PRIVATE

Code Assessment

of the Fast Bridge Smart Contracts

August 25, 2025

Produced for

Curve

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Curve Team,

Thank you for trusting us to help Curve with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Fast Bridge according to Scope to support you in forming an opinion on their security risks.

Curve implements Fast Bridge service leveraging LayerZero to allow users to quickly bridge crvUSD from Optimism or Arbitrum to Ethereum, without having to wait for the standard fault proof period.

The most critical subjects covered in our audit are fault proof risks assessment, integration with LayerZero and the native bridges, and functional correctness. Security regarding all the aforementioned subjects is high, we highlight the risk associated with the fast bridge in the informational issue Risk linked to the fast bridge.

The general subjects covered are integration with the existing Curve infrastructure, gas efficiency, and limit enforcement. Security regarding all the aforementioned subjects is high.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	3
• Code Corrected	3

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Fast Bridge repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	08 August 2025	61775532cbbdfe097fda1ec6f1e66547c7978cfc	Initial Version
2	25 August 2025	e94c92e347c4db45f28a0f231afcb5f350d58096	Fixes

For the Vyper smart contracts, the compiler version `0.4.3` was chosen.

The following contracts were included in the scope of the assessment:

```
contracts/bridgers/ArbitrumBridger.vy
contracts/bridgers/IBridger.vyi
contracts/bridgers/OptimismBridger.vy
contracts/FastBridgeL2.vy
contracts/FastBridgeVault.vy
contracts/messengers/L2MessengerLZ.vy
contracts/messengers/VaultMessengerLZ.vy
```

2.1.1 Excluded from scope

Anything not listed in scope of the assessment is considered out of scope. This includes tests, scripts and external libraries.

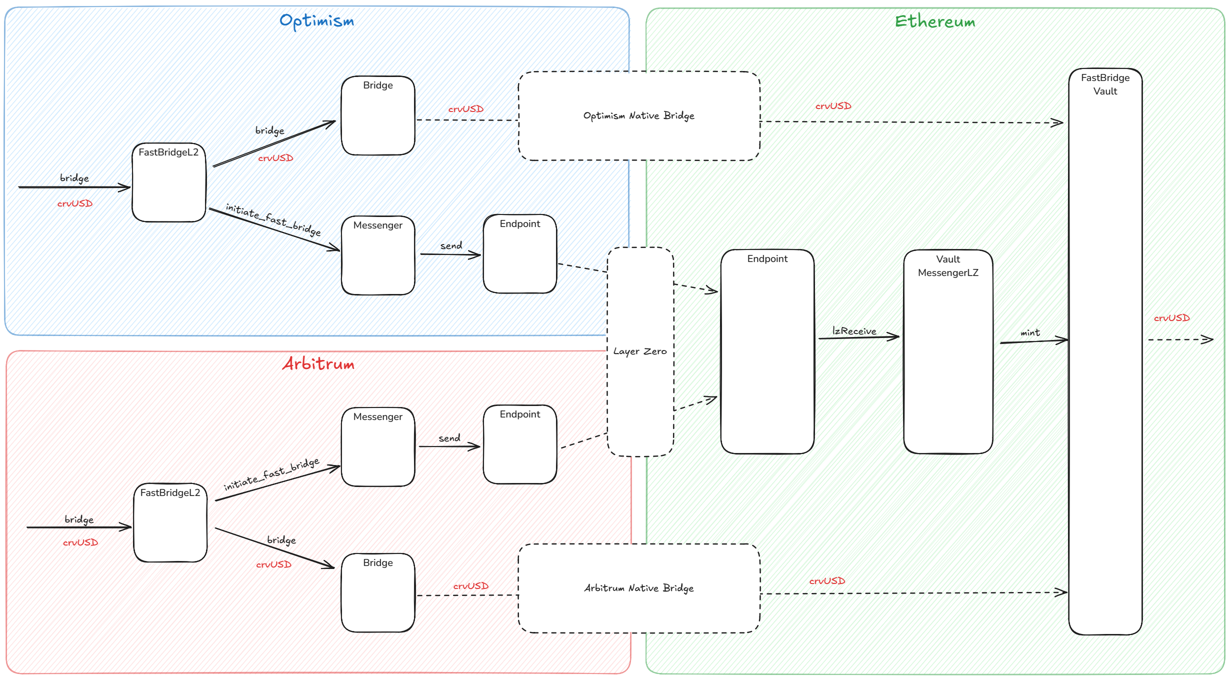
2.2 System Overview

This system overview describes the initially received version (**Version 1**) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Curve offers a Fast Bridge service to allow users to quickly bridge crvUSD from Optimism or Arbitrum to Ethereum, without having to wait for the standard fault proof period.

The architecture can be visualized as follows:



2.2.1 FastBridgeL2 Contract

The `FastBridgeL2` contract is the main contract enabling fast bridging of crvUSD from Optimism or Arbitrum to Ethereum, subject to a daily limit per chain. The entry point is the `bridge` function, which only permits bridging crvUSD. Requested amounts are capped to the remaining daily limit, but users can specify a minimum amount to enforce.

If enough funds are provided to cover the bridging costs, the function will:

1. Call the L2 `bridger` contract, which uses the chain's native bridge to transfer tokens (the native bridge involves a long fault-proof delay).
2. Send a LayerZero message to Ethereum to enable the fast bridge, allowing immediate receipt of crvUSD on Ethereum if the `FastBridgeVault` has sufficient funds.

Restricted Functions

The following administrative setter functions can only be called by the owner of the contract:

- `set_min_amount`: Sets the minimum amount of crvUSD that must be bridged in one transaction.
- `set_limit`: Sets the daily limit of crvUSD that can be bridged by all users.
- `set_bridger`: Sets the bridger contract.
- `set_messenger`: Sets the LayerZero messenger contract.

2.2.2 The Bridger contracts

The system supports two bridger contracts:

- `OptimismBridger` for Optimism
- `ArbitrumBridger` for Arbitrum

Both bridgers can be called directly or via `FastBridgeL2` to initiate the standard native bridge from L2 to Ethereum; they are unpermissioned and usable by anyone.

Their main entry is the bridge function. Implementations differ per L2 but perform the same action: initiate the chain's native bridge to transfer crvUSD to Ethereum.

- The OptimismBridger calls bridgeERC20To on the L2StandardBridge after burning tokens.
- The ArbitrumBridger calls outboundTransfer on the L2ERC20Gateway after burning tokens.

2.2.3 The L2MessengerLZ

After initiating the native bridge, the FastBridgeL2 calls initiate_fast_bridge on the L2MessengerLZ contract. Only FastBridgeL2 may call this function; it sends the VaultMessengerLZ on Ethereum a message containing the amount and recipient to indicate the native bridge is in flight so the recipient may receive tokens early.

Restricted functions:

Except for owner-only LayerZero OApp management functions, the following functions are owner-only:

- set_fast_bridge_l2: Sets the address of the FastBridgeL2 contract.
- set_gas_limit: Sets the gas limit to be enforced when delivering messages to Ethereum.

2.2.4 The FastBridgeVault

On Ethereum, Optimism's and Arbitrum's native bridges eventually release crvUSD to the FastBridgeVault contract (via L1ERC20Gateway and L1StandardBridge).

The FastBridgeVault holds bridged tokens until they can be minted to users.

The mint() function allows transferring crvUSD from the vault to a recipient if the recipient has an allocated balance. If called by an allowed minter (MINTER_ROLE), the contract attempts to transfer the requested amount immediately if it has sufficient balance; otherwise it allocates the amount to the recipient's vault balance for later withdrawal.

Restricted functions:

The following functions are restricted and can only be called by authorized roles:

- set_killed(): Kill the mint() functionality for a specific msg.sender or globally. Only callable by the KILLER_ROLE.
- set_fee(): Set the fee for minting crvUSD. Only callable by the DEFAULT_ADMIN_ROLE.
- set_fee_receiver(): Set the address that will receive fees from minting crvUSD. Only callable by the DEFAULT_ADMIN_ROLE.
- recover(): Recover tokens sent to the contract by mistake. Only callable by the DEFAULT_ADMIN_ROLE.

2.2.5 The VaultMessengerLZ

If a fast bridge was initiated, Ethereum's LZ endpoint will call VaultMessengerLZ.lzReceive. The contract holding MINTER_ROLE in the FastBridgeVault calls mint to mint or allocate the specified crvUSD to the recipient.

Restricted functions:

Except for owner-only LayerZero OApp management functions, the owner can call set_vault() to update the Vault address.

2.2.6 Receiving crvUSD on Ethereum

This section explains receiving crvUSD on Ethereum, partly described above.

The `FastBridgeVault` receives crvUSD from native bridges and mints to users, but native bridging can take about one week. To provide funds earlier, `VaultMessengerLZ` releases funds upon receiving a LayerZero message.

Since this requires available liquidity, Curve's `ControllerFactory` is expected to set a debt ceiling to the vault, by effectively minting unbacked tokens to the vault.

Thus availability depends not only on L2 daily limits but also on the vault's minting capability (debt ceilings).

If the vault cannot immediately release funds, it records the user's outstanding amount; users can call `mint` to withdraw once liquidity is available.

If the Curve DAO reduces the vault's debt ceiling, the `schedule_rug` function can be used to prevent withdrawals until the contract repays (burns) its debt.

2.2.7 Changelog

In Version 2 of the report, only fixes for findings raised by this review were implemented.

2.3 Trust Model

The Fast Bridge system involves several roles and trust relationships across multiple components:

2.3.1 General Architecture

- The `FastBridgeL2`, `OptimismBridger` and `L2MessengerLZ` are expected to be deployed on Optimism
- The `FastBridgeL2`, `ArbitrumBridger` and `L2MessengerLZ` are expected to be deployed on Arbitrum
- The `FastBridgeVault` and `VaultMessengerLZ` are expected to be deployed on Ethereum mainnet
- On every chain, `crvUSD` is compliant with the `ERC20` standard.
- On Optimism, `crvUSD` is the token deployed at `0xc52d7f23a2e460248db6ee192cb23dd12bddcbf6`
- On Arbitrum, `crvUSD` is the token deployed at `0x498bf2b1e120fed3ad3d42ea2165e9b73f99c1e5`
- The `VAULT` address of the L2's `FastBridgeL2` contracts is expected to be set to the `FastBridgeVault` address on Ethereum mainnet.
- The `VAULT_EID` of the L2's `L2MessengerLZ` contracts is expected to be set to the Ethereum Endpoint ID.
- Each of the L2's `L2MessengerLZ` contracts are expected to have exactly one peer; the corresponding L1 `VaultMessengerLZ` contract.
- The `VaultMessengerLZ` contracts are expected to have one peer for each of the L2's `L2MessengerLZ` contracts (as many peers as Rollup chains).
- The only address having `MINTER_ROLE` in the `FastBridgeVault` is the `VaultMessengerLZ`.

2.3.2 Roles and Trust Levels

System Admin

- Who:
  - `FastBridgeL2`, `VaultMessengerLZ`, and `L2MessengerLZ` owner
  - `VaultMessengerLZ` and `L2MessengerLZ` delegates
- Trust Level: Fully trusted
- Permissions: Can modify bridging parameters (min_amount, limit), replace bridger/messenger contracts, update LayerZero configurations
- Risk: Complete control over L2 bridging operations, could redirect funds or disable the system

Vault Admin

- Who: `FastBridgeVault`'s `DEFAULT_ADMIN_ROLE`
- Trust Level: Fully trusted
- Permissions: Can set fees, fee receiver, grant/revoke minter and killer roles, recover tokens
- Risk: Can drain the vault through token recovery, manipulate fees, control minting permissions

Emergency Killer

- Who: `FastBridgeVault`'s `KILLER_ROLE`
- Trust Level: Partially trusted
- Permissions: Can emergency-stop minting operations for specific addresses or globally
- Risk: Can halt the fast bridge service but cannot steal funds directly

End Users

- Trust Level: Untrusted
- Permissions: Can initiate bridge transactions, or mint their bridged token through the vault
- Risk: Limited to their own funds; protected by daily limits and minimum amounts

2.3.3 External Dependencies

LayerZero Protocol

LayerZero is out of scope for this review and is trusted to behave correctly and deliver messages to the correct destination.

The owner of the OApps can set arbitrary peers or update the `delegate` registered in the `Endpoint`. This could lead to draining or losing all funds, and hence, both the owner and the delegate are fully trusted.

The LayerZero executor is trusted to always deliver messages with the correct provided message value and gas limit. In the worst case, they could never deliver a message and keep what was paid to them for the gas limit and message value to forward on the receiving chain. The system would still be functional, but someone would have to pay for the message delivery costs by calling `EndpointV2.lzReceive()` on the receiving chain.

The configuration required for managing the LayerZero applications on multiple chains is considered out of scope for this review and should be performed by the `delegate` and `owner` of the OApps, this includes:

- Correctly setting peers to whitelist on each chain.

- Correctly setting the send and receive libraries to be used. If no send and receive libraries are explicitly set, the Endpoint will fall back to the default settings set by LayerZero Labs. In case LayerZero Labs changes the default settings, the oApps will be impacted and use the new default settings, which implies trust in LayerZero Labs.
- Correctly setting an Executor configuration, including the maximum message size and address of the executor.
- Correctly setting a DVN configuration, including block confirmations, required and optional DVN count and optional DVN threshold, required DVNs and optional DVNs. If the DVN was compromised, hypothetically transfers in flight would be censored / blocked, and arbitrary amount of tokens could be minted on the destination chain. The block confirmation is an important parameter that should reflect the trust model around the Rollup chain messages are sent from and should be carefully considered.

The system is not expected to be deployed on non-EVM chains.

Native Bridge and Fast Bridge

The native bridges on Optimism and Arbitrum are trusted to correctly work and ultimately allow releasing the crvUSD to the `FastBridgeVault` contract. The informational issue Risk linked to the fast bridge describes the risk associated with not waiting for finality and pausing of the native bridge. These risks should be closely monitored and managed through a combination of technical and governance measures.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Open Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Open Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	3

- Accounting Error When the Receiver Is Also the Fee Receiver Code Corrected
- Missing or Inconsistent Module Exports Code Corrected
- allowed_to_bridge Returns Incorrect Values Code Corrected

Informational Findings	5

- Daily Limit Computation Can Revert Due to Underflow Code Corrected
- Emergency Owner Cannot recover Tokens Specification Changed
- Event Not Indexed Code Corrected
- Hard-coded Endpoint ID Code Corrected
- Missing or Incorrect NatSpec Code Corrected

6.1 Accounting Error When the Receiver Is Also the Fee Receiver

Correctness Low Version 1 Code Corrected

CS-CURVE-FASTBRIDGE-001

In the `FastBridgeVault`, if `mint()` is called by a minter with `_receiver == fee_receiver`, the fee will be miscounted and locked in the contract until the default admin calls `recover` to unlock it.

```
@external
@nonreentrant
def mint(_receiver: address, _amount: uint256) -> uint256:
    """
    @notice Receive bridged crvUSD
    @param _receiver Receiver of crvUSD
    @param _amount Amount of crvUSD to mint (0 if not minter)
    @return Amount of crvUSD minted to receiver
    """
    assert not (self.is_killed[empty(address)] or self.is_killed[msg.sender])

    amount: uint256 = self.balanceOf[_receiver]
    if access_control.hasRole[MINTER_ROLE][msg.sender]:
        fee: uint256 = _amount * self.fee // 10 ** 18
        self.balanceOf[self.fee_receiver] += fee
        amount += _amount - fee

    available: uint256 = min(self._get_balance(), amount)
    if available != 0:
        extcall CRVUSD.transfer(_receiver, available)
    self.balanceOf[_receiver] = amount - available
    return available
```

This is because in such a case, `self.balanceOf[self.fee_receiver]` is first updated with the fee, and then `self.balanceOf[_receiver]` is overridden with the new amount, effectively losing the fee that was just added.

Code corrected:

A check was added in `mint()` to ensure that if `_receiver` is equal to `self.fee_receiver`, the fee is correctly accounted for.

6.2 Missing or Inconsistent Module Exports

Design Low Version 1 Code Corrected

CS-CURVE-FASTBRIDGE-014

The following module exports are missing:

- In `FastBridgeVault`, `access_control.set_role_admin`
- In `FastBridgeL2`, `ownable.renounce_ownership`

The following module exports are inconsistent:

- In `L2MessengerLZ`, `OApp.setReadChannel` is exported, but it is not in `VaultMessengerLZ`.

Code corrected:

In `FastBridgeVault` and `FastBridgeL2`, `access_control.set_role_admin` and `ownable.renounce_ownership` are now exported. In `L2MessengerLZ`, `OApp.setReadChannel` is no longer exported.

6.3 `allowed_to_bridge` Returns Incorrect Values

Correctness Low Version 1 Code Corrected

CS-CURVE-FASTBRIDGE-002

In `FastBridgeL2`, `allowed_to_bridge` is defined as:

```
@external
@view
```

```
def allowed_to_bridge(_ts: uint256=block.timestamp) -> (uint256, uint256):
    """
    @notice Get interval of allowed amounts to bridge
    @param _ts Timestamp at which to check (current by default)
    @return (minimum, maximum) amounts allowed to bridge
    """
    available: uint256 = self.limit - self.bridged[_ts // INTERVAL]

    balance: uint256 = staticcall CRVUSD.balanceOf(self)  # Someone threw money by mistake
    min_amount: uint256 = self.min_amount
    min_amount -= min(min_amount, balance)

    if available < min_amount:  # Not enough for bridge initiation
        return (0, 0)
    return (min_amount, available)
```

In general, the function is designed to return the minimum and maximum amounts allowed to bridge at a given timestamp.

However, the function accounts for the balance of the `FastBridgeL2` contract itself in the computation, assuming that users can use the contract's balance as part of their bridging amounts. This is however not the case and `bridge()` does not allow users to bridge the contract's own balance. If the contract's `crvUSD` balance is non-zero, the function `allowed_to_bridge()` will return incorrect values.

For example, assume that:

```
self.limit = 1000
self.bridged[block.timestamp // INTERVAL] = 800
self.min_amount = 400
CRVUSD.balanceOf(self) = 200
```

- `bridge()` would fail given the above values, as the contract will try to call `bridger.bridge()` with `amount = 1000 - 800 = 200` crvUSD, but with a `min_amount = 400`.
- As opposed, `allowed_to_bridge()` would return (200, 200) which is incorrect.

Code corrected:

The function was updated to match with the function `bridge`, and do not depends on the contract's balance anymore. Funds directly sent to the contract will be locked in the `FastBridgeL2` contract and will not be considered in the bridging amounts.

6.4 Daily Limit Computation Can Revert Due to Underflow

Informational | Version 1 | Code Corrected

CS-CURVE-FASTBRIDGE-003

In the `FastBridgeL2`, if the owner were to update `self.limit` in such a way that it is lower than the bridged amount for the current period, both `bridge()` and `allowed_to_bridge()` (with `_ts == block.timestamp`) will revert given the following computation:

```
self.limit - self.bridged[block.timestamp // INTERVAL]
```

Code corrected:

The function `_get_available` now ensures underflow cannot happen:

```
@view
def _get_available(ts: uint256=block.timestamp) -> uint256:
    limit: uint256 = self.limit
    bridged: uint256 = self.bridged[ts // INTERVAL]
    return limit - min(bridged, limit)
```

6.5 Emergency Owner Cannot `recover` Tokens

Informational | Version 1 | Specification Changed

CS-CURVE-FASTBRIDGE-005

In `FastBridgeVault`, the NatSpec of `recover` mentions:

```
Callable only by owner and emergency owner
```

However, the function is restricted to the owner only (`DEFAULT_ADMIN_ROLE`), and not the emergency owner.

Specification changed:

The documentation was updated to reflect the actual access control of the `recover` function.

6.6 Event Not Indexed

Informational | Version 1 | Code Corrected

CS-CURVE-FASTBRIDGE-006

The following events have no indexed fields, which may lead to difficulties in tracking important actions:

- All events in `FastBridgeL2`

Code corrected:

Curve added more events and indexed fields for events when found relevant.

6.7 Hard-coded Endpoint ID

Informational | Version 1 | Code Corrected

CS-CURVE-FASTBRIDGE-008

In `L2MessengerLZ`, the `VAULT_EID` is hard-coded as an immutable in the `__init__` function, however, LayerZero officially recommends using for example admin-restricted setters to configure endpoint IDs instead of hard-coding them.

Code corrected:

An owner-restricted setter function was added to allow the admin to configure the Vault EID instead of hard-coding it.

6.8 Missing or Incorrect NatSpec

Informational Version 1 Code Corrected

CS-CURVE-FASTBRIDGE-011

The following NatSpec comments are missing or incorrect:

FastBridgeL2:

- `__init__`: Missing
- `cost`: Missing return

ArbitrumBridger:

- `bridge`: NatSpec mentions Optimism and bridging from L1 to L2, where this is an Arbitrum-to-L1 bridge.

OptimismBridger:

- `bridge`: NatSpec mentions bridging from L1 to L2, where this is an L2-to-L1 bridge.

L2MessengerLZ:

- `quote_message_fee`: Missing return
- `initiate_fast_bridge`: Missing `_lz_fee_refund` param

FastBridgeVault:

- `__init__`: Missing

Code corrected:

All NatSpec comments have been added or corrected to accurately reflect the function behavior and parameters.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Duplicated Event Definition

Informational **Version 2**

CS-CURVE-FASTBRIDGE-015

In `FastBridgeL2`, the event `Bridge` is defined, although the contract implements `IBridger` which already includes the same event. `IBridger.Bridge` could hence be used instead of `Bridge`.

7.2 ERC20 Tokens Calls

Informational **Version 1** **Code Partially Corrected**

CS-CURVE-FASTBRIDGE-004

Across the codebase, ERC20 token `approve()`, `transfer()` and `transferFrom()` functions are called:

1. Without checking the return value.
2. Without specifying `default_return_value = True`

Although in some cases the token is known to be `crvUSD`, which is assumed to be a compliant ERC20 token on each chain it is deployed on, it is still a best practice to check the return value of these functions to ensure that the operation was successful, and to handle tokens not returning a value. Moreover, it makes the code more robust, reusable and easier to maintain.

In all the following cases the return value is not checked and `default_return_value` is not specified:

- `approve()`, called in `FastBridgeL2.__init__()`, `FastBridgeL2.set_bridger()`, `FastBridgeVault.__init__()`, `ArbitrumBridger.bridge()` and `OptimismBridger.bridge()`.
- `transfer()`, called in `FastBridgeVault.mint()`.
- `transferFrom()`, called in `FastBridgeL2.bridge()`.

Code partially corrected:

All ERC20 calls were made safe except for:

- `approve` in `FastBridgeVault.__init__()`, `ArbitrumBridger.bridge()` and `OptimismBridger.bridge()`.

7.3 Gas Savings

Informational **Version 1** **Code Partially Corrected**

CS-CURVE-FASTBRIDGE-007

The following gas optimizations were identified:

1. In `FastBridgeL2.__init__`, the two emitted events `SetMinAmount` and `SetLimit` each read from storage the value of the corresponding parameter, while they could use a local variable instead.
2. In `FastBridgeL2.bridge`, `self.bridged[block.timestamp // INTERVAL]` is read from storage twice, while it could be stored in a local variable.
3. In `FastBridgeVault.__init__`, `self.fee` is explicitly set to zero, while it is already initialized to zero by default.
4. In `FastBridgeVault.schedule_rug`, `self.schedule_rug` is read from storage after writing to it, while a local variable could be used.
5. In `FastBridgeVault.__init__` the role admin of `MINTER_ROLE` and `KILLER_ROLE` is explicitly set to the `DEFAULT_ADMIN_ROLE`, but in `snekmate.access_control`, by default, the admin role for all roles is already `DEFAULT_ADMIN_ROLE`.
6. In `FastBridgeVault.mint` the fee calculation and storage update is done even for zero fee.

Code partially corrected:

All gas optimizations were implemented except for points 1, 2, and 6. Curve acknowledges this and will not implement these optimizations.

7.4 Missing Sanity Checks

Informational **Version 1** **Code Partially Corrected**

CS-CURVE-FASTBRIDGE-009

The following functions do not perform sanity checks on their parameters:

1. `FastBridgeL2.__init__` does not check that the provided addresses are not zero.
2. `FastBridgeL2.bridge` does not check that the provided `_to` address is not zero, nor the address of `crvUSD` on Ethereum mainnet. In such case, the funds will be locked in the vault on Ethereum given the implementation of crvUSD:

```
@internal
def _transfer(_from: address, _to: address, _value: uint256):
    assert _to not in [self, empty(address)]

    self.balanceOf[_from] -= _value
    self.balanceOf[_to] += _value

    log Transfer(_from, _to, _value)
```

3. In `FastBridgeL2`, `set_bridger` and `set_messenger` do not check that the provided addresses are not zero.
4. In `FastBridgeL2`, `allowed_to_bridge` does not ensure the given timestamp is greater than or equal to the current block timestamp.
5. In `OptimismBridger` and `ArbitrumBridger`, the `bridge` function does not check that the provided `_to` address is not zero.

6. In `OptimismBridger`, the `bridge` function does not check that the given `_token` is a `OptimismMintableERC20` using EIP-165.
7. In `L2MessengerLZ`, `set_fast_bridge_l2` does not check that the provided address is not zero.
8. In `VaultMessengerLZ`, `set_vault` does not check that the provided address is not zero.
9. In `FastBridgeVault`, `__init__` does not check that the provided addresses are not zero.

Code partially corrected:

All above functions now include sanity checks for their parameters except for the following:

1. `FastBridgeL2.bridge` now checks that the provided `_to` address is not zero, but not that it is not the address of `crvUSD` on Ethereum mainnet.
2. In `FastBridgeL2`, `allowed_to_bridge` still allows for the given timestamp to be smaller than the current block timestamp.
3. In `OptimismBridger`, the `bridge` function does not check that the given `_token` is a `OptimismMintableERC20` using EIP-165. Curve acknowledges this and answered that EIP-165 check is skipped since it is assumed to be used with only compatible coins.

7.5 Missing and Unused Events

Informational **Version 1** **Code Partially Corrected**

CS-CURVE-FASTBRIDGE-010

The following functions perform important actions but do not emit events:

FastBridgeL2:

- `bridge`

ArbitrumBridger:

- `bridge`

OptimismBridger:

- `bridge`

L2MessengerLZ:

- `__init__`
- `set_fast_bridge_l2`
- `set_gas_limit`
- `initiate_fast_bridge`

FastBridgeVault:

- `__init__`
- `schedule_rug`
- `mint`
- `set_killed` (`SetKilled` is unused)
- `set_fee`

- `set_fee_receiver`
- `recover`

VaultMessengerLZ:

- `set_vault`
- `lzReceive`

Code partially corrected:

All above functions now emit events except for the following:

L2MessengerLZ:

- `__init__` (sets the vault EID but does not emit `SetVaultEid`)
- `initiate_fast_bridge`

FastBridgeVault:

- `schedule_rug`

VaultMessengerLZ:

- `lzReceive`

7.6 Unnecessary Approvals

Informational **Version 1** **Acknowledged**

CS-CURVE-FASTBRIDGE-012

Both the `OptimismBridger` and the `ArbitrumBridger` give approval to the native bridge of each chain before calling respectively `bridgeERC20To` and `outboundTransfer`. However, this approval is not necessary as in each case, the native bridge does not need an allowance to burn tokens from the `Bridger` contract.

Acknowledged:

Curve acknowledges the finding and decided to keep it as the behaviour of the system could be different with custom gateways/bridges.

7.7 Unused Code

Informational **Version 1** **Acknowledged**

CS-CURVE-FASTBRIDGE-013

In `OptimismBridger` the following code is commented and unused:

```
# OPTIMISM_L2_BRIDGE: constant(address) = 0x4200000000000000000000000000000000000010
```

Acknowledged:

Curve acknowledges the unused code and keeps it for convenience.

8 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

8.1 Limits Not Synced Between L1 and L2

Note Version 1

The current implementation allows to fast bridge at max `self.limit` per interval. This limit is enforced on L2. On L1 the maximum that is possible to fast bridge is the amount of crvUSD in the vault.

These are two independent limits with no enforced sync between them.

8.2 Risk Linked to the Fast Bridge

Note Version 1

The fast-bridge mechanism lets users receive crvUSD on Ethereum before the L2 output root passes the challenge window on L1. To enable that early payout, Curve's `ControllerFactory` effectively provides liquidity to the `FastBridgeVault` (by minting unbacked crvUSD) so the vault can honor fast withdrawals while the native bridge transfer is still finalizing.

The `ControllerFactory` covers this interim liquidity (via a debt/debt-ceiling mechanism). If the L2 transaction does NOT produce the intended effect (for example, the transaction is dropped, reverts, or its effects are invalidated by a fault proof or chain reorganization), the vault can be left short of assets because the fast-bridged crvUSD have already been paid out.

That shortfall is economic risk borne by the `ControllerFactory` (and by extension whoever controls its minting/repayment policy), creating potential protocol-level bad debt.

Clarifying failure modes

- Dropped: the L2 sequencer/batcher never includes the transaction in a published batch (or the batch is never posted), so the expected state changes never materialize on L1.
- Reverted: the transaction executes but reverts when verifying the rollup state, so the expected state change does not occur.
- fault proof / dispute / reorganization: a later fault proof or L1 reorg can invalidate a previously accepted L2 batch or change the canonical state produced by the L2, causing previously assumed transfers to be undone.

These are distinct failure modes but have the same consequence for the fast-bridge: the native-bridge transfer that was expected to back the fast payout may not finalize, producing a funding shortfall that the `ControllerFactory` must cover.

The following outlines risks that could lead to this situation:

L2 transaction inclusion / finality risk

- When a user fast-bridges from an L2, the vault may immediately supply crvUSD to the recipient based on a LayerZero message.
- Depending on the confirmation / DVN/ULN settings used by the OApp and endpoint, the message may be processed:

1. While the L2 transaction is still unsafe (e.g., included in a sequencer batch that has not been posted or sufficiently confirmed).
2. While the transaction appears included on L1 but is still within a reorg/fault-proof window.
3. Only after the transaction is finalized with a high confidence of irreversibility.

- In the first two scenarios, the L2 transaction could ultimately be dropped or invalidated, leaving the vault having already paid out fast-bridged crvUSD while the native-bridge transfer is invalid.

L2 transaction output mismatch risk

- Even when a transaction is posted, the posted state or outputs might not match the expected output (e.g., differences between the sequencer's claim and what verifiers gets). In such cases the transaction can revert while the fast-bridge payout has already been made.

Native bridge pause risk

- Optimism documents a pause mechanism for the Ethereum-side bridge: https://docs.optimism.io/stack/security/pause
- If the native bridge is paused, native-bridge receipts to the vault may be delayed or blocked while LayerZero-based fast-bridge messages continue to be processed, creating a liquidity mismatch until the bridge is resumed.
- Arbitrum does not expose the exact same pause primitive, however, bridge upgrades or operator actions that delay L1 settlement can produce similar effects.

Potential impacts and scenarios

The primary impact is vault insolvency: a successful fault proof, dropped batch, or prolonged bridge pause could render the vault illiquid and unable to repay the `ControllerFactory`.

A prolonged pause or operational delay in the native bridge exacerbates exposure because the vault cannot receive the underlying funds needed to cover fast withdrawals.

Mitigations and recommendations

This situation should be closely monitored and managed through a combination of technical and governance measures.

- Conservative confirmation / DVN settings: configure OApp/endpoint (DVN/ULN) confirmations and thresholds conservatively so messages are processed only after sufficient finality.
- Conservative debt ceilings: set and enforce low vault debt ceilings relative to expected native-bridge throughput.
- Tight caps and rate limits: enforce sensible per-day, per-chain limits for fast-bridging to limit peak exposure.
- Emergency controls: ensure fast-bridge minting can be paused quickly and that pause/parameter changes follow governance rules and timelocks.
- Monitoring and alerting: implement real-time monitoring for fast-bridge volumes, bridge confirmations, and unusual activity; trigger automated rate-limit reductions or pauses when thresholds are exceeded.
- As recommended by Optimism, monitor the native-bridge pause contract and take defensive actions when it is paused:

  > If you operate a centralized exchange or third party bridge, you should monitor this contract and pause withdrawals from the Superchain if you see that it has been paused.