SHERLOCK

Security Review For Alphix

Collaborative Audit Prepared For: **Alphix**
Lead Security Expert(s): **pkqs90**
vinica_boy
Date Audited: **December 1 - December 8, 2025**

Introduction

Alphix is an upgradeable Uniswap V4 Hook designed to combine multiple features into a Unified Pool. Think dynamic fees, liquidity rehypothecation, JIT liquidity protection, and more, all coexisting in the same AMM. The first product, and the subject of this audit, is a dynamic fee algorithm that monitors pool metrics such as the Volume/TVL ratio and adjusts fees toward an optimal state, itself evolving via an Exponential Moving Average. This architecture offers competitive price execution for traders while allowing LPs to benefit from reduced complexity and better returns. Read more.

Scope

Repository: alphixfi/alphix-core

Audited Commit: 6acc8c14887beaf42e759fc8d01aed4c34d1f0ca

Final Commit: c059d7f3f1af876928bb78a8d45a892a07ab7d64

Files:

- src/AlphixLogic.sol
- src/Alphix.sol
- src/BaseDynamicFee.sol
- src/interfaces/IAlphixLogic.sol
- src/interfaces/IAlphix.sol
- src/interfaces/IRegistry.sol
- src/libraries/AlphixGlobalConstants.sol
- src/libraries/DynamicFee.sol
- src/Registry.sol

Final Commit Hash

c059d7f3f1af876928bb78a8d45a892a07ab7d64

Findings

Each issue has an assigned severity:

- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.

- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
0	0	6

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue L-1: Pool information is not migrated when updating Registry [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-alphix-dec-1st/issues/10

Vulnerability Detail

The `Alphix` contract allows the owner to update the Registry address via `setRegistry()`. When called, it registers the `Alphix` and `AlphixLogic` contract addresses to the new Registry:

https://github.com/sherlock-audit/2025-11-alphix-dec-1st/blob/main/alphix-core/src/Alphix.sol#L270-L284

```
function setRegistry(address newRegistry) external override onlyOwner nonReentrant {
    ...
    registry = newRegistry;
    IRegistry(newRegistry).registerContract(IRegistry.ContractKey.Alphix,
    ↪  address(this));
    IRegistry(newRegistry).registerContract(IRegistry.ContractKey.AlphixLogic,
    ↪  logic);
}
```

However, pool information previously registered via `initializePool()` -> `registry.registerPool()` is not migrated to the new Registry. The new Registry will have no record of existing pools.

Impact

After a Registry update, `Registry.getPoolInfo()` and `Registry.listPools()` will return empty/stale data for existing pools.

Recommendation

Add a migration function to batch-register existing pools to the new Registry, or admin must manually re-register pools after a Registry update.

Issue L-2: Non-upgradeable Alphix hook hardcodes AlphixLogic-specific implementation details, limiting future upgradeability [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-alphix-dec-1st/issues/11

Vulnerability Detail

The protocol uses a two-contract architecture: `Alphix` (the hook, non-upgradeable) and `AlphixLogic` (upgradeable via UUPS). The intent is that `AlphixLogic` can be upgraded or replaced to modify the fee algorithm.

However, `Alphix` exposes multiple AlphixLogic-specific implementation details:

1. Hardcoded data structures: The `poke()` function uses `DynamicFeeLib.OobState` and `DynamicFeeLib.PoolTypeParams` structs that are algorithm-specific:

https://github.com/sherlock-audit/2025-11-alphix-dec-1st/blob/main/alphix-core/src/Alphix.sol#L248-L255

```
(uint24 newFee, uint256 oldTargetRatio, uint256 newTargetRatio,
↪  DynamicFeeLib.OobState memory sOut) =
    _getFee(key, currentRatio);
newTargetRatio = IAlphixLogic(logic).finalizeAfterFeeUpdate(key, newTargetRatio,
↪  sOut);
```

2. Logic-specific admin functions: `setPoolTypeParams()` and `setGlobalMaxAdjRate()` in `Alphix` are pass-through functions that only forward calls to `AlphixLogic`. These are implementation details that should be called directly on `AlphixLogic`.
3. Bidirectional state passing: `poke()` passes `OobState` to `AlphixLogic` and receives it back, tightly coupling `Alphix` to the current algorithm's internal state management.

https://github.com/sherlock-audit/2025-11-alphix-dec-1st/blob/main/alphix-core/src/Alphix.sol#L236

```
function poke(PoolKey calldata key, uint256 currentRatio)
    external
    override
    onlyValidPools(key.hooks)
    restricted
    nonReentrant
    whenNotPaused
{
    PoolId poolId = key.toId();
    (,,, uint24 oldFee) = poolManager.getSlot0(poolId);

    // Compute new fee and target ratio
```

```
    (uint24 newFee, uint256 oldTargetRatio, uint256 newTargetRatio,
    ↪  DynamicFeeLib.OobState memory sOut) =
        _getFee(key, currentRatio);

    // Update the fee
    _setDynamicFee(key, newFee);

    // Update the storage
    newTargetRatio = IAlphixLogic(logic).finalizeAfterFeeUpdate(key,
    ↪  newTargetRatio, sOut);

    emit FeeUpdated(poolId, oldFee, newFee, oldTargetRatio, currentRatio,
    ↪  newTargetRatio);
}
```

Since `Alphix` is non-upgradeable, any future `AlphixLogic` must maintain these exact interfaces, even if the new algorithm no longer needs them.

Impact

Future `AlphixLogic` upgrades are constrained to use the same struct signatures and function interfaces. New algorithms may need to carry legacy unused parameters or use workarounds to encode new data into existing fields, reducing the upgradeability benefit.

Recommendation

Refactor to minimize coupling:

- `Alphix.poke()` should only pass `currentRatio` and receive `newFee`; all internal state should be managed within `AlphixLogic`
- Move `setPoolTypeParams()` and `setGlobalMaxAdjRate()` to be called directly on `AlphixLogic` by admin
- `Alphix` should treat `AlphixLogic` as a black box and expose only `ratio -> fee` conversion

Issue L-3: Existing pool fee and targetRatio are not clamped when PoolTypeParams is updated [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-alphix-dec-1st/issues/12

Vulnerability Detail

Admin can update pool type parameters via `setPoolTypeParams()`, including `minFee`, `maxFee`, and `maxCurrentRatio`. However, when these bounds are changed, existing pools' `fee` (stored in `slot0.lpFee`) and `targetRatio` are not automatically clamped to the new bounds.

The clamping only occurs during the next `poke()` call:

https://github.com/sherlock-audit/2025-11-alphix-dec-1st/blob/main/alphix-core/src/AlphixLogic.sol#L325-L338

```
// In computeFeeAndTargetRatio()
if (oldTargetRatio > pp.maxCurrentRatio) {
    oldTargetRatio = pp.maxCurrentRatio;  // only clamped here
}
...
newFee = clampFee(fee, p.minFee, p.maxFee);  // only clamped here
```

Between the parameter update and the next `poke()`, swaps will use a fee that is outside the intended bounds.

Impact

If admin lowers `maxFee` from 1% to 0.3%, but a pool currently has `fee = 0.5%`, traders will continue paying 0.5% until the next `poke()`. This creates a window where the pool operates outside intended parameters.

Recommendation

Either trigger an automatic `poke()` for all affected pools when `setPoolTypeParams()` is called, or document this as expected behavior requiring admin to manually poke pools after parameter updates.

Issue L-4: BaseDynamicFee.poke() lacks access control, allowing anyone to manipulate fee with arbitrary ratio [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-alphix-dec-1st/issues/13

Vulnerability Detail

`BaseDynamicFee` is an abstract contract designed to be inherited by dynamic fee hooks. Its `poke()` function updates the pool's LP fee based on a `currentRatio` parameter:

https://github.com/sherlock-audit/2025-11-alphix-dec-1st/blob/main/alphix-core/src/BaseDynamicFee.sol#L66-L70

```
function poke(PoolKey calldata key, uint256 currentRatio) external virtual
↪   onlyValidPools(key.hooks) {
    (uint24 newFee,,,) = _getFee(key, currentRatio);
    poolManager.updateDynamicLPFee(key, newFee);
}
```

This function has no access control beyond `onlyValidPools`. Anyone can call it with an arbitrary `currentRatio`, directly manipulating the pool's fee.

The current `Alphix` hook overrides this function and adds the `restricted` modifier, so it is not vulnerable. However, `BaseDynamicFee` is intended as a reusable base contract. Future hooks inheriting from it may forget to override `poke()` with proper access control.

Impact

For the current `Alphix` implementation: no impact (function is overridden with access control).

For future hooks inheriting `BaseDynamicFee` without overriding `poke()`: anyone can manipulate pool fees by calling `poke()` with malicious `currentRatio` values.

Recommendation

Add explicit documentation warning that `poke()` must be overridden with access control, or make `poke()` a stub that reverts by default, forcing inheritors to implement their own version.

Issue L-5: Registry will point to the old logic address when changed [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-alphix-dec-1st/issues/14

Summary

The `Registry` contract keep information about the current pools' info and related contracts' addresses. Values are correctly set initially when pools are added and `logic` address is set, but if `logic` address is updated, the `Registry` will still point to the old address.

Vulnerability Detail

The `Alphix::setLogic()` function does not call `Registry::registerContract()` to properly update the state inside the registry.

Impact

Incorrect address returned in view-only functions in `Registry` contract.

Code Snippet

https://github.com/sherlock-audit/2025-11-alphix-dec-1st/blob/8e493fab7beefcfcbc17047ef9ef68f848c74251/alphix-core/src/Alphix.sol#L263C5-L265C6

Tool Used

Manual Review

Recommendation

Consider updating the registry stored value as well.

Issue L-6: Side factors lower bound can be extended to provide more flexibility [RESOLVED]

Source: https://github.com/sherlock-audit/2025-11-alphix-dec-1st/issues/16

Summary

Side factors are used to additionally adjust the fee movements apart from `linearSlope` parameter based on the direction of movement (increase/decrease).

Vulnerability Detail

The current implementation enforces a lower bound of `1e18` on both side factors:

```
if (
    params.upperSideFactor < AlphixGlobalConstants.ONE_WAD
        || params.upperSideFactor > AlphixGlobalConstants.TEN_WAD
) revert InvalidParameter();
if (
    params.lowerSideFactor < AlphixGlobalConstants.ONE_WAD
        || params.lowerSideFactor > AlphixGlobalConstants.TEN_WAD
) revert InvalidParameter();
```

This constraint creates a design coupling: `linearSlope` and side factors are interdependent. If an operator wants to decrease fee adjustment in one direction while keeping the base slope constant, they cannot do so if a side factor is already at its minimum (1e18). The only workaround is to adjust `linearSlope`, which affects fee adjustments in both directions equally and would require additional adjustments for the opposite factor if it has to be kept the same.

Impact

Limited flexibility regarding the adjustments of side factors.

Code Snippet

https://github.com/sherlock-audit/2025-11-alphix-dec-1st/blob/8e493fab7beefcfcbc17047ef9ef68f848c74251/alphix-core/src/AlphixLogic.sol#L546C7-L554C37

Tool Used

Manual Review

Recommendation

Consider allowing values lower than 1e18 for the side factors.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.