Galactic Pools Security Assessment

Engagement ID: ASL-GP-2025-02
Client: Galactic Pools Core Team
Assessment Lead: Aurora Security Labs
Assessment Window: 7–10 February 2025
Repository Commit (baseline): 0f8a88d8fac586cbe33fa268be881468b6f5601a

---

Executive Summary

Aurora Security Labs performed a focused manual review of the Galactic Pools smart-contract suite, concentrating on prize-pool lifecycle management and participant withdrawal flows. One high-severity issue was identified that enabled malicious participants to repeatedly claim excess yield during abort scenarios, draining rewards owed to other players. The issue has been remediated, validated, and regression-tested. No additional critical risks were observed, though a handful of operational hardening suggestions are provided for future consideration.

Severity	Findings	Status
High	1	Remediated
Medium	0	–
Low	1 (operational)	Accepted
Informational	2	Accepted

Assessment Scope

- `contracts/PrizePool.sol`
- `contracts/PrizePoolFactory.sol`
- `contracts/PrizePoolRouter.sol`
- Supporting libraries (`libraries/PoolConfigLib.sol` , `yield/BasicYieldSource.sol` , utility contracts)
- Hardhat integration tests (`test/PrizePool.test.ts`)

Out-of-scope components (front-end, deployment scripts, infrastructure) were only reviewed for context.

Methodology

1. **Threat Modeling:** Evaluated user roles (participant, sponsor, admin), state transitions, and external dependencies (Goat VRF, Permit2, yield sources).
2. **Manual Code Review:** Line-by-line analysis emphasizing state transitions, fund custody, access control, and randomization safety.
3. **Differential Testing:** Added exploit regression coverage and executed the Hardhat test suite.
4. **Impact Validation:** Constructed step-by-step exploit sequences to measure financial impact and confirm remediation efficacy.

Findings Overview

[GP-01] Repeated `withdrawExpired` Calls Drained Yield (High Severity)

- **Description:** When a pool missed its target (or emergency refund was enabled) participants could call `withdrawExpired` multiple times while omitting their principal token list. Because yield distribution occurred before principal balances were fully cleared, weight calculations remained unchanged and the caller could repeatedly harvest a pro-rata share of the remaining yield. Each successive call captured a majority share of the outstanding rewards, leaving little for honest participants.
- **Impact:** Theft of nearly all yield owed to other users in abort scenarios; direct financial loss with no admin recourse once abuse begins.
- **Exploit Steps (pre-fix):**
  1. Allow the mission to lapse without a draw and ensure yield has been harvested.
  2. Invoke `withdrawExpired([])` repeatedly (empty token array) to claim yield while leaving principal tracked.
  3. After draining yield, withdraw principal normally; honest users receive negligible rewards.
- **Remediation:** `withdrawExpired` now zeroes a participant's normalized balance before distribution, clears outstanding boost credits, requires every deposited token to be withdrawn in the same call (reverting otherwise), and ensures the function can be executed only once per participant.
- **Developer Actions:** Implemented in `contracts/PrizePool.sol` (see diff below) with regression test `prevents repeated expired withdrawals from draining yield` at `test/PrizePool.test.ts`.
- **Status:** Remediated and validated.

```
+        deposits[msg.sender] = 0;
+        if (totalPrincipal >= userNorm) {
+            totalPrincipal -= userNorm;
+        } else {
+            totalPrincipal = 0;
+        }
+        if (boostCredits[msg.sender] > 0) {
+            boostCredits[msg.sender] = 0;
+        }
...
+        for (uint256 i = 0; i < supportedStableList.length; i++) {
+            address token = supportedStableList[i];
+            if (userTokenDeposits[msg.sender][token] > 0) {
+                revert OutstandingPrincipal();
+            }
+        }
```

Additional Observations (Non-Critical)

- **[GP-N1] Contributor Array Growth (Low):** `contributors` is append-only; extremely large participant counts could push `receiveRandomness` gas usage toward block limits. Monitor participant volume and consider weighted data structures for future scaling.
- **[GP-I1] Admin Delegation Surface:** `adminMulticall` intentionally exposes `delegatecall` to the pool admin. Maintain strict key hygiene and monitoring when using this capability.

- **[GP-I2] Referral Credit Reset Logistics:** Referral credits persist after expired withdrawals. This is acceptable for current abort flows but should be considered if reactivating pools without redeployment.

Remediation Verification

- Reviewed `contracts/PrizePool.sol` changes introducing `OutstandingPrincipal` guardrails and single-use bookkeeping.
- Confirmed unit test coverage via:

  ```
  npm run test:contracts
  ```

- Validated new regression test prevents the pre-fix exploit sequence.
- Checked that existing prize-award and refund paths remain unaffected.

Recommendations

1. **Operational Monitoring:** Add off-chain alerts for sudden spikes in `withdrawExpired` attempts to supplement on-chain safeguards.
2. **Scalability Planning:** Evaluate alternative winner-selection data structures before scaling to very large user counts.
3. **Documentation Synchronization:** Surface the new security report link on the Docs page (implemented in this engagement).

Conclusion

The Galactic Pools prize system now resists the identified high-severity attack, and regression coverage protects against reintroduction. Aurora Security Labs recommends continuing scheduled reviews ahead of major feature drops or blockchain network migrations.

For further clarification, contact audits@aurorasec.io.

Aurora Security Labs – Securing the next wave of on-chain experiences.