Optimism u16 Security Review

Auditors

MiloTruck, Lead Security Researcher

R0bert, Lead Security Researcher

Report prepared by: Lucas Goiriz

October 7, 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 u16 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 2 days in total, OP Labs engaged with Spearbit to review the optimism protocol. In this period of time a total of **10** issues were found.

Summary

Project Name	OP Labs
Repository	optimism
Commit	47580169
Type of Project	Infrastructure, L2
Audit Timeline	Sep 9th to Sep 11th

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	3	1	2
Gas Optimizations	0	0	0
Informational	7	1	6
Total	**10**	**2**	**8**

5 Findings

5.1 Low Risk

5.1.1 Semver gate ignores prerelease metadata, blocking RC -> release SuperchainConfig upgrades

Severity: Low Risk

Context: OPContractsManager.sol#L934-L941

Description: The upgrade gate in the `SuperchainConfig` upgrader treats prerelease versions (e.g., "3.0.0-1") as equal to their corresponding release ("3.0.0") because the `SemverComp` parser drops prerelease/build metadata. The parser splits on "-" and "+" and only compares numeric major.minor.patch, so "3.0.0-1" and "3.0.0" both parse to 3.0.0:

```
function parse(string memory _semver) internal pure returns (Semver memory) {
    string[] memory parts = LibString.split(_semver, ".");
    string[] memory patchParts = LibString.split(parts[2], "-");
    string[] memory patchParts2 = LibString.split(patchParts[0], "+");
    return Semver({
        minor: JSONParserLib.parseUint(parts[1]),
        patch: JSONParserLib.parseUint(patchParts2[0])
    });
}
```

The upgrader gate reverts when the current `SuperchainConfig` version is greater-than-or-equal to the target impl's version:

```
if (SemverComp.gte(
        _superchainConfig.version(),
        ISuperchainConfig(getImplementations().superchainConfigImpl).version()
)) {
    revert OPContractsManagerUpgrader_SuperchainConfigAlreadyUpToDate();
}
```

Because prereleases compare numerically equal to releases, an intended upgrade from for example "3.0.0-1" (Bedrock pre-release) to "3.0.0" (Bedrock) will be blocked with an `OPContractsManagerUpgrader_SuperchainConfigAlreadyUpToDate()` error.

Therefore, RC → release upgrades can be improperly blocked, causing operational delays and requiring workarounds.

Recommendation: Add an explicit "RC step" flag to `upgradeSuperchainConfig` (e.g., `rcUpgrade`) that relaxes the equality block only for the RC -> release path. When `rcUpgrade` is `true`, use a strict "greater than" comparison. When `false`, keep the current "greater-than-or-equal" behavior.

OP Labs: Acknowledged. We no longer use release candidates in our process so this failure mode would not apply at the moment.

Spearbit: Acknowledged.

5.1.2 Unnecessary `DisputeGame`/`DelayedWETH` redeploy when ASR is not new (U16 → U16a)

Severity: Low Risk

Context: OPContractsManager.sol#L846-L847

Description: In the U16 → U16a upgrade context the `AnchorStateRegistry` is not "new", yet the upgrade flow proceeds to redeploy a fresh `DelayedWETH` and a new `DisputeGame` implementation, while a comment suggests this is because "the AnchorStateRegistry is new".

In the U16a upgrader, the redeploy step happens unconditionally even when the `AnchorStateRegistry` was successfully retrieved from the existing portal and only its implementation was upgraded. As a result, on U16 → U16a

upgrades the system deploys a new DelayedWETH proxy and resets the game implementation in the DisputeGameFactory, although the ASR reference didn't change. This creates address churn and fragments bond/liquidity across multiple DelayedWETH instances without a functional need tied to a "new" AnchorStateRegistry. It also increases operational overhead as there are new addresses to track, new validator configs to update and gas usage for no change in semantics tied to AnchorStateRegistry.

Recommendation: Gate the redeploy behind concrete changes that require it, instead of assuming that the "ASR is new":

- Only deploy a new DisputeGame implementation if any of the constructor parameters differ from the current game's params (e.g., vm changed, prestate override provided, or ASR address actually changed).
- Only create a new DelayedWETH if policy requires new instances per major change. Otherwise reuse the existing DelayedWETH (and skip the deploy/initialize).
- Update the misleading comment to reflect the actual condition (e.g., "Redeploy games when constructor params change (ASR/vm/prestate/WETH).").

OP Labs: Fixed in PR 17406.

Spearbit: Fix verified.

5.1.3 Chain can be accidentally unpaused when changing the ETH_LOCKBOX feature

Severity: Low Risk

Context: SystemConfig.sol#L526-L534

Description: In SystemConfig.paused(), the identifier used to check if the chain is paused is determined based on whether the ETH_LOCKBOX feature is enabled:

```
function paused() public view returns (bool) {
    // Determine the appropriate chain identifier based on the feature flags.
    address identifier = isFeatureEnabled[Features.ETH_LOCKBOX]
        ? address(IOptimismPortal2(payable(optimismPortal())).ethLockbox())
        : address(optimismPortal());

    // Check if either global or local pause is active.
    return superchainConfig.paused(address(0)) || superchainConfig.paused(identifier);
}
```

However, this implementation makes it possible to accidentally unpause a chain when enabling/disabling the ETH_LOCKBOX feature.

For example, assume that:

- A chain currently has ETH_LOCKBOX disabled.
- The chain is currently paused (i.e. the OptimismPortal2 address is set to paused in SuperchainConfig).

If ETH_LOCKBOX is enabled, the chain's pause identifier changes to the ETHLockbox address, which isn't set to paused in SuperchainConfig. As such, the chain is unpaused without actually calling unpause().

Recommendation: A potential solution would be to check if the chain is paused in setFeature() and revert. However, note that this removes the ability to disable features while the chain is paused, which might be needed (e.g. in case a feature has a bug).

OP Labs: Acknowledged. For U16a we are going to address this with better documentation. For U17 we might add some code in the SystemConfig to prevent the footgun/be more explicit about this concern.

Spearbit: Acknowledged.

5.2 Informational

5.2.1 Disabling `ETH_LOCKBOX` once enabled strands funds, splits liquidity, breaks withdrawals and future upgrades

Severity: Informational

Context: SystemConfig.sol#L503

Description: `SystemConfig` exposes `setFeature(bytes32 _feature, bool _enabled)`, allowing `ETH_LOCKBOX` to be disabled after a chain has migrated to use an `ETHLockbox`. In `OptimismPortal2` (u16a), deposits and withdrawals use the lockbox only when `_isUsingLockbox()` is true, defined as `systemConfig.isFeatureEnabled(Features.ETH_LOCKBOX) && address(ethLockbox) != address(0)`. Disabling the flag while a nonzero `ethLockbox` remains causes multiple inconsistencies.

First, liquidity splits. All ETH previously locked in the `ETHLockbox` remains there with no runtime path to "pull" it back into the portal, while new deposits stop calling `ethLockbox.lockETH` and therefore accrue to the portal's own balance. This bifurcates the L1 backing between the lockbox and the portal.

Second, withdrawals requiring ETH can fail irreversibly. `finalizeWithdrawalTransaction` marks the withdrawal as finalized before attempting the value transfer:

```
finalizedWithdrawals[withdrawalHash] = true;
bool success = SafeCall.callWithMinGas(_tx.target, _tx.gasLimit, _tx.value, _tx.data);
```

When `ETH_LOCKBOX` is disabled, the portal no longer calls `ethLockbox.unlockETH` to top up its balance before sending value. If the portal's own balance is insufficient, the SafeCall fails, yet the withdrawal stays permanently finalized with `success=false`, and cannot be retried. Users then do not receive their ETH on L1 without manual intervention. Third, governance/operational upgrades are impeded. The portal's upgrade(`IAnchorStateRegistry`) enforces `_assertValidLockboxState()`, which reverts if the feature is disabled while `ethLockbox != address(0)`. This blocks future portal upgrades in that inconsistent state. At the same time, there is no reverse migration primitive to drain the lockbox back into the portal. `ETHLockbox.migrateLiquidity` only moves funds between lockboxes, and `unlockETH` is only callable by the portal during a withdrawal flow.

Recommendation: Prevent this inconsistent state by adding an explicit, safe "decommission" path in the `setFeature` function. A simple, robust guard to disallow turning off `ETH_LOCKBOX` once enabled could be:

```
if (_feature == Features.ETH_LOCKBOX && !_enabled) {
    revert SystemConfig_CannotDisableEthLockbox();
}
```

OP Labs: Acknowledged. Not changing anything for the moment. We are aware of this vector.

Spearbit: Acknowledged.

5.2.2 Stale comment in OP2 suggests mode flag for Output Roots that no longer exists

Severity: Informational

Context: OptimismPortal2.sol#L342-L343

Description: In `OptimismPortal2`, the docstring above the Output Root proving function still states: "Only callable when the OptimismPortal is using Output Roots (superRootsActive flag is false)." In the current design, Super Roots support and the `superRootsActive` flag were moved to `OptimismPortalInterop`, `OptimismPortal2` always verifies withdrawals with an Output Root proof and no longer gates this path on any runtime "mode" flag. The function implementation in OP2 performs no `superRootsActive` check and directly verifies using Output Roots, so the comment is misleading and can cause some confusion about the expected behavior and prerequisites.

Recommendation: Update the comment to reflect the current architecture, for example: "Proves a withdrawal transaction using an Output Root proof. OP2 only supports Output Roots; Super Roots proofs are supported by OptimismPortalInterop.".

OP Labs: Fixed in PR 17482.

Spearbit: Fix verified.

5.2.3 Multi-chain upgrader assumes ProxyAdmin ownership and proxy admin alignment

Severity: Informational

Context: OPContractsManager.sol#L2039-L2048

Description: `OPContractsManager.upgrade` enforces `delegatecall` and then delegatecalls `OPContractsManagerUpgrader.upgrade`, which in turn executes `ProxyAdmin.upgrade/upgradeAndCall` for each target proxy. Because `ProxyAdmin` methods are `onlyOwner`, `msg.sender` (the Upgrade Controller Safe) must be the owner of every chain's `ProxyAdmin`. Furthermore, each target proxy must be administered by that same `ProxyAdmin`, otherwise the internal call to `IProxy.upgradeTo` will revert because the `ProxyAdmin` is not the proxy's admin.

If either assumption has drifted on any chain (e.g., `ProxyAdmin.owner()` changed, or a proxy's admin was switched away from that `ProxyAdmin`), the first offending upgrade reverts and the entire multi-chain upgrade transaction aborts.

Recommendation: Merely informative. Consider updating your documentation to reflect this restriction/requirement.

OP Labs: Acknowledged. This is sort of inherent to how we do our upgrades. We will consider documenting it better.

Spearbit: Acknowledged.

5.2.4 UpgradeTo relies on local `ProxyAdmin.upgrade`; not compatible with OZ v5 `ProxyAdmin`

Severity: Informational

Context: OPContractsManager.sol#L955-L961

Description: The internal upgrade helper calls the ProxyAdmin's legacy upgrade method:

```
function upgradeTo(IProxyAdmin _proxyAdmin, address _target, address _implementation) internal {
    assertValidContractAddress(_implementation);
    _proxyAdmin.upgrade(payable(address(_target)), _implementation);
}
```

This is compatible with the repository's `ProxyAdmin` (`src/universal/ProxyAdmin.sol`), which exposes `upgrade(address,address)`. However, OpenZeppelin ProxyAdmin v5.0.0+ removed `upgrade()` and requires `upgradeAndCall(proxy, implementation, data)`. If a deployment ever swaps to a different ProxyAdmin (e.g., OZ v5) or mixes admin types, this helper will revert or fail at compile time. The pattern also implicitly assumes the repo's `ProxyAdmin` for legacy proxy types (`CHUGSPLASH/RESOLVED`), which OZ's `ProxyAdmin` doesn't support.

Recommendation: Ensure all the deployed contracts are behind the intended `ProxyAdmin` (`src/universal/ProxyAdmin.sol`) contract.

OP Labs: Acknowledged. No plans to use any alternative `ProxyAdmin` so this is a non-issue for the moment.

Spearbit: Acknowledged.

5.2.5 Direct U15 → U16a prevents ever enabling ETHLockbox on OP2

Severity: Informational

Context: OptimismPortal2.sol#L123-L127

Description: `OptimismPortal2` v4.1.0 (U16a) retains the storage slot for `ethLockbox` but provides no method to set it. The comment explicitly states this: "as of v4.1.0 it is not possible to set this value in storage. Chains that skip v4.0.0 will not have any `ETHLockbox` set here." In the U16a upgrader path, pre-U16 portals are upgraded via `upgrade(anchorStateRegistry)` (single-arg) and the feature flag is not enabled when `ethLockbox` can't be read

on the old portal. Consequently, a chain that upgrades directly from U15 to U16a will end up with `ethLockbox == address(0)` and no way to set it later.

Enabling the `ETH_LOCKBOX` feature afterward is impossible because portal invariants require a nonzero `ethLockbox` whenever the feature is enabled. Because of this, chains that take U15 -> U16a can never adopt an `ETHLockbox` on OP2 without performing an intermediate U16 upgrade step.

Recommendation: Document and enforce the intended paths:

- If `ETHLockbox` is desired on OP2, require U15 → U16 (v4.0.0) first, where upgrade accepts an `IETHLockbox`, then U16 → U16a.
- Alternatively, offer an explicit one-time, onlyProxyAdmin/ownergated setter in OP2 v4.1.0 to set `ethLockbox` when it is currently zero.

OP Labs: Acknowledged. This is expected behavior for now.

Spearbit: Acknowledged.

5.2.6 U15 → u16a, 16 → 16a and u16a deploy: Parity, safety and verification gaps across upgrade/deploy paths

Severity: Informational

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

Description: During the review, three forkbased test cases were authored and executed to validate the u16a upgrade and deployment flows endtoend:

1. U15 → u16a direct upgrade.
2. U16 → u16a upgrade.
3. Fresh u16a deployment.

The corresponding artifacts are available here:

1. u15 → u16a upgrade.
2. u16 → u16a upgrade.
3. u16a deployment.

The u16a deployment test consumes ~13.15M gas, which should be acceptable operationally but is noteworthy for planning and cost predictability.

On the other hand, across the three paths, core behavior is consistent with design: the upgrade controller coordinates proxy upgrades through `ProxyAdmin`, the `DisputeGameFactory` rotates the `PermissionedDisputeGame` implementation and `SystemConfig`/`OptimismPortal`/L1 bridges and messenger are reinitialized on the target release. However, several issues emerge that are worth addressing to eliminate ambiguity and reduce operational risk.

First, the u15 → u16a flow necessarily deploys a new `AnchorStateRegistry` (ASR) proxy and repoints the `OptimismPortal` to that new proxy because the v15 portal does not expose an explicit ASR reference. The calltrace shows the new ASR being properly initialized and `OptimismPortal` upgraded to reference it. The legacy u15 ASR proxy remains deployed and unused. While functionally safe, this introduces an orphaned proxy that can confuse offchain diagnostics and operations that still reference the legacy ASR address. In the u15 test we added a postupgrade assertion to ensure `OptimismPortal2.anchorStateRegistry` returns the new proxy rather than the legacy one, but this operational reality should also be reflected in documentation so downstream systems do not read stale anchor data.

Second, the u15 → u16a trace shows `SuperchainConfig` being upgraded and initialized to the unified u16a config before any chainlocal upgrades, satisfying the upgrader's precondition. There is no corresponding `ProtocolVersions` upgrade in that path. If, as appears to be the case, `ProtocolVersions` is unchanged between u15, u16, and u16a, that is acceptable. However, consider documenting it so that operators do not expect a `ProtocolVersions` bump in this particular upgrade.

Third, the upgrade controller's authority is a hard precondition. All upgrade actions are routed through `ProxyAdmin.onlyOwner`. In the tests we preserved `msg.sender` across delegatecalls (to model execution by a Safe) so that `ProxyAdmin` permissions are satisfied. In production, if the upgrader is executed by a nonowner context, all upgrade calls will revert. A good improvement could be that the upgrade process asserted that the caller is the `ProxyAdmin` owner for every proxy before enacting state changes.

Finally, the u16a upgrader rebuilds the new `PermissionedDisputeGame`'s constructor parameters by reading from the existing PDG (including vm). This is correct when the source PDG is properly configured, but it could also propagate any misconfiguration silently.

Contracts deployed & upgraded:

u15 $\rightarrow$ u16a: New deployments:

- `AnchorStateRegistry` proxy: fresh proxy deployed during upgrade ("AnchorStateRegistry-U16"), `OptimismPortal` now points to this new ASR. The legacy u15 ASR proxy remains deployed but unused.
- No new `ETHLockbox` proxy (interop feature disabled in this run).

Upgraded proxies:

- `SystemConfig`: upgraded to u16a impl and reinitialized via `upgrade(l2ChainId, superchainConfig)`.
- `OptimismPortal2`: upgraded to u16a (noninterop path) and pointed at the new ASR via `upgrade(anchorStateRegistry)`.
- `L1CrossDomainMessenger` (`ResolvedDelegateProxy`): implementation switched via `AddressManager` and reinitialized.
- `L1StandardBridge` (`Chugsplash`): implementation upgraded and reinitialized.
- `L1ERC721Bridge`: implementation upgraded and reinitialized.
- `AnchorStateRegistry`: new proxy deployed and initialized (see above); the old proxy was not upgraded.

Not upgraded (in this path):

- `ProtocolVersions`: unchanged (consistent with u15/u16/u16a parity).
- `OptimismMintableERC20Factory`, `DelayedWETH`: not touched by the upgrader in this flow.

u16 $\rightarrow$ u16a: New deployments:

- None in the provided run (interop feature disabled). The upgrader reused the existing ASR and left the existing ETHLockbox proxy in place.
- Note: If the interop dev feature is enabled, the upgrader will deploy a new `ETHLockbox` proxy ("ETHLockboxU16a"), upgrade the portal to the interop impl, authorize the new lockbox and migrate liquidity from the portal. That did not occur in the test run.

Upgraded proxies:

- `SystemConfig`: upgraded to u16a impl and reinitialized via `upgrade(l2ChainId, superchainConfig)`.
- `OptimismPortal2`: upgraded to u16a (noninterop path) and repointed to the ASR via `upgrade(anchorStateRegistry)`.
- `L1CrossDomainMessenger` (`ResolvedDelegateProxy`): implementation switched via `AddressManager` and reinitialized.
- `L1StandardBridge` (`Chugsplash`): implementation upgraded and reinitialized.
- `L1ERC721Bridge`: implementation upgraded and reinitialized.
- `DisputeGameFactory`: implementation upgraded. PDG implementation rotated to the new PDG.
- `AnchorStateRegistry`: upgraded in place to the u16a impl (no new proxy).

OP Labs: Acknowledged. No action items were identified here.

Spearbit: Acknowledged.

5.2.7 Minor code improvement in `SystemConfig.setFeature()`

Severity: Informational

Context: SystemConfig.sol#L507-L511

Description: The following check in `SystemConfig.setFeature()` can be simplified:

```
  // As a sanity check, prevent users from enabling the feature if already enabled or
  // disabling the feature if already disabled. This helps to prevent accidental misuse.
- if ((_enabled && isFeatureEnabled[_feature]) || (!_enabled && !isFeatureEnabled[_feature])) {
+ if (_enabled == isFeatureEnabled[_feature]) {
      revert SystemConfig_InvalidFeatureState();
  }
```

OP Labs: Acknowledged. We will address this in the U17 implementation.

Spearbit: Acknowledged.

6 Appendix

6.1 File hashes

The following files were reviewed in the present engagement:

File	keccak256 digest
MIPS64State.sol	`0x7ff822ef2029fab4ae481f21dd9ce340603536b5c81e7d26dfb69b268c32f5c4`
MIPS64Syscalls.sol	`0xbf3cc1e7f22a67edc592564a0ef64f0e2b3f2b8aa34e69a58797534074a4d305`
MIPS64.sol	`0x0ab0b916807cf420b097335dddd03edf03b43a80674c14dc14cc45c449e8e742`
L1CrossDomainMessenger.sol	`0x91e5bb6dfb48bfc893293bded067d8b878cc6b906376edfb62835d33516b2710`
L1ERC721Bridge.sol	`0x260ba20d5820d22695ba3f79bb3b245d5d778a3a63af8b27e80cd70499ed3240`
L1StandardBridge.sol	`0x46cfe69fa1d0e2c7427cceea2fca67b1058a3a0b3e247c10ca87ea2dff49597d`
OptimismPortal2.sol	`0xfe308830f3096aadc55590f3ddf93638e2282c0bc9a5edb527a119e1d9051116`
SystemConfig.sol	`0x1f1cddfb85ba45ef007f73dd9f123d99cb9ccbc11247acb45238214db89d9de6`
OPContractsManager.sol	`0xd4243363204ffe984ed6319cd2e415821a3c287831ee3c653ed1ce6cd808e992`