

ABRACADABRA MONEY · SECURITY REVIEW

Abracadabra Money PurrSwap Final Review.

A security review of PurrSwap Final Review.

CLIENT	PROJECT	AUDITORS	AUDIT WINDOW	FINAL REPORT
Abracadabra Money	PurrSwap Final Review	Guardian	April 21 – April 30, 2025	April 30, 2025

§ 00

Contents.

17 pages: overview, confidence ranking, scope, 6 findings, disclaimer, and about Guardian.

§ 01	Overview	P. 03
§ 02	Confidence Ranking	P. 04
§ 03	Scope & Classifications	P. 05
	Files in Scope	P. 06
§ 04	Findings	P. 07
	Main Review · 6 findings	P. 08
§ 05	Disclaimer	P. 16
§ 06	About Guardian	P. 17

§ 01

Overview.

Guardian reviewed PurrSwap Final Review across a single review round. The final report includes 1 High, 1 Medium, 4 Low findings.

AUDIT FIRM	REPOSITORY
Guardian	https://github.com/Abracadabra-money/purrswap-contracts
CLIENT	METHODOLOGY
Abracadabra Money	Smart contract · Manual · Main Review
PROJECT	MAIN REVIEW COMMIT
PurrSwap Final Review	a8b18881cc35
PREPARED BY	
Guardian	
AUDIT WINDOW	
April 21 – April 30, 2025	
FINAL REPORT	
April 30, 2025	

1.1 Confidence Ranking

4 / 5

High Confidence

Guardian assigns a Confidence Ranking of 4 out of 5, combining finding severity, remediation outcome, and the reviewer’s assessment of latent risk. The full scale is on the next page.

1	2	3	4	5
VERY LOW CONFIDENCE	LOW CONFIDENCE	MODERATE CONFIDENCE	HIGH CONFIDENCE	VERY HIGH CONFIDENCE

1.2 Findings at a Glance

SEVERITY	TOTAL	PENDING	ACK	PARTIAL	RESOLVED
● Critical	0	0	0	0	0
● High	1	0	0	0	1
● Medium	1	0	0	1	0
● Low	4	0	3	0	1
● Informational	0	0	0	0	0
All severities	6	0	3	1	2

§ 02

Confidence Ranking.

Guardian assigns a project-level confidence score using the scale below. The ranking combines finding severity, remediation outcome, and the reviewer's assessment of latent risk.

CONFIDENCE RANKING	DEFINITION AND RECOMMENDATION	RISK PROFILE
5 Very High Confidence	Codebase is mature, clean, and secure. No High or Critical vulnerabilities were found. Follows modern best practices with high test coverage and thoughtful design. Recommendation: code is highly secure at time of audit.	0 High/Critical findings and few Low/Medium severity findings.
4 High Confidence ASSIGNED	Code is clean, well-structured, and adheres to best practices. Design patterns are sound and test coverage is strong. Recommendation: suitable for deployment after remediations; consider periodic review with changes.	0-1 High/Critical findings per engagement week and little to no Medium severity findings.
3 Moderate Confidence	Code is functional but shows gaps in defensive programming or test coverage. Recommendation: remediate the findings and re-review the changes before deployment.	Some High/Critical findings, with various Medium severity findings.
2 Low Confidence	Recurring issues suggest systemic gaps rather than isolated mistakes. Recommendation: remediate, then commission a follow-up review at a frozen commit.	Over 6 High and Critical findings across the engagement.
1 Very Low Confidence	The volume and severity of findings indicate the codebase is not ready for production. Recommendation: substantial rework, then a full re-review.	Over 12 High and Critical findings across the engagement.

§ 03

Scope & Classifications.

Guardian reviewed PurrSwap Final Review in

<https://github.com/Abracadabra-money/purrrswap-contracts>. The scope includes 1,259 source lines across 11 files.

3.1 Severity Matrix

	IMP · HIGH	IMP · MED	IMP · LOW
LIK · HIGH	Critical	High	Medium
LIK · MED	High	Medium	Low
LIK · LOW	Medium	Low	Low

IMPACT · HIGH

Significant loss of assets, harm to a group of users, or core protocol functionality is disrupted.

IMPACT · MEDIUM

Small amount of funds can be lost, or ancillary functionality is affected. Users may experience reduced or delayed receipt.

IMPACT · LOW

Unexpected behavior in some functionality that is notable but does not meet the criteria for higher severity.

LIKELIHOOD · HIGH

Possible with reasonable assumptions that mimic on-chain conditions; cost is low compared to amount gained.

LIKELIHOOD · MEDIUM

Possible only in uncommon cases or requires a large amount of capital relative to disruption caused.

LIKELIHOOD · LOW

Unlikely to ever occur in production.

Files in Scope.

FILE	NSLOC	TOTAL
contracts/Factory.sol	61	92
contracts/interfaces/IPurrFactory.sol	3	5
contracts/interfaces/IPurrLP.sol	3	5
contracts/interfaces/IPurrLPOracle.sol	3	6
contracts/interfaces/IWETH.sol	3	5
contracts/libraries/DecimalMath.sol	39	55
contracts/libraries/Math.sol	115	216
contracts/libraries/PMPricing.sol	99	136
contracts/PurrLP.sol	451	602
contracts/PurrLPFees.sol	23	29
contracts/Router.sol	459	611

§ 04

Findings.

Grouped by round. Visible IDs reset by severity within each round, so a finding is named the same way in the report as it was in the channel it was written in. Status is where each finding ended up; its own page says how.

ID	TITLE	CATEGORY	SEVERITY	STATUS	PAGE
MAIN REVIEW					
H-01	RState Manipulation Through Liquidity Donation	Math	HIGH	RESOLVED	p. 8
M-01	Reentrancy Allows Liquidity Donations	Reentrancy	MEDIUM	PARTIALLY RESOLVED	p. 10
L-01	Lazy Updating of Pool State When Oracle Price Changes	Warning	LOW	ACKNOWLEDGED	p. 12
L-02	Gas Optimization In sellBase And sellQuote	Gas Optimization	LOW	ACKNOWLEDGED	p. 13
L-03	Inefficient Oracle Design	Informational	LOW	RESOLVED	p. 14
L-04	Pools can be skewed when AUTO_COMPOUND_FEES is true	Informational	LOW	ACKNOWLEDGED	p. 15

HIGH **H-01**

RESOLVED

RState Manipulation Through Liquidity Donation

CATEGORY Math

LOCATION <https://github.com/GuardianOrg/purrrswap-contracts-fuzz/blob/a8b18881cc35adcee8e667223d03e0ff4e783659/contracts/PurrLP.sol#L387-L388>

REVIEW Main Review

— DESCRIPTION

A malicious user can create a new pool and steal funds from subsequent liquidity providers through a series of liquidity donations. The donated liquidity will over-inflate B and Q while not inflating B0 and Q0. A subsequent minimal swap (sellBase() or sellQuote()) will cause the RState to flip to 1 or 2 and cause a major target imbalance. The specific steps to perform the exploit are listed below:

- Malicious user (Alice) creates pool with min liquidity (1 ether quote/base)
- Alice donates a lot of base and a little bit of quote and buys shares. This increases base reserves while barely increasing total supply.
- Similarly Alice donates a lot of quote and a little bit of base and then buys shares.

At this point, the pool reserves are large but the total supply is still low. The base and quote targets are also very low.

- Bob comes along and mints liquidity as normal by transferring equal ratio base/quote and buying shares.
- Alice sells a single wei of quote. This puts the pool in a wonky state where B0 becomes much larger than B.
- Alice swaps some amount of base for a larger amount of quote.
- Alice sells all shares, makes a profit.

— RECOMMENDATION

Our initial recommendation is to send the excess Base or Quote received during `buyShares()` to the recipient `to`. If the malicious user intentionally sends a lot of one token and a little of the other, the extra value received from the first token will be returned and will not contribute to any imbalance.

In `buyShares()`:

```
```solidity
} else if (baseReserve > 0 && quoteReserve > 0) {
// case 2. normal case
uint256 baseInputRatio = DecimalMath.divFloor(baseInput, baseReserve);
```

**H-01** RState Manipulation Through Liquidity Donation

CONTINUED

```
uint256 quoteInputRatio = DecimalMath.divFloor(quoteInput, quoteReserve);
uint256 mintRatio = quoteInputRatio < baseInputRatio ? quoteInputRatio : baseInputRatio;
shares = DecimalMath.mulFloor(totalSupply(), mintRatio);

if (baseInputRatio < quoteInputRatio) {
// Return excess quote token to user
_QUOTE_TOKEN_.safeTransfer(to, (quoteInput * (quoteInputRatio - baseInputRatio)) / quoteInputRatio);
quoteBalance = _QUOTE_TOKEN_.balanceOf(address(this));
} else if (baseInputRatio > quoteInputRatio) {
// Return excess base token to user
_BASE_TOKEN_.safeTransfer(to, (baseInput * (baseInputRatio - quoteInputRatio)) / baseInputRatio);
baseBalance = _BASE_TOKEN_.balanceOf(address(this));
}
```

It's also important to block the donation path in `sellShares()`. Currently, it's easy to donate to the reserves by transferring base and quote to the LP contract and calling `sellShares()` with minimal shares.

**PROOF OF CONCEPT** <https://gist.github.com/fatherGoose1/91490beb80a739da2cf63ccda23b1e74>

**— RESOLUTION**

Status: Resolved.

<https://github.com/Abracadabra-money/purrrswap-contracts/commit/aec23e65105cf96a8188ab7f7860fbe478ea0483>

MEDIUM **M-01**

PARTIALLY RESOLVED

# Reentrancy Allows Liquidity Donations

CATEGORY Reentrancy

LOCATION <https://github.com/GuardianOrg/purrrswap-contracts-fuzz/blob/a8b18881cc35adcee8e667223d03e0ff4e783659/contracts/PurrrLP.sol#L290> <https://github.com/GuardianOrg/purrrswap-contracts-fuzz/blob/a8b18881cc35adcee8e667223d03e0ff4e783659/contracts/PurrrLP.sol#L319>

REVIEW Main Review

## DESCRIPTION

In the `sellBase` and `sellQuote` functions, the contract calls `_transferQuoteOut` or `_transferBaseOut` before updating the `_BASE_RESERVE` / `_QUOTE_RESERVE` through `_setReserve`. If the token supports "on-transfer" hooks (e.g. some tokens or ERC777-like hooks), an external actor could abuse this to deposit extra tokens into the contract during that outgoing transfer. Because the pool subsequently calls `balanceOf` to set its reserve, the newly donated tokens become part of `_BASE_RESERVE` / `_QUOTE_RESERVE`. This effectively forces the contract to accept an involuntary "donation," which can skew the pool's ratio and circumvent the intended "no donation" checks.

**\*\*Example\*\*:**

1. `sellBase` is called. The pool calculates `receiveQuoteAmount` and then does `_transferQuoteOut(to, receiveQuoteAmount)`.
2. During that transfer, the token's on-transfer hook triggers, sending additional tokens back into the pool.
3. After the transfer, `_setReserve` calls `*QUOTE_TOKEN*.balanceOf(address(this))`, which now includes that extra deposit. The pool's `_QUOTE_RESERVE` is updated to the new higher balance, accepting the donation.

## RECOMMENDATION

When sending tokens out, do not rely solely on post-transfer `balanceOf` to update `_BASE_RESERVE` / `_QUOTE_RESERVE`. Instead, record the pre-transfer token balance and compute the new balance by subtracting the known outgoing amount. For example, in `sellBase()` when transferring quote out:

```
```solidity
uint256 oldQuoteBal = _QUOTE_TOKEN.balanceOf(address(this));

// do the transfer
_transferQuoteOut(to, receiveQuoteAmount);
```

M-01 Reentrancy Allows Liquidity Donations

CONTINUED

```
// compute new balance by subtracting
// any donation from an on-transfer hook
uint256 newQuoteBal = oldQuoteBal - receiveQuoteAmount;

// now setReserve using newQuoteBal
_setReserve(baseBalance, newQuoteBal);
` ``
```

— RESOLUTION

Status: Partially resolved.

<https://github.com/Abracadabra-money/purrrswap-contracts/commit/7c1a8138bc1b818bb43c460fb6f43b73dffbe67c>

LOW **L-01**

ACKNOWLEDGED

Lazy Updating of Pool State When Oracle Price Changes

CATEGORY Warning
LOCATION `PurrLp.sol`
REVIEW Main Review

— DESCRIPTION

The PMM framework in `PurrLP` does not immediately re-label the pool's `RState` (e.g. from `BELOW_ONE` to `ABOVE_ONE`) as soon as the oracle price `I` changes. Instead, it employs a "lazy updating" design: the new price `_I_` only takes effect when a user action (such as a swap, a deposit, or a withdrawal) calls `(getPMMState)` and re-derives the equilibrium state via `_adjustTarget`. This can lead to a temporary mismatch where, for instance, the pool's records show it is short on quote when in fact, based on the updated oracle, it should be short on base.

— RECOMMENDATION

Retain or clearly document this lazy-updating approach. It is normal for a DODO-style PMM, as all actual arithmetic for trades or deposits uses the up-to-date oracle price at execution time. No additional immediate code changes are necessary, but clarity in documentation can help integrators understand that `RState` might look "behind" until a user triggers a recalculation through a swap.

— RESOLUTION

Status: Acknowledged.

LOW **L-02**

ACKNOWLEDGED

Gas Optimization In sellBase And sellQuote

CATEGORY	Gas Optimization
LOCATION	https://github.com/GuardianOrg/purrrswap-contracts-fuzz/blob/a8b18881cc35adcee8e667223d03e0ff4e783659/contracts/PurrLP.sol#L298 https://github.com/GuardianOrg/purrrswap-contracts-fuzz/blob/a8b18881cc35adcee8e667223d03e0ff4e783659/contracts/PurrLP.sol#L327
REVIEW	Main Review

— DESCRIPTION

In both `sellBase` and `sellQuote`, the contract performs

```
` `solidity
_BASE_TARGET_ = newBaseTarget.toUint112();
`
```

and

```
` `solidity
_QUOTE_TARGET_ = newQuoteTarget.toUint112();
`
```

respectively, even though `newBaseTarget == state.B0 == _BASE_TARGET_` and `newQuoteTarget == state.Q0 == _QUOTE_TARGET_`. This makes the assignments effectively redundant.

— RECOMMENDATION

Consider removing these assignments to reduce the gas costs.

— RESOLUTION

Status: Acknowledged.

LOW **L-03**

RESOLVED

Inefficient Oracle Design

CATEGORY	Informational
LOCATION	https://github.com/Abracadabra-money/purrrswap-contracts/blob/a8b18881cc35adcee8e667223d03e0ff4e783659/contracts/PurrLP.sol#L228
REVIEW	Main Review

DESCRIPTION

The price, `l`, is returned via a call to an oracle. The call is performed as `\ uint256 i = IPurrLP0racle(*ORACLE*).price(address(*BASE_TOKEN*)) ; \` The oracle contract does not allow for base/quote token pair prices as it only takes a single input, base token. Therefore, a separate oracle contract must be created for each quote token.

RECOMMENDATION

This could be optimized to take a base token and quote token parameter. A single oracle contract could serve multiple LP contracts using this approach.

RESOLUTION

Status: Resolved.

<https://github.com/Abracadabra-money/purrrswap-contracts/commit/4cf3d416e9da4a0c62335caa22cd539f842a1c21>

LOW **L-04**

ACKNOWLEDGED

Pools can be skewed when `AUTO_COMPOUND_FEES` is true

CATEGORY Informational

REVIEW Main Review

— DESCRIPTION

Similar to direct donations, the pool targets can be skewed when `AUTO_COMPOUND_FEES = true`. For a stable pool, it can always be expected that the pool will achieve equilibrium at roughly the same token balance of base and quote tokens. However, when the pool is auto-compounding, the fees remain in the pool and swaps that generate large fees result in increased target values for the token received on the swap.

For example, if a user swaps a large portion of the pool's base for quote, the LP fees generated from the swap will remain in the pool and will be added to the quote's target, thus moving it away from the base target. When the pool is rebalanced, the targets will still have some amount of delta.

This can cause minor fund loss in the scale of wei due to precision loss. In our testing, we found that swapping massive amounts of token in the scale of hundreds of millions can result in less than a gwei of unwarranted profit. This is not a valid theft vector in practice.

— RECOMMENDATION

Note this possibility and monitor for stable pools. Major skews of token targets are unlikely to occur.

PROOF OF CONCEPT <https://gist.githubusercontent.com/ZdravkoHr/e57ac4bba8af518fc5d51cc6002bee88/raw/1e5f6e0fdb7befce75b3c3214f9ca12bee56da4a/RStateTest.t.sol>

— RESOLUTION

Status: Acknowledged.

§ 05

Disclaimer.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian's position is that each company and individual are responsible for their own due diligence and continuous security. Guardian's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis.

Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract's safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

§ 06

About Guardian.

Guardian is a smart contract security firm. Every engagement is worked by two independent teams competing on the same scope, each with its own tooling and methods, so no single reviewer's blind spot survives.

\$45B+DIGITAL ASSETS UNDER
REVIEW**200+**

ENGAGEMENTS COMPLETED

500+CRITICAL & HIGH FINDINGS
REPORTED

Reach us at audits@guardianaudits.com.