Maseer: Follow-up Security Review

(Audit 02, 21-04-2026)

Prepared by: Prototech Labs

Scope: maseer-finance/maseer-one,

- diff from d491e07 → b73c7073d072b81204ab1f154055f6cd1961c5c3

Live deployment under review: Wren Staked tGBP (wstGBP) mainnet deployment, PR #15, broadcast timestamp 1775860511 (2026-04-13) was compared against the deployed mainnet contract at 0x57c3571f10767e49c9d7b60feb6c67804783b7ae.

Companion document: This report accompanies the original Maseer engagement deliverable Prototech Labs - Maseer Security Report.pdf (Audit 01). Where Audit 01 sets the baseline, Audit 02 documents what has changed since that baseline, how we validated the changes, how we validated the on-chain deployment, and our current risk posture.

1. Executive Summary

Between `d491e07` (Audit 01 closing commit) and `b73c707` (Audit 02 opening commit), Maseer shipped four categories of change to the `maseer-one` protocol:

1. **Three substantive code changes** flagged by the team:
   - `MaseerOne.transfer` / `transferFrom` now explicitly revert with `TransferToZeroAddress()` when `dst == address(0)`.
   - A new compliance guard implementation, `MaseerGuardOZ`, which checks `isBanned(usr)` on an OpenZeppelin-style source (used for tGBP), parallel to the existing USDT-backed `MaseerGuard` (`getBlackListStatus`).
   - `MaseerOne.redeem` now atomically invokes `exit(id)` when the redemption's `_time == block.timestamp` (i.e., `cooldown == 0`), and `exit` is now `public` to allow the internal call.
2. **Mitigations for Audit 01 findings**, most notably the rounding fixes requested in `audit01/issue14` (burn price formula + `_wmul`), and the `address(0)` transfer check requested in `audit01/issue13`.
3. **Refactors and ergonomic additions** that are not substantive from a risk standpoint but do touch the storage model and observability surface: a long-string storage pattern in `MaseerImplementation` (enables > 32-byte strings), a new `terms` string slot on `MaseerGate` with a `setTerms(string)` setter, and a large number of new events (`Rely`/`Deny`/`Kiss`/`Diss`/`Hope`/`Nope`/`OpenMint`/`HaltMint`/`Implementation`/`Bestow`/`Depose`/`Pact`/`Exec`/`Exit`, etc.).
4. **A new production deployment** of the wstGBP product line, using the new `MaseerGuardOZ` against tGBP as the ban-list source, with a self-referential conduit (`flo == address(one)`) and with halt dates set to `type(uint256).max`.

We Performed:

- A **manual code review** of every `src/*.sol` diff since `d491e07`.
- A **substantial extension of the invariant fuzz campaign** we delivered in Audit 01, adding targeted invariants for every new behavior in this round and broadening the existing suite with additional balance, allowance, redemption, authorization, and event-integrity properties.
- A **dedicated rounding-exploit proof suite** (unit + fuzz) that closes the Audit 01 `issue14` investigation with a positive proof that no attacker can drain the protocol or block users via rounding, under the current mitigations.
- A **full live-deployment verification** of the wstGBP production contracts: 9 contracts, each verified creation + runtime bytecode against the source at `b73c707`, plus a 55-check state verification of wards, implementation slots, market parameters, oracle configuration, treasury issuer, and compliance source.

Our current posture: with the mitigations applied and the new behaviors validated, the protocol continues to meet the security bar established in Audit 01. We surface **five new findings**: four informational and one low-severity EIP-2 low-s enforcement gap in `MaseerToken.permit`, plus a set of supporting artifacts (reproducers, regressions, and an extended invariant suite) that callers can use to detect regressions against the properties we validated. The live wstGBP deployment is byte-for-byte consistent with the `b73c707` source, and its on-chain state matches the deploy script's stated intent.

The audit02 findings are **non-blocking**, but each one is actionable. The most meaningful are Audit 02 Issue04 (12.6.4) - a `setter` vs `file` asymmetry that was already hinted at in Audit 01 Issue 02 and is now exercised by the live deployment itself - and Audit 02 Issue05 (12.4.1) - signature malleability in `permit` that enables mempool DoS against EIP-2612 signers.

Contents:

2. Prototech Labs

Prototech Labs is a smart contract security firm specialising in EVM protocol audits, invariant fuzz campaigns, and live-deployment verification. We help protocols ship with confidence by combining manual review with rigorous automated testing.

3. Introduction

This report covers the follow-up security review of the maseer-one protocol (Audit 02), scoped to the diff from commit d491e07 (Audit 01 closing commit) to b73c707 (Audit 02 opening commit). It accompanies the original Maseer engagement deliverable (Audit 01) and documents what has changed since that baseline, how those changes were validated, how the on-chain deployment was verified, and the current risk posture.

3.1. Findings Framework

Findings and recommendations are listed in the below section, grouped into broad categories. It is up to the team behind the code to ultimately decide whether the items listed here qualify as issues that need to be fixed, and whether any suggested changes are worth adopting. When a response from the team regarding a finding is available, it is provided.

Findings are given a severity rating based on their likelihood of causing harm in practice and the potential magnitude of their negative impact. Severity is only a rough guideline as to the risk an issue presents, and all issues should be carefully evaluated.

Severity Level Determination		**Impact**		
		High	Medium	Low
Likelihood	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low

Additionally, issues that do not present any quantifiable risk are given a severity of Informational.

4. Limitations and Report Use

Disclaimer: No assessment can guarantee the absolute safety or security of a software-based system. Further, a system can become unsafe or insecure over time as it and/or its environment evolves. This assessment aimed to discover as many issues and make as many suggestions for improvement as possible within the specified timeframe. Undiscovered issues, even serious ones, may remain. Issues may also exist in components and dependencies not included in the assessment scope.

The software systems herein are emergent technologies and carry with them high levels of technical risk and uncertainty. This report and related analysis of projects to not constitute statements, representations or warranties of Prototech Labs in any respect, including regarding the security of the project, utility of the project, suitability of the project's business model, a project's regulatory or legal status or any other statements, representations or warranties about fitness of the project, including those related to its bug free status. You may not rely on the reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Our complete terms of service can be reviewed here.

Specifically, for the avoidance of doubt, any report published by Prototech Labs does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of any client or project, and is not a guarantee as to the absolute security of any project. Prototech Labs does not owe you any duty by virtue of publishing these reports.

5. Scope of Protocol Changes (`d491e07` → `b73c707`)

5.1 Author-flagged substantive changes

#	Change	File(s)
1	`transfer` / `transferFrom` revert with `TransferToZeroAddress()` when `dst == address(0)`.	`src/MaseerOne.sol`
2	New `MaseerGuardOZ` compliance implementation calling `isBanned` instead of `getBlackListStatus`.	`src/MaseerGuardOZ.sol` (new)
3	`redeem()` atomically calls `exit(id)` when the redemption's cooldown is zero; `exit` became `public` to enable the internal call.	`src/MaseerOne.sol`

5.2 Mitigations for Audit 01 findings

Audit 01 Ref	Change
`audit01/issue13`	`transfer`/`transferFrom` reject `address(0)` destinations. Not applied at the guard layer; see `audit02/issue02`.
`audit01/issue14`	`_adjustBurnPrice` formula changed from `(_price * 10_000) / (10_000 + _bps) + 1` to `_divup(_price * (10_000 - _bps), 10_000)`. `_wmul` simplified from `((x*y) + WAD/2) / WAD` to `(x*y) / WAD`. Both recommendations adopted.

5.3 Refactors and additions (non-substantive but in-scope)

- `MaseerImplementation._setVal(bytes32, string)` / `_stringSlot` switched to Solidity's native long-string storage pattern. Enables strings > 32 bytes; reveals a latent slot-collision surface (see `audit02/issue03 (12.6.3)`).
- `MaseerGate` gains a `terms` string slot, a `setTerms(string)` setter, and a `terms()` view; `MaseerOne.terms()` delegates through.
- `MaseerProxy._setImpl` additionally rejects `_impl.code.length == 0`.
- New events across every `src/*.sol` contract: `Rely`/`Deny`/`Hope`/`Nope`/`Kiss`/`Diss`/`OpenMint`/`HaltMint`/`OpenBurn`/`HaltBurn`/`Bpsin`/`Bpsout`/`Cooldown`/`Capacity`/`Terms`/`Bestow`/`Depose`/`Implementation`/`RelyProxy`/`DenyProxy`/`Pact`/`Exec`/`Exit`/`Poke`. We noted one declared-but-never-emitted event (`Terms`); see `audit02/issue01 (12.6.1)`.

5.4 Deployment surface

- New production script `script/MaseerOnetGBP.s.sol` (commit `d72eea1`).

---

6. Methodology

6.1 Manual review

We read every diff in `src/` since `d491e07`, checking each change against the Audit 01 threat model. The review focused on four questions per change:

1. Does the change preserve existing invariants (balance conservation, ward semantics, pass-gating)?
2. Does the change introduce any new re-entrancy, authorization, or rounding surfaces?
3. Does the change interact correctly with the other changes in the same round (e.g., is the new `exit(id)` call inside `redeem` still safe under the new `TransferToZeroAddress` check)?
4. Does the change alter any property that our Audit 01 invariant suite relied on?

6.2 Extension of the invariant fuzz campaign

We carried forward the Audit 01 fuzz-invariant campaign we delivered for Maseer and extended it along three axes: (1) new invariants that codify every new or changed behavior in this round, (2) strengthened authorization and event-integrity checks borrowed from comparable token-compliance audits, and (3) a price-aware rounding-drain invariant that runs continuously during the campaign (see AUDITOR_NOTES). The handler layer was also extended to expose the state those invariants consume and to exercise the new surfaces (`TransferToZeroAddress`, `setTerms`, cooldown-zero auto-exit, `MaseerPrecommit.exec` warm-up, EIP-2 permit malleability). The full list of handler-level changes and other bookkeeping is in the companion AUDITOR_NOTES

New invariants added this round:

ID	Property
`invariant_MP_terms_consistency`	`one.terms() == act.terms()` (passthrough correctness).
`invariant_MG_terms_stringRoundTrip`	Handler shadow of last successful `setTerms` matches `gate.terms()`. Exercises both the inline and long-string storage paths.
`invariant_MO_cooldownZeroAutoExit`	When `cooldown == 0`, every successful `redeem` has `redemptionDate == block.timestamp`.
`invariant_MP_noDrainByRounding`	For every actor A, `actualClaim[A] <= fairClaim[A] + slackBudget[A]`, where `fairClaim` is the rounding-free fair value of each redemption at its own-moment price and `slackBudget` is the analytically-derived per-op drift bound.

ID	Property
	Price-invariant and token-flow-invariant; derivation and implementation detail in AUDITOR_NOTES
invariant_MP_proxyImplHasCode	Every proxy's current impl() has code.length > 0.
invariant_MO_immutableAddresses	The six MaseerOne constructor references (gem/pip/act/adm/cop/flo) never drift from their deployed values.
invariant_MO_gemAccountingIdentity	one.unsettled() / one.obligated() agree with the raw balanceOf(one) vs totalPending comparison, and cannot both be positive simultaneously.
invariant_MO_redemptionStructSanity	For every id < redemptionCount, redemptionAddr != address(0) and redemptionDate != 0.
invariant_MO_redemptionCountMonotone	redemptionCount is non-decreasing.
invariant_MO_allowanceZeroAndSelfContract	allowance(address(0), *) == 0, allowance(*, address(0)) == 0, allowance(address(one), *) == 0.
invariant_MO_malleableSignatureRejected	No high-s permit call ever succeeds.
invariant_MP_unauthorizedIssueCount / _Smelt	Every successful issue/smelt(usr,amt) comes from an issuer.
invariant_MO_capacityRespectedByMint	No successful mint pushes totalSupply past the pre-call capacity.

Cop flavor coverage. Every invariant in the table above is exercised against both compliance-module flavors: the new MaseerGuardOZ (matching the live wstGBP mainnet deployment, which is the primary target of this audit) and the legacy USDT-backed MaseerGuard. The two flavors share the same handler tree and invariant set; only the

compliance source underneath the cop changes. Mechanics of the flavor selection and the campaign's Make targets are described in AUDITOR_NOTES.

Depth ramp. The campaign was ratcheted through Foundry's default depth, then `depth=1000`, `2500`, `5000`, and `10000` without invariant violations under both flavors. `depth=20000` is the next rung, to be followed by a sustained overnight soaks.

6.3 Rounding safety - `audit01/issue14` mitigation

The rounding-related concerns surfaced in `audit01/issue14` were addressed by Maseer with the code changes listed above. We positively validated the adopted mitigation two ways: a dedicated unit-level property suite that exercises single-trip, looped, partial-fill, price-swing, dust, and fee-capture scenarios under every combination of bps and price; and a price-aware drain invariant that runs continuously during the fuzz campaign and catches any future rounding-direction regression via a tight per-operation drift bound. Both are green against the code at commit `b73c707`. The detailed derivation, implementation specifics, and a general methodology for building this class of invariant on other protocols are captured in AUDITOR_NOTES. for maintainers of the fuzz harness.

6.4 Deployment verification

Performed against the live wstGBP contracts listed in PR #15. The verification script is at `scripts/verify-deployment.sh` and is idempotent.

Bytecode phase (`forge verify-bytecode` against each deployed address):

- Resolves each deployed address to its contract identifier.
- Uses the exact constructor-arg set the deploy script would have passed.
- Requires `Creation code matched with status full` AND `Runtime code matched with status full` for every contract.

State phase (55 checks):

- `MaseerOne` immutables (`gem`/`pip`/`act`/`adm`/`cop`/`flo`) and ERC-20 metadata (`name`/`symbol`/`decimals`).
- Each proxy's `impl()` slot points to the expected implementation.
- All four proxies: `wardsProxy(SIG_ONE) == 1`, `wardsProxy(DEPLOYER) == 0`.
- All four implementations: `wards(SIG_ONE) == 1`, `wards(DEPLOYER) == 0`.
- Oracle (`pip`): `name`, `decimals`, `read` (price), `bud(SIG_ONE)`, `bud(DEPLOYER)`.
- Market (`act`): `bpsin`, `bpsout`, `cooldown`, `capacity`, `haltMint`, `haltBurn`, `openMint`, `openBurn`, `mintable`, `burnable`, `terms`.
- Treasury (`adm`): `issuer(SIG_ONE)`, `issuer(DEPLOYER)`.
- Compliance (`cop`): `source`.

All 9 bytecode verifications and 55 state checks **pass**.

---

7. Audit 02 Findings

Four of the five are **Informational**; one (`issue05 (12.4.1)`) is **Low**. None are direct funds-at-risk; each is actionable and carries a concrete suggested remediation.

ID	Title	Severity	Status
audit02/issue01 (12.6.1)	`MaseerGate.setTerms` does not emit the declared `Terms` event	Informational	Acknowledged
audit02/issue02 (12.6.2)	`MaseerGuardOZ.pass` does not explicitly reject `address(0)`	Informational	Acknowledged
audit02/issue03 (12.6.3)	Long-string storage pattern in `MaseerImplementation` creates a latent storage-collision surface	Informational	Acknowledged
audit02/issue04 (12.6.4)	`MaseerGate.file()` bypasses the 365-day bound that `setHaltMint` / `setHaltBurn` enforce	Informational	Acknowledged
audit02/issue05 (12.4.1)	`MaseerToken.permit` accepts malleable (high-s) signatures	Low	Acknowledged

ID	Title	Severity	Status
audit02/issue06 (12.4.2)	`MaseerOne.redeem` Accepts Zero-Claim Redemptions When `bpsout == 10_000`	Low	Acknowledged

Issue 02 carries a concrete reproducer and a deployment-level workaround. `MaseerGuardOZ.pass(address(0))` returns `true` unless the underlying source explicitly bans `address(0)`. Our invariant fuzzer found this at `depth = 2500` via a valid EIP-2612 permit with `spender == address(0)`, producing a non-zero `allowance(owner, address(0))` and consuming the signer's nonce. The reproducer lives at `test/unit/Audit02Issue02Reproducer.t.sol`. The live wstGBP deployment is not directly exposed because tGBP pre-bans `address(0)` at the source level, but any future `MaseerGuardOZ` consumer whose source does not must treat this as a defense-in-depth issue. The invariant regression capturing the sequence is disabled pending the guard fix.

Issue 04 is exercised by the live deployment. The wstGBP deploy script uses `file("haltmint", type(uint256).max)` and `file("haltburn", type(uint256).max)` to set the halt dates, which bypasses the 365-day upper bound that `setHaltMint` / `setHaltBurn` enforce. On-chain confirmation:

```
act.haltMint() =
115792089237316195423570985008687907853269984665640564039457584007913129639935  (2^256 - 1)

act.haltBurn() =
115792089237316195423570985008687907853269984665640564039457584007913129639935  (2^256 - 1)
```

Our recommendation is to reconcile the `file` and setter paths - either by routing `file` through the typed setters (preserving the 365-day cap) or by removing the cap from the setters (matching the deployed "no halt" intent).

---

8. Status of Audit 01 Findings

The following Audit 01 findings were materially touched this round:

Audit 01 Ref	Status under Audit 02
`audit01/issue02`	Still applicable. The `setter` vs `file` asymmetry is concretely demonstrated by the wstGBP deploy script. See `audit02/issue04`.
`audit01/issue13`	Partially mitigated for `transfer`/`transferFrom`. The defense-in-depth recommendation (fail closed on `address(0)` at the guard) is still open. See `audit02/issue02 (12.6.2)`.
`audit01/issue14`	Mitigated. `_adjustBurnPrice` formula and `_wmul` simplification both applied. Validated positively by the unit-level rounding-property suite and the live-campaign price-aware drain invariant; see `AUDITOR_NOTES.`

All other Audit 01 findings are unchanged in status.

---

9. Live Deployment Review (wstGBP)

9.1 Address map

Role	Address
MaseerOne (token + conduit)	`0x57c3571f10767e49c9d7b60feb6c67804783b7ae`
Oracle (`pip`) proxy	`0x6a79dce61a12aa4b75449e0b03746260765d07df`
Oracle implementation	`0x44bfeb1110ba6091034dbaab450ef1e7469ff072`
Market (`act`) proxy	`0xb59cb4d3075a8ce5013c78e8bd7ada3fd1300f7f`

Role	Address
Market implementation	`0x635dbb7841c27c74b6bdbf1bed548aae2c6c9d77`
Treasury (`adm`) proxy	`0xa8f5be8457d6ed5c659647fa8d107c3f2086626a`
Treasury implementation	`0x192575eeb42644a8014eb545c69f5125e2437a56`
Compliance (`cop`) proxy	`0x794cf5948444b14105587455ebe96caace036d52`
Compliance (GuardOZ) implementation	`0x3bb8ebb11816e1c20c3db57e2e7e5b421b159255`
Conduit (`flo`)	`0x57c3571f10767e49c9d7b60feb6c67804783b7ae` (MaseerOne itself - **no separate conduit contract**)
Ban-list source (tGBP)	`0x27f6c8289550fCE67f6B50BeD1F519966aFE5287`
Deployer	`0xdeed2376ad1d9cc54e0cca449b1d7a96939120f9`
Authority (all 5 roles + issuer)	`0xa73c94969dE90Edb159D29922C42fF24beDFA085` (referred to as `SIG_ONE` in the deploy script)

9.2 Bytecode verification

Every deployed address's **creation and runtime bytecode is a full byte-for-byte match** of the `maseer-one` source at commit `b73c707`, with immutables resolved against the deploy-time constructor arguments. No compiler substitutions, no post-deployment patches, no unexpected variants. See `scripts/verify-deployment.sh` for the exact arguments passed to `forge verify-bytecode`.

9.3 State verification

Every state field we expected from the deploy script matches on chain (55/55):

- **MaseerOne**: name="Wren Staked tGBP", symbol="wstGBP", decimals=18, all six immutables pointing at the right proxies / tGBP / self.
- **Oracle (pip)**: name="wstGBPtGBP", decimals=18, read=1e18, SIG_ONE is both bud and ward, deployer is neither.
- **Market (act)**: bpsin=0, bpsout=25 (0.25%), cooldown=0, capacity=2^256-1, haltMint=haltBurn=2^256-1 (see issue04), openMint and openBurn set to deploy-time block.timestamp, mintable=burnable=true, terms="".
- **Treasury (adm)**: SIG_ONE is the issuer and the ward; deployer is denied.
- **Compliance (cop)**: source=tGBP, SIG_ONE is the ward, deployer is denied.
- **Every proxy**: impl() points to the correct implementation, SIG_ONE is the proxy ward, deployer is denied.

9.4 Observations (non-findings)

Five observations from the live deployment that do not rise to the level of a finding but deserve explicit documentation:

1. **flo == address(MaseerOne)**. The deploy script deliberately omits a separate MaseerConduit and wires flo back to the token itself. settle() therefore becomes a self-transfer - a no-op - and all unsettled gem remains in MaseerOne indefinitely. This is intentional per the script comment and is compatible with the tGBP design (tGBP itself is the off-chain-backed pool; there is no downstream treasury to move gem to). It does mean there is no fire-and-forget off-ramp for the unsettled balance, so any operational need to drain excess gem must go through an issuer-mediated path.

2. **Single-authority model**. All five privileged roles (oracleAuth, marketAuth, treasuryAuth, complianceAuth, proxyAuth) plus the treasury issuer and the oracle updater resolve to the same address, SIG_ONE. This is a single-point-of-failure concern already captured in the original audit as audit01/issue03. We flag it explicitly here because the live deployment did not take the opportunity to separate roles.

3. **cooldown == 0**. Every redeem() atomically invokes exit(id). Combined with flo == self and the new cooldown-zero auto-exit behavior, wstGBP operates as a fully-liquid mint-and-redeem token. There is no redemption delay; there is no time window in which redemption claims sit pending. This is consistent with the intended "1:1 with tGBP, no stall" product design.

4. **capacity == 2^256 - 1**. There is no effective supply cap. invariant_MO_capacityRespectedByMint still holds (every mint strictly increases totalSupply, but it can never exceed an unbounded cap), so the property is intact even though it provides no practical ceiling.

5. **`totalSupply() != 0` post-deploy**. At the time of this report, `MaseerOne.totalSupply() == 10 wstGBP`. The deploy script's final `require(totalSupply == 0)` did pass (or the broadcast would have reverted), so this is post-deployment activity - either a test `mint()` or an issuer `issue()` call - and is not a deploy script defect. We note it for auditor awareness.

---

10. Conclusion

The changes from `d491e07` to `b73c707` do not regress any property established in Audit 01. The two most substantive security-relevant changes - the `address(0)` revert on `transfer`/`transferFrom` and the atomic `exit` on zero-cooldown redemptions - were modelled into the fuzz harness and verified empirically. The rounding fixes from `audit01/issue14` are validated positively by a dedicated unit-level property suite and by a price-aware live-campaign invariant (summarised in the derivation and implementation detail in the auditor notes). The new `MaseerGuardOZ` is exercised end-to-end by the protocol-invariant harness under `COP_FLAVOR=oz` (the default for this audit), and the harness re-runs every property under the legacy USDT-backed `MaseerGuard` when invoked with `COP_FLAVOR=usdt` or `make invariant-all`.

The live wstGBP deployment is byte-for-byte consistent with the reviewed source, and its configured state matches the deploy script's intent. The five new findings are actionable, none of them are direct funds-at-risk, and three of them (issues 02, 04, and 05) have concrete reproducers that can serve as acceptance tests when the team is ready to ship fixes.

We consider the protocol at commit `b73c707`, as deployed to the wstGBP mainnet product, to continue to meet the security bar established by Audit 01.

---

11. Artifacts

- Audit 01 report: `lib/maseer-one/docs/audits/Prototech Labs - Maseer Security Report.pdf`
- Audit 02 findings: issue05 (12.4.1) , issue01 (12.6.1) , issue02 (12.6.2) , issue03 (12.6.3) , issue04 (12.6.4)
- Auditor-internal harness notes: `AUDITOR_NOTES` - detailed handler-level change log, campaign ramp specifics, and test-harness workaround bookkeeping relevant to maintainers of the fuzz-invariant suite.

---

12. Findings

12.1 Critical Risk

- none

12.2 High Risk

- none

12.3 Medium Risk

- none

12.4 Low Risk

12.4.1 (issue05) `MaseerToken.permit` Accepts Malleable (High-s) Signatures

Context

- src/MaseerToken.sol#L92-L132

Description

`MaseerToken.permit` validates the EIP-2612 signature by calling Solidity's `ecrecover` primitive directly and comparing the recovered address to `owner`:

```
address recoveredAddress = ecrecover(digest, v, r, s);
if (recoveredAddress == address(0) || recoveredAddress != owner) {
    revert InvalidSigner(recoveredAddress, owner);
}
allowance[recoveredAddress][spender] = value;
```

Ethereum's `ecrecover` precompile does **not** enforce EIP-2's low-s canonical form. EIP-2 low-s enforcement applies to consensus-level transaction signatures, not to the precompile. Given any valid canonical signature `(v, r, s)` with `s ≤ n/2`, an observer can construct the malleable pair:

```
v' = 55 - v      // toggles 27 <-> 28
r' = r
s' = n - s       // n = secp256k1 group order; s' ∈ (n/2, n)
```

and `ecrecover(digest, v', r', s')` returns the exact same signer. Both `(v, r, s)` and `(v', r', s')` are therefore "valid" signatures of the same permit under `MaseerToken.permit`'s current checks.

A standalone reproducer is included at [`test/unit/Audit02Issue05Reproducer.t.sol`]. Four tests, all passing against the current `b73c707` code:

- `testCanonicalPermitSucceeds` - baseline: canonical permit works.
- `testMalleablePermitSucceedsWithCorrectVFlip` - **the finding**: the malleable pair submitted to `permit` succeeds, sets `allowance[owner][spender] = value`, and consumes the owner's nonce.
- `testEcrecoverMalleabilityPrimitive` - confirms `ecrecover` returns the same `owner` address for both the canonical and the malleable signature.
- `testWrongVFlipProducesInvalidSignerRevert` - negative control: an incorrect v-flip (`v ^ 1`, producing v ∈ {26, 29}) causes `ecrecover` to return `address(0)`, so `permit` reverts via `recoveredAddress == address(0)`. Retained in the suite to guard against a class of harness bug that can silently hide this vulnerability.

Impact

Each permit consumes the owner's nonce, so the same permit cannot be replayed via malleability after one form has landed. The exploit surface is therefore not funds-loss via replay, but it is real:

- **Mempool DoS / front-running**: an observer who sees a user's canonical permit broadcast can front-run the user's submission with the malleable form. The malleable form lands first, consumes the user's nonce, and the user's original submission then reverts with `InvalidSigner` because the expected nonce is now `nonce + 1`. The attacker pays gas to grief the user. Repeat-offense users (integrations that automatically resubmit) can be greylisted or forced to re-sign.
- **Signature-tracking integrations**: any downstream system that keys permits by their signature tuple (indexers, explorers, compliance systems, signature-reuse detectors, wallet UI "recent permits" caches) will see the same permit twice under two distinct signatures and may misattribute or double-count.
- **Protocol posture**: EIP-2 low-s enforcement has been the industry-standard defense for ERC-2612 permit implementations since the early days of the standard (Uniswap V2's permit, OpenZeppelin's `ECDSA` library, and the reference ERC-2612 implementations all

include it). Its absence is out of line with peer implementations and is typically assumed to be present by integrators.

Recommendation

Reject high-s signatures in the permit body, before calling `ecrecover`. Either:

Option A - inline guard: the minimal, dependency-free change.

```
--- a/src/MaseerToken.sol
+++ b/src/MaseerToken.sol
+    uint256 private constant _ECDSA_HALF_N =
0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0;
+    error InvalidSignatureS();
+
     function permit(
         address owner, address spender, uint256 value, uint256 deadline,
         uint8 v, bytes32 r, bytes32 s
     ) public virtual {
         if (deadline < block.timestamp) revert
PermitDeadlineExpired(deadline, block.timestamp);
+        if (uint256(s) > _ECDSA_HALF_N) revert InvalidSignatureS();
+        if (v != 27 && v != 28)         revert InvalidSigner(address(0),
owner);
         unchecked {
             address recoveredAddress = ecrecover(...);
             if (recoveredAddress == address(0) || recoveredAddress !=
owner) revert InvalidSigner(...);
             allowance[recoveredAddress][spender] = value;
         }
         emit Approval(owner, spender, value);
     }
```

Option B - switch to OpenZeppelin's `ECDSA.tryRecover`, which applies both the low-s and the v-range checks internally. Replace the `ecrecover` call accordingly. This matches how most mature ERC-2612 implementations (including OpenZeppelin's own `ERC20Permit`) handle the signature.

Both options fix the vulnerability. Option A is lower-churn and does not introduce an OpenZeppelin runtime dependency on `MaseerToken`; Option B is a slightly heavier change but aligns with the ecosystem idiom.

Severity

Low

No direct funds loss, but the DoS / front-run vector is practically exploitable against any user whose permit lands through the public mempool, and the issue is a well-known EIP-2 compliance gap in an ERC-2612 implementation.

Category
Cryptography Business Logic

Client Response
Acknowledged; no change planned for the deployed wstGBP contracts.

Heritage. `MaseerToken.permit` is derived from solmate's ERC-20 base, which accepts non-canonical (high-s) signatures as a deliberate minimalism choice. The same gap is present in the solady update of that lineage — as of `Vectorized/solady@main`, `ERC20.permit` also passes `(v, r, s)` to `ecrecover` without an `s <= n/2` guard. EIP-2 low-s enforcement is the distinguishing choice of the OpenZeppelin lineage rather than a universal convention across ERC-2612 implementations.

Practical exposure. We regard the vector as theoretically real but academic in practice. Each permit consumes the owner's nonce regardless of which signature form lands, so there is no replay or allowance-amplification vector. The concrete exploit is mempool-level griefing: an observer trivially constructs the malleable pair (two arithmetic ops on the canonical signature), front-runs the user's submission, and makes the user's original tx revert on the incremented nonce — paying real gas to accomplish nothing economically. Every major wallet and signing library produces canonical low-s signatures by default, so the attack has no opportunistic-MEV economics, only targeted-harassment incentives; the signer's recourse is to re-sign or to set the allowance directly via `approve`.

Forward consideration. Future token deployments will revisit this choice in light of EIP-2 ecosystem practice at that time. Whether to close the gap with an inline low-s guard or retain the solmate-lineage minimalism is a live design trade-off; this finding is a valid data point in that decision but not by itself conclusive.

Auditor Notes
Acknowledged

12.4.2 (issue06) `MaseerOne.redeem` Accepts Zero-Claim Redemptions When `bpsout == 10_000`

Context

src/MaseerOne.sol#L212-L252 src/MaseerGate.sol#L62-L64 src/MaseerGate.sol#L227-L230

Carries forward audit01/issue15 (no protocol change adopted). Interacts with audit02/issue04.

Description

`MaseerOne.redeem` validates the oracle price but not the fee-adjusted price used for the actual claim calculation:

```
function redeem(uint256 amt) external burnlive pass(msg.sender) returns
(uint256 _id) {
    if (amt < WAD) revert DustThreshold(WAD);
    uint256 _unit = _read();
    if (_unit == 0) revert InvalidPrice();   // <-- pre-adjustment check
    _unit = _burncost(_unit);                 // fee-adjust
    uint256 _claim = _wmul(amt, _unit);       // no post-adjustment check
    totalPending += _claim;
    _id = redemptionCount++;
    uint96 _time = uint96(block.timestamp + _cooldown());
    redemptions[_id] = Redemption({ amount: _claim, redeemer: msg.sender,
date: _time });
    _burn(msg.sender, amt);
    emit ContractRedemption(_id, _time, msg.sender, _claim);
    if (_time == block.timestamp) { exit(_id); }
}
```

`_burncost` is:

```
function _adjustBurnPrice(uint256 _price, uint256 _bps) internal pure
returns (uint256) {
    return _divup((_price * (10_000 - _bps)), 10_000);
}
```

When `bpsout == 10_000`, `_adjustBurnPrice` returns `0` for any `_price`, so `_claim = amt * 0 / WAD = 0`. The oracle guard (`if (_unit == 0) revert InvalidPrice()`) runs against the pre-adjustment `_unit` and does not fire. The redemption is recorded with `amount = 0` and `date = block.timestamp + cooldown()`, the user's tokens are burned, no gem is owed, and `ContractRedemption(..., amount: 0)` is emitted.

`mint()` has the symmetric guard `if (amt < _unit) revert DustThreshold(_unit)` that refuses a zero-output mint; `redeem()` has no symmetric guard on its output.

The original finding (audit01/issue15) described the same behavior and recommended adding a `ClaimIsZero()` revert. The recommendation was not adopted; the Audit 01 Client Response was `_No response_`. This finding is a carryover.

Path enumeration

`_claim = amt * _burncost(navprice) / WAD` reaches `0` only through:

Precondition	`_claim` behavior	Reachable in deployed config?
`navprice == 0`	`InvalidPrice()` revert	N/A — guarded
`bpsout == 10_000`	`_burncost = 0` → `_claim = 0`, no revert	**Yes** — `bpsout` is not capped below `10_000`
`bpsout < 10_000`, `_unit > 0`, `amt >= WAD`	`_claim >= 1`	N/A — cannot reach zero

The only reachable zero-claim path is `bpsout == 10_000`. `bpsout` can be set to `10_000` through either `MaseerGate.file("bpsout", 10_000)` or, if the typed setter is added in a future version, its equivalent. There is no upper cap below `10_000` on either path.

Interaction with `audit02/issue04`

audit02/issue04 documents that `MaseerGate.file` bypasses the 365-day bound that `setHaltMint`/`setHaltBurn` enforce. The same `file(bytes32,uint256)` entry point writes `bpsout` and shares the same asymmetry: no upper bound is enforced by either path, but `file` is the path the live deployment uses for window slots. Any operator procedure that treats `file` as the "universal setter" can push `bpsout` to `10_000` in a single transaction, which is the ignition source for this issue.

Impact

- **Not systemic funds-at-risk.** The protocol cannot be drained; each occurrence is a bounded per-user value loss (the burned tokens accrue to remaining holders via NAV).
- **Individual value loss / griefing.** A redeemer who hits `redeem()` while `bpsout == 10_000` burns their tokens and receives nothing. A malicious UI (or a front-running bot racing a brief `file("bpsout", 10_000)` admin operation) can cause this at scale.

- **Admin footgun.** Because `file` has no cap, a misconfigured ward transaction or a compromised `setter` actor can place the market in this state. The condition also produces no obvious on-chain signal besides the `Bpsout(10_000)` event, so a passive indexer watching only `ContractRedemption` events will see a stream of zero-amount redemptions with no context.
- **Support / UX burden.** Users have no on-chain recourse once their tokens are burned; recovery requires off-chain coordination with the issuer (per `Treasury.issue`).

The three substantive Audit 02 changes do not alter this risk surface:

- `exit` being `public` lets a third party call `exit(id)` on a zero-amount redemption, which is a no-op (transfers `0`, decrements `totalPending` by `0`). No new vector.
- The `cooldown == 0` auto-exit (`if (_time == block.timestamp) { exit(_id); }`) does not fire here (the regression that surfaced this has `cooldown = 3600`). Even when `cooldown == 0`, the auto-exit on a zero-amount redemption is a no-op: tokens are burned, nothing is transferred, faster.
- The new `TransferToZeroAddress` check is not in the redemption path.

Reproducer

The invariant fuzzer surfaces this on a four-call sequence. The regression is captured as `test_regression_invariant_MO_redemptionDateAmountRelations_3ea8ff11_failure` in `test/invariant/regressions/MaseerProtocolRegressions.t.sol` and is currently **disabled** behind a documented action-item block until the protocol fix lands.

Minimal shape of the failing sequence:

1. An actor mints or is issued MaseerOne tokens.
2. An auth-gated actor calls `MaseerGate.file("bpsout", 10_000)` — legal under the current contract.
3. A MaseerOne holder calls `redeem(amt)` with any `amt >= WAD`.
4. The protocol burns `amt`, records `redemptions[id] = { amount: 0, date: now + cooldown, redeemer: holder }`, emits `ContractRedemption(id, now + cooldown, holder, 0)`, and the call returns.

The invariant `invariant_MO_redemptionDateAmountRelations` — introduced in Audit 01 and retained in Audit 02 — requires that any redemption with `amount == 0` have `date <= block.timestamp`. The sequence above violates this property.

Recommendation

Preferred — apply the `audit01/issue15` recommendation (add a post-adjustment zero-claim guard):

```
--- a/src/MaseerOne.sol
+++ b/src/MaseerOne.sol
@@ -88,6 +88,7 @@ contract MaseerOne is MaseerToken {
     error ClaimableAfter(uint256 time);
     error NoPendingClaim();
     error DustThreshold(uint256 min);
+    error ClaimIsZero();
     // ...
@@ -222,6 +223,9 @@ contract MaseerOne is MaseerToken {
         _unit = _burncost(_unit);
         uint256 _claim = _wmul(amt, _unit);
+        if (_claim == 0) {
+            revert ClaimIsZero();
+        }
         totalPending += _claim;
```

Equivalent alternative — re-check the fee-adjusted unit after `_burncost`, mirroring the pre-adjustment guard:

```
-        _unit = _burncost(_unit);
+        _unit = _burncost(_unit);
+        if (_unit == 0) revert InvalidPrice();
```

Either fix closes the only reachable zero-claim path (`bpsout == 10_000`). Adding an explicit upper bound `bpsout < 10_000` in `_setBpsout` is a weaker, complementary fix; it closes the specific ignition path but does not protect against future code that might introduce a second route to a zero fee-adjusted unit (e.g., a new fee modifier).

Severity
Low Risk

Category
Code Quality Invariant Correctness

Client Response
Acknowledged; no change planned. The `bpsout == 10_000` state is an intentional operational affordance, not an unintended boundary condition.

Design intent. `bpsout` governs the burn-side fee up to the limiting case of 100%. Setting `bpsout = 10_000` is the mechanism by which the ward can soft-disable the standard redemption path — for example, to route redemptions through a future upgrade bridge, execute an orderly migration, or interpose a novel corrective mechanism without pausing mint/burn windows outright. A `ClaimIsZero()` revert would forbid this state and remove an affordance we want `bpsout` to retain over the token's lifetime. Future token deployments may revisit the balance between this flexibility and stricter guardrails as the state of the art for novel-asset tokens evolves, but the current choice is deliberate.

Trust model. Real-world-asset tokenization rests on trust in the issuing entity to hold and redeem the underlying asset. Every RWA product, including this one, is load-bearing on that assumption; the `auth`-gated entry points on `MaseerGate` — `bpsout` among them — are expressions of that trust, not exceptions to it. The ward also holds strictly more destructive authority over the upgradeable modules (changing the oracle, pausing markets, altering capacity, revoking issuers), so the residual surface opened by this path is fully inside the trust envelope already documented in `audit01/issue03`.

Auditor Notes

The invariant `invariant_MO_redemptionDateAmountRelations` remains in the protocol-invariant suite; the corresponding regression is kept in `test/invariant/regressions/MaseerProtocolRegressions.t.sol` but disabled behind a `// --- DISABLED: audit02-issue06 ---` block with the action items for uncommenting it once the fix lands. The `MaseerOneHandler.redeem` handler carries a matching workaround (`if (_unit == 0) return;`, tagged `audit02-issue06`) so the live fuzz campaign no longer trips this state; the workaround is a one-line change and is scoped to the single precondition that reaches a zero claim.

12.5 Gas Optimizations

- none

12.6 Informational

12.6.1 (issue01) `MaseerGate.setTerms` Does Not Emit the Declared `Terms` Event

Context

- src/MaseerGate.sol#L42
- src/MaseerGate.sol#L168-L170

Description

`MaseerGate` declares a `Terms(string)` event at the top of the contract:

```
event Terms(string terms);
```

The setter that writes the corresponding storage slot does not emit it:

```
function setTerms(string calldata terms_) external auth {
    _setVal(_TERMS_SLOT, terms_);
}
```

Every other parameter update on `MaseerGate` - `openMint`, `haltMint`, `openBurn`, `haltBurn`, `bpsin`, `bpsout`, `cooldown`, `capacity` - goes through an internal `_set…` helper that both stores the value and emits a matching event. `setTerms` is the only auth-gated state mutation on the contract that produces no log output, and the declared `Terms` event is never referenced anywhere in the source. The structural pattern is consistent enough across the rest of the contract that the missing `emit Terms(terms_)` appears to be an omission rather than an intentional divergence.

Impact

`terms()` is exposed through `MaseerOne.terms()` and is expected to carry legal / T&C data for the token. Any off-chain indexer, monitoring pipeline, or front-end that observes `MaseerGate` state through event logs - as is standard for governance-relevant parameter changes - will not see terms updates at all. Compliance audit trails that consume event logs will be silently incomplete for this parameter, and users viewing front-end-rendered T&C may see stale text until a consumer is updated to read the storage slot directly.

Recommendation

Emit `Terms(terms_)` from `setTerms`, matching the pattern used for every other setter on this contract.

Suggested Code Changes

```
--- a/src/MaseerGate.sol
+++ b/src/MaseerGate.sol
     function setTerms(string calldata terms_) external auth {
         _setVal(_TERMS_SLOT, terms_);
+        emit Terms(terms_);
     }
```

Severity
Informational

Category
Code Quality Monitoring / Observability

Client Response
Acknowledged; no change planned for the deployed wstGBP contracts.

Scope. This is an observability gap, not a functional one. `setTerms` writes the intended storage value correctly; only the confirmation log is missing. The `terms` slot on the live wstGBP market is currently the empty string, so no downstream indexer has material state to miss today, and any consumer can read the value directly via `terms()`.

Forward consideration. Should a future revision of the `MaseerGate` implementation ship for other reasons, this one-line `emit Terms(terms_)` addition can be bundled into that upgrade at no marginal audit cost. We do not plan to ship a dedicated implementation upgrade solely for this finding.

Auditor Notes
Acknowledged

12.6.2 (issue02) `MaseerGuardOZ.pass` Does Not Explicitly Reject `address(0)`

Context
- src/MaseerGuardOZ.sol#L39-L41
- src/MaseerGuard.sol#L39-L41
- src/MaseerOne.sol#L393-L403
- src/MaseerToken.sol#L105-L132

Description
`MaseerGuardOZ`, the new compliance implementation intended for OpenZeppelin-style `AccessControl` ban-list sources, returns whether a user is allowed to pass by negating the source's `isBanned` view:

```
function pass(address usr) external view returns (bool) {
    return !GuardSource(source).isBanned(usr);
```

```
}
```

The original `MaseerGuard` (wrapping USDT's `getBlackListStatus`) has the structurally identical property. When the underlying source has not explicitly banned `address(0)`, `pass(address(0))` returns `true` and every `pass(...)`-gated code path on `MaseerOne` will accept `address(0)` as a valid principal.

This is the live state on the wstGBP mainnet deployment. tGBP (`0x27f6c8289550fCE67f6B50BeD1F519966aFE5287`) does not ban `address(0)` at the source layer:

```
cast call 0x27f6c8289550fCE67f6B50BeD1F519966aFE5287
'isBanned(address)(bool)' 0x0000000000000000000000000000000000000000
  → false
cast call 0x794cf5948444b14105587455ebe96caace036d52 'pass(address)(bool)'
0x0000000000000000000000000000000000000000
  → true
```

As a result, every `pass(...)`-gated entry point on `MaseerOne` at `0x57c3571f10767e49c9d7b60feb6c67804783b7ae` currently accepts `address(0)` as a valid address argument, except for `transfer` and `transferFrom`, which defend themselves with an inline body-level `TransferToZeroAddress()` revert introduced in response to `audit01/issue13`.

This is the same class as `audit01/issue13`, now widened to include `MaseerGuardOZ`. In response to `audit01/issue13`, Maseer added inline `TransferToZeroAddress()` reverts to `MaseerOne.transfer` / `transferFrom`:

```
function transfer(address dst, uint256 wad) public override
pass(msg.sender) pass(dst) returns (bool) {
    if (dst == address(this)) revert TransferToContract();
    if (dst == address(0)) revert TransferToZeroAddress();
    return super.transfer(dst, wad);
}
```

That correctly closes the direct funds-loss surface on the two transfer functions. It does not address the broader property that the guard itself accepts `address(0)` - and on this deployment, every other `pass(...)`-gated surface is still reachable with `address(0)` arguments.

Per-callsite exposure on the live deployment

The following functions are `pass(...)`-gated and accept `address(0)` as the gated argument today. None of them are funds-loss vectors in isolation, but several produce anomalous on-chain state that consumers of event logs and allowance accounting will have to work around:

Function	**Gated arg that can be `address(0)`**	**Effect of a call with `address(0)`**
`approve(address usr, uint256 wad)`	`usr`	Sets `allowance[caller][address(0)] = wad`. No one can ever spend from `address(0)`, so no funds move, but the storage write succeeds and the resulting non-zero allowance to `address(0)` is visible to any downstream integrator that scans allowances.
`approve(address usr)`	`usr`	Same as above with `type(uint256).max`.
`permit(owner, spender, value, …)`	`owner`, `spender`	See below - the protocol is safe here because of an inline `ecrecover` check, but the property is fragile.
`smelt(address usr, uint256 amt)` [issuer]	`usr`	`balanceOf(address(0)) == 0`, so the burn is a no-op. The call still succeeds and emits `Smelted(address(0), amt)`, polluting the event log and any compliance-oriented downstream process that audits issuer activity.

`mint`, `redeem`, `exit`, `settle`, `issue`, and the user-level `smelt(uint256)` are all gated on `pass(msg.sender)` only; `msg.sender` is never `address(0)` on Ethereum, so those paths are safe by construction.

The `permit` / `ecrecover` interaction

`ecrecover` returns `address(0)` for malformed or unrecoverable signatures. Any code that treats its return value as "a valid signer" without explicitly rejecting `address(0)` is vulnerable to a classic bypass where an attacker submits a garbage signature whose recovered address happens to be `address(0)`.

For this protocol, the permit path DOES defend itself inline:

```
// MaseerToken.sol:129
if (recoveredAddress == address(0) || recoveredAddress != owner) revert
InvalidSigner(recoveredAddress, owner);
allowance[recoveredAddress][spender] = value;
```

Consequently the combined `owner == address(0)` + `ecrecover → address(0)` attack is closed: even though `pass(owner)` accepts `owner == address(0)`, `recoveredAddress == address(0)` aborts the call before any state mutation.

However, the protocol's safety against this class of bypass is load-bearing on **one inline check inside `MaseerToken.permit`**. The guard - the one piece of code every `pass(...)`-gated caller already consults - does not centralize the defense. Any future `pass(...)`-gated entry point that accepts a caller-controlled address argument and does not independently defend against the `address(0)` sentinel inherits this class of bug. Treating the guard as the single source of truth for "this address is not allowed to transact" closes the whole category at once, and it is how every other ban-list consumer in the codebase behaves (USDT itself blacklists `address(0)` by convention on many deployments; tGBP, the source used here, happens not to).

Impact

- On the live wstGBP deployment today: any holder can call `MaseerOne.approve(address(0), N)` or submit a valid signed `permit(_, address(0), _, _, _, _, _)` and leave a non-zero allowance entry pointing at `address(0)` in the token's storage. Any issuer can call `smelt(address(0), amt)` and emit a misleading `Smelted(address(0), amt)` event. None of these move funds.
- Any future `MaseerGuardOZ` consumer whose ban-list source does not explicitly ban `address(0)` inherits the full surface (including the `ecrecover` combined attack if a future pass-gated path forgets its local defense).
- Integrators and compliance tooling that rely on "accounts allowed to transact cannot include `address(0)`" as an invariant at the guard layer are operating on an incorrect

model. The guarantee holds today only because of a single inline check in `MaseerToken.permit`; the guard itself does not uphold it.

Recommendation

In both `MaseerGuardOZ` and `MaseerGuard`, short-circuit `pass(address(0))` to `false`. This mirrors the "fail closed on `address(0)`" choice Maseer already made for the `MaseerOne.transfer` / `transferFrom` body checks, and centralizes the decision in the single contract every caller already consults.

Suggested Code Changes

```
--- a/src/MaseerGuardOZ.sol
+++ b/src/MaseerGuardOZ.sol
     function pass(address usr) external view returns (bool) {
+        if (usr == address(0)) return false;
         return !GuardSource(source).isBanned(usr);
     }
```

```
--- a/src/MaseerGuard.sol
+++ b/src/MaseerGuard.sol
     function pass(address usr) external view returns (bool) {
+        if (usr == address(0)) return false;
         return !GuardSource(source).getBlackListStatus(usr);
     }
```

After this change, `MaseerOne.transfer` / `transferFrom` can drop their inline `TransferToZeroAddress()` reverts as technically redundant - the `pass(dst)` modifier will now revert `NotAuthorized(address(0))` before the body runs. We recommend **keeping** the inline reverts anyway: they are ~20 gas each, produce a more descriptive error for integrators, and preserve the protection if a future cop flavor (or a forked deployment) is ever wired in without the short-circuit. Either choice is defensible.

Either code change also requires a one-time on-chain remediation on the live wstGBP deployment: deploy a new `MaseerGuardOZ` implementation with the fix and `file(_newImpl)` the compliance proxy at `0x794cf5948444b14105587455ebe96caace036d52` to point at it.

Reproducer

`test/unit/Audit02Issue02Reproducer.t.sol`. Three tests, runnable with:

```
make unit mc=Audit02Issue02Reproducer
```

- `testGuardOZPassReturnsTrueForZeroAddress` - demonstrates the defect directly at the guard layer.
- `testPermitToZeroAddressBypassesGuard` - shows that a valid EIP-2612 permit with `spender == address(0)` succeeds and mutates protocol state (allowance, nonce).
- `testBanningZeroAddressInSourceClosesTheHole` - shows that configuration-level mitigation (banning `address(0)` at the source) closes the hole, but is strictly weaker than fixing the guard: it only protects deployments that remember to do it.

Severity
Informational

Category
Business Logic Defense in Depth

Client Response
Acknowledged; no change planned for the deployed wstGBP contracts.

The only path by which `address(0)` could cause inadvertent loss of funds — transfers to a zero destination — is defended inline in `MaseerOne.transfer` / `transferFrom` via the `TransferToZeroAddress()` revert introduced in response to `audit01/issue13`. The remaining `pass(...)`-gated paths that accept `address(0)` — `approve`, `permit`, issuer `smelt` — all require voluntary action by an authorized caller and produce at most dead storage, log pollution, or nonce consumption; none move value. The `permit` + `ecrecover → address(0)` combined bypass that would have been the one real funds-loss vector is already closed by the inline `recoveredAddress == address(0)` revert in `MaseerToken.permit`.

Auditor Notes
Acknowledged

12.6.3 (issue03) Long-String Storage Pattern in `MaseerImplementation` Creates a Latent Storage Collision Surface

Context
- src/MaseerImplementation.sol#L27-L29
- src/MaseerImplementation.sol#L82-L88
- src/MaseerImplementation.sol#L108-L114

- src/MaseerGate.sol#L29 src/MaseerPrice.sol#L21

Description

The string setter in `MaseerImplementation` was refactored to remove the previous 32-byte cap and to use Solidity's native long-string storage layout:

```
struct StringData {
    string val;
}
```

```
function _setVal(bytes32 slot, string memory val) internal {
    StringData storage data;
    assembly {
        data.slot := slot
    }
    data.val = val;
}
```

```
function _stringSlot(bytes32 slot) internal view returns (string memory) {
    StringData storage data;
    assembly {
        data.slot := slot
    }
    return data.val;
}
```

For a string `s` stored at `slot`:

- If `bytes(s).length <= 31`, the value is packed inline into `slot` itself.
- If `bytes(s).length >= 32`, `slot` stores `(2 * length + 1)` and the actual bytes are written starting at `keccak256(slot)`, then `keccak256(slot) + 1`, and so on, for `ceil(length / 32)` slots.

This setter is used at two call sites today:

- `MaseerGate._TERMS_SLOT = keccak256("maseer.gate.terms")`
- `MaseerPrice._NAME_SLOT = keccak256("maseer.price.name")`

Both contracts share their storage namespace with every other slot written by `MaseerImplementation`, in particular:

- `_WARD_SLOT = keccak256("maseer.wards")` - per-user ward flags are stored at `keccak256(abi.encode(_WARD_SLOT, usr))`.

Because long-string storage writes into consecutive `keccak256(_TERMS_SLOT) + k` slots, and those slots live in the same flat 2^256-sized storage space as the ward-flag slots, there is a latent collision surface: if an address `usr` happened to hash such that `keccak256(abi.encode(_WARD_SLOT, usr))` falls inside the range `[keccak256(_TERMS_SLOT), keccak256(_TERMS_SLOT) + ceil(len / 32))`, then a `setTerms` call with a long-enough string could overwrite that user's ward bit, or vice versa.

For the two slots in use today, finding such an overlap requires a second preimage of the keccak256-derived ward-slot hash, which is cryptographically infeasible. The concern is therefore not an exploitable collision on the current surface - it is a bookkeeping hazard that compounds as the contract evolves. The previous 32-byte hard cap in `_setVal(bytes32, string)` was an implicit guarantee that string slots could not tail into adjacent storage; removing it, without introducing an explicit reservation for the tail range, means any future addition of a new mapping keyed off `_WARD_SLOT` - or a second long-string slot near `_TERMS_SLOT` / `_NAME_SLOT` - can silently begin to overlap if a string grows past 31 bytes.

Impact

- No current exploitability: overlap requires a keccak256 second-preimage.
- Forward-looking concern: the long-string pattern spreads writes across an unbounded-length tail of consecutive storage slots, and every slot in that tail shares the same namespace as every mapping and every other `bytes32 internal constant` slot on the contract. Future additions to `MaseerImplementation` and its subclasses need to treat the tail of each string slot as reserved storage. Without an explicit reservation mechanism, that reasoning is implicit and easy to forget across future changes.

Recommendation

Any one of the following closes the forward-looking hazard. They are ordered from lightest to heaviest:

1. **Document the reservation.** Add a comment at `_setVal(bytes32, string)` stating that every slot used with this setter reserves a length-dependent suffix range `[keccak256(slot), keccak256(slot) + ceil(maxLen / 32))`, and that no other storage in the contract hierarchy may alias that range. Pair with the existing

`__gap[50]` entries on the concrete contracts to make the reservation visible at the consumer.

2. **Bound the length at the call site.** Add a conservative length cap to `setTerms` (and to any future string setter). A 512-byte cap covers realistic T&C payloads by an order of magnitude and limits the reserved tail to 16 consecutive slots, which can be explicitly carved out of the storage gap:

```
--- a/src/MaseerGate.sol
+++ b/src/MaseerGate.sol
     function setTerms(string calldata terms_) external auth {
+        if (bytes(terms_).length > 512) revert
UnrecognizedParam(bytes32(bytes(terms_)));
         _setVal(_TERMS_SLOT, terms_);
+        emit Terms(terms_);
     }
```

3. **Switch to ERC-7201-style slot derivation.** Derive string slots as `keccak256(abi.encode(<contract>, "<name>"))` and reserve the corresponding tail explicitly in the contract's storage gap. This is the approach OpenZeppelin uses for their namespaced storage pattern and makes the reservation both machine-checkable and documentation-adjacent. For the two current uses, the derivation is equivalent to the existing constants, but making it explicit preserves the property across future refactors.

Severity
Informational

Category
Storage Layout Code Quality

Client Response
Acknowledged; no change planned for the deployed wstGBP contracts.

The concern is forward-looking: the long-string storage pattern leaves a length-dependent tail of consecutive slots reserved, and future additions to `MaseerImplementation` or its subclasses need to respect that reservation. There is no current exploitability — constructing an overlap between `keccak256(_TERMS_SLOT) + k` and `keccak256(abi.encode(_WARD_SLOT, usr))` requires a second preimage of a keccak256-derived slot hash, which is cryptographically infeasible. The hazard is bookkeeping discipline across future code evolution, not an in-situ attack surface.

Auditor Notes

No response

12.6.4 (issue04) `MaseerGate.file()` Bypasses the 365-Day Bound That `setHaltMint` / `setHaltBurn` Enforce

Context

- src/MaseerGate.sol#L122-L140
- src/MaseerGate.sol#L172-L182
- script/MaseerOnetGBP.s.sol#L111-L114

Description

`MaseerGate` exposes two different auth-gated code paths for writing each window slot (`openmint`, `haltmint`, `openburn`, `haltburn`). Only the typed setters enforce the 365-day forward cap:

```
function setHaltMint(uint256 halt_) external auth {
    if (halt_ > block.timestamp + 365 days) revert
UnrecognizedParam(bytes32(halt_));
    _setHaltMint(halt_);
}
```

```
function setHaltBurn(uint256 halt_) external auth {
    if (halt_ > block.timestamp + 365 days) revert
UnrecognizedParam(bytes32(halt_));
    _setHaltBurn(halt_);
}
```

```
function file(bytes32 what, uint256 data) external auth {
    if      (what == "openmint") _setOpenMint(data);
    else if (what == "haltmint") _setHaltMint(data);  // <-- no bound check
    else if (what == "openburn") _setOpenBurn(data);
    else if (what == "haltburn") _setHaltBurn(data);  // <-- no bound check
    ...
}
```

`file(bytes32,uint256)` writes the same four slots via the internal `_set…` helpers without the bound, so a caller with ward access can set any of the four windows to an arbitrary `uint256` through the `file` entry point.

The wstGBP deployment script on mainnet (`script/MaseerOnetGBP.s.sol#L111-L114`) relies on this asymmetry to set the halt dates to `type(uint256).max` - effectively "never halts":

```
MaseerGate(MASEER_MARKET_PROXY).setOpenMint(block.timestamp);
MaseerGate(MASEER_MARKET_PROXY).file("haltmint", type(uint256).max);
MaseerGate(MASEER_MARKET_PROXY).setOpenBurn(block.timestamp);
MaseerGate(MASEER_MARKET_PROXY).file("haltburn", type(uint256).max);
```

On-chain state (queried against the live wstGBP deployment at `act = 0xb59cb4d3075a8ce5013c78e8bd7ada3fd1300f7f`):

```
act.haltMint() =
115792089237316195423570985008687907853269984665640564039457584007913129639
935  (2^256 - 1)
act.haltBurn() =
115792089237316195423570985008687907853269984665640564039457584007913129639
935  (2^256 - 1)
```

The sequence is legal given the current code, but it surfaces a design inconsistency in `MaseerGate` that materially changes what a reader of the contract should assume is true about the protocol:

- A reader of `setHaltMint` / `setHaltBurn` would conclude that the protocol enforces a forward limit of ≤ 365 days on halt dates. That conclusion is false in general - the same slot is writable to any `uint256` through `file(...)`.
- Any off-chain system, invariant, or safety argument that relies on "halt dates are bounded to ≤ 1 year from now" is unsound against deployments that exercise the `file` path.

This is the concrete consequence of the broader asymmetry already surfaced in `audit01/issue02` ("MaseerGate mixes paradigms for setting state"): two code paths that write the same slot have different semantics, and a live deployment now depends on that difference.

Impact

- Not funds-at-risk. The live deployment is legal given the contract's actual behavior.

- The 365-day bound advertised by the typed setters is not a protocol invariant; any reasoning or tooling that depends on it is wrong whenever `file(...)` is used.
- Safety assumptions about the window state - for example, "the market will always close within 1 year unless manually rolled" - do not hold. Operators who want that guarantee need additional off-chain discipline (never call `file("haltmint"/"haltburn", ...)` with a value beyond `block.timestamp + 365 days`), which is not self-enforcing.

Recommendation

Either:

1. **Enforce the bound in `file` too** (preferred; aligns both code paths with the setter-level invariant):

```
--- a/src/MaseerGate.sol

+++ b/src/MaseerGate.sol

     function file(bytes32 what, uint256 data) external auth {
-        if      (what == "openmint") _setOpenMint(data);
-        else if (what == "haltmint") _setHaltMint(data);
-        else if (what == "openburn") _setOpenBurn(data);
-        else if (what == "haltburn") _setHaltBurn(data);
+        if      (what == "openmint") setOpenMint(data);
+        else if (what == "haltmint") setHaltMint(data);
+        else if (what == "openburn") setOpenBurn(data);
+        else if (what == "haltburn") setHaltBurn(data);
         else if (what == "bpsin")    _setBpsin(data);
         else if (what == "bpsout")   _setBpsout(data);
         else if (what == "cooldown") _setCooldown(data);
         else if (what == "capacity") _setCapacity(data);
         else    revert UnrecognizedParam(what);
     }
```

   …and change `setOpenMint` / `setHaltMint` / `setOpenBurn` / `setHaltBurn` from `external auth` to `public auth` so `file` can call them internally.

2. **Remove the bound from the setters** if "no halt limit" is the intended product behavior, and make the setter and `file` paths agree on that.

Option 1 keeps the current deployment valid only if the intent is actually to cap halt dates at 365 days - in which case the current state `haltMint == 2^256 - 1` is itself a deploy-time

violation and needs a corrective call. If the intent is "no upper bound", option 2 is the correct fix and the current deployment is fine.

Severity
Informational

Category
Code Quality Invariant Correctness

Client Response
Acknowledged; no change planned for the deployed wstGBP contracts.

The asymmetry between the typed setters and `file(bytes32, uint256)` is intentional: typed setters carry bounds that model the normal-case operational envelope (365-day halt rolls, sub-100% fees), while `file(...)` is the unconstrained configuration path for deploy-time and product-lifecycle decisions that sit deliberately outside that envelope. The wstGBP deployment uses `file("haltmint", type(uint256).max)` and `file("haltburn", type(uint256).max)` by design — the product is configured never to halt under routine operation, and the on-chain state matches the script's stated intent.

Auditor Notes
Acknowledged

---

13. Maseer - Audit 02 Auditor Notes

Audience: Maseer engineering team and future reviewers with access to the `pit-maseer` audit harness repository. Everything here is complementary to the above public report. It documents the internal mechanics of the Audit 02 test harness, handler conventions, regression bookkeeping, and workarounds that a reader without `pit-maseer` checked out could not make use of directly.

13.1 Invariant fuzz campaign - handler-level changes

This round's extensions to the `pit-maseer` handler tree:

- `MaseerOneHandler.transfer` / `transferFrom`: now expect `TransferToZeroAddress()` when `dst == address(0)`.
- `MaseerOneHandler.redeem`: tracks `cooldown == 0` auto-exit and, on success, captures the pre-call price and `bpsout` to accumulate `fairClaim` / `actualClaim` /

`slackBudget` per actor for `invariant_MP_noDrainByRounding` (see Deployment Verification §6.4).

- `MaseerOneHandler.redeem` also records `redemptionCount` monotonicity violations and cooldown-zero auto-exit violations into dedicated counters.
- `MaseerOneHandler.issue` / `smeltUser`: bump unauthorized-caller counters when a non-issuer succeeds.
- `MaseerOneHandler.mint`: bumps a counter if `totalSupply` exceeds the pre-call capacity after a successful mint (for `invariant_MO_capacityRespectedByMint`).
- `MaseerOneHandler.permit`: full EIP-2 malleability test - 10% of calls compute the canonical `(s' = n - s, v' = v ^ 1)` malleable pair of a valid signature. Any such call succeeding bumps `malleableSignatureSuccessCount`.
- `MaseerGateHandler.setTerms`: new selector, intentionally oscillates between ≤ 31 and ≤ 128 byte strings to exercise both the inline and long-string storage paths.
- `MaseerGateHandler.pauseMarket`: inline `require(!gate.mintable() && !gate.burnable())` on the success branch.
- `MaseerPrecommitHandler.exec`: maintains a live-deal-id list so `exec` picks known-non-empty commitments, and warms up the oracle / mint window via `vm.prank(pipAuth)` / `vm.prank(actAuth)` when required. Lifted the `exec` success rate from ~2% to ~15%, materially improving the signal-to-noise of the `MaseerPrecommit` fuzz branch.

13.2 Cop flavor parametrization

The compliance-module deployment inside `MaseerOneInvariants.setUp` is controlled by a `useOzGuard` flag on `BaseMaseerProtocolInvariants`. `BaseMaseerHandler._onBlacklist` dispatches between `USDT.getBlackListStatus` and `MockGuardSourceOZ.isBanned` based on the active flavor. `addBlacklist` / `removeBlacklist` handlers route to `ban` / `unban` on the OZ variant and to `USDT.addBlackList` / `removeBlackList` (pranked as the Tether owner) on the USDT variant. Every handler and every invariant is reused across both flavors without duplication.

The flavor is selected per-run via the `COP_FLAVOR` environment variable:

Command	**Flavor**
`make invariant`	`oz` (default, matches the live wstGBP deployment)
`COP_FLAVOR=usdt make invariant`	Legacy USDT-backed `MaseerGuard`

Command	Flavor
`make invariant-all`	Runs both flavors back-to-back

Regression replay also honors `COP_FLAVOR`; each captured regression's doc comment records the flavor under which it was originally observed so a reviewer can reproduce in the matching environment.

13.3 Depth ramp

Each `make invariant` run exercises **96 tests across 7 test suites** in the current layout (six per-module invariant suites plus one protocol-wide integration suite). The campaign was ratcheted through the following depths without invariant violations after the price-aware reformulation of `invariant_MP_noDrainByRounding`:

- `make invariant` (Foundry default depth ≈ 500)
- `make invariant depth=1000`
- `make invariant depth=2500`
- `make invariant depth=5000`
- `make invariant depth=10000`

`depth=20000` is the next rung, to be followed by `make overnight`. Handed back to the client for further long-soak runs, but Prototech will run many iterations of these over the course of the next month too.

13.4 Test-harness workarounds and process notes

13.4.1 `audit02-issue02 (12.6.2)` source-layer workaround

Both cop flavors (`MaseerGuard` on USDT, `MaseerGuardOZ` on the mock) pre-ban `address(0)` at the source layer inside `MaseerOneInvariants.setUp` so the fuzz campaign does not re-discover the `address(0)` bypass on every run.

- **OZ branch**: `ozSource.ban(address(0));`
- **USDT branch**: `vm.prank(TETHER_OWNER); USDT.addBlackList(address(0));`

Each workaround site is tagged with the string `audit02-issue02` in the source tree. To find and remove every workaround once `MaseerGuardOZ.pass` / `MaseerGuard.pass` are patched:

```
grep -rn "audit02-issue02" test/
```

The corresponding invariant regression is committed but **commented out** inside `test/invariant/regressions/MaseerProtocolRegressions.t.sol` (`test_regression_invariant_MO_allowanceZeroAndSelfContract_08082f7b_failure`). Re-enable instructions are inline on that test. The standalone reproducer at `test/unit/Audit02Issue02Reproducer.t.sol` does not depend on the workaround and can be run independently at any time.

This is the only handler-level workaround currently in place.

13.4.2 `audit01/issue02` + `audit02/issue04 (12.6.4)` - paused-window invariant

The two pre-existing invariants

- `invariant_MG_mintWindowBounds`
- `invariant_MG_burnWindowBounds`

are commented out in `test/invariant/MaseerGateInvariants.t.sol`. They encode the 365-day cap that the typed `setOpenMint`/`setHaltMint`/`setOpenBurn`/`setHaltBurn` setters enforce, but that is bypassed by the `file(bytes32, uint256)` path - see `audit02/issue04 (12.6.4)`. Re-enable instructions are inline above the commented block, tagged with both `audit01/issue02` and `audit02/issue04`.

13.4.3 `audit02-issue05` (12.4.1) signature-malleability workaround

`MaseerToken.permit` does not enforce EIP-2 low-s canonical signatures (see `audit02/issue05`). A correct malleable-pair transform in the `MaseerOneHandler.permit` chaos-monkey branch would therefore trip `invariant_MO_malleableSignatureRejected` on every invariant run, drowning out signal for other invariants.

Until the protocol adopts the fix, the handler suppresses malleable-pair generation and lets the chaos-monkey branch always take the invalid-sig path. The suppression site is tagged with `audit02-issue05` in the source tree; grep to find it:

```
grep -rn "audit02-issue05" test/
```

The standalone reproducer at `test/unit/Audit02Issue05Reproducer.t.sol` preserves the proof of the protocol vulnerability independent of the campaign. When the protocol patch lands, the workaround is deleted (restoration snippet is inline) and the invariant becomes a proper regression detector for future changes.

13.4.4 Earlier shadow-based paused-market invariant

The first iteration of `invariant_MG_pausedMarketConsistent` used a shadow bool on `MaseerGateHandler` to remember "`pauseMarket` was the last observed window mutation." Once the `MaseerPrecommitHandler.exec` success-path warm-up was added (which pranks as `actAuth` and re-opens the mint window to lift `exec`'s success rate), the shadow became stale and the invariant tripped. We replaced it with an inline `require(!gate.mintable() && !gate.burnable())` on the `pauseMarket` success branch. This is a narrower guarantee than the original - the inline check only proves `pauseMarket` was effective at the moment of the call, not that the market stays closed through subsequent handler activity - but it avoids giving false positives under legitimate cross-handler re-warming. Restoring the stronger property cleanly would require threading the gate handler into every other handler that can open windows, or switching to an event-driven shadow; we've left it at the narrower check for now but note the option explicitly here.

13.5. Rounding safety - `audit01/issue14` mitigation proof

`audit01/issue14` ("Rounding Issues in MaseerGate and MaseerOne") closed with a code-level recommendation for Maseer: change the `_adjustBurnPrice` formula and simplify `_wmul` to remove a drain surface that the Audit 01 invariant campaign had surfaced. Maseer adopted both changes in the diff under review. This section records the positive-proof work we did this round to validate that the adopted changes fully close the drain surface - not just that the Audit 01 regression does not reproduce, but that no rounding-drain can be constructed against the revised formulas. It is maintainer-facing because it refers to the `pit-maseer` unit suite and invariant implementation; the public report summarizes the outcome without the detail.

13.5.1 Unit-level property proofs

The property-test suite at `test/unit/MaseerRounding.t.sol` (21 tests, all passing) exercises:

1. **Single-trip no-drain** - concrete and fuzzed. User mints `amt` gem → receives tokens → redeems all → total gem out ≤ gem in, under every combination of `bpsin`, `bpsout`, `p`.
2. **Looped no-drain** - `N ∈ {2, 10, 100, 10 000}` cycles of mint/redeem, at `bps ∈ {0, 50}`, both uniform cycles and adversarial splitting (one mint → 1 000 tiny redeems).

3. **Partial-fill no-drain** - gem starved out of the contract via `settle`, redemption paid partial, then topped up and claimed. Total gem out ≤ gem in.
4. **Price-swing no-drain** - oracle moved `×10` up, `×10` down, and paused (`== 0`) between mint and redeem. Any gain bounded by the price ratio; rounding cannot compound on top.
5. **Dust no-DoS** - minimum mintable and redeemable amounts produce ≥ 1 token / ≥ 1 wei respectively, except when `bpsout == 10 000` (100% burn fee), which is by-design economically zero.
6. **Fee-capture lower bound** - with `bpsin > 0` or `bpsout > 0`, the protocol retains at least `floor(amt * bps / (10_000 + bps))` wei per cycle, across single and looped round-trips.

The analytical write-up of why each step in the round-trip rounds in the protocol's favor is at the top of `test/unit/MaseerRounding.t.sol` and reproduced here:

- `_unit_mint = divup(p*(10_000+bpsin), 10_000)` - rounds **up** (protocol favor).
- `_out = floor(amt * WAD / _unit_mint)` via `_wdiv` - rounds **down** (protocol favor).
- `_unit_burn = divup(p*(10_000-bpsout), 10_000)` - rounds **up**. At `bpsout = 0` this is exactly `p` (no rounding surplus); at `bpsout > 0` the ceiling can exceed the rational value by up to < 1 wei per unit of price. This is a locally user-favorable rounding direction.
- `_claim = floor(_out * _unit_burn / WAD)` via `_wmul` - rounds **down** (protocol favor).

Every step except the burn-side divup rounds in the protocol's direction. The divup surplus (< 1 wei per unit of price, only when `bpsout > 0`) is multiplied by `_out < amt·WAD / p` and then floored by `WAD`, which is already absorbed by the mint-side `_wdiv` floor and by the `bpsin` fee in all real round trips. Looping cannot turn this local divup drift into net gain: the unit suite includes `testLoopedNoDrain_N10000_bps0` and `testLoopedNoDrain_N10000_bps50` which assert exactly this across 10,000 round trips at both `bps = 0` and `bps = 50`; the live-campaign invariant below additionally bounds per-redemption drift to `amt/WAD + 1` wei and would trip if any deployment ever exceeded that bound.

13.5.2 Live-campaign invariant and general methodology

The unit-level suite proves the rounding claim under controlled inputs. To catch regressions that only surface under the combinatorial state the invariant campaign explores, we also carry a price-aware drain invariant that is active throughout every run.

The invariant. For every actor `A`:

```
actualClaim[A] <= fairClaim[A] + slackBudget[A]
```

where each term is accumulated on successful `redeem(amt)` calls, using the pre-call oracle price `p` and the pre-call `bpsout`:

- `fairClaim[A] += floor(amt · p · (10_000 − bpsout) / (10_000 · WAD))` - the rounding-free fair claim for that redemption, computed in 512-bit precision via `Math.mulDiv`.
- `actualClaim[A] += floor(amt · burncost_pre / WAD)` where `burncost_pre = divup(p · (10_000 − bpsout), 10_000)` is exactly what `MaseerOne.burncost()` returns at the pre-call instant. This matches the value the protocol stores in `redemptions[_id].amount`.
- `slackBudget[A] += amt / WAD + 1` - the per-operation drift bound derived below.

The invariant is implemented in `test/invariant/MaseerProtocolInvariants.t.sol::invariant_MP_noDrainByRounding`; the shadow-tracking is in `test/invariant/handlers/MaseerOneHandler.sol`.

Why it is price-invariant. `fairClaim` is evaluated at the redemption's own-moment price and bps. Moving the oracle between a mint and a subsequent redeem changes both sides of the inequality by the same amount (modulo per-op slack), so legitimate price-move-induced gains do not produce false positives. This is exactly the case that tripped an earlier, non-price-aware version of the invariant (regression `test_regression_invariant_MP_noDrainByRounding_16820f7c` in `test/invariant/regressions/MaseerProtocolRegressions.t.sol` captures the exact sequence; it now passes under the price-aware formulation).

Why it is token-flow-invariant. The invariant compares the protocol's recorded claim against its own fair-value computation at the time of the redeem. It does not depend on who minted the tokens, who holds them, whether they were transferred, or whether they were issued without gem backing. Cross-actor transfers, `issue()` flows, and `MaseerPrecommit.exec()` flows are all orthogonal to this property; no tainting logic is required.

Derivation of the slack bound. For a single redemption of `amt` tokens at price `p` with fee `bpsout`, the protocol computes the burn unit as:

```
burncost_pre = ceil( p * (10_000 - bpsout) / 10_000 )       // = divup(...)
```

Let `fair_unit = p * (10_000 - bpsout) / 10_000` as an exact rational. Then `burncost_pre - fair_unit` lies in `[0, 1)` wei. The protocol's recorded claim is:

```
actual_claim = floor( amt * burncost_pre / WAD )            // = _wmul(amt, burncost_pre)
             <= amt * burncost_pre / WAD
             <= amt * (fair_unit + 1) / WAD
              = fair_exact + amt / WAD                        // where fair_exact = amt * fair_unit / WAD
```

Taking the floor of `fair_exact` to make it representable in `uint256` costs at most one additional wei. Summing across any N redemptions yields:

```
actualClaim[A] <= fairClaim[A] + sum_j ( amt_j / WAD + 1 )
               = fairClaim[A] + slackBudget[A]
```

The slack is **tight**: any change to the protocol's rounding direction that lifts `actualClaim` even slightly above this bound (for example, swapping the `_wmul` floor for a ceiling, or replacing `divup` with a round-half-up variant) will accumulate drift above the slack budget after enough redemptions and trip the invariant.

General methodology. The pattern above generalises to any protocol with a conversion formula of the form `out = f(in, state)` where `f` applies rounding. The recipe we used here, and that callers may lift into their own audit harnesses:

1. **Identify every rounding operation** in the conversion path: every `floor`, `ceil`, `divup`, `_wmul`, `_wdiv`, etc. Record the direction of each.
2. **Write the rounding-free "fair" formula** as a rational expression. For MaseerOne redeem: `fair = amt * p * (10_000 - bpsout) / (10_000 · WAD)`.
3. **Bound the per-op drift**: compute an upper bound on `actual_i - fair_i` using the rounding-direction analysis from step 1. For this protocol, the bound is `amt/WAD + 1` wei per redemption.
4. **Shadow-track in the handler** at every successful conversion: capture the pre-call state (price, fees, inputs), compute the fair value with 512-bit precision (`Math.mulDiv(a, b, c)` from OpenZeppelin handles this safely), and accumulate per-actor `fair[actor]`, `actual[actor]`, `slack[actor]`.
5. **Assert the invariant** as `actual[A] <= fair[A] + slack[A]` for every actor `A`. Because `fair` is recomputed per operation, this is automatically price-invariant; because it checks the protocol's computation rather than economic outcome, it is automatically token-flow-invariant.

The scaffolding is small enough to sit inside an audit-specific handler; if the same pattern recurs across enough engagements, a generic `RoundingDrainTracker` mixin with `fairAccum` / `actualAccum` / `slackAccum` mappings plus a `checkNoDrain(address)` view could be promoted into `pit-base` for reuse.

13.6. Campaign artifacts and layout

Artifact	**Path**
Invariant suite (integration + per-module)	`test/invariant/`
Handlers	`test/invariant/handlers/`
Regression harness	`test/invariant/regressions/MaseerProtocolRegressions.t.sol`
Standalone rounding proofs	`test/unit/MaseerRounding.t.sol`
Issue-02 reproducer	`test/unit/Audit02Issue02Reproducer.t.sol`
Deployment verification script	`scripts/verify-deployment.sol`
Methodology writeup for the price-aware no-drain invariant	§13.5 of this document

`MaseerProtocolInvariants` is the env-var-parameterized integration suite. `MaseerProtocolRegressionTests` is the single regression-harness contract that replays historical captures; each test in it records its original-capture flavor in its doc comment so a reviewer can reproduce in the matching environment by setting `COP_FLAVOR` appropriately.

13.7. How to triage a failing campaign run

1. `make invariant depth=<N>` (or `COP_FLAVOR=usdt make invariant depth=<N>`). Non-zero exit means one or more invariants fired.
2. If a failure is captured, Foundry writes a log under `logs/`. Use `make gen-regression-all` to synthesize regression tests from those logs into `test/invariant/regressions/`. The generator derives the target file from the

failing suite name (`MaseerProtocolInvariants` → `MaseerProtocolRegressions.t.sol`).

3. Replay with `make regression mc=<contract> mt=<test>`.
4. If the failure is a legitimate bug: open a new `issues/audit02/issueNN.md`, write a standalone reproducer under `test/unit/`, tag any handler workarounds with `audit02-issueNN` so they can be reverted once the protocol is patched, and either keep the captured sequence in the regression file or note why it was disabled.
5. If the failure is a false positive (invariant over-claims against legitimate behavior): re-derive the invariant and, if necessary, retire it. Leave a comment describing what went wrong so a future change doesn't re-introduce the same mistake.

Section 13.5 of this document describes the recipe for deriving a rounding-drain invariant correctly (identify every rounding op, derive the fair formula, bound per-op drift, shadow-track at every successful conversion). Apply the same recipe when authoring invariants for new protocol surfaces.
