Optimism Security Review

Auditors

M4rio.eth, Lead Security Researcher

MiloTruck, Lead Security Researcher

LonelySloth, Lead Security Researcher

0xDeadbeef, Associate Security Researcher

Report prepared by: Lucas Goiriz

March 5, 2025

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Optimism is a Collective of companies, communities, and citizens working together to reward public goods and build a sustainable future for Ethereum.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of `optimism` according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- 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.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 20 days in total, Optimism engaged with Spearbit to review the optimism protocol. In this period of time a total of **19** issues were found.

Summary

Project Name	Optimism
Repository	optimism
Commit	7d6d15...87e1
Type of Project	Infrastructure, L2
Audit Timeline	Jan 21 to Feb 21

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	2	1	1
Low Risk	6	3	3
Gas Optimizations	0	0	0
Informational	11	3	8
Total	**19**	**7**	**12**

5 Findings

5.1 Medium Risk

5.1.1 EIP-1559 parameters are not set in `SystemConfig.initialize()`

Severity: Medium Risk

Context: SystemConfig.sol#L179

Description: According to the Holocene specs, `ConfigUpdate.UNSAFE_BLOCK_SIGNER` and `ConfigUpdate.EIP_1559_PARAMS` should be emitted during initialization:

> The following actions should happen during the initialization of the SystemConfig:
>
> - emit ConfigUpdate.BATCHER
> - emit ConfigUpdate.FEE_SCALARS
> - emit ConfigUpdate.GAS_LIMIT
> - emit ConfigUpdate.UNSAFE_BLOCK_SIGNER
> - emit ConfigUpdate.EIP_1559_PARAMS

However, `SystemConfig.initialize()` does not call `_setEIP1559Params()` to set `eip1559Denominator` and `eip1559Elasticity`. As a result, upgraded or newly deployed chains will not have these parameters initialized, causing the EIP-1559 constants declared in the chain's config to be used when calculating gas costs on L2, even after the chain has been upgraded to Holocene.

Additionally, `initialize()` sets `UNSAFE_BLOCK_SIGNER_SLOT` directly instead of calling `_setUnsafeBlockSigner()`:

```
Storage.setAddress(UNSAFE_BLOCK_SIGNER_SLOT, _unsafeBlockSigner);
```

As such, the `ConfigUpdate` event will not be emitted for `UpdateType.UNSAFE_BLOCK_SIGNER` on initialization.

Recommendation: In `initialize()`:

- Call `_setEIP1559Params()` to set the EIP-1559 parameters.
- Call `_setUnsafeBlockSigner()` instead of setting `UNSAFE_BLOCK_SIGNER_SLOT` directly.

Optimism: Acknowledged. As long as these values are unset, the default values will be used. We ended up not going with this design because it would add a strict requirement that the L2 hardforks before the L1 contracts, and this is complicated to manage in a world of many chains, until they are all on the same version of everything.

Spearbit: Acknowledged.

5.1.2 Blacklisted `anchorGame` in `AnchorStateRegistry` prevents submitting L2 root claims to L1

Severity: Medium Risk

Context: AnchorStateRegistry.sol#L109-L111, AnchorStateRegistry.sol#L266-L273, FaultDisputeGame.sol#L299-L300, OPContractsManager.sol#L541-L544

Description: `AnchorStateRegistry.getAnchorRoot()` checks if the current `anchorGame` is blacklisted, and reverts if so:

```
if (isGameBlacklisted(anchorGame)) {
    revert AnchorStateRegistry_AnchorGameBlacklisted();
}
```

While this is meant to prevent invalid anchor roots from being used, `getAnchorRoot()` reverting will break several important functions in the protocol.

`anchorGame` could be blacklisted if the following sequence occurs:

1. Dispute game incorrectly resolves to DEFENDER_WINS.
2. DISPUTE_GAME_FINALITY_DELAY_SECONDS passes.
3. Guardian calls blacklistDisputeGame() to blacklist the dispute game.
4. Attacker front-runs guardian, calling setAnchorState() to set the dispute game as anchorGame.

When this occurs, due to getAnchorRoot() always reverting when called:

1. New dispute games cannot be created as FaultDisputeGame.initialize() reverts.
2. anchorGame cannot be updated as AnchorStateRegistry.setAnchorState() reverts.
3. The chain cannot be upgraded as OPContractsManager.upgrade() reverts.

(1) is particularly severe as there is no way for Optimism's proposer (or anyone else) to submit root claims to L1 until the situation is manually handled. This violates the L2Beat's Stage 1 requirements.

Additionally, since setAnchorState() cannot be called as outlined in (2), anchorGame is permanently set to the blacklisted dispute game. As such, there is no way for such a situation to be resolved without an upgrade to AnchorStateRegistry.

Recommendation: Remove the blacklist check from getAnchorRoot(). If there is a need to ensure the anchor root is not invalid, it should be checked before calling getAnchorRoot(). For example, OPContractsManager.upgrade() could check if anchorGame is blacklisted before calling anchors().

Note that the only way to prevent a blacklisted anchorGame from occurring is to ensure the guardian will always blacklist an incorrectly resolved dispute game before DISPUTE_GAME_FINALITY_DELAY_SECONDS has passed (currently 3.5 days after the dispute game is resolved). This is fully reliant on the guardian and nothing in the code prevents anchorGame from being blacklisted.

Optimism: Fixed in PR 14232 by removing the blacklist check. This should be safe under the assumption that invalid dispute games are invalidated *before* they are finalized. Additionally, we will begin to formalize the L2Beat Stage 1 definition as an invariant that any future design must maintain.

Spearbit: Verified, the blacklist check in getAnchorRoot() was removed.

5.2 Low Risk

5.2.1 Missing checks in OPCM

Severity: Low Risk

Context: See each case below

Description: The OPContractsManager functions as a deployment script for various on-chain deploys and upgrades. To prevent potential mistakes that could occur during these operations, it should enforce comprehensive validation checks, which are currently incomplete.

This issue will enumerate the missing checks that should be implemented to improve security and reliability.

1. OPContractsManager.sol#L875: DeployInput.startingAnchorRoot in the deploy function is missing a check against 0x000...0000, currently it's just checking the length if (_input.startingAnchorRoot.length == 0) revert InvalidStartingAnchorRoot();. Also, the length check could be more specific since the only valid length is known.
2. OPContractsManager.sol#L660-L670, OPContractsManager.sol#L684-L694: The addGameType is missing checks against the configurability parameters, see: Spec:configurability.
3. OPContractsManager.sol#L448: Missing checks for the OpChainConfig parameters, this should be checked at least in the scripts same as you do for the deploy.
4. OPContractsManager.sol#L483-L485: We only verify the superchainConfig for optimismPortal while we should verify for every contract that does contain this.

Recommendation: Consider adding the missing checks.

Optimism: Fixed in PR 14357.

We addressed items 3 and 4. The first check is redundant with the `abi.decode()` call which does a length check. The second checks are not desirable as we intend for the method to be maximally flexible. This allows it to be used on devnets and testnets to setup chains with non-standard configs for testing.

Spearbit: Verified.

5.2.2 `OptimismPortal2.proveWithdrawalTransaction()` rejects valid dispute games when `respectedGameType` is updated

Severity: Low Risk

Context: OptimismPortal2.sol#L309-L310

Description: In `OptimismPortal2.sol`, `checkWithdrawal()` does not check `gameType == respectedGameType`, so a dispute game can be used to finalize withdrawals even if its `gameType` does not match the current `respectedGameType`.

However, `proveWithdrawalTransaction()` has the following check:

```
// The game type of the dispute game must be the respected game type.
if (gameType.raw() != respectedGameType.raw()) revert InvalidGameType();
```

This check prevents users from proving withdrawals with a game when its `gameType` does not match `respectedGameType`.

As a result, if `respectedGameType` is changed, users may not be able to prove withdrawals with dispute games that are valid. For example:

- Alice and Bob have withdrawal transactions.
- Alice calls `proveWithdrawalTransaction()` to prove a withdrawal with a dispute game.
- Guardian changes `respectedGameType` without updating `respectedGameTypeUpdatedAt`.
- Alice can still finalize her withdrawal with `finalizeWithdrawalTransaction()`.
- However, if Bob tries to prove a withdrawal with the same dispute game, he will be unable to do so.

Since Alice can finalize her withdrawal with the dispute game, it is implicitly valid. Therefore, Bob should be able to prove his withdrawal with the same dispute game. Note that this does not permanently prevent Bob from proving his withdrawal; he can always prove it with a future dispute game.

Additionally, a strange quirk of this check is it will pass if `respectedGameType` was initially changed but subsequently switched back to the original game type.

Recommendation: Consider removing the `gameType == respectedGameType` check in `proveWithdrawalTransaction()`.

Optimism: Acknowledged, we do not intend to fix this issue as part of this audit but have plans for an additional set of contract changes that would include this fix.

Spearbit: Acknowledged.

5.2.3 `SystemConfig._setGasConfig()` should validate `overhead` and `scalar` according to the specs

Severity: Low Risk

Context: SystemConfig.sol#L312-L320

Description: `SystemConfig._setGasConfig()` only checks that the first byte of `_scalar` is `0x0`:

```
function _setGasConfig(uint256 _overhead, uint256 _scalar) internal {
    require((uint256(0xff) << 248) & _scalar == 0, "SystemConfig: scalar exceeds max.");

    overhead = _overhead;
    scalar = _scalar;

    bytes memory data = abi.encode(_overhead, _scalar);
    emit ConfigUpdate(VERSION, UpdateType.FEE_SCALARS, data);
}
```

As such, there is no input validation ensuring `_scalar` matches the requirements specified in the Ecotone specs for version 0:

- `0`: scalar-version byte
- `[1, 32)`: depending scalar-version:
  - Scalar-version 0:
    - `[1, 28)`: padding, should be zero.
    - `[28, 32)`: big-endian `uint32`, encoding the L1-fee `baseFeeScalar`
    - This version implies the L1-fee `blobBaseFeeScalar` is set to 0.
    - In the event there are non-zero bytes in the padding area, `baseFeeScalar` must be set to MaxUint32.
    - This version is compatible with the pre-Ecotone `scalar` value (assuming a `uint32` range).

More specifically, the function should check that:

1. `blobBaseFeeScalar == 0`.
2. Either bytes 1-28 are zeroed out or `baseFeeScalar == type(uint32).max`.

Recommendation: To prevent the caller from violating the specs while maintaining backwards compatibility, check that `_scalar` doesn't exceed `uint32`:

```
  function _setGasConfig(uint256 _overhead, uint256 _scalar) internal {
-     require((uint256(0xff) << 248) & _scalar == 0, "SystemConfig: scalar exceeds max.");
+     require(_scalar <= type(uint32).max, "SystemConfig: scalar exceeds max.");

      overhead = _overhead;
      scalar = _scalar;

      bytes memory data = abi.encode(_overhead, _scalar);
      emit ConfigUpdate(VERSION, UpdateType.FEE_SCALARS, data);
  }
```

This works as:

- Pre-ecotone networks can set `scalar` to any `uint32` value.
- Post-ecotone networks would decode `scalar` as version 0, with `blobBaseFeeScalar = 0` and `baseFeeScalar` as the `uint32` value. `overhead` doesn't matter as its value is ignored.

The only restriction is pre-ecotone networks can't set `scalar` to larger than `uint32`.

Optimism: Acknowledged, we will plan to fix it in the future. This function exists to enable the `SystemConfig` to be backwards compatible with pre-ecotone networks. Until we have a strong and battle tested versioning system in place, we try to keep the contracts backwards compatible and not add in strict dependencies on the contract upgrading or the network upgrading first.

Spearbit: Acknowledged.

5.2.4 OPCM: Failure when `addGameType` due to common `saltMixer`

Severity: Low Risk

Context: OPContractsManager.sol#L634

Description: The function `addGameType` allows for multiple game types to be simultaneously deployed/upgraded. Additionally, the function allows for the `delayedWETH` address to be set to `address(0)`, meaning the game type needs a new deployment of the `DelayedWETH` rather than an upgrade.

However, if two game types (for the same chain id) have their `delayedWETH` address set to zero, then a collision in the create2 address will prevent the second one from being deployed, reverting the entire transaction -- if the `saltMixer` parameter is the same, which is likely to happen, as these values are usually version tags.

Furthermore, the same `saltMixer` is used to deploy the game implementations as well in the `addGameType`.

Therefore because the same `saltMixer` is used to deploy the deplayedWETH and the dispute games, it creates inconsistencies on how this random salt is used.

We can notice throughout the contract that we have various ways of how this salt is chosen but there is no consistent strategy.

For example, at OPContractsManager.sol#L1125, the `saltMixer` is the game version:

```
computeSalt(_l2ChainId, "v2.0.0", "PermissionedDisputeGame"),
```

Recommendation: A different salt should be used for the `delayedWETH` for each game type -- or the deployment of a new proxy should be skipped if the `create2` address already has code deployed to it.

Overall, the way this salt is generated should be standardized so that it can be unique for every target and every deployment, and that it can assure no deploy conflicts will happen in the future.

Optimism: Fixed in PR 14364 and PR 14430.

Spearbit: Verified.

5.2.5 Possible DoS of creating valid games for specific claims, by creating games before their type is respected

Severity: Low Risk

Context: DisputeGameFactory.sol#L145

Description: The DGF (Dispute Game Factory) allows for the creation of games that are of types other than the one respected at the time of creation.

While such games can't be used to update the Anchor State or withdraw from Optimism Portal, they prevent the creation of other games for the same type, root claim, and block height.

As the process for deploying new game types sets its implementation in the DGF before the respected game type is set in the Portal and/or previous games are invalidated, there is a window of time allowing an attacker to create games of the new type and DoS the creation of valid games for the specific root claims and block heights.

Since it's not possible to know in advance the root claim for future blocks, the impact is quite limited. Also note anchor state updates and withdrawals can still be preformed using new blocks for root claims, so delays imposed on users are very limited. Meanwhile the cost of the attack is considerable, as bonds must be deposited for each game created.

Recommendation: If the small risk involved is considered not acceptable, a mechanism should be created to prevent the DoS -- e.g. requiring only games of the respected type be created.

If the risk is considered acceptable, documenting the fact that the possibility of creating games for every valid claim isn't guaranteed.

Optimism: Acknowledged, given that this does not impact withdrawals we do not plan to address this concern as part of this audit. We will document this system property and plan to do some user research to determine if it's necessary to resolve this at some point in the future.

Spearbit: Acknowledged.

5.2.6 Possible DoS preventing Anchor State from being timely updated

Severity: Low Risk

Context: FaultDisputeGame.sol#L1016

Description: The usual process for updating the Anchor State in the ASR (Anchor State Registry) is implemented as a call from an FDG (Fault Dispute Game) to the ASR. However such call occurs within a try/catch statement that ignores reverts, resulting in the possibility of silent failures. Since that update is triggered by a permissionless function call to `closeGame()` in FDG, a malicious user might force an out of gas exception during the Anchor State update, while the overall call to `closeGame` succeeds.

The above scenario would result in the Anchor State not being updated by that FDG, possibly staying out of date for a length of time.

The impact is limited given that `updateAnchorState` can be called in a permissionless way on the ASR, meaning any user aware of the problem can submit a transaction to perform the call and fix the issue.

Recommendation: A process (ideally automated) should exist to track if Anchor State hasn't been updated for some time, and call the `updateAnchorState` function so as to correctly update it.

Optimism: Fixed in PR 14144. We chose not to change `FaultDisputeGame` but to instead introduce a fuzz test checking that the `closeGame()` function will either revert or correctly update the anchor state. Additionally, we've added a new semgrep rule that forces developers to explicitly mark try/catch blocks as "safe" from issues with EIP-150. We'll also be adding monitoring on the age of the anchor state in private repositories.

Spearbit: Verified.

5.3 Informational

5.3.1 Missing event in `addGameType`

Severity: Informational

Context: OPContractsManager.sol#L708

Description: The `addGameType` function does not emit any event. While in the docs is not specified that this should emit an event, we recommend to add one to better track what exactly was deployed.

Recommendation: Consider adding a special event for `addGameType`.

Optimism: Fixed in PR 14196.

Spearbit: Verified.

5.3.2 Incomplete Specs for OPCM

Severity: Informational

Context: op-contracts-manager.html

Description: The `OPContractsManager` specs are out of date with the current implementation. Various functions are missing or function signatures do not correspond with what we have in the specs, for example:

```
/// @notice Returns the latest approved release of the OP Stack contracts are named with the
///         format `op-contracts/vX.Y.Z`.
function release() external view returns (string memory);
/// @notice Represents the interface version so consumers know how to decode the DeployOutput struct
/// @notice Maps an L2 Chain ID to the SystemConfig for that chain.
function systemConfigs(uint256 _l2ChainId) external view returns (address);
```

These functions are missing from the implementation. The `Deployed` event is different. In the specs:

```
event Deployed(uint256 indexed outputVersion, uint256 indexed l2ChainId, address indexed deployer,
↪   bytes deployOutput);
```

While in the implementation:

```
event Deployed(uint256 indexed l2ChainId, address indexed deployer, bytes deployOutput);
```

The `upgrade` signature is different. In the specs:

```
struct IsthmusConfig {
    uint32 public operatorFeeScalar;
    uint64 public operatorFeeConstant;
}

function upgrade(ISystemConfig[] _systemConfigs, IProxyAdmin[] _proxyAdmins, IsthmusConfig[]
↪   _isthmusConfigs) public;
```

While in the implementation:

```
 struct OpChainConfig {
        ISystemConfig systemConfigProxy;
        IProxyAdmin proxyAdmin;
        Claim absolutePrestate;
    }

function upgrade(OpChainConfig[] memory _opChainConfigs) external {
```

Recommendation: Consider modifying the specs to be in sync with the actual implementation.

Optimism: Fixed in PR 572.

Spearbit: Verified.

5.3.3 Major functionality changes in `FaultDisputeGame`

Severity: Informational

Context: FaultDisputeGame.sol#L968-L974, FaultDisputeGame.sol#L1020-L1029

Description: This issue describes two non-trivial changes in `FaultDisputeGame` from the previous implementation that dispute game participants should be aware of.

1. `WETH.unlock()` is now called in `claimCredit()` instead of `_distributeBond()`:

   ```
       if (!hasUnlockedCredit[_recipient]) {
           hasUnlockedCredit[_recipient] = true;
           WETH.unlock(_recipient, recipientCredit);
           return;
       }
   ```

   This change has the following impacts:

- Users now have to call `claimCredit()` to start the unlock process; if the user is inactive and doesn't call `claimCredit()`, the withdrawal delay will never start. Whereas previously, `WETH.unlock()` will be called automatically when resolving claims.
- Previously, the withdrawal delay starts once `_distributeBond()` is called in `resolveClaim()`. Now, users have to wait for the game to be resolved, the airgap to pass (i.e. `DISPUTE_GAME_FINALITY_DELAY_SECONDS`) and the withdrawal delay before they can withdraw their bonds.

2. Bonds are now refunded to their original depositor if the dispute game is not "proper":

```
// Check if the game is a proper game, which will determine the bond distribution mode.
bool properGame = ANCHOR_STATE_REGISTRY.isGameProper(IDisputeGame(address(this)));

// If the game is a proper game, the bonds should be distributed normally. Otherwise, go
// into refund mode and distribute bonds back to their original depositors.
if (properGame) {
    bondDistributionMode = BondDistributionMode.NORMAL;
} else {
    bondDistributionMode = BondDistributionMode.REFUND;
}
```

   Therefore, if either of the following occur:

   - Guardian updates `respectedGameTypeUpdatedAt` in `OptimismPortal2`.
   - Guardian blacklists the dispute game.

   Bonds will be refunded to their original depositors. Therefore, honest participants could end up not getting bonds for disputing invalid claims if the game is invalidated by the guardian.

Recommendation: Consider documenting these changes in functionality.

Optimism: Acknowledged.

Spearbit: Acknowledged.

5.3.4 Inconsistent data type for gas limit constants in `OptimismPortal2`

Severity: Informational

Context: OptimismPortal2.sol#L72-L76

Description: In `OptimismPortal2`, `RECEIVE_DEFAULT_GAS_LIMIT` is declared as `uint64` while `SYSTEM_DEPOSIT_GAS_LIMIT` is declared as `uint32`:

```
/// @notice The L2 gas limit set when eth is deposited using the receive() function.
uint64 internal constant RECEIVE_DEFAULT_GAS_LIMIT = 100_000;

/// @notice The L2 gas limit for system deposit transactions that are initiated from L1.
uint32 internal constant SYSTEM_DEPOSIT_GAS_LIMIT = 200_000;
```

However, this is inconsistent as both constants represent gas limits for L1 -> L2 deposit transactions. Additionally, variables representing gas limits are declared as `uint64` in other parts of the protocols, such as in `SystemConfig`.

Recommendation: Consider declaring `RECEIVE_DEFAULT_GAS_LIMIT` as `uint64`. `ResourceMetering.useGas()` would have to be refactored to take in a `uint64` parameter as well.

Optimism: Acknowledged.

Spearbit: Acknowledged.

5.3.5 Inconsistent data type when calculating the minimum gas limit in `SystemConfig`

Severity: Informational

Context: SystemConfig.sol#L199-L201, SystemConfig.sol#L351, SystemConfig.sol#L414

Description: In `SystemConfig`, `minimumGasLimit()` casts `maxResourceLimit` and `systemTxMaxGas` to `uint64` before they are added and compared in `_setGasLimit()`:

```
function minimumGasLimit() public view returns (uint64) {
    return uint64(_resourceConfig.maxResourceLimit) + uint64(_resourceConfig.systemTxMaxGas);
}
```

```
require(_gasLimit >= minimumGasLimit(), "SystemConfig: gas limit too low");
```

In contrast, `_setResourceConfig()` directly adds both `maxResourceLimit` and `systemTxMaxGas` as `uint32` without upcasting them. Note that both parameters are `uint32` below:

```
require(_config.maxResourceLimit + _config.systemTxMaxGas <= gasLimit, "SystemConfig: gas limit too
↪   low");
```

However, there is currently no impact as `gasLimit` will not exceed `uint32.max` (~4 billion gas) in the near future.

Recommendation: In `_setResourceConfig()`, cast both parameters to `uint64` before adding them:

```
- require(_config.maxResourceLimit + _config.systemTxMaxGas <= gasLimit, "SystemConfig: gas limit too
↪   low");
+ require(uint64(_config.maxResourceLimit) + uint64(_config.systemTxMaxGas) <= gasLimit, "SystemConfig:
↪   gas limit too low");
```

Optimism: Acknowledged.

Spearbit: Acknowledged.

5.3.6 `ConfigUpdate` event in `SuperchainConfig` is missing its `version` parameter

Severity: Informational

Context: SuperchainConfig.sol#L38-L41, ProtocolVersions.sol#L37-L41, SystemConfig.sol#L125-L129

Description: The `ConfigUpdate` event in `SuperchainConfig.sol` does not have the `version` parameter:

```
/// @notice Emitted when configuration is updated.
/// @param updateType Type of update.
/// @param data       Encoded update data.
event ConfigUpdate(UpdateType indexed updateType, bytes data);
```

This is inconsistent with the `ConfigUpdate` event in `ProtocolVersions.sol` and `SystemConfig.sol`:

```
/// @notice Emitted when configuration is updated.
/// @param version    ProtocolVersion version.
/// @param updateType Type of update.
/// @param data       Encoded update data.
event ConfigUpdate(uint256 indexed version, UpdateType indexed updateType, bytes data);
```

```
/// @notice Emitted when configuration is updated.
/// @param version    SystemConfig version.
/// @param updateType Type of update.
/// @param data       Encoded update data.
event ConfigUpdate(uint256 indexed version, UpdateType indexed updateType, bytes data);
```

Recommendation: Modify the `ConfigUpdate` event in `SuperchainConfig.sol` to include the `version` parameter.

Optimism: Acknowledged. This is intentional because the other two event types are decoded by the op-node, but the one in `SuperchainConfig` is not.

Spearbit: Acknowledged.

5.3.7 AnchorStateRegistry.isGameAirgapped() returns true for unresolved dispute games

Severity: Informational

Context: AnchorStateRegistry.sol#L167-L169

Description: AnchorStateRegistry.isGameAirgapped() checks if DISPUTE_GAME_FINALITY_DELAY_SECONDS has passed since a dispute game was resolved:

```
function isGameAirgapped(IDisputeGame _game) public view returns (bool) {
    return block.timestamp - _game.resolvedAt().raw() > portal.disputeGameFinalityDelaySeconds();
}
```

However, the function does not check resolvedAt != 0. Therefore, it will wrongly return true when the game has not been resolved. Note that there is no impact since isGameAirgapped() is used alongside isGameResolved(), which checks resolvedAt != 0.

Recommendation: Check resolvedAt != 0 as well:

```
  function isGameAirgapped(IDisputeGame _game) public view returns (bool) {
-     return block.timestamp - _game.resolvedAt().raw() > portal.disputeGameFinalityDelaySeconds();
+     uint256 resolvedAt = _game.resolvedAt().raw();
+     return resolvedAt != 0 && block.timestamp - resolvedAt > portal.disputeGameFinalityDelaySeconds();
  }
```

Optimism: Fixed in PR 14139. We decided to simply remove isGameAirgapped() because a "fixed" version of this function would be identical to isGameFinalized().

Spearbit: Verified.

5.3.8 Code improvements

Severity: Informational

Context: See each case below.

Description/Recommendation:

1. FaultDisputeGame.sol#L976-L977: This check should be performed before calling WETH.unlock() to avoid calling unlock() for recipients with no credit.
2. FaultDisputeGame.sol#L999-L1002: There's no need for else if here:

```
- } else if (bondDistributionMode != BondDistributionMode.UNDECIDED) {
+ }
+
+ if (bondDistributionMode != BondDistributionMode.UNDECIDED) {
      // We shouldn't get here, but sanity check just in case.
      revert InvalidBondDistributionMode();
  }
```

3. AnchorStateRegistry.sol#L88, AnchorStateRegistry.sol#L252:respectedGameType() and setAnchorState() can be declared external as they are never called internally.
4. SystemConfig.sol#L166:SystemConfig.Addresses can just be Addresses:

```
- SystemConfig.Addresses memory _addresses
+ Addresses memory _addresses
```

5. OptimismPortal2.sol#L52: SafeERC20 is no longer needed as the portal does not handle ERC20 tokens. The import for IERC20 and SafeERC20 should also be removed.
6. AnchorStateRegistry.sol#L46: The AnchorNotUpdated event not used.
7. AnchorStateRegistry.sol#L46: The AnchorNotUpdated event not used.

8. OPContractsManager.sol#L1106-L1107: The `IFaultDisputeGame.GameConstructorParams memory params` construction can be moved at the first instruction in the function then instead of `getWETH` call you could just use `params.weth`.

Optimism: Acknowledged.

Spearbit: Acknowledged.

5.3.9 Comments and documentation improvements

Severity: Informational

Context: See each case below.

Description/Recommendation: Throughout the codebase, we've identified various comments and documentation text that would require an improvement:

1. L1StandardBridge.sol#L17: Typo: `transfering` ⇒ `transferring`.
2. L1StandardBridge.sol#L280-L281: Comment should say that it emits `ETHWithdrawalFinalized` and `ETHBridgeFinalized`.
3. OptimismPortal2.sol#L193-L201: `_initialRespectedGameType` is missing a NatSpec comment.
4. ProtocolVersions.sol#L55: Typo: `to operate on thi chain` ⇒ `to operate on this chain`.
5. FaultDisputeGame.sol: Consider documenting why `FaultDisputeGame` can receive ETH even though it doesn't have a `receive/fallback` function. This could be a potential footgun if the codebase doesn't use the current version of Solady's `LibClone` in the future.
6. FaultDisputeGame.sol#L968-L974: This comment is incorrect, credit is always unlocked regardless of whether `bondDistributionMode` is `REFUND` or `NORMAL`.
7. FaultDisputeGame.sol#L1065-L1075: Consider adding a comment stating the value returned by `credit()` is not the actual credit owed to `_recipient` until `closeGame()` is called.
8. FaultDisputeGame.sol#L222-L222: The `refundModeCredit` should be better documented that this variable will be 0 only after `claimCredit` is called even if the credit is moved to the normal mode.
9. AnchorStateRegistry.sol#L195: Comment is outdated as `isGameRetired()` now checks `createdAt <= respectedGameTypeUpdatedAt`. It should say "Must be created after the respectedGameTypeUpdatedAt timestamp".
10. SystemConfig.sol#L20-L32: Natspec for `EIP_1559_PARAMS` is missing.
11. DisputeGameFactory.sol#L201-L213: Missing `@return` NatSpec.
12. DelayedWETH.sol#L111: `of WETH from a specific owner.` ⇒ `of WETH from a specific user.`
13. OPContractsManager.sol: The overall NatSpec comments should be improved across the contract, various functions have incomplete or missing comments.

Optimism: Acknowledged.

Spearbit: Acknowledged.

5.3.10 OPCM: Silent failure if `RESOLVED` proxies aren't deployed

Severity: Informational

Context: OPContractsManager.sol#L492

Description: `ProxyAdmin` contracts fail silently when an upgrade to a contract set as "`RESOLVED`" fails due to the contract not being deployed. If OPCM `upgrade` is called and the `l1CrossDomainMessenger` contract has been set as "RESOLVED" but either not deployed or somehow destructed after deployment -- the transaction would complete successfully, even though the upgrade wasn't complete.

This is a very unlikely scenario, but would break the assumption of atomicity of the upgrade.

Recommendation: An assertion that `l1CrossDomainMessenger` has code deployed at its address would prevent the issue.

Optimism: Acknowledged. The resolved proxy is different from others because it reads from the `AddressManager`, so there is no interaction with the proxy during an upgrade, only the `AddressManager`. Therefore, this is an unlikely issue and will choose not to fix.

Spearbit: Acknowledged.

5.3.11 Overflow and memory corruption in `findLatestGames` leading to incorrect values.

Severity: Informational

Context: DisputeGameFactory.sol#L222

Description: The view `findLatestGames` taken a parameter `n` for the maximum number of items returned. However that parameter is used within an inline assembly statement that performs memory allocation, without any checks on its range. That results in possible overflow in a memory location calculation, leading to memory corruption, and the view returning unpredictable, invalid results. That is particularly relevant if the parameter is very large (e.g. `type(uint256).max`).

While this issue is unlikely to be exploitable -- as the parameter wouldn't be under control of an attacker that hasn't already compromised other components of the system (e.g. UI), it's possible mistakes in coding of other components would lead to triggering the issue and giving incorrect results.

Recommendation: The view should assert that the parameter `n` is within reasonable bounds (e.g. $0 < n < 2^{16}$).

Optimism: Acknowledged.

Spearbit: Acknowledged.