero-value checks for `denominator`, `window_duration_seconds`, `gas_cost_scaler_dp`, and `block_interval_requirement` within the `initialize()` function and their corresponding setter functions. Ensure these variables are always greater than zero before updating the state.

Coinbase: Fixed in commit f02692a0.

Cantina Managed: Fix verified.

3.2.4 Pausing does not stop message registration on Base

Severity: Low Risk

Context: BridgeValidator.sol#L142-L159, register_output_root.rs#L62-L65

Description: The pausing mechanism is inconsistent across chains. On Solana, `register_output_root()` is paused when the bridge is paused. However, on Base, `registerMessages()` at `base/src/BridgeValidator.sol` does not check for the paused state. This allows registering messages on Base while paused, which is unexpected behavior.

Recommendation: Pause the `registerMessages()` function when the bridge is paused. This will align its behavior with the Solana implementation, ensuring a uniform pausing mechanism.

Coinbase: Fixed in commit 669e4099.

Cantina Managed: Fix verified.

3.2.5 Stale output root cannot be overridden

Severity: Low Risk

Context: register_output_root.rs#L28-L35

Description: The `register_output_root()` instruction initializes a new `OutputRoot` account for each root update. This design could be problematic.

Suppose a registered `OutputRoot` is incorrect or malicious (e.g., due to a compromised guardian set allowing a faulty root), and a new, correct root is subsequently registered. In that case, the old, incorrect root will still exist on-chain. This is because a new account is created for the new root, leaving the old one untouched.

Consequently, there is no mechanism to invalidate or override the stale root, potentially allowing malicious transactions validated against it to be processed.

Using a single, updatable account would mitigate this risk. While this approach introduces a minor race condition where a proof could be generated against a root that changes before submission, this is a manageable trade-off for the increased security against stale roots.

Recommendation: It is recommended to use a single, persistent `OutputRoot` account. The logic should be modified to initialize this account once, and then update its value in subsequent calls to `register_output_root()`, rather than creating a new account for each root. This ensures that there is only one canonical `OutputRoot` at any given time, preventing issues with stale or malicious roots.

Alternatively, we can pause the bridge, upgrade the program with a new function to remove the malicious `OutputRoot` account.

Cantina Managed: Acknowledged.

Coinbase: Acknowledging that we considered this and are choosing not to address it since we could change OUTPUT_ROOT_SEED in the future to invalidate output roots.

3.3 Gas Optimization

3.3.1 Unnecessary twin contract deployment for simple transfers

Severity: Gas Optimization

Context: Bridge.sol#L218-L226

Description: In `Bridge.sol` contract, when a message is relayed, the contract deploys a deterministic "twin" contract for the `message.sender` if one does not already exist. This deployment occurs for all types of messages.

However, for simple token transfers where `message.ty` is `MessageType.Transfer`, this twin contract is not needed. The unconditional deployment logic leads to unnecessary gas expenditure on the Base chain for users who only intend to transfer tokens and not execute calls.

Recommendation: To optimize gas usage and avoid creating unnecessary contracts, the twin deployment logic should only be executed for message types that require it, such as `MessageType.Call` and `MessageType.TransferAndCall`.

Coinbase: Fixed in commit babf00cb.

Cantina Managed: Fix verified.

3.4 Informational

3.4.1 Remove unused struct, constants and functions

Severity: Informational

Context: SVMBridgeLib.sol#L8-L22, SVMLib.sol#L76, SVMLib.sol#L99, mod.rs#L23-L29

Description: There are multiple instances across the entire codebase where unused structs, constants, and functions are found. Removing them will result in increased code quality.

1. The `TransferParams` struct in `solana_to_base/instructions/mod.rs` is unused and can be removed.
2. The SVMLib contains two functions: `serializePubkeyAccount` and `serializePdaAccount`, which are not used anywhere in the code and are redundant.
3. The SVMBridgeLib.sol file contains three declared constants `_TOKEN_PROGRAM_ID`, `_TOKEN_PROGRAM_2022_ID`, and `_SYSTEM_PROGRAM_ID` that aren't used anywhere in the entire code scope.

Recommendation: Consider deleting the unused or redundant declarations mentioned above.

Coinbase: Fixed in commits 61782283 and 54a2575d.

Cantina Managed: Fix verified.

3.4.2 Anchor discriminator magic constant

Severity: Informational

Context: prove_message.rs#L35, register_output_root.rs#L31, initialize.rs#L27, bridge_call.rs#L46, bridge_sol.rs#L57, bridge_spl.rs#L75, bridge_wrapped_token.rs#L61, bridge_call.rs#L62, bridge_sol.rs#L71, bridge_spl.rs#L87, bridge_wrapped_token.rs#L77, initialize_call_buffer.rs#L33, wrap_token.rs#L75

Description: The Anchor docs state:

> Prior to Anchor v0.31, discriminators were always 8 bytes in size. However, starting with Anchor v0.31, it is possible to override the default discriminators, and the discriminator length is no longer fixed, which means this trait can also be implemented for non-Anchor programs.

Recommendation: Prefer using a constant or for easy update of the discriminator size as needed.

Coinbase: Fixed in commit 4a83b609.

Cantina Managed: Fix verified.

3.4.3 `executeCall()` does not need to be `payable`

Severity: Informational

Context: Twin.sol#L84

Description: The `executeCall()` function in the `Twin` contracts is marked `payable`. This is not necessary, as it has a `receive()` function, the bridge is the only caller (apart from the contract itself) and the bridge pushes funds to the contract before calling `executeCall()`. Marking functions as `payable` is a valid strategy for reducing gas costs, but this is not practiced throughout the rest of the code base, which leads us to believe it is a mistake in this case.

Recommendation: Remove the `payable` type on `executeCall()`.

```diff
- function execute(Call calldata call) external payable {
+ function execute(Call calldata call) external {
```

Coinbase: Fixed in commit 594c2253.

Cantina Managed: Fix verified.

3.4.4 Unused `payer` account in `RelayMessage`

Severity: Informational

Context: relay_message.rs#L16-L18

Description: The `RelayMessage` instruction requires a `payer` account to be provided and signed. However, this `payer` account is not utilized within the instruction's logic for any operation. Its inclusion is unnecessary and can be safely removed.

Recommendation: Remove the `payer` account from the `RelayMessage` accounts struct to improve code clarity and gas efficiency.

```diff
  pub struct RelayMessage<'info> {

-     pub payer: Signer<'info>,


      #[account(mut)]
      pub message: Account<'info, IncomingMessage>,


      #[account(
          seeds = [BRIDGE_SEED],
          bump
      )]
      pub bridge: Account<'info, Bridge>,
  }
```

Coinbase: Fixed in commit d98b00e7.

Cantina Managed: Fix verified.

3.4.5 Unnecessary `Pda` struct increases code complexity

Severity: Informational

Context: ix.rs#L30-L36

Description: In `solana/programs/bridge/src/base_to_solana/internal/ix.rs`, the `Pda` struct, which contains `seeds` and a `program_id`, appears to be unnecessary. When users relay messages on Solana, they are required to include the `Pubkey` of the PDAs in the instruction to call the `relay_message()` function. Since the `Pubkey` is already provided by the user, storing `seeds` and `program_id` to derive the PDA on-chain is a redundant step.

Recommendation: For simplicity and gas efficiency, consider refactoring the logic to use the `Pubkey` directly, removing the need for the `Pda` struct. This would streamline the data structures and reduce on-chain computations.

Coinbase: Fixed in commit 61782283.

Cantina Managed: Fix verified.

3.4.6 Unnecessary signature length check

Severity: Informational

Context: signatures.rs#L49-L52

Description: In `solana/programs/bridge/src/base_to_solana/internal/signatures.rs`, the `recover_eth_address()` function accepts the `signature` parameter as a dynamically sized slice `&[u8]`. Consequently, it must perform a runtime check to ensure the slice's length is 65 bytes. The use of a slice for a fixed-size input is the root cause, leading to redundant validation logic. This adds unnecessary code and a slight overhead, making the implementation less robust than it could be by leveraging the type system.

Recommendation: Change the `signature` parameter's type from a slice `&[u8]` to a fixed-size array reference `&[u8; 65]`. This enforces the correct length at compile time, removes the need for the manual length check, and makes the function signature more precise.

```diff
- pub fn recover_eth_address(signature: &[u8], message_hash: &[u8; 32]) -> Result<[u8;
↪  20]> {
+ pub fn recover_eth_address(signature: &[u8; 65], message_hash: &[u8; 32]) ->
↪  Result<[u8; 20]> {
-     if signature.len() != 65 {
-         return err!(SignatureError::InvalidSignatureLength);
-     }
```

Coinbase: Fixed in commit 4de91af2.

Cantina Managed: Fix verified.

3.4.7 Incorrect validation for signature recovery ID

Severity: Informational

Context: signatures.rs#L56-L58

Description: The signature verification logic at `solana/programs/bridge/src/base_to_solana/internal/signatures.rs` incorrectly validates the `recovery_id`. The current check, `if recovery_id >= 4`, permits recovery IDs of 2 and 3. However, according to the `secp256k1_recover()` natspec:

> In practice this function will not succeed if given a recovery ID of 2 or 3, as these values represent an "overflowing" signature, and this function returns an error when parsing overflowing signatures.

The validation should only permit `recovery_id` values of 0 or 1.

Recommendation: The check should be modified to strictly allow only valid recovery IDs, which are 0 and 1.

```
- if recovery_id >= 4 {
+ if recovery_id >= 2 {
      return err!(SignatureError::InvalidRecoveryId);
  }
```

Coinbase: Fixed in commit 3d6d1785.

Cantina Managed: Fix verified.

3.4.8 Misleading variable name for mint account

Severity: Informational

Context: metadata.rs#L122-L125, metadata.rs#L124-L127

Description: In `try_from()` at `solana/programs/bridge/src/common/internal/metadata.rs`, the parameter `value` of type `&AccountInfo<'_>` is ambiguously named. The name `value` is too generic and does not clearly indicate that the account is expected to be a token mint.

Recommendation: Rename the `value` parameter to better reflect its purpose. Suggested names include `mint` or `mint_account`.

Coinbase: Fixed in commit e62a957c.

Cantina Managed: Fix verified.

3.4.9 Sol vault balance dropping below rent exemption can halt relaying message on Solana

Severity: Informational

Context: finalize_sol_transfer.rs#L33

Description: In `finalize_sol_transfer()` at `solana/programs/bridge/src/base_to_solana/instructions/token/finalize_sol_transfer.rs`, the `sol_vault_info` account transfers native SOL out to recipients. If a transfer causes the vault's balance to fall below the rent-exemption threshold (even when the `sol_vault_info` account doesn't contain any data) and above 0, the instruction will fail.

Recommendation: Consider adding minimum lamports to the `sol_vault_info` account that surpass the rent-exemption threshold for zero data during the deployment phase.

Coinbase: We have added a transfer to SOL vault to our deployment runbook.

Cantina Managed: Acknowledged.

3.4.10 Unnecessary mutable attribute for signer account

Severity: Informational

Context: append_to_call_buffer.rs#L10-L13

Unnecessary mutable attribute for signer account:

Description: The `owner` account in the `AppendToCallBuffer` struct at `solana/programs/bridge/src/solana_to_base/instructions/buffered/append_to_call_buffer.rs` is unnecessarily marked as mutable. The `owner` is a `Signer` used for authorization via the `has_one = owner` constraint on `call_buffer` and is not modified by `append_to_call_buffer_handler()`.

Recommendation: Remove the mutable attribute for the `owner` account to follow security best practices.

```
  pub struct AppendToCallBuffer<'info> {
-     #[account(mut)]
      pub owner: Signer<'info>,
```

Coinbase: Fixed in commit 6f18b574.

Cantina Managed: Fix verified.

3.4.11 `initialize_call_buffer()` reverts for buffer sizes over 10KB

Severity: Informational

Context: initialize_call_buffer.rs#L30-L36

Description: The `initialize_call_buffer()` instruction is responsible for initializing a `CallBuffer` account. According to Solana's constraints, an account's size cannot be larger than 10KB in a single transaction. If a `buffer_size` greater than 10KB is provided to this function, the transaction will revert, preventing the creation of larger call buffers. This limitation can hinder the system's ability to handle a high volume of buffered calls.

Recommendation: To address this, two different approaches can be considered:

1. Enforce an upper bound on `BufferConfig.max_call_buffer_size` to ensure `CallBuffer` account cannot be set higher than 10,240 bytes. This validation should be performed in the `initialize()` and `set_max_call_buffer_size()` functions where this value is configured. This prevents oversized buffer allocation attempts from the outset.
2. Create a new instruction that allows for resizing an existing `CallBuffer` account. This would enable allocating buffers larger than 10KB in a separate transaction after initial creation.

Coinbase: Acknowledged.

Cantina Managed: Acknowledged.

3.4.12 Inaccurate error documentation in `VerificationLib`

Severity: Informational

Context: VerificationLib.sol#L59

Description: The `InvalidThreshold` error in `VerificationLib.sol` has misleading documentation that only describes one of its two failure conditions.

```
/// @notice Thrown when threshold is 0.
error InvalidThreshold();
```

Actual Usage:

The error is thrown when:

- Threshold == 0 (as documented).
- Threshold > validatorCount (not documented).

This occurs in both `initialize()` and `setThreshold()` functions where the validation is:

```
require(threshold > 0 && threshold <= validators.length, InvalidThreshold());
```

Recommendation: Update the error documentation to reflect both failure conditions accurately:

```
/// @notice Thrown when threshold is invalid (0 or exceeds validator count).
error InvalidThreshold();
error InvalidThreshold();
```

Coinbase: Fixed in commit da6ac2fd.

Cantina Managed: Fix verified.

3.4.13 Integer division truncation in `TokenLib`

Severity: Informational

Context: TokenLib.sol#L169-L175

Description: The `TokenLib.initializeTransfer()` function is used by the Bridge contract to initiate token bridging. This function has an edge case where it allows zero-amount bridging due to division truncation.

When users bridge ERC20 tokens that charge transfer fees, the contract:

1. Transfers tokens from the user (fees are deducted).
2. Calculates how much was actually received.
3. Converts this amount to remote chain units using division.

Problem: If the received amount is smaller than the conversion scalar, division truncates to zero.

Recommendation: Add a check to prevent zero-amount transfers:

```
uint256 receivedRemoteAmount = receivedLocalAmount / scalar;
require(receivedRemoteAmount > 0, "Transfer amount too small after fees");
```

Coinbase: Fixed in commit 68eb89e5.

Cantina Managed: Fix verified.

3.4.14 Inconsistent threshold configuration for validators

Severity: Informational

Context: BridgeValidator.sol#L29-L31

Description: The threshold for the base validator is configurable via `BridgeValidator.setThreshold()`, while the partner validator's threshold is not. This creates an inconsistent management model where one validator's setting is mutable and the other's is effectively immutable without a contract upgrade. This disparity can lead to operational complexity and potential misconfiguration when managing validator settings.

Recommendation: To ensure consistent and predictable behavior, both validator thresholds should be managed symmetrically. Implement a setter function for the partner validator's threshold, or alternatively, make both thresholds immutable and only settable via contract upgrade.

Coinbase: Fixed in commit 0cdc5021.

Cantina Managed: Fix verified.

3.4.15 Inconsistent event emission on the Solana program

Severity: Informational

Context: guardian.rs#L18-L21

Description: The Solana program only emits a `GuardianTransferred` event within the `transfer_guardian()` instruction. As the sole event in the program, this creates an inconsistent and incomplete monitoring mechanism. The absence of events for other critical state transitions, such as message proving, hinders off-chain tracking and reduces the protocol's transparency.

Recommendation: A consistent event strategy should be implemented. Either add events for all significant state changes to enhance monitorability or remove the existing `GuardianTransferred` event for consistency. Adding more events is recommended for improved security and transparency.

Coinbase: Fixed in commit c317c66d.

Cantina Managed: Fix verified.

3.4.16 Inconsistent validator limit between contracts

Severity: Informational

Context: VerificationLib.sol#L11-L15, VerificationLib.sol#L91-L103, VerificationLib.sol#L120-L132, bridge.rs#L201-L202

Description: The `BaseOracleConfig` struct in `solana/programs/bridge/src/common/state/bridge.rs` restricts the number of signers to a maximum of 16, as defined by `MAX_SIGNER_COUNT`. However, the corresponding `base/src/BridgeValidator.sol` contract on the Base chain does not enforce an upper limit for its `validatorCount`.

The `initialize()` and `addValidator()` functions can be called to increase the number of validators indefinitely. This creates a discrepancy between the two systems.

Recommendation: To ensure consistency across both chains, a maximum limit for validators should be enforced in the `BridgeValidator` contract, mirroring the `MAX_SIGNER_COUNT` of 16 on Solana. This can be achieved by introducing a check in the `initialize()` and `addValidator()` functions within the `VerificationLib` library.

Coinbase: Fixed in commit b98a9798.

Cantina Managed: Fix verified.

3.4.17 Unhelpful error message when trying to register a 0-length array

Severity: Informational

Context: BridgeValidator.sol#L142

Description: If `registerMessages()` is called with empty arrays, then a `BaseThresholdNotMet()` error will be thrown. This is because there will be an attempt to validate the signatures, even though there are no messages to register. Of course, if the validators have signed the empty array, then the call succeeds as a no-op.

Recommendation: Check if there are any messages to register, and if not, throw a new error message named `NoMessagesRegistered()` or similar, or return, making it a no-op.

Coinbase: Fixed in commit 5afd42cf.

Cantina Managed: Fix verified.

3.4.18 Message execution flag could be set before CPI calls for enhanced safety

Severity: Informational

Context: relay_message.rs#L84

Description: The executed flag is currently set after all CPI calls complete, which could theoretically cause issues via self-cpi, especially with the current call depth increase from 4 to 8. While the current implementation has existing protections, setting the flag earlier would provide additional defense-in-depth.

```rust
// Process transfer if it exists
if let Some(transfer) = transfer {
    // ... transfer processing
}

// Execute instructions via CPI
for ix in ixs {
    solana_program::program::invoke_signed(
        &ix.into(),
        ctx.remaining_accounts,
        &[bridge_cpi_authority_seeds],
    )?;
}

// Flag set AFTER all operations
ctx.accounts.message.executed = true;
```

Recommendation: Move `ctx.accounts.message.executed = true;` to immediately after the existing validation checks and before any transfer processing or CPI calls:

```rust
// Set executed flag BEFORE any external calls
ctx.accounts.message.executed = true;

// Process transfer if it exists
if let Some(transfer) = transfer {
    // ... transfer processing
}

// Execute instructions via CPI
for ix in ixs {
    solana_program::program::invoke_signed(
        &ix.into(),
        ctx.remaining_accounts,
        &[bridge_cpi_authority_seeds],
    )?;
}
```

Coinbase: Fixed in PR 118.

Cantina Managed: Fix verified.

3.4.19 In fix: overly permissive account encoding of Solana accounts

Severity: Informational

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

Description: The fix for the issue "Unbounded message size could cause temporary fund lock due to Solana transaction limits" introduces a buffering mechanism for transactions from base to Solana. The file `SVMLib.sol` the new function `validateIxs()` iterates over all the Solana accounts in a transaction. Each account is serialized as follows:

```
32-byte pubkey || 1-byte is_writable || 1-byte is_signer
```

And so `validateIxs()` contains the following line:

```
// Last byte stores is_signer bit in LSB
bool isSigner = (uint8(acct[SERIALIZED_ACCOUNT_LENGTH - 1]) & 1) == 1;
```

There is no reason to allow any other byte than specifically `0x01` to indicate a signer since the whole byte is used for the flag, so no need to bitmask.

Recommendation: Check that `uint8(acct[SERIALIZED_ACCOUNT_LENGTH - 1]) == 1`, without bit masking. Optionally, also check that `uint8(acct[SERIALIZED_ACCOUNT_LENGTH - 2])` is either `0` or `1` to ensure that the `is_writable` flag is correctly set.

Coinbase: Fixed in commit 7d870f4f.

Cantina Managed: Fix verified.