OpenZeppelin | security

Optimism Bedrock and Periphery Audit

September 2, 2022

This security assessment was prepared by **OpenZeppelin**, protecting the open economy.

Table of Contents

Summary

Type	L1/L2 Bridge
Timeline	From 2022-07-18 To 2022-08-12
Languages	Solidity

Total Issues	34 (18 resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	8 (5 resolved)
Low Severity Issues	14 (7 resolved)
Notes & Additional Information	12 (6 resolved)

Scope

We audited the "ethereum-optimism/optimism" repository at the "93d3bd411a8ae75702539ac9c5fe00bad21d4104" commit.

In scope were the following contracts:

- `packages/contracts/contracts/chugsplash/L1ChugSplashProxy.sol`
- `packages/contracts/contracts/libraries/bridge/CrossDomainEnabled.sol`
- `packages/contracts/contracts/libraries/constants/Lib_PredeployAddresses.sol`
- `packages/contracts-bedrock/contracts/L1/L1CrossDomainMessenger.sol`
- `packages/contracts-bedrock/contracts/L1/L1StandardBridge.sol`
- `packages/contracts-bedrock/contracts/L1/L2OutputOracle.sol`
- `packages/contracts-bedrock/contracts/L1/OptimismPortal.sol`
- `packages/contracts-bedrock/contracts/L1/ResourceMetering.sol`
- `packages/contracts-bedrock/contracts/L2/L2CrossDomainMessenger.sol`
- `packages/contracts-bedrock/contracts/L2/L2StandardBridge.sol`
- `packages/contracts-bedrock/contracts/L2/L2ToL1MessagePasser.sol`
- `packages/contracts-bedrock/contracts/libraries/Hashing.sol`
- `packages/contracts-bedrock/contracts/universal/CrossDomainMessenger.sol`
- `packages/contracts-bedrock/contracts/universal/Proxy.sol`
- `packages/contracts-bedrock/contracts/universal/ProxyAdmin.sol`
- `packages/contracts-bedrock/contracts/universal/Semver.sol`
- `packages/contracts-bedrock/contracts/universal/StandardBridge.sol`
- `packages/contracts-periphery/contracts/L1/L1ERC721Bridge.sol`
- `packages/contracts-periphery/contracts/L2/L2ERC721Bridge.sol`

System Overview

This audit covers a newly added ERC721 bridge as well as some updates made to the standard bridge.

The ERC721 bridge is a pair of smart contracts that accept deposits of ERC721 tokens on L1 and mints a representation of them on L2. The bridge uses the underlying Optimism cross-domain messaging system to relay transactions between L1 and L2. In order to use the ERC721 bridge, the assets have to be initially minted on L1 and bridged over to L2. It does not support L2 native ERC721 bridging over to L1.

The standard bridge supports both ETH and ERC20 tokens. It accepts deposits on L1 and mints representation of these tokens on L2. It supports L1 natively minted tokens that are bridged to L2, and L2 native tokens bridged to L1 if they are `OptimismMintable`. These bridges are permissionless, meaning users can bridge any supported assets. However, not all asset types are natively supported (e.g., tokens with transfer fees are explicitly unsupported). It is up to users to ensure that any assets they intend to bridge are legitimate and supported as failing to do so could lead to loss of funds.

The system also fully supports bridging ETH between layers either via the standard bridge or even at a layer beneath that using the Optimism Portal contract.

Security assumptions

The system is designed with some trusted entities and privileged roles. We assume these are trustworthy during our audit, however they are still worth pointing out.

- Some of the contracts e.g. standard bridges, ERC721 bridges, cross-domain messengers, and the Optimism portal are designed to be upgradable. A trusted admin is capable of upgrading these contracts and all logic involved.
- Cross-chain messages are relayed by off-chain nodes which play a critical role in the system. How these are operated is out of the scope of this audit, however any malicious acts, delays or failures could impact the overall bridging process.
- The cross-domain messenger contracts on both L1 and L2 can have message relaying paused by their owner.
- The owner of the `L2OutputOracle` contract can delete block proposals submitted by node runners.

Findings

Here we present our findings.

Medium Severity

M-01 Asymmetric failure behavior of ERC721 bridges

The `L1ERC721Bridge` and `L2ERC721Bridge` contracts are not symmetric in the way they deal with failing cross-domain token transfers.

Specifically, if a layer one (L1) "native" ERC721 token is sent from the `L1ERC721Bridge` to the `L2ERC721Bridge` on the other domain, the layer two (L2) bridge checks that the local token supports the `IOptimismMintableERC721` interface and that it identifies itself as being mapped to the expected remote token. If either of these conditions do not hold, then an "auto withdrawal" message is sent back to L1 to facilitate refunding the ERC721 token to the original sender on L1.

This fail-safe behavior is not present within the `L1ERC721Bridge` contract's `finalizeBridgeERC721` function. An `IOptimismMintableERC721` token could be created natively on L2, and it would pass all checks required to initiate a bridge to L1. However, upon arrival at the L1 bridge, there is no support for `IOptimismMintableERC721` tokens. Since the token was never deposited into the L1 bridge, and given that it is L2 native, the finalization transaction would always fail.

A cross-chain transfer of an L2 "native" ERC721 that fails during transfer finalization on L1 does not automatically create a withdrawal message to send the token back to the original sender on L2. Hence, such transfers would effectively leave the ERC721 token completely inaccessible on either L1 or L2.

Consider making the bridge behavior more symmetric by adding a mechanism to automatically refund ERC721 tokens bridged from L2 to L1 when they encounter permanent conditions that prevent their successful transfer. Alternatively, consider heavily documenting the lack of symmetry to avoid user confusion and the unintentional locking of assets.

Update: Fixed by commits `0a0eeeba1bd7d5b30bd2e9b6fad5792436d69f42` and `efc31f3b2131d6d166bf7ebe5c206e532ec67cac` in pull request 20. The `L1ERC721Bridge` has been updated to refund assets if a bridge transaction fails to finalize.

M-02 Confusing deprecated `OptimismMintableERC20` interface considerations

In the `StandardBridge` contract, the internal `_isOptimismMintableERC20` function is meant to check that a provided token is an `OptimismMintableERC20`.

The codebase also seems to indicate that, going forward, an `OptimismMintableERC20` token may come in two variations, the `IL1Token` type (which exposes the `l1Token` function which is being depreciated) and the `IRemoteToken` type which exposes the `remoteToken` function that replaces the `l1Token` function.

Currently, the `_isOptimismMintableERC20` function only checks for the deprecated interface. This means all "new" style tokens that only conform to the `IRemoteToken` interface would not be usable with the StandardBridge. Additionally, the `_isCorrectTokenPair` function only works with `IL1Token` interface type tokens.

The `OptimismMintableERC20` contract supports both interfaces, indicating that the legacy `IL1Token` interface should continue to be available.

To clarify the `IL1Token` interface deprecation status, consider adding more extensive documentation that covers what interfaces external contracts should support. To support tokens that only implement the `IRemoteToken` interface, if permitted, consider checking for this interface type in addition to the deprecated interface type.

Update: Fixed by commits `8257593f8e1f545151c66b2677e356dd09ed0060` and `3bef594f30743613a14fdd4c515c2bd010f313a9` in pull request 4.

M-03 ETH can be sent to undeliverable recipient

The `StandardBridge` contract is the base contract for the domain-specific standard bridge contracts. The standard bridge contracts facilitate the cross-domain transmission of ETH and ERC20 tokens.

The `StandardBridge` contract does not validate within the `bridgeETHTo` function that the provided `_to` address is not the bridge contract on the other domain (`otherBridge`).

However, when it arrives at the standard bridge on the other domain the `finalizeBridgeETH` function requires that `_to != (current domain bridge` `address)`. As a result, any of the ETH sent in such a transaction would be undeliverable to the standard bridge, and irretrievable by the sender.

Consider disallowing ETH to be sent across the bridge if the `_to` address is the address of the other bridge. Alternatively, consider documenting the result of such a transaction.

Update: Fixed in commit `41ae72d0007ec10a1260603d40eadb4f55dc8a95` in pull request 27 where extra documentation has been added to warn users about such transactions.

M-04 Unsatisfiable `gasLimit` values could lead to frozen assets

Throughout the codebase whenever a cross-domain transaction is initiated, the initiating function generally accepts a `gasLimit` (or similarly named) value. This value is meant to indicate the minimum amount of gas that should be provided to yield a successful execution of the transaction on the target domain. This is necessarily a user provided value, because the exact execution costs of a message on the target domain are not programmatically computable.

However, there is no explicitly enforced upper bound on this value at the time a cross-domain transaction is being initiated. Where there are bounds placed on this value, they are implicit. For example, the `metering` modifier applied to the `depositTransaction` function of the `OptimismPortal` contract sets an upper bound on the `_gasLimit` for all cross-domain transactions from the portal's domain (L1) to the target domain (L2). Transactions that pass through a `CrossDomainMessenger` contract also have an implicit upper bound on `_gasLimit` as a result of the `baseGas` function call and the fact that it will revert if the minimum gas limit is too large.

However, for lower-level transactions that are flowing in the other direction, from L2 *to* L1, there is no upper bound, implicit or otherwise, on the `gasLimit` value.

This is problematic because it can result in cross-domain transactions being created that are unintentionally unexecutable on L1. If a `gasLimit` is specified that is larger than the number of ETH in existence, for instance, then it could never pass the finalization check requiring that at least `gasLimit` amount of gas be made available for transaction execution.

In practice, a much smaller `gasLimit` could cause transaction finalization issues for most users because they would not have the funds to satisfy excessively large gas requirements.

Consider setting explicit upper bounds on the user-provided `gasLimit` values for all cross-domain transactions. Alternatively, consider modifying the finalization mechanism so that is better capable of handling unsatisfiable `gasLimit` values.

Update: Not fixed. The Optimism team's response:

> *We have not yet encountered this issue while operating our existing bridges, and will decline to make this change in favor of reduced complexity.*

M-05 Incomplete backwards compatibility

The standard bridge contracts are designed to maintain compatibility with contracts that were designed to integrate with previous implementations of the bridge. However the `L2StandardBridge` contract does not emit a `DepositFailed` event when a `finalizeDeposit` call fails. Instead, it always emits a `DepositFinalized` event. It emits a *different*, albeit similarly named, `ERC20BridgeFailed` event only if the child call to `finalizeBridgeERC20` fails. This could be misleading for legacy off-chain applications monitoring bridge activity.

To fully maintain backwards compatibility, consider emitting a `DepositFailed` event when a `finalizeDeposit` call fails.

Update: Fixed in commit `730a9c74496aaf1dd46953719f8a5c337e94c225` in pull request 21.

M-06 Lack of input validation

There is a general lack of input validation throughout the codebase, especially in initializers and constructors. Some examples are:

- In the `L1ERC721Bridge` and `L2ERC721Bridge` contracts, the `_messenger` and `_otherBridge` address inputs are not subject to any validation in the respective contract's `constructor` nor in its `initialize` function.
- In the `L1StandardBridge` contract, the `_messenger` address input is not subject to any validation in the `constructor`, the `initialize` function, nor in the inherited `__StandardBridge_init` function.
- In the `L2StandardBridge` contract, the `_otherBridge` address input is not subject to any validation in the `constructor`, the `initialize` function, nor in the inherited `__StandardBridge_init` function.

- In the `L2OutputOracle` contract, the majority of the arguments to the `constructor` and to the `initialize` function are not subject to any validation.
- In the `OptimismPortal` contract, neither the `_l2Oracle` address input nor the `_finalizationPeriodSeconds` integer input is subject to any validation in the `constructor`.

Additionally, several of the functions that initiate cross-domain transfers allow zero-value inputs. For instance:

- In the `StandardBridge` contract, neither the `bridgeETH` nor the `bridgeETHTo` functions validate that user-provided inputs are non-zero. In the case of the latter function, a zero-value `_to` address does not seem like it should be valid input, for example.
- In the `L1ERC721Bridge` contract, the `bridgeERC721` and `bridgeERC721To` functions do not validate for non-zero address inputs. This is potentially problematic for the `_to` and `_remoteToken` address inputs, for example.

These lists are non-exhaustive; the lack of input validation throughout the codebase is extensive.

The lack of validation on user-controlled parameters may result in erroneous or failing transactions and could lead to the unintentional loss of user assets. Note also that some user interfaces may default to sending null parameters if none are specified and this could be particularly problematic for users.

To avoid the potential for erroneous values to result in unexpected behaviors or wasted gas, consider adding input validation for all user-controlled input, including administrator-only functions.

Update: Partially fixed in commit `3505f204f84b64acf5bcd7a1518e5e0000174f4b` in pull request 20. No relevant changes were made to the `L2OutputOracle` and `OptimismPortal` contracts cited in issue.

M-07 `CrossDomainMessenger` allows sending messages to unrelayable addresses

The `CrossDomainMessenger` contract allows anyone to send a message to any `_target` address via the `sendMessage` function. There are no restrictions on the `_target` at the point of cross-domain message initiation. However, there are restrictions on that `_target` address at the point of message finalization. This can lead to unrelayable transactions.

For instance, a valid cross-domain message can be created on layer one (L1) with a `_target` address that corresponds to the other messenger address (`otherMessenger`) on layer 2 (L2). After this message makes its way over to L2, the L2 cross-domain messenger will attempt to relay the message. However, the transaction will always revert because the message's `_target` address is the L2 cross-domain messenger itself which is a blocked system address on L2.

The same scenario will be true of any other system address set for the domain, including `Predeploys.L2_TO_L1_MESSAGE_PASSER` for L2 and the cross domain messenger contract for L1.

Consider disallowing cross-domain messages to be created if the `_target` address is an unrelayable address on the other domain. If this is not possible, then consider auto-refunding these kinds of unrelayable transactions or heavily documenting that such messages will be unrelayable.

Update: Fixed in commit `5e2f6b3ba642af68919a79b52eef15e6be2517ce` in pull request 27 where extra documentation has been added to warn users about this behavior.

M-08 Upgradeability inconsistencies

Throughout the codebase, many of the contracts are meant to be upgradeable. This is accomplished via means of proxy contracts in conjunction with proxy admin contract(s). However, the upgradeable contracts throughout the codebase are not consistent about reserving storage slots for future upgrades. In particular, the use of storage "gaps" to reserve storage slots and to avoid potential storage slot collisions in the event that contract storage requirements change in the future is inconsistent. Some top-level upgradeable contracts and, more importantly, the contracts they inherit from, safely reserve storage slots via "gap" arrays while others do not. This is problematic because inconsistency around the storage layout of upgradeable contracts can make upgrades more error prone and can lead to missteps that may result in unpredictable, undesirable outcomes after the contracts are upgraded.

Several contracts that are inherited by upgradeable contracts do not reserve any storage space for their own future upgrades. For instance, the `CrossDomainMessenger`, `StandardBridge`, and `CrossDomainEnabled` contracts do not reserve storage for upgrades. On the other hand, the `ResourceMetering` contract *does* reserve storage for upgrades.

Most of the top-level contracts (contracts which no other contracts currently inherit from) do not reserve storage gaps. This can be safe if the upgrade strategy will only ever add additional storage variables to these contracts. However, this is inconsistent throughout the codebase. The `OptimismPortal` contract is one such contract that *does* use a storage gap.

Consider standardizing the use of storage gaps in all contracts that may be upgraded, *especially* contracts that may be upgraded and are inherited by other contracts. Further, consider more thoroughly documenting the intended upgrade process, which contracts are meant to be upgradable and which are not, and any assumptions about the storage layouts of contracts that are meant to be upgradable.

Update: Partially fixed in commit `def7e8942739c2a78192246ff196ead158797cf3` in pull request 23. No relevant changes made to the `CrossDomainEnabled` contract cited in the issue.

Low Severity

L-01 Documentation could be improved

Certain parts of the deposit documentation could be improved in order to better clarify the deposit process.

For example, in the `Execution` section of the document, when a new EVM call frame targeting the `to` address is created, the CALLER and ORIGIN terms should be clarified as being synonymous for `msg.sender` and `tx.origin`, respectively. The document should also clarify that because of that behavior, any `tx.origin == msg.sender` checks would not be checking that an EOA caller during a deposit transaction. Instead, the check could only be useful for identifying the first call in the L2 deposit transaction.

To favor explicitness and help users understand the transaction bridging process better, consider being more verbose in the documentation.

Update: Fixed in commit `25f3e27d044f90bc71124ad0d3581b99f6d04dcd` in pull request 27 where additional clarifying documentation has been added.

L-02 Multiple OpenZeppelin contracts versions in use

Throughout the codebase there are different versions of OpenZeppelin contracts being used. For example, the `contracts-periphery` folder uses 4.6.0 while the `contracts` folder uses 4.3.2 and `contracts-bedrock` folder uses 4.5.0 and 4.5.2.

To avoid unexpected behaviors and to increase the overall consistency of the codebase, consider updating the codebase to use the latest OpenZeppelin contracts.

Update: Partially fixed in commit `1ff3e03380d1f7e200ac44eec28b8b3e15f12a1a` in pull request 22. The OpenZeppelin contracts version was updated only for the `bedrock` contracts.

L-03 Not inheriting supported interfaces

The `OptimismMintableERC20` contract exposes support for three interfaces via the `supportsInterface` function in accordance with the ERC165 standard.

The supported interfaces are `IERC165`, `IL1Token` and `IRemoteToken`. However, the contract does not inherit any of these interfaces.

In favor of explicitness, to increase the readability of the codebase, and to benefit from compiler-level error checking, consider updating the `OptimismMintableERC20` contract so that it inherits from all of the interfaces it supports.

Update: Not fixed. The Optimism team's response:

> *These interfaces were created for the specific purpose of making the interface matching code cleaner, and we do not feel that inheriting will increase clarity for users.*

L-04 Standard Bridge does not support tokens with transfer fees

The `StandardBridge` contract is designed without support for any tokens with a transfer fee. In fact, a relevant brief disclaimer appears in the `L2StandardBridge` contract, but it is absent from the `L1StandardBridge` contract.

Any bridging transactions involving these unsupported tokens will not revert. Instead, they will potentially put the bridge in an under-collateralized position. The undercollateralization would disproportionately impact the last withdrawing user/users, who would essentially pay all of the transfer fees for all prior users.

Consider using the delta of the balance before and after deposit to support fee-charging tokens within the internal accounting logic. Alternatively, consider reverting any bridging attempts involving token types with transfer-related accounting behaviors that are known to be unsupported.

Update: Partially fixed in commit `76b8ff65fab5cf73cd1ee07b6e654d9fcf18b787` in pull request 27 where extra documentation has been added in the `L1StandardBridge` contract to warn users about potential risk.

L-05 Auto withdrawal transactions can be misleading

When a cross-domain transaction fails, it is sometimes bounced back to the origination chain in the form of an auto-withdrawal transaction. Transactions involving ERC20 cross-domain transfers are auto-refunded by the `StandardBridge` if the transfer is not successful. Transactions involving ERC721 tokens are auto-refunded back to L1 if the L2 token is not as expected.

In all cases, the auto-refunded transactions are created by flipping the `from` and `to` addresses associated with the original transaction. This is somewhat intuitive, because the "refund" should go back *to* the address that it was originally *from*.

However, what is less intuitive is that the new `from` address on the refund transaction is an address that never had control of the token involved. Indeed, the auto-withdrawal transaction itself is precisely because the transfer to the `to` address did not successfully complete.

And yet, auto-withdrawal transactions are sent back to the originating chain with no indication that they are distinct from any other transactions. Without any such indication that these transactions are auto-withdrawal transactions, off-chain observers could mistakenly believe that the `from` address had control of the token involved in the transaction.

In reality, the `from` address is completely arbitrary - it could, for instance, be the zero address or any other arbitrary address selected by the original transaction sender. Although this does not undermine the security of the cross-domain bridges, it is misleading and prone to misinterpretation.

Consider making auto-withdrawal transactions distinct from regular transactions. Also consider making these auto-withdrawal transactions `from` the same address that initiated the corresponding original transaction to better convey which addresses were actually ever in control of the token.

Update: Fixed in commit `45766d702daffd8a1837dc8e4c8ffce497e3d4b8` in pull request 20.

L-06 Circumventable requirement that `owner` and `proposer` are distinct

The `L2OutputOracle` contract has a `changeProposer` function that lets the contract owner update the `proposer` address.

It enforces the requirement that even the contract owner cannot set the `proposer` to the `owner` address.

However, this requirement can easily be circumvented by the owner. Because the contract inherits from the OpenZeppelin `OwnableUpgradeable` contract, it also inherits a public `transferOwnership` function that would allow the owner to set the `owner` address to the `proposer` address.

Relatedly, there is no requirement in the contract's `initializer` function that the `owner` and `proposer` are distinct.

If the requirement that the `owner` and `proposer` addresses are distinct is important, consider overriding the inherited `transferOwnership` function to enforce this requirement and use this same function within the `initializer` function as well. Otherwise, consider documenting why the requirement is needed in one case but not the others.

Update: Fixed in commit `ba0857dca0c75c6cea0174759b925da6ab6dc141` in pull request 24.

L-07 Deprecated math library

The parent npm package of the `FixedPointMathLib` that is imported by the `ResourceMetering` contract has been deprecated. The library is still being actively maintained within this github repo.

Although the deprecated npm package and the maintained repo do not currently diverge with regard to the logic of the `FixedPointMathLib`, this could change as the maintained version is iterated upon. Consider updating the implementation so that it is not dependent on the deprecated npm package.

Update: Not fixed.

L-08 ERC721 bridge contracts not using `safeTransferFrom`

Cross-chain ERC721 transfers *back* to the Ethereum network (L1) are finalized via the `finalizeBridgeERC721` function of the `L1ERC721Bridge` contract. There, the ERC721 token is transferred to the intended recipient's (`_to`) address via the ERC721 token's `transferFrom` method.

In the event that the recipient is a smart contract, it may not have a mechanism to interact with ERC721 tokens. This could result in the token becoming unmovable after this transfer. The ERC721 spec attempts to mitigate this scenario via the `safeTransferFrom` method, which provides additional checks to ensure that the recipient of the transfer can safely accept ERC721 tokens.

Consider using the ERC721 `safeTransferFrom` method unless there are specific reasons why its additional safety checks would not be desirable. In the case that `safeTransferFrom` is not desirable, consider explicitly documenting the reason why.

Update: Fixed in commit `e0556413f0d3d1e6bab3bfb2864bca02c9c37c66` in pull request 20.

L-09 Misleading inline documentation

There are instances of potentially misleading inline documentation that should be fixed. For instance:

- The NatSpec `@notice` for the `successfulMessages` mapping is the same as that for `receivedMessages` mapping. It makes sense for the latter, but not the former.
- The NatSpec `@notice` for the Standard Bridge contracts says that "ERC20 tokens sent to {this domain} are escrowed within this contract" (here and here); this is not strictly true. `OptimismMintableERC20` tokens are not escrowed.
- The NatSpec `@notice` tag for `computeL2Timestamp` errantly says that it "Returns a null output proposal if none is found."

Clear inline documentation is fundamental for outlining the intentions of the code. Mismatches between the inline documentation and the implementation can lead to serious misconceptions about how the system is expected to behave. Consider clarifying the referenced inline documentation to avoid confusion for developers, users, and auditors alike.

Update: Fixed in commit `0f8057e5979b5b947ce99ed32dd3b7fd60c563da` in pull request 27.

L-10 Specs do not cover all aspects of current implementation

The deposits specification document is incomplete with regard to the current implementation of the codebase. For instance:

- The specification says that "deposited transactions" should contain an additional byte that indicates the version of the transaction encoding, but no such byte is encoded in the `encodeDepositTransaction` function.
- The `UserDepositTransaction` struct has `l1BlockHash` and `logIndex` attributes that are not covered in the specification.

To increase the overall readability and maintainability of the codebase, consider bringing the specification in line with the current implementation.

Update: Not fixed.

L-11 Potentially confusing naming

There are instances in the codebase where naming could be improved or where naming practices could be more consistent.

For instance, in the `L1ChugSplashProxy` contract the `onlyWhenNotPaused` modifier is actually concerned with whether or not there is an upgrade in progress rather than some sort of arbitrary "pause". A name such as `onlyWhenNotUpgrading` would better reflect the actual implementation.

A more general issue around naming is that domain agnostic names ("localToken" rather than "L1Token", for instance) are used for inconsistent ends. In a contract that functions as the base contract for all domains, such as the `StandardBridge` contract, domain agnostic names can make sense. There is a trade off in terms of readability, because names with relative terms require more mental overhead, but the benefit in such a case is that much less code is duplicated. In these cases, the ends (code reuse) can justify the means (harder to read code).

However, in the `L1ERC721Bridge` and the `L2ERC721Bridge` contracts, there are domain agnostic terms that are not attributable to code reuse. Thus, they are harder to read and understand without any benefit.

Consider renaming elements of the codebase so that they are as easy to understand as possible. Further, as suggested elsewhere in this report, consider reusing code that is domain agnostic as much as possible. If reuse is undesirable, then consider renaming the relevant domain agnostic code to be domain specific so that it is easier to understand.

Update: Partially fixed in commit `8603864482eb734cbeb79e7b5fc994f16685b577` in pull request 10. The Optimism team's response:

> *We have only one Chugsplash contract deployed, and won't deploy any new ones, so prefer not to modify it. Regarding direction agnostic code-reuse, this was fixed for L13.*

L-12 Potential revert-inducing overflow

The `baseGas` function in the `crossDomainMessenger` contract is responsible for computing the amount of gas required to ensure that a given message will be received on the other chain without running out of gas.

Inside this function the return value, `_minGasLimit`, and `MIN_GAS_DYNAMIC_OVERHEAD_NUMERATOR` are all of type `uint32`.

Given that `MIN_GAS_DYNAMIC_OVERHEAD_NUMERATOR = 1016`, this leaves `_minGasLimit` with a max value around 4.2 million before the function encounters revert-inducing overflows on line 321.

Although this may be unlikely given the current state of block-level gas considerations and the fact that current message passing logic is not that gas heavy, considering fully documenting this design limitation.

Update: Fixed in commit `bdfad3cd03ce163d2cb9accc6f0ccb1e3e134d39` in pull request 30.

L-13 Duplicated code

There are instances of duplicated code between the `L1ERC721Bridge` and `L2ERC721Bridge` contracts. Duplicated code can lead to issues later in the development lifecycle and leaves the project more prone to the introduction of errors. Such errors can inadvertently be introduced when functionality changes are not replicated across all instances of code that should be identical. Examples of duplicated code across these two contracts include the following:

- The `ERC721BridgeInitiated` event (L1, L2)
- The `ERC721BridgeFinalized` event (L1, L2)

- The `otherBridge` public state variable (L1, L2)
- The `initialize` function (L1, L2)
- The `bridgeERC721To` function (L1, L2)
- The `bridgeERC721` function, aside from a single error message being different (L1, L2)

Rather than duplicating code, consider having just one contract or library containing the duplicated code and using it whenever the duplicated functionality is required.

Update: Fixed in commit `8603864482eb734cbeb79e7b5fc994f16685b577` in pull request 10.

L-14 Variables missing the `immutable` keyword

There are instances in the codebase where state variables are set only once and the system does not have logic to update those values by any means other than an upgrade.

For instance, the `otherBridge` address in the `L1ERC721Bridge`, `L2ERC721Bridge`, and the `StandardBridge` contracts and the `messenger` address in the `CrossDomainEnabled` and `StandardBridge` contracts are capable of being set only once for any given proxy via the relevant `Initializer` functions. If those values need to be changed after initialization, then an upgrade would need to be performed.

Given that this is the case, unless the intention is to have several proxies using the same logic contract with different values for these addresses, making them `immutable` would offer protection against potential storage slot collisions during upgrades and make the system more robust overall.

To more explicitly signal that these variables are not meant to be updated outside of upgrades and to reduce transaction gas costs by statically encoding their values, consider changing them to immutable variables. If the intended use case demands they are not immutable, then consider adding inline documentation highlighting the reason why.

Update: Not fixed.

Notes & Additional Information

N-01 Functions fail later than required

Throughout the codebase there are a couple of instances where state variables are mutated before relevant checks are evaluated. For instance:

- The `_initiateBridgeERC721()` function on line 270 of the `L2ERC721Bridge` contract burns the L2 NFT *before* checking if the destination token address matches the expected value on line 275.
- The `finalizeWithdrawalTransaction()` function on line 183 of the `OptimismPortal` contract sets the value of the public `finalizedWithdrawals` mapping *before* checking if there is enough gas on line 188.

In favor of failing early to save gas and to follow the best practice of using the checks-effects-interactions pattern, consider refactoring these functions to ensure that state variables are only updated after all checks are complete.

Update: Partially fixed in commit `c333d0d1fe979c094e7371c997f91256946a69f8` in pull request 18. No relevant changes made to the `OptimismPortal` contract cited in the issue.

N-02 Lack of indexed parameters in event

The `sender` address parameter for the `SentMessage` event defined in the `CrossDomainMessenger` contract is not indexed. The lack of indexing is inconsistent with the majority of other `address` parameters in events throughout the codebase.

Consider indexing the `sender` parameter to allow off-chain searching and filtering. Alternatively, to clarify intent, consider adding inline documentation conveying the reasoning behind not indexing the parameter.

Update: Not fixed. The Optimism team's response:

> *Need to preserve for backwards compatibility.*

N-03 Require used for condition "that will never happen"

The `relayMessage` function inside the `crossDomainMessenger` contract checks for a required condition on line 233. This condition is annotated with the inline comment, "Should never happen".

To better convey intent and to allow for better debugging, consider using an `assert` statement rather than a `require` statement to check for critical system error conditions.

Update: Fixed in commit `2fa8f29cea85ea4952db60d2dad8a29a6a4859cd` in pull request 26.

N-04 Confusing `else` conditions

The `ProxyAdmin` contract is meant to be the administrator/owner of various types of proxy contracts. It defines an `enum` meant to enumerate three supported kinds of proxies that it can be the administrator of. Namely, it supports proxy types: `ERC1967, CHUGSPLASH, RESOLVED`.

The `setProxyType` function allows the contract owner to map an address to an allowable proxy type. It implicitly acknowledges that the "default" proxy type will be `ERC1967` based on the `0` value of the `enum` when it explicitly states that the proxy type only needs to be set if it is one of the other ("legacy") allowable types.

Several administrative functions of the `ProxyAdmin` contract check which proxy type a given address maps to. At the end of such checks there is an `else` statement that reverts with the message "ProxyAdmin: unknown proxy type".

Given that a proxy type cannot be set to anything other than a supported type, and given the fact that the default (0) case maps to the default supported proxy type, the `else` statements meant to trigger for some "other" proxy type will never be reachable.

To improve the overall readability and intentionality of the codebase, consider documenting why the `else` clause is present or replacing the nested `require` statement with an `assert` statement if this is a condition that should never be reachable.

Update: Fixed in commit `d04de66a39b8e42d1c0661bc66cdec046d857759` in pull request 26.

N-05 Inconsistent approach to auto-refunding failed token transfers

The `StandardBridge` contract will automatically create refund transactions for ERC20 token transfers that fail. Since it checks for generic transaction failure via a `try` block, it will auto-refund a wide variety of failure cases.

On the other hand, the `L2ERC721Bridge` contract only auto-refunds ERC721 token transfers if the token on the target domain has the wrong interface or disagrees about the address of the associated L1 token. The `L1ERC721Bridge` contract has no mechanism to auto-refund token transfers that would be unsuccessful.

To reduce user confusion, consider standardizing the mechanisms the bridges use to auto-refund unsuccessful token transfers. In the case where these mechanisms cannot be standardized, consider thoroughly documenting the failure cases that each bridge endpoint is capable of creating refund transactions for.

Update: Fixed in commit `efc31f3b2131d6d166bf7ebe5c206e532ec67cac` in pull request 20.

N-06 Inconsistent terminology

Domain sensitive terminology used across the codebase can be inconsistent, even when accounting for the specifics of the individual contracts.

For instance, the `L2CrossDomainMessenger` contract uses layer-specific terminology to describe the `_isOtherMessenger` function:

> Checks that the message sender is the L1CrossDomainMessenger on L1.

Whereas the `L1CrossDomainMessenger` contract uses layer-agnostic terminology to describe its `_isOtherMessenger` function:

> Checks whether the message being sent from the other messenger. (*sic*)

Consider adopting layer specific terminology wherever the code being documented is layer specific and using layer agnostic terminology only alongside code that is, itself, layer agnostic.

Update: Fixed in commit `c49926220e01399a6edeffa3c0be7cfe8f8bd77e` in pull request 27.

N-07 Unexplained literal values

Within the codebase there are occurrences of literal values being used with unexplained meaning. For instance:

- The literal storage slot used where the `ProxyAdmin` sets the storage slot of an `L1ChugSplashProxy`
- The literal value `0x7e` used as a prefix for deposit transaction encoding
- The value `int256(uint256(type(uint128).max))` that is used (here and here) as the upper bound for the base fee in the `ResourceMetering` contract

Literal values in the codebase without an explained meaning make the code harder to read, understand and maintain for developers, auditors, and external contributors alike.

Consider defining a `constant` variable for every magic value used, giving it a clear and self-explanatory name. Additionally, for complex values, inline comments explaining how they were calculated or why they were chosen are highly recommended.

Update: Partially fixed in commit `9cba510e6eb3878714047e3478d18c0266370429` in pull request 28. No relevant changes were made to address the third bullet point raised in the issue.

N-08 Easily bypass-able requirement

The `L2ERC721Bridge` contract has two functions to initiate a withdrawal process, namely, `bridgeERC721` and `bridgeERC721To`. The same pattern is used in the `L1ERC721Bridge` contract.

The `bridgeERC721` function requires that `msg.sender` is not a contract, but this restriction can be easily bypassed by using `bridgeERC721To` which has no such restriction.

Consider removing the restriction and simplifying the number of functions if the restriction is not necessary. Alternatively, consider more thoroughly documenting why the restriction is only required for one function but not the other.

Update: Fixed in commit `f6916d56971441e5daa70f3964942b85c88a29e2` in pull request 15.

N-09 Typographical errors

The following typographical errors were identified in the codebase:

- "recieve" should be "receive"
- "/**q" should be "/**"
- "Intializer" should be "Initializer"
- "being sent from" should be "being sent is from"
- "deposted" should be "deposited"
- "transfering" should be "transferring"
- "access" should be "address"
- "transfering" should be "transferring"
- "erreneous" should be "erroneous"
- "rerutn" should be "return"
- "If the of this" should be "If the value of this"
- "based an amount" should be "based on the amount"
- "depoisited" should be "deposited"

To improve the overall readability of the codebase, consider correcting these typographical errors.

Update: Partially fixed in commit `4df56a057a31958911366f976121574b9a371314` in pull request 14. Not all typos were addressed.

N-10 Undocumented implicit approval requirements

In the `StandardBridge` and `L1ERC721Bridge` contracts, when a token is transferred into the system via its `safeTransferFrom` or `transferFrom` functions, there is an implicit requirement that the address the token is being transferred from has already granted the appropriate approvals.

In favor of explicitness and to improve the overall clarity of the codebase, consider documenting all approval requirements in the relevant functions' inline documentation.

Update: Partially fixed in commit `e97820c3e2eb5b0df59b15e30a7bbb26d5ddd168` in pull request 8. No relevant changes were made to the `StandardBridge` contract cited in the issue.

N-11 Unused inherited contract

The `L1ERC721Bridge` and `L2ERC721Bridge` contracts import (here and here, respectively) and inherit from the OpenZeppelin `OwnableUpgradeable` contract, but do not use any of its functionality.

To simplify the codebase and avoid potential confusion, consider not importing or inheriting from contracts unless their logic is necessary for the inheriting contract to operate as intended.

Update: Fixed in commit `09ac4b02e1e277a44d6f15a906e2d6b2522baaa0` in pull request 7.

N-12 Virtual functions never overridden

In the `StandardBridge` contract, the `bridgeERC20` and `bridgeERC20To` functions are marked `virtual`, but neither the `L1StandardBridge` contract nor the `L2StandardBridge` contract that inherits from `StandardBridge` overrides these functions.

To improve the overall explicitness and intentionality of the codebase, consider removing the `virtual` keyword from functions if it is not needed or documenting why functions are marked `virtual` if they are never overridden.

Update: Not fixed. The Optimism team's response:

> *This is a reusable contract intended to be used by other projects as well.*

Conclusions

No critical or high severity issues were found. Some suggestions to improve code cleanliness and quality were made.