SPEARBIT

Optimism Security Review

Auditors

Chris Smith, Lead Security Researcher

MiloTruck, Lead Security Researcher

Report prepared by: Lucas Goiriz

May 20, 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 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.

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 5 days in total, Op Labs engaged with Spearbit to review the optimism protocol. In this period of time a total of **14** issues were found.

Summary

Project Name	Op Labs
Repository	optimism
Commit	7cd84f...48f6
Type of Project	Infrastructure, L2
Audit Timeline	May 5 to May 12
Two week fix period	May 12 - May 16

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	5	3	2
Gas Optimizations	1	0	1
Informational	8	6	2
Total	**14**	**9**	**5**

5 Findings

5.1 Low Risk

5.1.1 `ReinitializableBase` pattern allows `initialize()` to be incorrectly called during an upgrade

Severity: Low Risk

Context: SuperchainConfig.sol#L69, SuperchainConfig.sol#L78

Description: All upgradeable contracts have been modified for their `initialize()` and `upgrade()` functions to use the `reinitializer` modifier with `initVersion()`. Using `SuperchainConfig` as an example:

```
function initialize(address _guardian) external reinitializer(initVersion()) {
```

```
function upgrade() external reinitializer(initVersion()) {
```

However, this new pattern allows `initialize()` to be called in place of `upgrade()` during an upgrade. For example:

- Assume a contract's initialized version is currently `1`.
- In the new implementation, `INIT_VERSION` is set to `2` (i.e. `initVersion() = 2`).
- After the contract is upgraded to the new implementation, calling `initialize()` passes as it uses `reinitializer(2)`.
- However, `upgrade()` was supposed to be called instead.

Recommendation: An ideal pattern would ensure the version that `initialize()` is called with never increases (e.g. a hardcoded constant), such that it can only ever be called once.

Optimism: Acknowledged, we will log this as feedback for OPCM and have prioritized appropriately.

Spearbit: Acknowledged.

5.1.2 Version enforcement could be violated for future versions

Severity: Low Risk

Context: Encoding.sol#L237-L247

Description: The literal `0x01` is used for encoding after confirming the `_superRootProof.version` matches. However, this leaves open a potential footgun where the check is updated for a future version but the literal is still used for the encoding.

Recommendation: Use the version after confirming it is correct so future updates only have to update one place in the library or create an internal `CONSTANT` for the `VERSION` and use that instead of the literals.

- Option 1:

```
uint64 internal constant VERSION = 0x01;
...
        if (_superRootProof.version != VERSION) {
            revert Encoding_InvalidSuperRootVersion();
        }
...
        bytes memory encoded = bytes.concat(bytes1(VERSION), bytes8(_superRootProof.timestamp));
```

- Option 2:

```
...
        if (_superRootProof.version != 0x01) {
            revert Encoding_InvalidSuperRootVersion();
        }
...
        bytes memory encoded = bytes.concat(bytes1(_superRootProof.version),
        ↪   bytes8(_superRootProof.timestamp));
```

Optimism: Fixed in commit 7134b4c0.

Spearbit: Fixed.

5.1.3 `OptimismPortal2.migrateToSuperRoots()` could accidentally unpause a chain

Severity: Low Risk

Context: SystemConfig.sol#L488-L491, OptimismPortal2.sol#L405-L434

Description: In `SystemConfig.paused()`, a chain is paused if either `address(0)` or the `ethLockbox` address set in `OptimismPortal2` is paused:

```
function paused() public view returns (bool) {
    IETHLockbox lockbox = IOptimismPortal2(payable(optimismPortal())).ethLockbox();
    return superchainConfig.paused(address(lockbox)) || superchainConfig.paused(address(0));
}
```

However, since the `ethLockbox` address in `OptimismPortal2` can be changed, an upgrade could accidentally unpause a chain by changing the `lockbox` address stored in `OptimismPortal2`. For example:

- Assume the chain is paused.
- `OptimismPortal2.migrateToSuperRoots()` is called to set a new `ethLockbox` address.
- The new `ethLockbox` address is not paused.
- The chain automatically becomes unpaused without calling `SuperchainConfig.unpause()`.

Recommendation: In `OptimismPortal2`, consider adding the `whenNotPaused` modifier to `migrateToSuperRoots()`:

```
- function migrateToSuperRoots(IETHLockbox _newLockbox, IAnchorStateRegistry _newAnchorStateRegistry)
↪   external {
+ function migrateToSuperRoots(IETHLockbox _newLockbox, IAnchorStateRegistry _newAnchorStateRegistry)
↪   external whenNotPaused {
```

Additionally, ensure that the `ethLockbox` address cannot change while the chain is paused in future upgrades.

Optimism: Fixed in commit 33695ee.

Spearbit: Verified, the recommended fix was implemented.

5.1.4 Mismatching version passed into EIP-712 for `DeputyPauseModule`

Severity: Low Risk

Context: DeputyPauseModule.sol#L82-L84, DeputyPauseModule.sol#L98

Description: The version of `DeputyPauseModule` is declared in a string constant as such:

```
/// @notice Semantic version.
/// @custom:semver 2.0.0
string public constant version = "2.0.0";
```

However, the `version` passed to `EIP712` is `1`, as seen from the second argument in the constructor, instead of the correct version:

```
constructor(
    // ...
)
    EIP712("DeputyPauseModule", "1")
{
```

As a result, the domain separator for all signatures related to `DeputyPauseModule` will contain the wrong version.

Recommendation: Pass the `version` string into `EIP712` instead of `1`:

```
  constructor(
      // ...
  )
-     EIP712("DeputyPauseModule", "1")
+     EIP712("DeputyPauseModule", version)
  {
```

Optimism: Acknowledged. Since we always redeploy `DeputyPauseModule` when making changes, the version string here is essentially cosmetic. It would be important if we deployed `DeputyPauseModule` behind a proxy, but we do not intend to do that.

Spearbit: Acknowledged.

5.1.5 `ProxyAdminOwnedBase.proxyAdmin()` risk for future contracts

Severity: Low Risk

Context: ProxyAdminOwnedBase.sol#L41-L72

Description: `ProxyAdminOwnedBase.proxyAdmin()` fetches the proxy admin for `L1CrossDomainMessenger` by manually checking the first two mappings in `ResolvedDelegateProxy`:

```
if (
    Storage.getBytes32(keccak256(abi.encode(address(this), uint256(0))))
        != bytes32(
            uint256(bytes32("OVM_L1CrossDomainMessenger")) |
            ↪  uint256(bytes("OVM_L1CrossDomainMessenger").length * 2)
        )
) {
    revert ProxyAdminOwnedBase_NotResolvedDelegateProxy();
}

// Ok, now we'll try to read the AddressManager slot.
address addressManagerAddress = Storage.getAddress(keccak256(abi.encode(address(this), uint256(1))));
if (addressManagerAddress != address(0)) {
    return IProxyAdmin(IAddressManager(addressManagerAddress).owner());
}
```

More specifically, it checks that `implementationName[address(this)]` is "OVM_L1CrossDomainMessenger" and fetches the address manager at `addressManager[address(this)]`.

However, if a future (possibly new) contract:

1. Does not store the `ProxyAdmin` address at `Constants.PROXY_OWNER_ADDRESS`.
2. Has address mappings at its first and second slot.

These checks could mistakenly pass and return a wrong address.

Additionally, note that such an implementation means `L1CrossDomainMessenger` must always be named `OVM_L1CrossDomainMessenger` in `ResolvedDelegateProxy` for all chains.

Recommendation: Consider explicitly documenting the proxies which are compatible with `ProxyAdminOwnedBase`. Additionally, ensure that future contracts are not deployed behind a `ResolvedDelegateProxy`.

Optimism: Documented in commit c637130.

Spearbit: Verified, the risk has been documented in the comments.

5.2 Gas Optimization

5.2.1 Saved storage slot in `AnchorStateRegistry.sol`

Severity: Gas Optimization

Context: AnchorStateRegistry.sol#L181-L190

Description: This adds `asr` to the stack just to use it in the return assessment.

Recommendation: This is could be gas optimized as the storage slot is only used in the return as part of the comparison. Trade off is this is will make the code and comments harder to read something like:

```
// Return whether the game is factory registered and uses this AnchorStateRegistry. We
// check for both of these conditions because the game could be using a different
// AnchorStateRegistry if the registry was updated at some point. We mitigate the risks of
// an outdated AnchorStateRegistry by invalidating all previous games in the initializer of
// this contract, but an explicit check avoids potential footguns in the future.
return address(factoryRegisteredGame) == address(_game)
       &&
       // Grab the AnchorStateRegistry from the game. Awkward type conversion here but
       // IDisputeGame probably needs to have this function eventually anyway.
       address(IFaultDisputeGame(address(_game)).anchorStateRegistry()) == address(this);
```

Optimism: Acknowledged, but won't fix, readability is worth more than the gas in this case.

Spearbit: Agreed.

5.3 Informational

5.3.1 `FaultDisputegame.claimCredit()` cannot be called while the system is paused

Severity: Informational

Context: FaultDisputeGame.sol#L1005-L1007, FaultDisputeGame.sol#L957-L961

Description: In `FaultDisputeGame`, `closeGame()` now reverts if the `AnchorStateRegistry` is currently paused:

```
if (ANCHOR_STATE_REGISTRY.paused()) {
    revert GamePaused();
}
```

However, `claimCredit()` also calls `closeGame()`. As such, `claimCredit()` also cannot be called while the system is paused for games where the bond distribution mode has already been determined, which is an unintended consequence of this change. For example:

- Assume the system is unpaused.
- `closeGame()` is called for a dispute game and `bondDistributionMode` is set to `BondDistributionMode.NORMAL`.
- Guardian pauses the system.

Recommendation: Move the pause check to after the early return from checking `bondDistributionMode`:

```
- if (ANCHOR_STATE_REGISTRY.paused()) {
-     revert GamePaused();
- }

  // If the bond distribution mode has already been determined, we can return early.
  if (bondDistributionMode == BondDistributionMode.REFUND || bondDistributionMode ==
  ↪  BondDistributionMode.NORMAL)
  {
      // We can't revert or we'd break claimCredit().
      return;
  } else if (bondDistributionMode != BondDistributionMode.UNDECIDED) {
      // We shouldn't get here, but sanity check just in case.
      revert InvalidBondDistributionMode();
  }

+ if (ANCHOR_STATE_REGISTRY.paused()) {
+     revert GamePaused();
+ }
```

This allows `claimCredit()` to be called if `bondDistributionMode` is already determined, even when the system is paused (i.e. the example above).

Optimism: Fixed in commit 3ad1915.

Spearbit: Verified, the recommended fix was implemented.

5.3.2 Reduce duplicate code and follow "***internal modifer***" pattern in `AnchorStateRegistry`

Severity: Informational

Context: AnchorStateRegistry.sol#L125-L146

Description: These three functions all contain: `if (msg.sender != systemConfig.guardian()) revert AnchorStateRegistry_Unauthorized();` which could be moved to a helper function similar to `_assertOnlyProxyAdminOwner()` in OptimismPortal2.sol.

Optimism: Fixed in commit c7c974d1.

Spearbit: Fixed.

5.3.3 Inconsistent variable declaration patterns with upgradeability

Severity: Informational

Context: L1ERC721Bridge.sol#L28-L40

Description: Several contract use different patterns (some with `constants` and `immutables` first and some with them after storage variables. However, the `L1ERC721Bridge` mixes a constant in the middle of the storage declarations. This complicates upgrading the contract and leads to potential memory corruption with future upgrades.

Another example of this is the `internal` variable in the middle of `public` variables in AnchorStateRegistry.sol.

Recommendation: Move the `constant` above the `mapping` or below the `systemConfig`. Ideally, it would be good to establish a style guide for how upgradable contracts should have their variables laid out.

Optimism: Acknowledged, we will fix this in a future cleanup of the contracts.

Spearbit: Acknowledged.

5.3.4 `OPContractsManager` Contract deployment naming inconsistency

Severity: Informational

Context: OPContractsManager.sol#L675, OPContractsManager.sol#L717

Description: There is some inconsistency with naming and version suffixes in deploying contracts in the OPCM.

Recommendation: Establish a pattern that can be used consistently for these suffixes so there is a clear versioning system within the contract names.

Optimism: Fixed in commit 62dcc8a8.

Spearbit: Fixed.

5.3.5 Contracts Manager could better optimize use of `encodeETHLockboxInitializer`

Severity: Informational

Context: OPContractsManager.sol#L728-L733, OPContractsManager.sol#L1238-L1249

Description: The `upgrade` function does not take advantage of the `encodeETHLockboxInitializer` helper function duplicating code and potentially leading to a divergence if future updates are not make in both places. Further, it appears that the `encodeETHLockboxInitializer` function is only ever used with one portal in the array, so it could remove some pressure from the stack to move this:

```
IOptimismPortal[] memory portals = new IOptimismPortal[](1);
portals[0] = optimismPortal;
```

Into the function itself and only pass the optimism portal.

Recommendation: Consider using the helper function in `upgrade` and moving the array construction in both places to that function.

Optimism: Acknowledged.

Spearbit: Acknowledged.

5.3.6 `OptimismPortal2.upgrade()` sets the wrong state variables

Severity: Informational

Context: OptimismPortal2.sol#L296-L300

Description: `OptimismPortal2.upgrade()` sets the following state variables:

```
// Now perform upgrade logic.
anchorStateRegistry = _anchorStateRegistry;
ethLockbox = _ethLockbox;
systemConfig = _systemConfig;
spacer_53_1_20 = address(0);
```

However, `systemConfig` does not need to be set as it is already set in the current `OptimismPortal2` that is live. Additionally, the following spacers are not cleared:

- `spacer_56_0_20`, previously the `disputeGameFactory` address
- `spacer_58_0_32`, previously the `disputeGameBlacklist` mapping
- `spacer_59_0_4`, previously `respectedGameType`
- `spacer_59_4_8`, previously `respectedGameTypeUpdatedAt`

Recommendation: Consider modifying `upgrade()` as suggested above:

```
  // Now perform upgrade logic.
  anchorStateRegistry = _anchorStateRegistry;
  ethLockbox = _ethLockbox;
- systemConfig = _systemConfig;
  spacer_53_1_20 = address(0);
+ spacer_56_0_20 = address(0);
+ spacer_58_0_32 = bytes32(0)
+ spacer_59_0_4 = GameType.wrap(0);
+ spacer_59_4_8 = 0;
```

Optimism: Fixed in commit 3f77c1b.

Optimism: Verified, `OptimismPortal2.upgrade()` has been modified to not set `systemConfig`. Additionally, spacers are no longer cleared in all `upgrade()` functions.

5.3.7 `SuperchainConfig.extend()` can be used to pause a chain

Severity: Informational

Context: SuperchainConfig.sol#L136-L145

Description: `SuperchainConfig.extend()` does not check that `_identifier` is already paused:

```
function extend(address _identifier) external {
    // Only the Guardian can extend the pause.
    if (msg.sender != guardian) {
        revert SuperchainConfig_OnlyGuardian();
    }

    // Reset the pause timestamp.
    pauseTimestamps[_identifier] = block.timestamp;
    emit Paused(_identifier);
}
```

This allows the guardian to call `extend()` instead of `pause()` to pause an identifier. While there technically is no issue with the guardian using `extend()` instead of `pause()`, it would be good to ensure `extend()` can only be used to extend an existing pause.

Recommendation: Add the following check to `extend()`:

```
if (pauseTimestamps[_identifier] == 0) {
    revert SuperchainConfig_NotPaused(_identifier);
}
```

Optimism: Fixed in commit 1891f66.

Spearbit: Verified, the recommended fix was implemented.

5.3.8 Minor issues with code and comments

Severity: Informational

Context: (*See each case below*).

Description/Recommendation:

1. SuperchainConfig.sol#L157 -- `paused()` can be declared external.
2. SystemConfig.sol#L213 -- Unnecessary cast of `_superchainConfig` to `ISuperchainConfig`:

```
- superchainConfig = ISuperchainConfig(_superchainConfig);
+ superchainConfig = _superchainConfig;
```

3. SystemConfig.sol#L230-L232 -- Consider clearing slots using `Storage.setBytes32()` to ensure the entire slot is cleared. This would also be consistent with how slots are cleared in `SuperchainConfig`:

```
- Storage.setAddress(disputeGameFactorySlot, address(0));
+ Storage.setBytes32(disputeGameFactorySlot, bytes32(0));
```

4. SystemConfig.sol#L278 -- The code here can be simplified:

```
- IOptimismPortal2 portal = IOptimismPortal2(payable(Storage.getAddress(OPTIMISM_PORTAL_SLOT)));
+ IOptimismPortal2 portal = IOptimismPortal2(payable(optimismPortal()));
```

5. DeputyPauseModule.sol#L149 -- This comment as incorrect as `DeputyPauseModule.pause()` no longer calls the Foundation Safe.

6. ProxyAdminOwnedBase.sol#L57-L59 -- Consider making the encoded `L1CrossDomainMessenger`'s name a constant, for example:

```
bytes32 private constant L1CDM_IMPLEMENTATION_NAME =
0x4f564d5f4c3143726f7373446f6d61696e4d657373656e676572000000000034
```

7. ProxyAdminOwnedBase.sol#L80 -- Typo, there is an extra + at the end of the comment.

8. DisputeGameFactory.sol#L122-L123 -- This comment is incorrect, `DisputeGameFactory.gameAtIndex()` does not return `address(0)` for non-existent proxies.

9. OptimismPortal2.sol#L675-L678 -- Consider calling `EthLockbox.lockETH()` only when `_tx.value` is non-zero:

```
  // Send ETH back to the Lockbox or it'll get stuck here.
- if (!success) {
+ if (!success && _tx.value > 0) {
      ethLockbox.lockETH{ value: _tx.value }();
  }
```

10. ProxyAdminOwnedBase.sol#L74-L81 -- Consider documenting in the comments that multiple OP stack chains could have the same `ProxyAdmin` owner address. Since this isn't obvious when looking at the `ProxyAdminOwnedBase` contract, this could become a potential footgun in the future. For example, when calling `_assertSharedProxyAdminOwner()`, passing the address of a proxy from another chain would pass if both chains share the same `ProxyAdminOwner`.

11. OptimismPortal2.sol#L159-L169 -- Consider indexing addresses for the event to make it easier to query (at least the two new addresses).

12. OptimismPortal2.sol#L318-L322 -- `superchainConfig()` can be set to `external` and might need a `@custom:legacy` notation.

13. OptimismPortal2.sol#L675 -- This comment is misleading. ETH can always be recovered with the new `migrateLiquidity` function.

14. OptimismPortal2.sol#L324-L328 -- Guardian legacy function could be `external`.

15. OPContractsManager.sol#L1920-L1921 -- Incorrect comment on the `migrate` function about being a "stub".

16. OPContractsManager.sol#L626-L627 -- It seemed to me with a little testing that it would be possible to cache the config without a `stack too deep` error which would optimize the code some: `OPContractsManager.OpChainConfig memory config = _opChainConfigs[i];`.

Optimism:

1. Fixed in commit 4fc9181a.
2. Fixed in commit 4fc9181a.
3. Fixed in commit 4fc9181a.
4. Fixed in commit 4fc9181a.

5. Fixed in commit 4fc9181a.
6. Acknowledged.
7. Acknowledged.
8. Fixed in commit 4fc9181a.
9. Fixed in commit 4fc9181a.
10. Fixed in commit 4fc9181a.
11. Acknowledged.
12. Fixed in commit 4fc9181a.
13. Fixed in commit 4fc9181a.
14. Fixed in commit 4fc9181a.
15. Fixed in commit 4fc9181a.
16. Acknowledged.

Spearbit: Verified fixes and acknowledged the rest.