Optimism: Custom Gas Token

Security Review

Cantina Managed review by:
R0bert, Lead Security Researcher
Sujith Somraaj, Lead Security Researcher

November 18, 2025

Contents

1 Introduction **2**
- 1.1 About Cantina . . . 2
- 1.2 Disclaimer . . . 2
- 1.3 Risk assessment . . . 2
  - 1.3.1 Severity Classification . . . 2

2 Security Review Summary **3**
- 2.1 Scope . . . 3

3 Findings **4**
- 3.1 Low Risk . . . 4
  - 3.1.1 Missing CGT predeploy mirrors . . . 4
  - 3.1.2 CGT mode strands `donateETH` funds . . . 4
  - 3.1.3 Missing zero-value validation in `burn()` function . . . 5
  - 3.1.4 OptimismPortal2 lacks isCustomGasToken flag . . . 5
  - 3.1.5 `NativeAssetLiquidity` lacks `fund()` function . . . 6
  - 3.1.6 `LiquidityController` events off-spec . . . 6
- 3.2 Gas Optimization . . . 6
  - 3.2.1 Nested `if`-statements can be combined to save gas . . . 6
  - 3.2.2 Inefficient boolean comparison in `setCustomGasToken()` function . . . 7
- 3.3 Informational . . . 7
  - 3.3.1 Split CGT flags can cause locked funds . . . 7
  - 3.3.2 CLI truncates large init bond values . . . 8
  - 3.3.3 LegacyERC20 name lookup reverts . . . 8
  - 3.3.4 CLI can't enable V2 dispute games . . . 9
  - 3.3.5 Redundant indexed parameter `caller` in events . . . 10
  - 3.3.6 CGT override bypasses intent validation . . . 10
  - 3.3.7 Superchain Registry queries invalid CGT ABIs . . . 11
  - 3.3.8 Superchain Registry staging drops CGT metadata . . . 12
  - 3.3.9 Unnecessary `ETHLockbox` deployment for custom gas token chains . . . 12

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

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 are 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 Nov 2nd to Nov 6th the Cantina team conducted a review of optimism on commit hash 1f888ede. The team identified a total of **17** issues:

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	0	0	0
Low Risk	6	5	1
Gas Optimizations	2	0	2
Informational	9	1	8
Total	**17**	**6**	**11**

2.1 Scope

The security review had the following components in scope for optimism on commit hash 1f888ede:

```
packages/contracts-bedrock
├── scripts
│   ├── deploy
│   │   ├── DeployConfig.s.sol
│   │   └── DeployOPChain.s.sol
│   └── L2Genesis.s.sol
└── src
    ├── L1
    │   ├── OPContractsManager.sol
    │   ├── OptimismPortal2.sol
    │   └── SystemConfig.sol
    └── L2
        ├── L1BlockCGT.sol
        ├── L2ToL1MessagePasserCGT.sol
        ├── LiquidityController.sol
        └── NativeAssetLiquidity.sol
```

3 Findings

3.1 Low Risk

3.1.1 Missing CGT predeploy mirrors

Severity: Low Risk

Context: addresses.go#L5-L47

Description: Two Go mirrors of the predeploy address table were never updated when the custom gas token contracts were introduced. `op-service/predeploys/addresses.go` still enumerates only the legacy constants and `init()` omits every CGT/superchain predeploy, so downstream tooling that depends on this package can neither discover nor label the new contracts. The same omission exists in `devnet-sdk/contracts/constants/constants.go`, whose exported `types.Address` values also stop at `SuperchainTokenBridge`. Tools that rely on either package like the `op-service` state dump and metadata handlers, tx-intent builders, and the devnet SDK registry, treat the new deployments as nonexistent, leaving custom-gas-token flows unusable even though the Solidity predeploys exist. The trimmed snippet below shows the stale address table in both mirrors:

```
const (
    L2ToL1MessagePasser   = "0x4200000000000000000000000000000000000016"
    // ...
    SuperchainTokenBridge = "0x4200000000000000000000000000000000000028"
    Create2Deployer       = "0x13b0D85CcB8bf860b6b79AF3029fCA081AE9beF2"
    // ...
    EntryPoint_v070       = "0x0000000071727De22E5E9d8BAf0edAc6f37da032"
)
```

Neither file defines `OptimismSuperchainERC20Factory (0x4200000000000000000000000000000000000026)`, `OptimismSuperchainERC20Beacon (0x4200000000000000000000000000000000000027)`, `NativeAssetLiquidity (0x4200000000000000000000000000000000000029)`, or `LiquidityController (0x420000000000000000000000000000000000002a)` and the `Predeploys` map never inserts them, so the entire CGT stack is invisible to any consumer of these mirrors.

Recommendation: Add the four missing address constants to both `op-service/predeploys/addresses.go` and `devnet-sdk/contracts/constants/constants.go`, then extend the `Predeploys` map to include each new predeploy so every tooling layer remains consistent with `Predeploys.sol`.

Optimism: Fixed in commit fd6865e9.

Cantina Managed: Fix verified.

3.1.2 CGT mode strands `donateETH` funds

Severity: Low Risk

Context: OptimismPortal2.sol#L327-L330

Description: When Custom Gas Token mode is enabled the portal defends against value-bearing withdrawals by rejecting any transaction that finalizes with nonzero ETH, yet the `donateETH` hook keeps accepting deposits that never forward to L2. This directly violates the Custom Gas Token spec, which states that `donateETH` MUST revert whenever `isCustomGasToken()` is true and `msg.value > 0` (see the Custom Gas Token spec). The relevant implementation is:

```
function donateETH() external payable {
    // Intentionally empty.
}
```

and the finalization path enforces the following gate whenever CGT is active:

```
function finalizeWithdrawalTransactionExternalProof(
    Types.WithdrawalTransaction memory _tx,
    address _proofSubmitter
)
```

```
    public
{
    _assertNotPaused();
    if (_isUsingCustomGasToken()) {
        if (_tx.value > 0) revert OptimismPortal_NotAllowedOnCGTMode();
    }
    // ...
}
```

As soon as CGT mode is switched on, ETH can still be pushed into the portal by calling `donateETH`, but any attempt to bridge that value out will revert during finalization because `_tx.value` is greater than zero. The deposited ether has no escape hatch, so every donation made under CGT effectively strands funds in the contract and inflates its balance without a supported withdrawal mechanism. In practice this produces permanent trapped value.

Recommendation: Consider blocking `donateETH` while CGT mode is active.

Optimism: Fixed in commit abad267.

Cantina Managed: Fix verified.

3.1.3 Missing zero-value validation in `burn()` function

Severity: Low Risk

Context: LiquidityController.sol#L99

Description: The `burn()` function in the `LiquidityController.sol` contract does not include validation to prevent zero-value burns. This contradicts the spec, which states that the function must revert when msg.value is zero.

Recommendation: Consider validating and reverting to zero value burns:

```
  function burn() external payable {
    // ....
+   if (msg.value == 0) revert LiquidityController_InvalidAmount();
    // ....
  }
```

Optimism: Acknowledged.

Cantina Managed: Acknowledged.

3.1.4 OptimismPortal2 lacks isCustomGasToken flag

Severity: Low Risk

Context: OptimismPortal2.sol#L222-L250

Description: The Custom Gas Token specification mandates that `OptimismPortal2` owns a native `isCustomGasToken` boolean which is set during `initialize` and exposed through a public getter so external components can attest to the chain's mode (see the Custom Gas Token spec). In `OptimismPortal2` the initializer only stores the `SystemConfig` and `AnchorStateRegistry` and every caller is forced to infer the mode indirectly via `_isUsingCustomGasToken()` which just forwards to `systemConfig.isFeatureEnabled(Features.CUSTOM_GAS_TOKEN)`. There is no portal-local storage or public `isCustomGasToken()` function, so any consumer following the spec cannot read the flag from the portal. This is a direct violation of the spec and breaks compatibility for tooling that depends on querying the portal for the flag.

Recommendation: Introduce a dedicated `bool private isCustomGasToken_;` storage slot on `OptimismPortal2`, set it during `initialize` (or migration) based on the deployment configuration and expose the required `function isCustomGasToken() external view returns (bool)` that returns that slot. Keep `_isUsingCustomGasToken()` as an internal helper if desired but ensure the portal presents the stateful getter expected by the spec.

Optimism: Fixed in commit cdc2aa0.

Cantina Managed: Fix verified.

3.1.5 `NativeAssetLiquidity` lacks `fund()` function

Severity: Low Risk

Context: NativeAssetLiquidity.sol#L16

Description: The Custom Gas Token predeploy spec requires `NativeAssetLiquidity` to expose a standalone `fund()` function callable by any address, reverting when `msg.value == 0` and emitting `LiquidityFunded` so operators can seed the pool during fresh deployments or migrations (see the Custom Gas Token spec). The implementation only provides `deposit()` and `withdraw()` and never declares either the `fund()` entry point or the accompanying `LiquidityFunded` event. As a result there is no supported on-chain primitive to pre-fund the liquidity vault, making migration procedures described in the spec impossible and violating the spec.

Recommendation: Add the missing `function fund() external payable` that reverts on zero value, accepts arbitrary callers and emits `LiquidityFunded(msg.sender, msg.value)` before crediting the contract balance. This ensures the predeploy matches the spec and enables operators to seed liquidity safely.

Optimism: Fixed in commit 3f85e88

Cantina Managed: Fix verified.

3.1.6 `LiquidityController` events off-spec

Severity: Low Risk

Context: LiquidityController.sol#L22-L39

Description: Per the Custom Gas Token spec, `LiquidityController` must emit `MinterAuthorized(minter, authorizer)` and `MinterDeauthorized(minter, deauthorizer)` so governance actions can be audited, and the mint/burn lifecycle must use `AssetsMinted` / `AssetsBurned` (see the Custom Gas Token spec).

The implementation instead defines `event MinterAuthorized(address indexed minter)` and `event MinterDeauthorized(address indexed minter)` and emits them without authorizer context, while minting/burning emits `LiquidityMinted` and `LiquidityBurned`. Tooling expecting the spec events never receives authorizer addresses or the canonical event names, preventing reliable on-chain auditing and not respecting the documented interface.

Recommendation: Refactor the events to match the spec by declaring `event MinterAuthorized(address indexed minter, address indexed authorizer)` / `event MinterDeauthorized(address indexed minter, address indexed deauthorizer)` and emitting the caller as the authorizer when minters are added or removed and rename/align the mint/burn events to `AssetsMinted` and `AssetsBurned` with the same parameter structure as specified.

Optimism: Fixed in commit 573fdb3.

Cantina Managed: Fix verified.

3.2 Gas Optimization

3.2.1 Nested `if`-statements can be combined to save gas

Severity: Gas Optimization

Context: OptimismPortal2.sol#L353-L356, OptimismPortal2.sol#L449-L452, OptimismPortal2.sol#L576-L578

Description: The `proveWithdrawalTransaction()`, `depositTransaction()` and `finalizeWithdrawalTransaction()` functions in OptimismPortal2.sol contains nested if-statements that check two conditions sequentially:

```
if (_isUsingCustomGasToken()) {
 if (_tx.value > 0) revert OptimismPortal_NotAllowedOnCGTMode();
}
```

The current implementation uses nested if-statements, which results in two separate JUMPI opcodes being executed when both conditions need to be evaluated. This pattern is less gas-efficient than combining the conditions with a logical AND operator.

Recommendation: Consider combining the nested if-statements as following:

```
if (_isUsingCustomGasToken() && _tx.value > 0) revert
↪  OptimismPortal_NotAllowedOnCGTMode();
```

Optimism: Acknowledged.

Cantina Managed: Acknowledged.

3.2.2 Inefficient boolean comparison in `setCustomGasToken()` function

Severity: Gas Optimization

Context: L1BlockCGT.sol#L65

Description: The `setCustomGasToken()` function explicitly compares a boolean value to false, which generates unnecessary bytecode compared to using the negation operator.

Recommendation: Replace the explicit boolean comparison with the negation operator:

```
- require(isCustomGasToken() == false, "L1Block: CustomGasToken already active");
+ require(!isCustomGasToken(), "L1Block: CustomGasToken already active");
```

Optimism: Acknowledged.

Cantina Managed: Acknowledged.

3.3 Informational

3.3.1 Split CGT flags can cause locked funds

Severity: Informational

Context: L2ToL1MessagePasserCGT.sol#L30-L36

Description: `OptimismPortal2` rejects any transaction with value when `systemConfig.isFeatureEnabled(Features.CUSTOM_GAS_TOKEN)` is true, yet `L2ToL1MessagePasserCGT` only blocks value-bearing withdrawals when `IL1Block(Predeploys.L1_BLOCK_ATTRIBUTES).isCustomGasToken()` has already been flipped. The L2 withdrawal gate looks like:

```
function initiateWithdrawal(address _target, uint256 _gasLimit, bytes memory _data)
↪  public payable override {
    if (IL1Block(Predeploys.L1_BLOCK_ATTRIBUTES).isCustomGasToken() && msg.value > 0) {
        revert L2ToL1MessagePasserCGT_NotAllowedOnCGTMode();
    }
    super.initiateWithdrawal(_target, _gasLimit, _data);
}
```

The `SystemConfig` toggle is switched on governance's L1 call:

```
function setFeature(bytes32 _feature, bool _enabled) external {
    if (_enabled == isFeatureEnabled[_feature]) revert
    ↪  SystemConfig_InvalidFeatureState();
    isFeatureEnabled[_feature] = _enabled;
    emit FeatureSet(_feature, _enabled);
}
```

while the depositor separately flips the L2 flag once inside `L1BlockCGT`:

```
function setCustomGasToken() external {
    require(msg.sender == Constants.DEPOSITOR_ACCOUNT);
    require(isCustomGasToken() == false);
```

```
    assembly { sstore(IS_CUSTOM_GAS_TOKEN_SLOT, 1) }
}
```

Because the transactions are independent, the portal guard can activate before the L2 gate (blocking value withdrawals only after they reach L1), or the L2 gate can remain active after the portal is turned off. In the first case users submit withdrawals with `msg.value > 0`, they pass the L2 check, but later revert forever inside `OptimismPortal2`:

```
if (_isUsingCustomGasToken()) {
    if (_tx.value > 0) revert OptimismPortal_NotAllowedOnCGTMode();
}
```

In the second case the portal accepts value deposits again while L2 still rejects them, making ETH one-way to L2. Any mismatch traps user funds until governance re-aligns both flags.

Recommendation: Pause both `OptimismPortal2` and `L2ToL1MessagePasserCGT`, wait until there are no inflight deposits or withdrawals, then enable CGT mode by having L1 governance call `SystemConfig.setFeature(Features.CUSTOM_GAS_TOKEN, true)` and the depositor account call `L1BlockCGT.setCustomGasToken()` on the L1_BLOCK_ATTRIBUTES predeploy, and finally unpause. This orchestrated sequence keeps the two toggles aligned and avoids permanently stranding ETH during the feature transition.

Optimism: Acknowledged. Chain operators should not change the L1 flag once set.

Cantina Managed: Acknowledged.

3.3.2 CLI truncates large init bond values

Severity: Informational

Context: migrate.go#L107

Description: The interop migration CLI reads `--initial-bond` with `cliCtx.Uint64` and immediately passes the result through `big.NewInt(int64(...))` (`op-deployer/pkg/deployer/manage/migrate.go`). The flag itself is defined as a string to support values wider than 64 bits yet the current handling forces it back into a signed 64-bit window. Any bond equal to or above 2^63 wei (9.223372036854775808 wei ≈ 9.22 ETH) wraps negative when cast to `int64`, so the subsequent ABI encoder rejects it and the migration run reverts. The shipping default is 1 ETH:

```
InitialBondFlag = &cli.StringFlag{
    Name:    "initial-bond",
    Usage:   "Initial bond amount required for the dispute game (value as string, in
    ↪  wei). Defaults to 1 ETH.",
    EnvVars: deployer.PrefixEnvVar("INITIAL_BOND"),
    Value:   "1000000000000000000",
}
```

so a modest governance decision to raise the bond just an order of magnitude, well within operational expectations, would immediately hit this overflow. On-chain governance can set larger bonds (e.g. via `OPContractsManagerInteropMigrator.migrate` calling `DisputeGameFactory.setInitBond`), but any attempt to do so through the `op-deployer manage migrate` CLI helper fails before the transaction is even sent.

Recommendation: Parse the flag into a `*big.Int` without narrowing conversions-e.g. reuse the existing `cliutil.BigIntFlag` helper from the add-game-type path or call `new(big.Int).SetString(...)` on the string value and validate it is non-negative before passing it to `InteropMigrationInput.InitBond`.

Optimism: Acknowledged.

Cantina Managed: Acknowledged.

3.3.3 LegacyERC20 name lookup reverts

Severity: Informational

Context: Predeploys.sol#L146

Description: `Predeploys.getName` first enforces that every queried address lives inside the 0x4200... namespace and only then compares it against the known predeploy constants. The helper looks like:

```
function getName(address _addr) internal pure returns (string memory out_) {
    require(isPredeployNamespace(_addr), "Predeploys: address must be a predeploy");
    if (_addr == LEGACY_MESSAGE_PASSER) return "LegacyMessagePasser";
    // ...
    if (_addr == LEGACY_ERC20_ETH) return "LegacyERC20ETH";
    // ...
    revert("Predeploys: unnamed predeploy");
}
```

but `LEGACY_ERC20_ETH` lives at `0xDeadDeAddeAddEAddeadDEaDDEAdDeaDDeAD0000`, outside the namespace that `isPredeployNamespace` allows. As a result the guard reverts before ever hitting the branch that would return `"LegacyERC20ETH"`, and any tooling that loops through the exported predeploy constants like labeling scripts, deployment metadata generators, monitors... crashes when it reaches the legacy token address. The same issue affects `OPTIMISM_SUPERCHAIN_ERC20` (also outside of 0x4200...).

Recommendation: Special-case the out-of-namespace predeploys before asserting the namespace (or relax the predicate for those constants) so both `LEGACY_ERC20_ETH` and `OPTIMISM_SUPERCHAIN_ERC20` can be labeled without reverting. Add a regression test that calls `getName` for every exported predeploy constant to guarantee future additions remain reachable.

Optimism: Acknowledged.

Cantina Managed: Acknowledged.

3.3.4 CLI can't enable V2 dispute games

Severity: Informational

Context: implementations.go#L42-L45

Description: `op-deployer bootstrap implementations` now requires four `fault-game-*` parameters whenever `DeployV2DisputeGames` is set in the dev-feature bitmap, but the CLI no longer exposes flags or env vars that populate those fields. `ImplementationsConfig` expects:

```
type ImplementationsConfig struct {
    DevFeatureBitmap          common.Hash `cli:"dev-feature-bitmap"`
    FaultGameMaxGameDepth     uint64      `cli:"fault-game-max-game-depth"`
    FaultGameSplitDepth       uint64      `cli:"fault-game-split-depth"`
    FaultGameClockExtension   uint64      `cli:"fault-game-clock-extension"`
    FaultGameMaxClockDuration uint64      `cli:"fault-game-max-clock-duration"`
}
```

and its validator aborts when the bitmap enables V2 dispute games but any of the required fields remain zero:

```
if deployer.IsDevFeatureEnabled(c.DevFeatureBitmap, deployer.DeployV2DisputeGames) {
    if c.FaultGameMaxGameDepth == 0 {
        return errors.New("fault game max game depth must be specified when V2 dispute
        ↪  games feature is enabled")
    }
    // ...
}
```

However, `bootstrap/flags.go` dropped the old `dispute-*` flags without adding replacements using the new names, so `cliutil.PopulateStruct` can never populate the `fault-game-*` fields. Any operator who sets the V2 bit runs the documented command, the config fails validation, and the tool exits before deploying the contracts. This regression DoS's the CLI path for rolling out V2 dispute games until the flags return.

Recommendation: Reintroduce CLI (and env) flags for `fault-game-max-game-depth`, `fault-game-split-depth`, `fault-game-clock-extension`, and `fault-game-max-clock-duration` so the struct receives non-zero values when V2 is enabled. Update docs and helper scripts to

use the new flag names, and add a regression test that ensures every `cli:"..."` struct tag in `ImplementationsConfig` has a corresponding CLI flag.

Optimism: Acknowledged.

Cantina Managed: Acknowledged.

3.3.5 Redundant indexed parameter `caller` in events

Severity: Informational

Context: NativeAssetLiquidity.sol#L37, NativeAssetLiquidity.sol#L49

Description: The `NativeAssetLiquidity` contract emits LiquidityDeposited and LiquidityWithdrawn events with an indexed caller parameter. However, both the `deposit()` and `withdraw()` functions enforce that the msg.sender must be Predeploys.LIQUIDITY_CONTROLLER, otherwise the transaction reverts:

```
function deposit() external payable {
 if (msg.sender != Predeploys.LIQUIDITY_CONTROLLER) revert
  ↪  NativeAssetLiquidity_Unauthorized();

 emit LiquidityDeposited(msg.sender, msg.value);  // msg.sender is always
  ↪  LIQUIDITY_CONTROLLER
}

function withdraw(uint256 _amount) external {
  if (msg.sender != Predeploys.LIQUIDITY_CONTROLLER) revert
   ↪  NativeAssetLiquidity_Unauthorized();

  // ...

  emit LiquidityWithdrawn(msg.sender, _amount);  // msg.sender is always
   ↪  LIQUIDITY_CONTROLLER
}
```

Since the caller parameter will always be the same constant value (`Predeploys.LIQUIDITY_CONTROLLER`), including it as an indexed parameter in the events provides no filtering utility and wastes gas.

Recommendation: Remove the redundant caller parameter from both events since it provides no variable information.

Optimism: Acknowledged.

Cantina Managed: Acknowledged.

3.3.6 CGT override bypasses intent validation

Severity: Informational

Context: l2genesis.go#L157-L165

Description: `IntentTypeStandard` deployments assume ETH-native behavior and reject custom gas token configs during validation. The check in `op-deployer/pkg/deployer/state/intent.go` explicitly fails if `chain.CustomGasToken.Enabled` is true:

```
if chain.CustomGasToken.Enabled {
  return fmt.Errorf("%w: chainId=%s custom gas token must be disabled for standard
   ↪  chains", ErrNonStandardValue, chain.ID)
}
```

ensuring standard chains launch without CGT support. However, once per-chain overrides are merged, `calculateL2GenesisOverrides` rewrites the validated intent whenever CGT was originally disabled:

```
// op-deployer/pkg/deployer/pipeline/l2genesis.go:157-164
// If CustomGasToken is not enabled, update it with override values
if !thisIntent.CustomGasToken.Enabled {
    thisIntent.CustomGasToken = state.CustomGasToken{
```

```
        Enabled:          overrides.UseCustomGasToken,
        Name:             overrides.GasPayingTokenName,
        Symbol:           overrides.GasPayingTokenSymbol,
        InitialLiquidity: overrides.NativeAssetLiquidityAmount,
    }
}
```

Overrides are merged after all validation completes and no subsequent `ChainIntent.Check()` call re-runs. A simple JSON override (e.g., `{"useCustomGasToken": true, "gasPayingTokenName": "...", ...}`) therefore mutates the chain to CGT mode even when the signed intent, code review and standard template all forbade it. The deployer only needs to run the normal apply command:

```
op-deployer apply \
  --intent standard-intent.toml \
  --overrides overrides/devnet.json \
  --l1-rpc $L1_RPC \
  --wallet $DEPLOYER_KEY
```

If that override happens to flip `useCustomGasToken`, the generated genesis now installs `LiquidityController/NativeAssetLiquidity`, blocks L1 ETH deposits and requires the special liquidity runbooks, yet operators, tooling and users still believe they launched a standard ETH chain. This mismatch can strand deposits, reject withdrawals and break integrations that were never configured for CGT.

Recommendation: Treat override-driven CGT toggles as a high-risk change. After merging overrides, rerun `thisIntent.Check()` (or at least the portion that enforces `CustomGasToken.Enabled == false` for standard configs) before proceeding and fail if the override flips the flag. Alternatively, explicitly forbid `useCustomGasToken` inside both global and per-chain override files unless the base intent already opted into CGT mode. This keeps the reviewed intent as the single source of truth and prevents accidental or malicious overrides from silently changing the chain's economic model.

Optimism: Fixed in commit 2e44c5f

Cantina Managed: Fix verified.

3.3.7 Superchain Registry queries invalid CGT ABIs

Severity: Informational

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

Description: The Superchain Registry report generator (Go code under `packages/contracts-bedrock/lib/superchain-registry/ops/internal/report/`) builds an `L1Report` by issuing batched RPC calls to the deployed `SystemConfig` proxy. Once the release tag exceeds `Semver160`, it unconditionally appends calls for `gasPayingToken()`, `gasPayingTokenName()` and `gasPayingTokenSymbol()`:

```
// packages/contracts-bedrock/lib/superchain-registry/ops/internal/report/l1.go
calls = append(
    calls,
    ...,
    makeBatchCall(isCustomGasTokenABI, &report.IsGasPayingToken),
    BatchCall{
        To: addr, // SystemConfig proxy
        Encoder: func() ([]byte, error) {
            return gasPayingTokenABI.EncodeArgs()
        },
        Decoder: func(rawOutput []byte) error {
            return gasPayingTokenABI.DecodeReturns(rawOutput, &report.GasPayingToken,
            ↪  &report.GasPayingTokenDecimals)
        },
    },
    makeBatchCall(gasPayingTokenNameABI, &report.GasPayingTokenName),
    makeBatchCall(gasPayingTokenSymbolABI, &report.GasPayingTokenSymbol),
)
```

Those ABIs are defined in `bindings.go` but rely on the called contract actually implementing the selectors. `SystemConfig` does not expose any of the `gasPayingToken*` getters, only `L1Block/L1BlockCGT` (and `LiquidityController`) do. As soon as the registry scanner contacts a `SystemConfig` on a release that uses this branch, each `eth_call` reverts, `CallBatch` returns an error and the tooling fails to emit a report. Practically, every attempt to collect CGT metadata after PR #18076 will break the registry publishing workflow.

Recommendation: Adjust the reporter so it fetches CGT metadata from the correct source. Either add proxy functions on `SystemConfig` that expose `gasPayingToken*` and call those, or point the scanner at the L2 predeploys (`L1Block/LiquidityController`) when `isCustomGasToken` is true. The goal is to keep the RPC batch addressing contracts that actually implement the queried selectors, otherwise the report generator will continue to revert and no registry entries can be produced for CGT-enabled chains.

Optimism: Acknowledged. After the release we will update the superchain-registry repository.

Cantina Managed: Acknowledged.

3.3.8 Superchain Registry staging drops CGT metadata

Severity: Informational

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

Description: The Superchain Registry staging step (`packages/contracts-bedrock/lib/superchain-registry/ops/internal/manage/staging.go`) still assumes CGT deployments expose an ERC20 address on L1 via the now-removed `DeployConfig.CustomGasTokenAddress`. The only logic that records a gas-paying token is:

```
if dc.CustomGasTokenAddress != (common.Address{}) {
    cfg.GasPayingToken = config.NewChecksummedAddress(dc.CustomGasTokenAddress)
}
```

In the custom gas token architecture introduced by PR #18076, there is no standalone ERC20 on L1. Liquidity is handled by the new `LiquidityController/NativeAssetLiquidity` predeploys and the configuration lives in `chainIntent.CustomGasToken` plus flags such as `useCustomGasToken/gasPayingTokenName`. Because the staging code never reads those fields, every CGT-enabled chain is exported to the registry as if it were an ETH-native chain. Downstream tooling (dashboards, wallets, governance systems) can no longer tell whether the chain requires custom gas token handling or what metadata (name/symbol/liquidity) to display, leading to incorrect UX and potentially misconfigured integrations.

Recommendation: Update `InflateChainConfig` to source CGT data from the new intent/deploy config fields. At minimum, persist whether `chainIntent.CustomGasToken.Enabled` is true and copy the configured `Name`, `Symbol` and `InitialLiquidity` (or the equivalent values from `DeployConfig.GasTokenDeployConfig`). Extend the staged-chain schema to surface this information so registry consumers can distinguish CGT chains from standard ETH chains and display the correct token metadata.

Optimism: Acknowledged. After the release we will update the superchain-registry repository.

Cantina Managed: Acknowledged.

3.3.9 Unnecessary `ETHLockbox` deployment for custom gas token chains

Severity: Informational

Context: OPContractsManager.sol#L1322

Summary: The `deploy()` function in `OPContractsManager.sol` unconditionally deploys and initializes the ETHLockbox proxy contract for all chains, including those configured with custom gas tokens. This is redundant.

Recommendation: Conditionally deploy and initialize the ETHLockbox only when the CUSTOM_GAS_TOKEN feature is disabled. Additionally, consider adding validation to prevent both CUSTOM_GAS_TOKEN and OPTIMISM_PORTAL_INTEROP features are not enabled simultaneously, as they appear to be mutually exclusive use cases at this point.

Optimism: Acknowledged. We will do further work with feature flags compatibilities.

Cantina Managed: Acknowledged.