Stonewall
Smart Contract Security

Lemonad Gaming Suite Security Review

Introduction

A time-boxed security review of the **Lemonad Gaming Suite** contracts was conducted by **Stonewall**, with a focus on the security aspects of the smart contract implementation.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

About Stonewall

Stonewall is an independent smart contract security firm delivering immovable protection for Web3 protocols. Our team brings deep expertise in DeFi security, having reviewed DEXs, yield farming protocols, gaming contracts, and complex financial systems.

About Lemonad Gaming Suite

The Lemonad Gaming Suite provides on-chain gaming experiences:

- **LemonDice**: Provably fair dice rolling game
- **LemonLotto**: Lottery system with random draws
- **LemonPredict**: Prediction markets using Pyth oracle price feeds
- **LemonBattles**: PvP battle system with matchmaking
- **SqueezeRacing**: Multi-player racing game
- **GameRegistry**: Central registry for game configurations

- **EntropyManager**: VRF provider for randomness across all games
- **NFTStatsRegistry**: Tracks NFT statistics for game participation

Privileged Roles & Actors

Role	Description
Owner	Can modify game parameters, pause games, withdraw emergency funds
EntropyManager	Centralized VRF provider for randomness in games
GameRegistry	Controls which games are active and can modify configurations

Observations

- Protocol uses Pyth Network for price feeds in prediction markets
- Games utilize commit-reveal pattern with VRF for fair randomness
- NFT integration for game participation and yield boosting
- Centralized entropy management for all randomness needs

Risk Classification

	High Impact	Medium Impact	Low Impact
High Likelihood	Critical	High	Medium
Medium Likelihood	High	Medium	Low
Low Likelihood	Medium	Low	Low

Impact

- **High**: Leads to significant loss of user funds, protocol insolvency, or complete protocol failure
- **Medium**: Leads to partial loss of funds, temporary denial of service, or governance manipulation
- **Low**: Leads to minor issues, inconvenience, or suboptimal behavior

Likelihood

- **High**: Attack is easy to perform and likely to happen
- **Medium**: Attack requires specific conditions but is feasible
- **Low**: Attack requires significant effort, resources, or unlikely conditions

Security Assessment Summary

Review Details	
Protocol Name	Lemonad Gaming Suite
Repository	Private
Commit	`4bcdafa9703e197329cbc0cff193a50457decc6c`
Review Date	January 2026
Methods	Manual review, static analysis
Network	Monad Mainnet (Chain ID: 143)

Deployed Contract Addresses

Contract	Address
GameRegistry	`0x564F6AE37410B96dde6F4D6cBBc42A7954222CF9`
LemonBattles	`0xBeD26CB7cd6a9a510A721e15F4E2d3Fe7973d08D`
SqueezeRacing	`0xDcE5a579A11B82F7533767B028DC9f2DA19D3791`
LemonLotto	`0xA008B0FB38F9f90175834F0C28f6dA34ed7bc894`
LemonDice	`0x9306F5c8d95f182639a115Ec5f416d67E2321676`
LemonPredict	`0xc3130BBE86FFFD91e2495a8BdDb4bAD091A17078`
NFTStatsRegistry	`0xB008B5623722A3b50b940a63d3C1778Ce1Db9463`
EntropyManager	`0xf22Fb668F1e9129e12C3AdE0f7A52aa72b844474`

Project Links

Platform	Link
Website	lemonad.one
Twitter	@LeMONAD_Factory
Telegram	LeMONAD_Factory
Discord	Join Discord
Docs	GitBook

Scope

Contract	SLOC
`games/LemonDice.sol`	~200
`games/LemonLotto.sol`	~250
`games/LemonPredict.sol`	~350
`games/LemonBattles.sol`	~370
`games/SqueezeRacing.sol`	~400
`games/EntropyManager.sol`	~100
`games/GameRegistry.sol`	~120
`games/NFTStatsRegistry.sol`	~80

Findings Summary

ID	Title	Severity	Status
[M-01]	Stale oracle price can resolve prediction markets incorrectly	Medium	Open
[M-02]	Emergency refund logic creates confusing user experience	Medium	Open
[L-01]	Unbounded loop in matchmaking can cause DoS	Low	Open
[L-02]	Active races array iteration may become expensive	Low	Open

[L-03]	Multiple unbounded arrays never cleaned up	Low	Open

Findings

[M-01] Stale oracle price can resolve prediction markets incorrectly

Severity: Medium

Impact: High - Markets could be resolved with outdated prices, causing incorrect outcomes and financial losses.

Likelihood: Low - Requires oracle to return stale data, which is uncommon but possible.

Location: `games/LemonPredict.sol:157`

Description:

The contract uses `pyth.getPriceUnsafe()` which returns price data without validating freshness:

```
IPyth.Price memory priceData = pyth.getPriceUnsafe(market.priceId);
```

During periods of high volatility or network congestion, the price data could be significantly outdated, leading to incorrect market resolution.

Attack Scenario:

1. Market closes at timestamp T
2. Price at T is $100, but cached oracle price is from T-1 hour showing $95
3. Users who bet "under $97" win incorrectly
4. Legitimate winners based on actual price lose funds

Recommendation:

Use `pyth.getPrice()` which validates freshness, or add explicit staleness check:

```
IPyth.Price memory priceData = pyth.getPriceUnsafe(market.priceId);
require(block.timestamp - priceData.publishTime < MAX_PRICE_AGE, "Stale price");
```

[M-02] Emergency refund logic creates confusing user experience

Severity: Medium

Impact: Low - Poor user experience and failed transactions.

Likelihood: Medium - Occurs whenever emergency refund is triggered.

Location: `games/LemonPredict.sol:323-333`

Description:

When `emergencyRefund` is called, it sets `finalPrice = 0`. However, `claimWinnings` doesn't check for this condition, so users attempt to claim winnings (which fails) before realizing they need to call `claimRefund` instead.

Recommendation:

Add a check in `claimWinnings`:

```
function claimWinnings(uint256 marketId) external {
    Market storage market = markets[marketId];
    require(market.resolved, "Market not resolved");
    require(market.finalPrice != 0, "Market was refunded - use claimRefund");
    // ...
}
```

[L-01] Unbounded loop in matchmaking can cause DoS

Severity: Low

Location: `games/LemonBattles.sol:197-210`

Description:

The `_tryMatchmaking` function iterates through all pending entries:

```
for (uint256 i = 0; i < pendingEntryIds.length; i++) {
    // ...
}
```

If the pending queue grows large, gas costs become prohibitively expensive.

Recommendation:

Add maximum iteration limit:

```
uint256 maxIterations = pendingEntryIds.length > 100 ? 100 : pendingEntryIds.length;
for (uint256 i = 0; i < maxIterations; i++) {
```

[L-02] Active races array iteration may become expensive

Severity: Low

Location: `games/SqueezeRacing.sol - _removeFromActive()`

Description:

Removing races iterates through the active array. With many concurrent races, this becomes expensive.

Recommendation:

Consider limiting maximum concurrent races or using more efficient data structures.

[L-03] Multiple unbounded arrays never cleaned up

Severity: Low

Location: Multiple files

Description:

Arrays tracking battles, entries, and races grow indefinitely without cleanup, increasing gas costs over time.

Recommendation:

Implement periodic cleanup or use mappings with separate counters.

Informational Findings

[I-01] Centralized VRF Management

The EntropyManager creates a single point of failure for randomness. If compromised or offline:

- Dice games cannot resolve
- Lotto draws cannot complete
- Battle outcomes cannot be determined
- Race results cannot be finalized

Consider documenting trust assumptions and planning migration to decentralized VRF (e.g., Chainlink VRF when available on Monad).

[I-02] Commit-Reveal Pattern Implementation

Games properly implement commit-reveal pattern preventing front-running of random outcomes. This is a positive security pattern.

[I-03] NFT Integration for Games

NFT stats tracking is properly isolated and doesn't introduce security concerns to the gaming contracts.

Security Patterns Observed

Positive

- Solidity ^0.8.20+ with overflow protection
- ReentrancyGuard on state-changing functions
- Proper VRF implementation for randomness
- Commit-reveal pattern prevents front-running
- House edge properly enforced
- Game pausing capability for emergencies

Concerns

- Centralized entropy management
- Oracle dependency for predictions
- Unbounded array growth

- No circuit breakers for large losses

Conclusion

The Lemonad Gaming Suite implements solid patterns for on-chain gaming, including proper randomness via commit-reveal and VRF. The main concerns are:

1. **Oracle staleness** in prediction markets
2. **Centralized entropy** creating single point of failure
3. **Unbounded arrays** causing potential gas issues

Overall Risk Assessment: Medium

This security review was conducted by Stonewall. For questions or clarifications, contact our team.