Optimism Cycle 19
Security Review

Cantina Managed review by:
Cccz, Security Researcher
Jeiwan, Security Researcher
Christos Pap, Associate Security Researcher

February 15, 2024

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

Optimism is a fast, stable, and scalable L2 blockchain built by Ethereum developers, for Ethereum developers. Built as a minimal extension to existing Ethereum software, Optimism's EVM-equivalent architecture scales your Ethereum apps without surprises. If it works on Ethereum, it works on Optimism at a fraction of the cost.

From Jan 22nd to Feb 5th the Cantina team conducted a review of optimism on commit hash e6ef3a90. The team identified a total of **14** issues in the following risk categories:

- Critical Risk: 0
- High Risk: 0
- Medium Risk: 0
- Low Risk: 7
- Gas Optimizations: 0
- Informational: 7

3 Findings

3.1 Low Risk

3.1.1 Avoid using real values in the implementation

Severity: Low Risk

Context: L1CrossDomainMessenger.sol#L28-L39

Description: Calling `initialize()` in the `constructor()` initializes `superchainConfig` and `portal` to `address(0)`, but `otherMessenger` remains `Predeploys.L2_CROSS_DOMAIN_MESSENGER`:

```
constructor() CrossDomainMessenger() {
    initialize({ _superchainConfig: SuperchainConfig(address(0)), _portal: OptimismPortal(payable(address(0)))
↪  });
}

/// @notice Initializes the contract.
/// @param _superchainConfig Contract of the SuperchainConfig contract on this network.
/// @param _portal Contract of the OptimismPortal contract on this network.
function initialize(SuperchainConfig _superchainConfig, OptimismPortal _portal) public initializer {
    superchainConfig = _superchainConfig;
    portal = _portal;
    __CrossDomainMessenger_init({ _otherMessenger: CrossDomainMessenger(Predeploys.L2_CROSS_DOMAIN_MESSENGER)
↪  });
}
```

Although the correct checks are made in the tests, however, we should not use the real values in the implementation to ensure that there is a difference between the proxy and the implementation, thus avoiding confusion.

And, considering that this implementation will be used by multiple proxies, it is not a good choice to keep it as a certain L2 value.

Recommendation: It is recommended not to use real values in the implementation.

3.1.2 Different sources of values for checking and deployment

Severity: Low Risk

Context: l1.go#L121-L123, L1CrossDomainMessenger.sol#L38

Description: In `L1()`, we check that the values in the configuration (from `list/config/chainConfig` parameter) are the same as on the chain, and call the `initialize()` with them later. However, for `predeploys`, the address in `op-bindings/predeploys/addresses.go` is used for checking and the address in `packages/contracts-bedrock/src/libraries/Predeploys.sol` is used for deploying. If the addresses in the two files are not synchronized, then this will result in incorrect addresses being used when deployment.

```
import (
    //...
"github.com/ethereum-optimism/optimism/op-bindings/predeploys"
    //...
    if otherMessenger != predeploys.L2CrossDomainMessengerAddr {
        return fmt.Errorf("upgrading L1CrossDomainMessenger: OtherMessenger address doesn't match config")
    }
```

```
import { Predeploys } from "src/libraries/Predeploys.sol";
// ...
    function initialize(SuperchainConfig _superchainConfig, OptimismPortal _portal) public initializer {
        superchainConfig = _superchainConfig;
        portal = _portal;
        __CrossDomainMessenger_init({ _otherMessenger:
↪  CrossDomainMessenger(Predeploys.L2_CROSS_DOMAIN_MESSENGER) });
    }
```

Recommendation: It is recommended to add checks to ensure that the values to be used are the same in the two files.

3.1.3 Gaps should be reserved for `FeeVault`

Severity: Low Risk

Context: FeeVault.sol#L32

Description: There is no gaps reserved in `FeeVault` abstract contract.

The `BaseFeeVault`, `L1FeeVault`, and `SequencerFeeVault` that inherit from `FeeVault` don't have their own state variables yet, so it's possible to reserve gaps in `FeeVault` now.

If any contracts that inherit from `FeeVault` have their own state variables, then it will not be possible to add any new state variables to `FeeVault`. This will break `FeeVault` contract's upgradability.

Recommendation: It is recommended to reserve gaps in `FeeVault`.

```
  abstract contract FeeVault {
      /// @notice Enum representing where the FeeVault withdraws funds to.
      /// @custom:value L1 FeeVault withdraws funds to L1.
      /// @custom:value L2 FeeVault withdraws funds to L2.
      enum WithdrawalNetwork {
          L1,
          L2
      }

      /// @notice Minimum balance before a withdrawal can be triggered.
      uint256 public immutable MIN_WITHDRAWAL_AMOUNT;

      /// @notice Wallet that will receive the fees.
      address public immutable RECIPIENT;

      /// @notice Network which the RECIPIENT will receive fees on.
      WithdrawalNetwork public immutable WITHDRAWAL_NETWORK;

      /// @notice The minimum gas limit for the FeeVault withdrawal transaction.
      uint32 internal constant WITHDRAWAL_MIN_GAS = 35_000;

      /// @notice Total amount of wei processed by the contract.
      uint256 public totalProcessed;

+     uint256[49] private __gap;
```

3.1.4 OP Chain deployment process is not compatible with MCP L1

Severity: Low Risk

Context: Deploy.s.sol#L303-L305

Description: The reviewed MCP L1 upgrade makes L1 contract implementations re-usable by different OP Chains, via their respective L1 proxy contracts. However, the process of deploying OP Chains wasn't updated accordingly: when setting up a new chain, new L1 implementations are deployed and set in respective proxies (Deploy.s.sol#L304-L305):

```
function setupOpChain() public {
    console.log("Deploying OP Chain");
    // ...
    deployProxies();
    deployImplementations(); // @audit deploys new implementations instead of re-using existing ones
    initializeImplementations();
    // ...
}
```

- Deploy.s.sol#L329-L342:

```
function deployImplementations() public {
    console.log("Deploying implementations");
    deployOptimismPortal();
    deployL1CrossDomainMessenger();
    deployL2OutputOracle();
    deployOptimismMintableERC20Factory();
    deploySystemConfig();
    deployL1StandardBridge();
    deployL1ERC721Bridge();
    deployDisputeGameFactory();
    deployPreimageOracle();
    deployMips();
}
```

As a result, the next deployed OP Chain will be using its implementation contracts, which won't be covered by the atomic upgrades feature.

Recommendation: Consider updating the deployment script to re-use existing implementation addresses in newly deployed OP Chains, instead of deploying new ones.

3.1.5 Inconsistent version updates

Severity: Low Risk

Context: L2CrossDomainMessenger.sol#L18-L19, L1CrossDomainMessenger.sol#L23-L25

Description: The reviewed MCP L1 upgrade doesn't assume backward incompatible changes. Indeed, the changes made in L1CrossDomainMessenger, L2CrossDomainMessenger, and CrossDomainMessenger didn't break backward compatibility.

However, `L1CrossDomainMessenger` and `L2CrossDomainMessenger` received different version updates:

1. `L1CrossDomainMessenger` was updated from 2.2.0 to 2.3.0 (a minor update).
2. `L2CrossDomainMessenger` was updated from 1.9.0 to 2.0.0 (a major update).

As a result, the upgrade will signal an incorrect version update, which may impact third-party integrations and off-chain tools.

Recommendation: In `L2CrossDomainMessenger`, consider setting version to 1.10.0.

3.1.6 Incorrect implementation initialization testing

Severity: Low Risk

Context: L2CrossDomainMessenger.t.sol#L22-L28, L2StandardBridge.t.sol#L25-L33

Description: The L2CrossDomainMessenger_Test.test_constructor_succeeds() is intended to check that the implementation of `L2CrossDomainMessenger` was initialized with zero values. However, the test ensures that the address of the other messenger was set to a real value:

```
L2CrossDomainMessenger impl = L2CrossDomainMessenger(deploy.mustGetAddress("L2CrossDomainMessenger"));
assertEq(address(impl.OTHER_MESSENGER()), address(l1CrossDomainMessenger));
assertEq(address(impl.otherMessenger()), address(l1CrossDomainMessenger));
assertEq(address(impl.l1CrossDomainMessenger()), address(l1CrossDomainMessenger));
```

The cause of this is that `deploy.mustGetAddress("L2CrossDomainMessenger")` returns the address of the proxy, not the implementation (Artifacts.s.sol#L134-L135):

```
if (digest == keccak256(bytes("L2CrossDomainMessenger"))) {
    return payable(Predeploys.L2_CROSS_DOMAIN_MESSENGER);
```

This issue (with an identical root cause) is also present in the L2StandardBridge_Test.test_constructor_succeeds() test case.

As a result, the tests will fail to detect when the implementation of the L2 contract is initialized with real values. It's recommended that implementation contracts are always initialized with zero values so they cannot be confused and/or used as the main contracts.

Recommendation: In the `L2CrossDomainMessenger_Test.test_constructor_succeeds()` test case, consider running the checks against the implementation address, not the proxy address. The implementation address can be obtained by calling the `Proxy.implementation()` function as an admin, or by reading the address from the corresponding storage slot.

This recommendation is also applicable to `L2StandardBridge_Test.test_constructor_succeeds()`.

3.1.7 Resource config values can be modified during the MCP L1 upgrade

Severity: Low Risk

Context: l1.go#L728

Description: The deployment process implemented in the `op-chain-ops/upgrades/l1.go` file is intended to carry over the values of the removed immutable variables to their mutable counterparts. Additionally, the process sets values of newly added variables (e.g. `SystemConfig's stratBlock`). If the initialization of a contract requires setting variables that were not part of the MCP L1 upgrade, their values are carried over without modification (e.g. the `SystemConfig` settings).

However, the resource config of `SystemConfig` is set to the default value during the upgrade of the contract (l1.go#L728):

```
calldata, err := systemConfigABI.Pack(
    "initialize",
    finalSystemOwner,
    gasPriceOracleOverhead,
    gasPriceOracleScalar,
    batcherHash,
    l2GenesisBlockGasLimit,
    p2pSequencerAddress,
    genesis.DefaultResourceConfig, // @audit sets default values
    chainConfig.BatchInboxAddr,
    bindings.SystemConfigAddresses{
        L1CrossDomainMessenger:       common.Address(list.L1CrossDomainMessengerProxy),
        L1ERC721Bridge:               common.Address(list.L1ERC721BridgeProxy),
        L1StandardBridge:             common.Address(list.L1StandardBridgeProxy),
        L2OutputOracle:               common.Address(list.L2OutputOracleProxy),
        OptimismPortal:               common.Address(list.OptimismPortalProxy),
        OptimismMintableERC20Factory: common.Address(list.OptimismMintableERC20FactoryProxy),
    },
)
```

The resource config of a `SystemConfig` can have values that differ from the default ones as a result of a call to SystemConfig.setResourceConfig().

As a result, the upgrade can modify the resource config of a deployed `SystemConfig` contract. The resource config represents the configuration for the EIP-1559 based curve for the deposit gas market. Unintentional modification of the values can impact the gas metering of deposits of an OP chain.

Recommendation: In the upgrades.SystemConfig() function, consider reading the current values of the `SystemConfig`'s resource config and carrying them over to the `initialize()` function call.

3.2 Informational

3.2.1 Incorrect comment in the `ERC721Bridge` contract regarding storage layout

Severity: Informational

Context: ERC721Bridge.sol#L25-26

Description: The comment in the ERC721Bridge contract states that there are `50` reserved storage slots. However, it should indicate that 49 slots are reserved, not 50. This discrepancy arises because the combined storage slots of the `ERC721Bridge` and `Initializable` total 49 slots rather than the stated 50.

The storage slot for the deposit mapping in the parent contract starts at 49 not 50.

Recommendation: It is recommended to modify the comment in the ERC721Bridge contract:

```
- /// @notice Reserve extra slots (to a total of 50) in the storage layout for future upgrades.
+ /// @notice Reserve extra slots (to a total of 49) in the storage layout for future upgrades.
  uint256[46] private __gap;
```

3.2.2 `address(0)` checks are missing from the `ERC721Bridge` contract

Severity: Informational

Context: ERC721Bridge.sol#L73-L82

Description: The `__ERC721Bridge_init` function is called during the initialization of both the L1ERC721Bridge and in the L2ERC721Bridge. However, the `address(0)` checks that were present in the previous version are now missing from the `ERC721Bridge` contract.

```
/// @notice Initializer.
/// @param _messenger   Contract of the CrossDomainMessenger on this network.
/// @param _otherBridge Contract of the ERC721 bridge on the other network.
// solhint-disable-next-line func-name-mixedcase
function __ERC721Bridge_init(
    CrossDomainMessenger _messenger,
    StandardBridge _otherBridge
)
    internal
    onlyInitializing
{
    messenger = _messenger;
    otherBridge = _otherBridge;
}
```

Recommendation: Since the `address(0)` are used to initialize the `_messenger` or `_otherBridge`, `address(0xdEaD)` can be used on both the `L1ERC721Bridge` and `L2ERC721Bridge` contracts while also including the zero-address checks. Additionally, instead of initializing with zero or dead values, you can consider using the _disableInitializers method on the constructor.

3.2.3 Missing or Incomplete NatSpec

Severity: Informational

Context: SystemConfig.sol#L215, SystemConfig.sol#L220, SystemConfig.sol#L225, SystemConfig.sol#L230, SystemConfig.sol#L235, SystemConfig.sol#L240, SystemConfig.sol#L245

Description: Some instances of incomplete or missing NatSpec documentation have been found in the codebase. There are listed in the recommendation section with the suggested fix.

Recommendation:

- In SystemConfig.sol, the `l1ERC721Bridge()`, `l1StandardBridge()`, `l2OutputOracle()`, `optimismPortal()`, `optimismMintableERC20Factory()`, `batchInbox()` and the `startBlock()` functions are missing the NatSpec `@return`.

3.2.4 L2OutputOracle.t.sol removed check for `latestBlockNumber`

Severity: Informational

Context: L2OutputOracle.t.sol#L41-L61

Description: `L2OutputOracle.t.sol` seems to have mistakenly removed the check for `latestBlockNumber` in `test_initialize_succeeds()` when upgrading.

Recommendation: It is recommended to add back the check for `latestBlockNumber` in `test_initialize_succeeds()`.

```
  function test_initialize_succeeds() external {
      address proposer = deploy.cfg().l2OutputOracleProposer();
      address challenger = deploy.cfg().l2OutputOracleChallenger();
      uint256 submissionInterval = deploy.cfg().l2OutputOracleSubmissionInterval();
      uint256 startingBlockNumber = deploy.cfg().l2OutputOracleStartingBlockNumber();
      uint256 startingTimestamp = deploy.cfg().l2OutputOracleStartingTimestamp();
      uint256 l2BlockTime = deploy.cfg().l2BlockTime();
      uint256 finalizationPeriodSeconds = deploy.cfg().finalizationPeriodSeconds();

      assertEq(l2OutputOracle.SUBMISSION_INTERVAL(), submissionInterval);
      assertEq(l2OutputOracle.submissionInterval(), submissionInterval);
      assertEq(l2OutputOracle.L2_BLOCK_TIME(), l2BlockTime);
      assertEq(l2OutputOracle.l2BlockTime(), l2BlockTime);
+     assertEq(l2OutputOracle.latestBlockNumber(), startingBlockNumber);
      assertEq(l2OutputOracle.startingBlockNumber(), startingBlockNumber);
      assertEq(l2OutputOracle.startingTimestamp(), startingTimestamp);
      assertEq(l2OutputOracle.finalizationPeriodSeconds(), finalizationPeriodSeconds);
      assertEq(l2OutputOracle.PROPOSER(), proposer);
      assertEq(l2OutputOracle.proposer(), proposer);
      assertEq(l2OutputOracle.CHALLENGER(), challenger);
      assertEq(l2OutputOracle.FINALIZATION_PERIOD_SECONDS(), finalizationPeriodSeconds);
      assertEq(l2OutputOracle.challenger(), challenger);
  }
```

3.2.5 `OptimismPortal` initialization tests lack resource config checks

Severity: Informational

Context: OptimismPortal.t.sol#L34-L42, OptimismPortal.t.sol#L45-L56

Description: The test_constructor_succeeds and test_initialize_succeeds test cases check the initial values in the implementation and the proxy of `OptimismPortal`. However, those checks are not complete: the OptimismPortal.initialize() function also initializes the ResourceMetering contracts and sets the initial resource config–these config values are not checked in the tests.

Recommendation: In above mentioned test cases, consider checking the initial resource config values of `OptimismPortal`.

3.2.6 Typos

Severity: Informational

Context: L2OutputOracle.t.sol#L454, OptimismPortal.sol#L26

Description: During the review, the following typos were found in the codebase:

1. L2OutputOracle.t.sol#L454: "test_**initalize**_l2BlockTimeZero_reverts" should be "test_**initialize**_l2BlockTimeZero_reverts";
2. In OptimismPortal.sol#L26 `whcih` should be `which`.

```
  /// @notice Represents a proven withdrawal.
  /// @custom:field outputRoot    Root of the L2 output this was proven against.
- /// @custom:field timestamp     Timestamp at whcih the withdrawal was proven.
+ /// @custom:field timestamp     Timestamp at which the withdrawal was proven.
  /// @custom:field l2OutputIndex Index of the output this was proven against.
```

Recommendation: Consider fixing the typos shown above.

3.2.7 Upgrading poses the risk of unintentional modification of state variables

Severity: Informational

Context: SystemConfig.sol#L153-L188, l1.go#L720-L738

Description: The upgradeable contracts implement only one state initialization function (`initialize()`), which is used both during deployments and upgrades. This doesn't allow to initialize state variables selectively during an upgrade.

Consider SystemConfig.initialize() as an example. The reviewed MCP L1 upgrade has added new state variables to the contract, which were all added to the `initialize()` for initialization during deployment. But, since the same initializer is used during upgrades, the values of the previously defined state variables need to be carried carried over without modification, when upgrading the contract (l1.go#L664-L738):

```
gasPriceOracleOverhead, err := systemConfig.Overhead(&bind.CallOpts{})
// ...
gasPriceOracleScalar, err := systemConfig.Scalar(&bind.CallOpts{})
// ...
batcherHash, err := systemConfig.BatcherHash(&bind.CallOpts{})
// ...
l2GenesisBlockGasLimit, err := systemConfig.GasLimit(&bind.CallOpts{})
// ...
p2pSequencerAddress, err := systemConfig.UnsafeBlockSigner(&bind.CallOpts{})
// ...
finalSystemOwner, err := systemConfig.Owner(&bind.CallOpts{})
// ...
calldata, err := systemConfigABI.Pack(
    "initialize",                // @audit old variables are carried over
    finalSystemOwner,            //
    gasPriceOracleOverhead,      //
    gasPriceOracleScalar,        //
    batcherHash,                 //
    l2GenesisBlockGasLimit,      //
    p2pSequencerAddress,         //
    genesis.DefaultResourceConfig, //
    chainConfig.BatchInboxAddr,  //
    bindings.SystemConfigAddresses{ // @audit new variables are set
        L1CrossDomainMessenger:       common.Address(list.L1CrossDomainMessengerProxy),
        L1ERC721Bridge:               common.Address(list.L1ERC721BridgeProxy),
        L1StandardBridge:             common.Address(list.L1StandardBridgeProxy),
        L2OutputOracle:               common.Address(list.L2OutputOracleProxy),
        OptimismPortal:               common.Address(list.OptimismPortalProxy),
        OptimismMintableERC20Factory: common.Address(list.OptimismMintableERC20FactoryProxy),
    },
)
```

This creates the risk of mistakenly modifying an existing storage value during an upgrade. Since proxy addresses of L1 contracts are now stored in storage, an unintentional modification of a mutable proxy address can disrupt the work of an OP chain.

Recommendation: Consider having multiple initializing functions in the upgradeable contracts:

1. The main initializer, e.g. `initialize()`, could be used to initialize state during deployments. It'd initialize all state variables of a contract.
2. The upgrade initializer, e.g. `initialize_upgrade()`, would initialize only upgrade-specific storage variables.

To avoid initialization conflicts, consider using versioning via the `reinitializer` modifier.