CANTINA

Clique Bera Contracts
Security Review

Cantina Managed review by:

Eric Wang, Lead Security Researcher

Riley Holterhus, Lead Security Researcher
RustyRabbit, Security Researcher

February 5, 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	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

Clique is a cryptographic infrastructure project specializing in secure and high-performance off-chain computations for blockchain applications. The company leverages a network of nodes operating within a Trusted Execution Environment (TEE) to perform confidential operations. These computations integrate seamlessly with blockchain smart contracts, ensuring reliability, verifiability, and privacy.

From Jan 10th to Jan 13th the Cantina team conducted a review of clique-bera-contracts on commit hash 13be6727. In addition, a follow up review took place on Jan 28th and Jan 29th of clique-bera-contracts on commit hash f7d0d503; and LayerZero's devtools PR 1184. The team identified a total of **15** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	4	4	0
Low Risk	4	2	2
Gas Optimizations	0	0	0
Informational	7	4	3
Total	**15**	**10**	**5**

The Cantina Managed team reviewed Clique's `clique-bera-contracts` holistically on commit hash 1f73a78b for the initial review, and on commit hash a4d2bf9b for the follow up review, and concluded that all the issues were addressed and no new vulnerabilities were introduced.

3 Findings

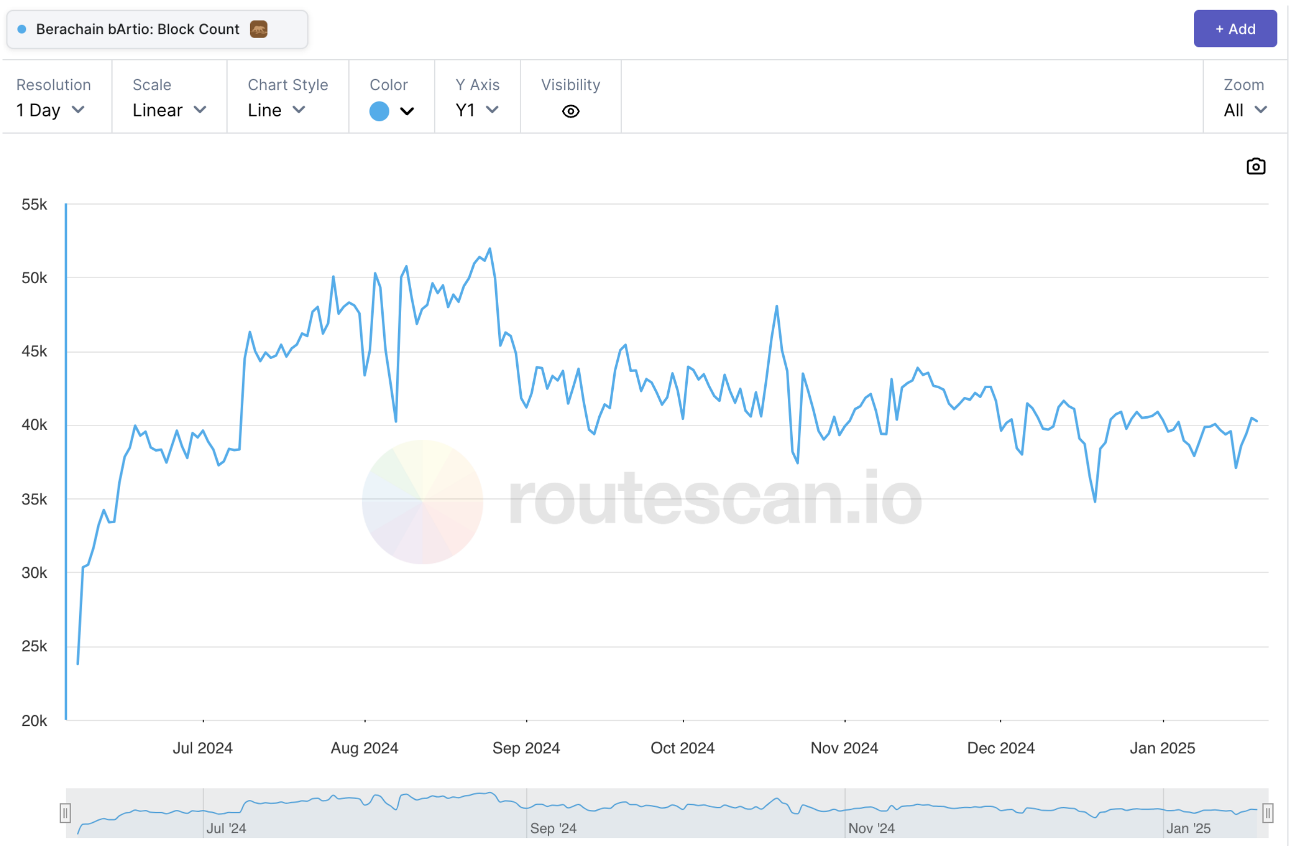
3.1 Medium Risk

3.1.1 Vesting period may not be accurate due to non-constant block time on Berachain

Severity: Medium Risk

Context: StreamingNFT.sol#L54-L55

Description: The vesting duration is defined as `vestingDuration` blocks. Note that on Berachain, block time is not a constant as it depends on the network congestion. See what-do-berachain-s-performance-metrics-look-like docs and glossary#block-time docs. Also, see the graph of the history block count per day for more information:



Since the block time may vary in the future, the estimated vesting period in terms of seconds may likely be inaccurate in the long term. For example, suppose the owner expects the vesting period to be 1 year and sets `vestingDuration = 1 year / T` based on the current average block time `T`. However, after 3 months, Berachain starts to produce blocks faster to reflect the current network congestion. If the trend continues, the vesting period will be reduced to less than 1 year, allowing users to receive their vested funds early.

On the other hand, if the block time increases for any reason, the vesting period will become longer than expected.

Recommendation: In general, it is recommended to calculate the vesting or reward periods using block timestamps on chains with a non-constant block time. Consider adopting this approach to avoid changes in block time from affecting the vesting period.

Clique: Fixed in commit 4194f992 for the follow-up review.

Cantina Managed: Verified. The `StreamingNFT` contract now uses block timestamps.

3.1.2 Potential issues of determining if a transaction is sent by the paymaster

Severity: Medium Risk

Context: StreamingNFT.sol#L82-L87, StreamingNFT.sol#L105

Description: The `createStream()` and `createBatchStream()` functions in `StreamingNFT`, and the `claim()` function in `Distributor1` use `tx.origin != onbehalfOf` to determine if the transaction is sent by the paymaster. This approach is not recommended since it has several downsides:

1. Consider a scenario where the owner of the NFT is a smart contract wallet, and the owner of the wallet calls `createStream()`. The wallet owner won't lose funds because the gas fees are sent to him. However, this may cause poor UX if the owner wants to keep all the funds in the smart contract wallet as they need to transfer them back.
2. The design discourages the integration with ERC-4337 accounts. The owner of ERC-4337 accounts creates user operations off-chain and sends them to bundlers. Bundlers will then execute the operations on-chain and get compensated for the spent gas fees. However, in this scenario, `tx.origin`, the bundler, will get the compensation for gas fees again. Effectively, the users are paying the bundler double gas fees.
3. The design allows anyone to act as a paymaster, create streams for others, and get compensated for the gas fee. The gas fees are set manually by the contract owner. The price of the native token of Berachain may fluctuate, and the contract owner has to ensure that the gas fees are updated accordingly. In case the gas fee fails to be updated timely, arbitragers may extract value from NFT holders by creating the stream for them, causing the users to pay more on gas fees than needed.

Recommendation: Given that users creating streams for others is not an expected use case. Consider adopting the following approach: The contract owner can set a `paymaster` state variable in the contract. The functions check if `tx.origin == paymaster` is true. If so, transfer tokens to `paymaster`. Otherwise, transfer the entire amount to `onbehalfOf`.

Clique: Fixed in commit 04c52bd.

Cantina Managed: Verified. The contract implemented a mapping `isPayMaster` to determine if the `tx.origin` is a paymaster. Besides that, access control on the `createStream()` function is implemented so that either the `tx.origin` should be a paymaster or the same as `onbehalfOf`. The latter condition prevents a regular user from creating streams for others.

Similar access control is also implemented in the `_claimVestedRewards()` function so that only the EOA owner of the NFT can claim the rewards. The client has clarified that the airdrop system only supports EOA owners of NFTs and not smart contract owners.

3.1.3 WrappedNFT cannot receive ERC-1155 tokens from OpenSea Shared Storefront

Severity: Medium Risk

Context: WrappedNFT.sol#L50

Description: According to the provided documentation, the `WrappedNFT` contract is designed to receive ERC-1155 tokens from the `OpenSea Shared Storefront` contract on Ethereum and wrap them to ERC-721 tokens. The ERC-1155 tokens to be bridged to Berachain are those in the Bong Bears collection.

The `WrappedNFT` contract requires the received ERC-1155 token's ID to be less than `1 << 128`. However, by observation, we can see that the token IDs in the collection are in the format of `creator (160 bits) || index (56 bits) || supply cap (40 bits)`.

For example, Bong Bear #57 has an ID of `66075445032688988859229341194671037535804503065310441849644897923713034354689` or `0x921560673f20465c118072ff3a70d0057096c1230000000000003a0000000001` in hex, where `0x921560673f20465c118072ff3a70d0057096c123` is the creator of the token, `0x3a` is the index for this token, and `1` is the supply cap for this token.

Since the first 160 bits of the token IDs are the creator address, the IDs will be larger than `1 << 128`. As a result, the `WrappedNFT` contract cannot receive or bridge them.

Recommendation: Consider creating one `WrappedNFT` contract per collection, where the `creator` bits are the same among the tokens. Add a check in the `_wrap()` function so that only tokens with an ID belonging to the known `creator` are accepted. Also, validate that the `index` of the token IDs is within the range of `[0x01, 0x6c]` (inclusive), in case there are other tokens with the same creator but not belonging to Bong Bears.

Remove the requirement that IDs should be less than `1 << 128`, and construct the ERC-721 token ID by discarding the `creator` bits in the original ID and combining it with the corresponding `nextId` value as the lower bits.

Clique: Fixed in commit 1f73a78.

Cantina Managed: Verified. Note that the index of the token IDs is not validated in the contract since all the ERC-1155 tokens created by `0x921560673f20465c118072ff3a70d0057096c123` belong to the Bong Bear collection at the time of writing. If the creator creates new ERC-1155 tokens outside the Bong Bear collection in the future, consider verifying the `index` values as well.

3.1.4 Missing blacklist check in `createBatchStream()`

Severity: Medium Risk

Context: StreamingNFT.sol#L128-L172

Description: A blacklist for token ids has been added to the `createStream()` function of the `StreamingNFT` contract. However, the `createBatchStream()` function, which performs the same logic in a loop over multiple ids, does not include this blacklist check. As a result, the blacklist in `createStream()` can always be bypassed by calling `createBatchStream()` with a singleton array.

Recommendation: Ensure that the blacklist applies to all functions by adding the blacklist check to `createBatchStream()`. For example:

```
  function createBatchStream(uint256[] calldata tokenIds, address onBehalfOfOwner)
      external
      nonReentrant
      whenNotPaused
  {
      // ...
      for (uint256 i = 0; i < tokenIds.length; i++) {
          require(claimedTimestamp[tokenIds[i]] == 0, "Stream already created");
+         require(!isBlacklistedTokenId[tokenIds[i]], "TokenId is blacklisted");
          claimedTimestamp[tokenIds[i]] = cliffEndTimestamp;

          // ...
      }

      // ...
  }
```

Clique: Fixed in PR 3.

Cantina Managed: Verified.

3.2 Low Risk

3.2.1 Users can create streams before the vesting period starts

Severity: Low Risk

Context: StreamingNFT.sol#L73

Description: The `createStream()` function of `StreamingNFT` does not check that `block.number >= vestingStartBlock`. However, the same check is presented in a similar function, `createBatchStream()`. Without this check, users can create streams and receive the unlock amount anytime before the start of the vesting period. The test `testFailCreateStreamBeforeStart()` fails because of the absence of the check.

Recommendation: Consider clarifying whether users are allowed to create streams before the vesting period. If not, consider adding the missing. Otherwise, consider removing the check in the `createBatchStream()` function if users are allowed to do so.

Clique: Removed the check in the `createBatchStream()` function in commit edd7128.

Cantina Managed: Verified. Note that NFT holders can create streams anytime, e.g., before or after the cliff release time or even after the vesting period ends. The time when a user creates a stream does not affect the total received tokens at the end since the vesting period of all the NFTs is synchronized.

3.2.2 Missing check on the vesting start block when creating streams

Severity: Low Risk

Context: StreamingNFT.sol#L76

Description: When a user creates a stream with the `createStream()` function, there is no check to ensure that the owner has set `vestingStartBlock` variable. The variable may remain uninitialized as 0.

A better approach is to include a `vestingStartBlock != 0` check in the `createStream()` function. There is a `whenNotPaused` modifier, so the creation of streams is paused after this contract is deployed until the owner sets the vesting start block. However, if the owner accidentally calls `unpause()` before setting the start block, creating streams with a start block of 0 would be possible.

A start block of 0 would cause the user's rewards to become unclaimable, as the `getClaimableRewards()` function ensures the last claimed block is not 0.

Recommendation: Consider checking `vestingStartBlock != 0` in the `createStream()` and `createBatch-Stream()` functions.

Clique: Acknowledged.

Cantina Managed: Acknowledged. Note that `cliffEndBlock` (named `vestingStartBlock` before) is not checked in either the `createStream()` or `createBatchStream()` functions. To prevent the above scenario, the owner should ensure that they do not accidentally call `unpause()` before setting `cliffEndBlock`.

3.2.3 `instantAmmount` can be locked when the NFT is held by a non-compatible contract

Severity: Low Risk

Context: StreamingNFT.sol#L73

Description: The `createStream()` function is public and allows anyone to create the stream on behalf of any holder of a supported NFT. In doing so the `instantAmount` of tokens is immediately transferred to the NFT owner.

If at the launch of the airdrop such an NFT happens to be held by a contract that is unable to deal with the unexpected increase in tokens they may be locked in the contract. If it's a pool type contract they would be donated to the pool instead of attributed to the underlying owner of the NFT. An attacker could use this to increase the value of their pool tokens, especially if multiple qualifying NFTs are held in the pool.

Recommendation: Consider limiting the access control of the `createstream()` function to the actual owner of the NFT. The paymaster can safely be allowed to create the stream if the NFT owner is an EOA.

Clique: Acknowledged. The streaming contract is designed to support EOA holders only.

Cantina Managed: Acknowledged. Consider clarifying in public documentation or website/front-end so that users can be aware that the system only supports EOA holders instead of smart contract wallets or other contracts.

3.2.4 Rounding causes lost of dust amounts on each `claimReward`

Severity: Low Risk

Context: StreamingNFT.sol#L208

Description: The `claimableAmount` calculation is based on a proportional formula using `elapsedBlocks / vestingDuration`. However, since stream claims are tracked by block number rather than actual claimed amounts, rounding discrepancies in this division are not properly accounted for. This creates an inconsistency where users who claim frequently may receive a different total amount compared to users who claim only once at the stream's conclusion. While this discrepancy may be negligible for tokens with 18 decimal places and relatively lower value (unlike WBTC), it represents a potential accounting imprecision in the system.

Recommendation: To ensure precise and consistent token distribution, we recommend implementing a claimed amount tracking system instead of relying on the last claimed block number. This modification would:

1. Guarantee accurate final disbursement amounts for all users regardless of their claiming frequency.
2. Eliminate rounding-related discrepancies in the vesting calculations.
3. Provide better accounting transparency and reliability.

This approach would ensure mathematical accuracy and maintain fairness across different claiming patterns.

Clique: Fixed by tracing `claimedAmount`. Fixed in commit 1637e50.

Cantina Managed: Verified.

3.3 Informational

3.3.1 Minor code and specification improvements

Severity: Informational

Context: Distributor1.sol#L43-L44, Distributor1.sol#L58-L60, StreamingNFT.sol#L15, StreamingNFT.sol#L130-L132, Transferable.sol#L10

Description: 1. Distributor1.sol#L43-L44: Unused errors. It is best practice to check that the constructor parameter, `signer`, is not `address(0)`. Consider adding the check or removing the unused errors.

1. Distributor1.sol#L58-L60: Consider emitting events for important state changes or configuration updates. This would allow off-chain components to monitor or keep track of the changes more easily. Same for:
   - Distributor1.sol#L64-L66.
   - Distributor1.sol#L70-L72.
   - Distributor1.sol#L76-L79.
   - StreamingNFT.sol#L61-L63.
2. StreamingNFT.sol: Consider documenting the following to make users aware: If user A transfers their NFT to user B without claiming the rewards first, those unclaimed rewards will then be claimable by user B. This contract does not automatically claim rewards for the old owner before the transfer of the NFT.
3. StreamingNFT.sol#L15: Consider replacing `Ownable` with `Ownable2Step` to prevent accidentally transferring the contract's ownership to uncontrolled addresses.
4. StreamingNFT.sol#L130-L132: Since Solidity 0.8.22, the optimizer will omit overflow checks on the loop increments if the `for` loop is a simple counter loop (see Solidity 0.8.22 release announcement). Since the project is using Solidity 0.8.25, and this `for` loop is a simple counter loop, consider changing it to `for (uint256 i = 0; i < tokenIds.length; i++) { ... }` to benefit from compiler optimizations, while maintaining better code readability. Same for:
   - StreamingNFT.sol#L158-L160.
   - ClaimBatchProcessor.sol#L39-L41.
   - ClaimBatchProcessor.sol#L62-L64.
   - ClaimBatchProcessor.sol#L71-L73.
5. Transferable.sol#L10: The `token` state variable can be declared as `immutable` since its value does not change after the contract is deployed. This would reduce the gas cost of each read of the variable.

Clique: Partially fixed in commit 6ee97b3.

Cantina Managed: Note that 1. is partially fixed and 6. is acknowledged. 2., 4., and 5. are verified.

3.3.2 Handle ERC-20 token transfers with the SafeERC20 library

Severity: Informational

Context: Transferable.sol#L31

Description: The `transfer()` function of the `Transferable` contract transfers native tokens or ERC-20 tokens to a recipient. In the case of ERC-20 tokens, the following code is executed:

```
require(IERC20(token).transfer(recipient, amount), "ERC20 transfer failed");
```

It requires the ERC-20 token to comply with the ERC-20 standard and returns `true` when the transfer succeeds. However, there are known ERC-20 tokens that do not comply with the standard and do not return a boolean. This `transfer()` function fails to handle those non-standard ERC-20 tokens and always reverts.

Recommendation: Consider using the SafeERC20 library to handle ERC-20 token transfers for better compatibility with different ERC-20 token implementations. Otherwise, clearly document that the token must comply with the ERC-20 standard.

Clique: Acknowledged. ERC20 is only for test purpose.

Cantina Managed: Acknowledged.

3.3.3 VestedRewards and instantAmount can be immutables

Severity: Informational

Context: StreamingNFT.sol#L190

Description: `vestedRewards` and `instantAmount` on the fly when streams are created and on each reward claim. They are however calculated based purely on `immutables` and as such can also be calculated as immutables in the constructor which saves gas and improves readability.

Recommendation: Consider calculating `vestedRewards` and `instantAmount` as immutables in the constructor.

Clique: Fixed in commit 1637e50.

Cantina Managed: Verified.

3.3.4 Fee must be less than the `instantAmount`

Severity: Informational

Context: StreamingNFT.sol#L85

Description: During stream creation by the Paymaster, the `fee` amount is deducted from the stream's `instantAmount`. If the `fee` exceeds the `instantAmount`, the transaction will revert when executed by the Paymaster. While this issue can be temporarily resolved by setting the `fee` equal to the `instantAmount`, this approach may not be optimal for all use cases.

Recommendation: Consider deducting the fees from the total stream amount rather than solely from the `instantAmount`.

Clique: Acknowledged, unlikely happenning.

Cantina Managed: Acknowledged.

3.3.5 Redundant check in `onERC1155Received()`

Severity: Informational

Context: WrappedNFT.sol#L29-L39

Description: In the `WrappedNFT` contract, ERC1155 tokens are transferred in, and an ERC721 is minted to the user in return. This logic is handled by the `onERC1155Received()` and `onERC1155BatchReceived()` functions, which are triggered when the ERC1155 contract is executing a `safeTransferFrom()` or a `safeBatchTransferFrom()`. To prevent unauthorized calls, the `WrappedNFT` functions should verify that the `msg.sender` is indeed the ERC1155 contract.

In the current codebase, the `onERC1155Received()` function does this `msg.sender` check directly:

```
function onERC1155Received(/* ... */) /* ... */ {
    require(msg.sender == oriToken, "Invalid sender");
    // ...
}
```

On the other hand, the `onERC1155BatchReceived()` does not check the `msg.sender`, and instead relies on the `_wrap()` function (which is used in both cases) to do the check:

```
function _wrap(/* ... */) /* ... */ {
    require(msg.sender == oriToken, "Invalid sender");
    // ...
}
```

So, the `onERC1155Received()` function is doing a redundant check, since the `msg.sender` is already checked in the `_wrap()` function. Moreover, the `onERC1155BatchReceived()` is checking the `msg.sender` multiple times, since each individual `_wrap()` call will do the check.

Recommendation: Consider streamlining this logic. To ensure the `msg.sender` check is performed the least number of times, the check can be removed from `_wrap()` and added to `onERC1155BatchReceived()`. For example:

```diff
  function onERC1155BatchReceived(
      address,
      address from,
      uint256[] calldata ids,
      uint256[] calldata values,
      bytes calldata
  ) external nonReentrant whenNotPaused returns (bytes4) {
+     require(msg.sender == oriToken, "Invalid sender");
      for (uint256 i = 0; i < ids.length; i++) {
          _wrap(from, ids[i], values[i]);
      }
      return this.onERC1155BatchReceived.selector;
  }

  function _wrap(address from, uint256 id, uint256 value) internal {
-     require(msg.sender == oriToken, "Invalid sender");

      require(value == 1, "Value must equal 1");

      uint160 tokenCreator = uint160(id >> 96);
      require(tokenCreator == creator, "Invalid creator");

      require(uint40(id) == uint40(1), "Total supply must be 1");

      _mint(from, id);

      emit Mint(from, id);
      emit Wrap(msg.sender, from, id, value);
  }
```

Clique: Fixed in PR 4.

Cantina Managed: Verified.

3.3.6 LayerZero receive function not paused in `WrappedNFT`

Severity: Informational

Context: WrappedNFT.sol#L93

Description: The `_buildMsgAndOptions()` inherited from Layer Zero's `ONFT721Core` is overwritten to pause sending messages over LayerZero via the `whenNotPaused`modifier when the contract is paused.

```
function _buildMsgAndOptions(SendParam calldata _sendParam) override whenNotPaused
        returns (bytes memory message, bytes memory options)
    {
        return super._buildMsgAndOptions(_sendParam);
    }
```

Receiving messages however is not paused within the contract.

Although the intent is to only configure LayerZero to function in one direction (Ethereum to Berachain), if at any point this is changed to be bi-directional, receiving messages should also be paused.

Recommendation: Consider overriding `ONFTCore`'s `_lzReceive()` in the same way to pause receiving messages from Berachain when the contract is paused regardless of the underlying LayerZero configuration.

Clique: The plan is to not implement this change and to rely on the LayerZero configuration to prevent messages from being received on Ethereum.

Cantina Managed: Acknowledged.

3.3.7 One-way bridging considerations

Severity: Informational

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

Description: The `BeraNftAdapter`, `BeraNft`, and `WrappedNFT` contracts are meant to facilitate the bridging of a specific set of ERC721 and ERC1155 tokens from Ethereum mainnet to Berachain. After discussing with the team, it was clarified that this bridging is intended to be strictly one-way, meaning there is no intended support for bridging back from Berachain to Ethereum mainnet.

However, note that with the most common deployment approach for ONFT contracts, two-way bridging is the default. A specific configuration would therefore be required to enforce one-way bridging.

This default two-way behavior exists because the `OAppCore` contract (which the ONFT contracts inherit) controls both sending and receiving messages through its `peers` mapping. The contract can only send messages if its `peers` mapping is set, and it can only receive messages using the same mapping. As a result, receiving messages automatically enables message sending, and vice versa.

So to implement one-way bridging, a configuration change beyond the `peers` mapping would be required. One hypothetical solution is to use the endpoint's `setSendLibrary()` function on Berachain to specify a library that blocks outgoing messages, while using the `setReceiveLibrary()` function on Ethereum mainnet to prevent incoming messages.

Recommendation: Consider the configuration options for enforcing one-way bridging. Beyond just configuration choices, another approach is to create modified smart contracts that override the sending and receiving functions to revert.

Clique: To accomplish one-way bridging, the "Dead DVN" configuration will be used. This is a configuration in LayerZero that always reverts the relevant function calls.

Cantina Managed: Verified. Using the "Dead DVN" configuration for sending messages on Berachain and for receiving messages on Ethereum will result in the intended one-way bridging behavior.