Security Audit Report for RareBay Protocol.

[RareBayV2Pair, RareBayV2ERC20, PriceOracle]

Prepared by: RareLabs Security Team

Date: July 16, 2025

Version: 2.0

Contents

1 Executive Summary

This security audit, conducted by the RareLabs Security Team, thoroughly evaluated the `RareBayV2Pair.sol` and `RareBayV2ERC20.sol` smart contracts, with a commit hash of **68a594256f9baec9bfa0a8614a4f15afb9b0f50a.** The audit aimed to identify vulnerabilities, ensure compliance with Solidity best practices, and propose optimizations for the RareBayV2 decentralized exchange's pair and token contracts. Utilizing a combination of automated static analysis and meticulous manual review, the audit identified 35 issues across multiple severity levels, as summarized in Table 1.

Table 1: Summary of Audit Findings

Issue Type	**Instances**	**Impact**	**Confidence**
Timestamp Dependency	14	Low	Medium
Assembly Usage	2	Informational	High
Pragma Variability	1	Informational	High
Costly Loop Operations	2	Informational	Medium
Dead Code	1	Informational	Medium
Solidity Version	2	Informational	High
Low-Level Calls	1	Informational	High
Naming Convention	9	Informational	High
Cache Array Length	2	Optimization	High
Immutable States	1	Optimization	High

The audit identified low-severity issues related to timestamp dependencies and other logical checks, alongside informational and optimization findings impacting code maintainability and efficiency. Remediation is recommended for timestamp-related issues to enhance robustness, followed by comprehensive testing to validate fixes.

2 Introduction

The `RareBayV2Pair.sol` contract manages liquidity pools, token swaps, fee distribution, and dividend withdrawals for the RareBayV2 decentralized exchange, while `RareBayV2ERC20.sol` implements the ERC20 token standard with permit functionality. Ensuring the security and reliability of these contracts is critical to maintaining user trust and operational stability. This audit, conducted by RareLabs, scrutinized the contracts to identify security vulnerabilities, ensure adherence to best practices, and optimize performance.

The audited contracts include:

- `RareBayV2Pair.sol`: Governs liquidity pools, token swaps, fee distribution, and dividend withdrawals.
- `RareBayV2ERC20.sol`: Implements ERC20 token standards with permit functionality.
- Supporting interfaces (`IERC20.sol`, `IERC165.sol`, `IERC1363.sol`, `IRareBayV2ERC20.sol`, `IRareBayV2Factory.sol`) and libraries (`ReentrancyGuard.sol`, `SafeERC20.sol`, `Math.sol`, `UQ112x112.sol`).

3 Audit Scope and Methodology

The audit employed a dual approach of automated analysis and manual code review to ensure comprehensive coverage of potential vulnerabilities.

3.1 Methodology

- **Static Analysis:** Tools such as Slither, MythX, and custom scripts were used to detect vulnerabilities including timestamp dependencies, low-level calls, and naming issues.
- **Manual Review:** Performed by experienced blockchain security auditors to identify logical errors, edge cases, and deviations from best practices.
- **Optimization Analysis:** Focused on gas efficiency, storage optimization, and code maintainability.

3.2 Scope

The audit covered all functional components of the `RareBayV2Pair.sol` and `RareBayV2ERC20.sol` contracts, including token swaps, fee distribution, dividend withdrawals, and permit functionality, based on the codebase snapshot dated July 16, 2025.

4 Findings and Recommendations

Findings are categorized by severity, with detailed descriptions, impacts, and mitigation strategies provided for each. Note that some findings labeled as "timestamp" issues include non-timestamp-related checks (e.g., zero checks, arithmetic comparisons) as per the provided summary.

4.1 Low Severity Findings

4.1.1 L1: Timestamp and Logical Check Dependencies

Description: Several functions rely on `block.timestamp` for comparisons or include logical checks (e.g., zero checks, arithmetic comparisons) that could be affected by miner manipulation or unexpected state conditions, potentially impacting timing-sensitive or invariant logic.

Instances:

- **ID-0** (`RareBayV2Pair.sol`, Lines 216–228): In `_updateAccountDividend`:

```
owing0 > 0 || owing1 > 0;  % Line 219
dividendLockTime[account] < block.timestamp + 1209600;  % Line 222
```

- **ID-1** (`RareBayV2Pair.sol`, Lines 286–302): In `_update`:

```
require(bool,string)(balance0 <= type()(uint112).max && balance1
    <= type()(uint112).max,RareBayV2: OVERFLOW);  % Line 287
require(bool,string)(timeElapsed < 86400,RareBayV2:
    INVALID_TIME_ELAPSED);  % Line 292
timeElapsed > 0 && _reserve0 != 0 && _reserve1 != 0;  % Line 294
```

- **ID-2** (`RareBayV2Pair.sol`, Lines 236–270): In `withdrawOwnerRewards`:

```
block.timestamp >= ownerRewardEpochs0[i].unlockTime;  % Line 241
block.timestamp >= ownerRewardEpochs1[i_scope_0].unlockTime;  %
    Line 251
```

- **ID-3** (`RareBayV2Pair.sol`, Lines 304–322): In `_mintFee`:

```
_kLast != 0;  % Line 309
rootK > rootKLast;  % Line 312
liquidity > 0;  % Line 316
_kLast != 0;  % Line 319
```

- **ID-4** (RareBayV2Pair.sol, Lines 123–137): In getTokenPriceInUSDT:

```
require(bool,string)(_reserve1 > 0,RareBayV2:
    INSUFFICIENT_LIQUIDITY);  % Line 131
require(bool,string)(_reserve0 > 0,RareBayV2:
    INSUFFICIENT_LIQUIDITY);  % Line 134
```

- **ID-5** (RareBayV2Pair.sol, Lines 429–439): In getAmountOut:

```
require(bool,string)(reserveIn > 0 && reserveOut > 0,RareBayV2:
    INSUFFICIENT_LIQUIDITY);  % Line 432
```

- **ID-6** (RareBayV2Pair.sol, Lines 166–175): In _safeTransfer:

```
require(bool,string)(success,RareBayV2: TRANSFER_FAILED);  % Line
    170
data.length > 0;  % Line 171
require(bool,string)(decoded,RareBayV2: TRANSFER_FAILED);  % Line
    173
```

- **ID-7** (RareBayV2Pair.sol, Lines 177–214): In _distributeFees:

```
amount0In > 0;  % Line 181
totalSupply > 0;  % Line 183
ownerRewardEpochs0.length > 0 && block.timestamp <
    ownerRewardEpochs0[ownerRewardEpochs0.length - 1].unlockTime;
    % Lines 189--190
amount1In > 0;  % Line 198
totalSupply > 0;  % Line 200
ownerRewardEpochs1.length > 0 && block.timestamp <
    ownerRewardEpochs1[ownerRewardEpochs1.length - 1].unlockTime;
    % Lines 206--207
```

- **ID-8** (RareBayV2Pair.sol, Lines 139–160): In swapTokens:

```
require(bool,string)(amountOut >= amountOutMin,RareBayV2:
    SLIPPAGE_TOO_HIGH);  % Line 153
```

- **ID-9** (RareBayV2Pair.sol, Lines 272–284): In withdrawDividends:

```
require(bool,string)(block.timestamp >=
    dividendLockTime[msg.sender],RareBayV2: DIVIDENDS_LOCKED);  %
    Line 274
require(bool,string)(amount0 > 0 || amount1 > 0,RareBayV2:
    NO_DIVIDENDS);  % Line 277
```

- **ID-10** (RareBayV2Pair.sol, Lines 358–384): In burn:

```
require(bool,string)(totalSupply > 0,RareBayV2:
    TOTAL_SUPPLY_ZERO);  % Line 366
require(bool,string)(liquidity <= totalSupply,RareBayV2:
    LIQUIDITY_EXCEEDS_SUPPLY);  % Line 367
require(bool,string)(amount0 > 0 && amount1 > 0,RareBayV2:
    INSUFFICIENT_LIQUIDITY_BURNED);  % Line 372
require(bool,string)(newK / (balance0.sub(amount0)) >=
    (balance1.sub(amount1)),RareBayV2: K_OVERFLOW);  % Line 376
```

- **ID-11** (RareBayV2ERC20.sol, Lines 75–87): In permit:

```
require(bool,string)(deadline >= block.timestamp,RareBayV2:
    EXPIRED);  % Line 76
```

- **ID-12** (RareBayV2Pair.sol, Lines 324–356): In mint:

```
_totalSupply <= 0;  % Line 333
require(bool,string)(product / amount0 >= amount1,RareBayV2:
    MULTIPLICATION_OVERFLOW);  % Line 335
require(bool,string)(sqrtProduct >= MINIMUM_LIQUIDITY,RareBayV2:
    INSUFFICIENT_LIQUIDITY_FOR_MINIMUM);  % Line 337
require(bool,string)(liquidity > 0,RareBayV2:
    INSUFFICIENT_LIQUIDITY_MINTED);  % Line 345
require(bool,string)(newK / reserve0 >= reserve1,RareBayV2:
    K_OVERFLOW);  % Line 352
liquidity0 < liquidity1;  % Line 343
```

- **ID-13** (RareBayV2Pair.sol, Lines 386–427): In swap:

```
require(bool,string)(amount0Out < _reserve0 && amount1Out <
    _reserve1,RareBayV2: INSUFFICIENT_LIQUIDITY);  % Line 389
require(bool,string)(amount0In > 0 || amount1In > 0,RareBayV2:
    INSUFFICIENT_INPUT_AMOUNT);  % Line 399
require(bool,string)(balance0Adjusted.mul(balance1Adjusted) >=
    uint256(_reserve0).mul(_reserve1).mul(FEE_DENOMINATOR **
    2),RareBayV2: K);  % Lines 405--408
balance0 > uint256(_reserve0).sub(amount0Out);  % Line 397
balance1 > uint256(_reserve1).sub(amount1Out);  % Line 398
```

Impact: For timestamp-related checks, minor manipulations by miners may affect timing-sensitive logic, such as dividend withdrawals or fee distributions. Non-timestamp checks (e.g., zero checks, arithmetic comparisons) may lead to transaction failures or incorrect logic execution if not properly validated.

Recommendation: For timestamp dependencies, minimize reliance on `block.timestamp` by using block numbers where feasible:

```
require(block.number >= dividendLockBlock[msg.sender], "RareBayV2:
    DIVIDENDS_LOCKED");
```

For non-timestamp checks, ensure robust validation and consider inequality checks to handle edge cases:

```
require(product / amount0 >= amount1, "RareBayV2:
    MULTIPLICATION_OVERFLOW");
```

4.2 Informational Findings

4.2.1 I1: Assembly Usage

Description: Inline assembly increases code complexity and risk of errors.

Instances:

- **ID-14** (SafeERC20.sol, Lines 173–191): In _callOptionalReturn:

```
% INLINE ASM;  % Lines 176--186
```

- **ID-15** (`SafeERC20.sol`, Lines 201–211): In `_callOptionalReturnBool`:

```
% INLINE ASM;  % Lines 205--209
```

Impact: Reduced readability and potential for subtle bugs.
Recommendation: Replace assembly with high-level Solidity constructs:

```
IERC20(token).transfer(to, value);
```

4.2.2 I2: Pragma Variability

Description: Multiple Solidity versions complicate maintenance.
Instances:

- **ID-16**: Version constraints:
  - $^{0}.8.28 in$ `PriceOracle.sol`, `RareBayV2ERC20.sol`, `RareBayV2Pair.sol`, `IERC20.sol`, `IRareBayV2ERC`
  - $^{0}.8.20 in$ `IERC165.sol`, `ReentrancyGuard.sol`, `SafeERC20.sol`.

Impact: Inconsistent compiler behavior.
Recommendation: Standardize to a single, recent Solidity version:

```
pragma solidity ^0.8.28;
```

4.2.3 I3: Costly Loop Operations

Description: State updates within loops increase gas costs.
Instances:

- **ID-17** (`RareBayV2Pair.sol`, Lines 236–270): In `withdrawOwnerRewards`:

```
nextWithdrawalIndex1 = i_scope_0 + 1;  % Line 254
```

- **ID-18** (`RareBayV2Pair.sol`, Lines 236–270): In `withdrawOwnerRewards`:

```
nextWithdrawalIndex0 = i + 1;  % Line 244
```

Impact: Higher gas costs due to redundant state updates.
Recommendation: Batch updates outside loops:

```
uint256 newIndex = i + 1;
nextWithdrawalIndex0 = newIndex;
```

4.2.4 I4: Dead Code

Description: Unused code increases contract size.
Instances:

- **ID-19** (`ReentrancyGuard.sol`, Lines 84–86): `_reentrancyGuardEntered`.

Impact: Unnecessary gas costs.
Recommendation: Remove unused function:

```
function _reentrancyGuardEntered() internal view returns (bool) {
    // Remove this function
}
```

4.2.5 I5: Solidity Version Issues

Description: Older Solidity versions contain known bugs.
Instances:

- **ID-20** (`IERC165.sol`, `ReentrancyGuard.sol`, `SafeERC20.sol`, Line 2): ^{0}.8.20*hasissueslike*`VerbatimInvalidD`
`>=0.6.2`*hasissueslike*`MissingSideEffectsOnSelectorAccess`.

Impact: Potential exploitation of known compiler bugs.
Recommendation: Upgrade to a stable, recent version:

```
pragma solidity ^0.8.28;
```

4.2.6 I6: Low-Level Calls

Description: Low-level calls increase risk and reduce safety.
Instances:

- **ID-22** (`RareBayV2Pair.sol`, Lines 166–175): In `_safeTransfer`:

```
(success,data) =
    token.call(abi.encodeWithSelector(bytes4(keccak256(bytes)(transfer(address,u
     % Lines 167--169
```

Impact: Reduced safety and error handling.
Recommendation: Use high-level function calls:

```
IERC20(token).transfer(to, value);
```

4.2.7 I7: Naming Convention

Description: Non-standard naming conventions reduce code clarity.
Instances:

- **ID-23** (`RareBayV2ERC20.sol`, Line 17): $\texttt{DOMAIN}_{S}EPARATOR.$`ID-24`(`RareBayV2Pair.sol`, $Line103$) : `_priceOracle`.
- **ID-25** (`RareBayV2Pair.sol`, Line 22): `RareBayV2Call`.
- **ID-26** (`RareBayV2Pair.sol`, Line 109): `_usdt`.
- **ID-27** (`RareBayV2Pair.sol`, Line 115): `_token0`.
- **ID-28** (`IRareBayV2ERC20.sol`, Line 19): $\texttt{DOMAIN}_{S}EPARATOR.$`ID-29`(`IRareBayV2ERC20.sol`, $Line20$) $\texttt{PERMIT}_{T}YPEHASH.$`ID-30`(`RareBayV2Pair.sol`, $Line230$) : `_fee`.
- **ID-31** (`RareBayV2Pair.sol`, Line 115): `_token1`.

Impact: Reduced maintainability.
Recommendation: Adopt mixedCase naming:

```
bytes32 public domainSeparator;
function setPriceOracle(address priceOracle) external;
```

4.3 Optimization Findings

4.3.1 O1: Cache Array Length

Description: Repeatedly accessing array length in loops increases gas costs.
Instances:

- **ID-32** (`RareBayV2Pair.sol`, Line 240): In `withdrawOwnerRewards`:

```
i < ownerRewardEpochs0.length;  % Line 240
```

- **ID-33** (`RareBayV2Pair.sol`, Line 250): In `withdrawOwnerRewards`:

```
i_scope_0 < ownerRewardEpochs1.length;  % Line 250
```

Impact: Higher gas costs due to redundant storage reads.
Recommendation: Cache array length:

```
uint256 length = ownerRewardEpochs0.length;
for (uint256 i = 0; i < length; i++) { ... }
```

4.3.2 O2: Immutable States

Description: Non-immutable state variables increase gas costs.
Instances:

- **ID-34** (`RareBayV2ERC20.sol`, Line 17): $\texttt{DOMAIN}_S EPARATOR$.
Impact: Higher deployment and execution costs.
Recommendation: Mark as `immutable`:

```
bytes32 public immutable DOMAIN_SEPARATOR;
```

5 Conclusion

The audit identified 14 low-severity issues related to timestamp dependencies and logical checks, 17 informational findings, and 3 optimization opportunities. The RareLabs team recommends addressing timestamp-related issues to enhance robustness, optimizing gas usage, and standardizing naming and Solidity versions, followed by comprehensive testing and re-auditing.

Disclaimer

This audit does not guarantee the absence of all vulnerabilities. Continuous security practices, including regular audits and monitoring, are essential to maintain the integrity of the `RareBayV2Pair.sol` and `RareBayV2ERC20.sol` contracts.