Coinbase Verified Pools Security Review

Auditors

Christoph Michel, Lead Security Researcher

Noah Marconi, Lead Security Researcher

Phaze, Security Researcher

0xDjango, Associate Security Researcher

Report prepared by: Lucas Goiriz

October 31, 2024

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Coinbase is a secure online platform for buying, selling, transferring, and storing cryptocurrency.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of Coinbase Verified Pools according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 10 days in total, Coinbase engaged with Spearbit to review the verified-pools-contracts-audit protocol. In this period of time a total of **28** issues were found.

Summary

Project Name	Coinbase
Repository	verified-pools-contracts-audit
Commit	dc821d56
Type of Project	DeFi
Audit Timeline	Aug 7th to Aug 17th
Two week fix period	Sep 13

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	4	0	4
Low Risk	7	4	3
Gas Optimizations	4	3	1
Informational	13	5	8
Total	**28**	**12**	**16**

5 Findings

5.1 Medium Risk

5.1.1 Users can increase liquidity for token owned by sanctioned or non-verified user

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: `VerifiedPoolsPositionManager.sol` ensures that users can only mint, decrease liquidity, and burn positions for themselves via the `_ensureOwner()` check. The policy contracts check that the caller is verified and not on a list of sanctioned addresses. Anyone is able to increase the liquidity of any position, even if the position owner has been subsequently added to the sanctions list or has lost their KYC/country verifications.

This may result in regulatory issues. In the case where liquidity is added to a position owned by an address placed on the sanctions list, the liquidity will be frozen.

Proof of concept:

```
contract IncreaseLiquidityTest is VerifiedPoolsBasicHookTest {
    using PoolIdLibrary for PoolKey;
    using CurrencyLibrary for Currency;
    using Planner for Plan;

    PoolKey private _poolKey;
    PoolId private _poolId;
    address private _lp;
    uint256 private _tokenId;
    PositionConfig private _config;
    bytes32 private _vaUid;
    bytes32 private _vcUid;

    event ModifyLiquidity(
        PoolId indexed id,
        address indexed sender,
        int24 tickLower,
        int24 tickUpper,
        int256 liquidityDelta
    );

    error Wrap__FailedHookCall(address hook, bytes revertReason);
    error InvalidVerifiedAccountAttestation();
    error IncorrectPositionConfigForTokenId(uint256 tokenId);
    error NotApproved(address caller);

    function setUp() public virtual override {
        super.setUp();

        (_poolKey, _poolId) = _initializePool();
        _lp = makeAddr("Liquidity Provider");
        _fundAddress(_lp, 1_000_000 ether);
        _approveLP(_lp, _erc20(currency0));
        _approveLP(_lp, _erc20(currency1));

        (_vaUid, _vcUid) = _issueAttestations(_lp);
        _config = _defaultPositionConfig(_poolKey);
        vm.startPrank(_lp);
        (_tokenId, ) = _addLiquidityWithParams(
            _config,
            100 ether,
            100 ether,
            ZERO_BYTES,
```

```
            _lp
        );
        vm.stopPrank();
    }

    modifier whenCallerVerified() {
        _;
    }

    function test_WhenNotTokenOwnerAndOwnerIsSanctioned()
        external
        whenCallerVerified
    {
        address anotherLp = makeAddr("Another Liquidity Provider");
        _issueAttestations(anotherLp);
        _fundAddress(anotherLp, 1_000_000 ether);
        _approveLP(anotherLp, _erc20(currency0));
        _approveLP(anotherLp, _erc20(currency1));

        // Add sanction to owner of LP position (_lp)
        MockSanctionsList(address(_sanctionsList)).addSanction(_lp);

        // It should increase liquidity
        vm.startPrank(anotherLp);
        _assertIncreaseLiquidityWithHookData(ZERO_BYTES);
        vm.stopPrank();
    }

    function _assertIncreaseLiquidityWithHookData(
        bytes memory hookData
    ) private {
        vm.expectEmit();
        emit ModifyLiquidity(
            _poolId,
            address(_posm),
            _config.tickLower,
            _config.tickUpper,
            100 ether
        );
        increaseLiquidity(_tokenId, _config, 100 ether, hookData);
    }
}
```

Recommendation: Either option would effectively mitigate this issue:

- Restrict INCREASE_LIQUIDITY action for only the owner of position (similar to mint, decrease liquidity, and burn).
- Check verification and inclusion on sanctions list for owner of position as well as the caller.

Coinbase: Acknowledging the issue. We are currently OK accepting this as-is and will consider the first recommendation if we ultimately address it.

Spearbit: Acknowledged.

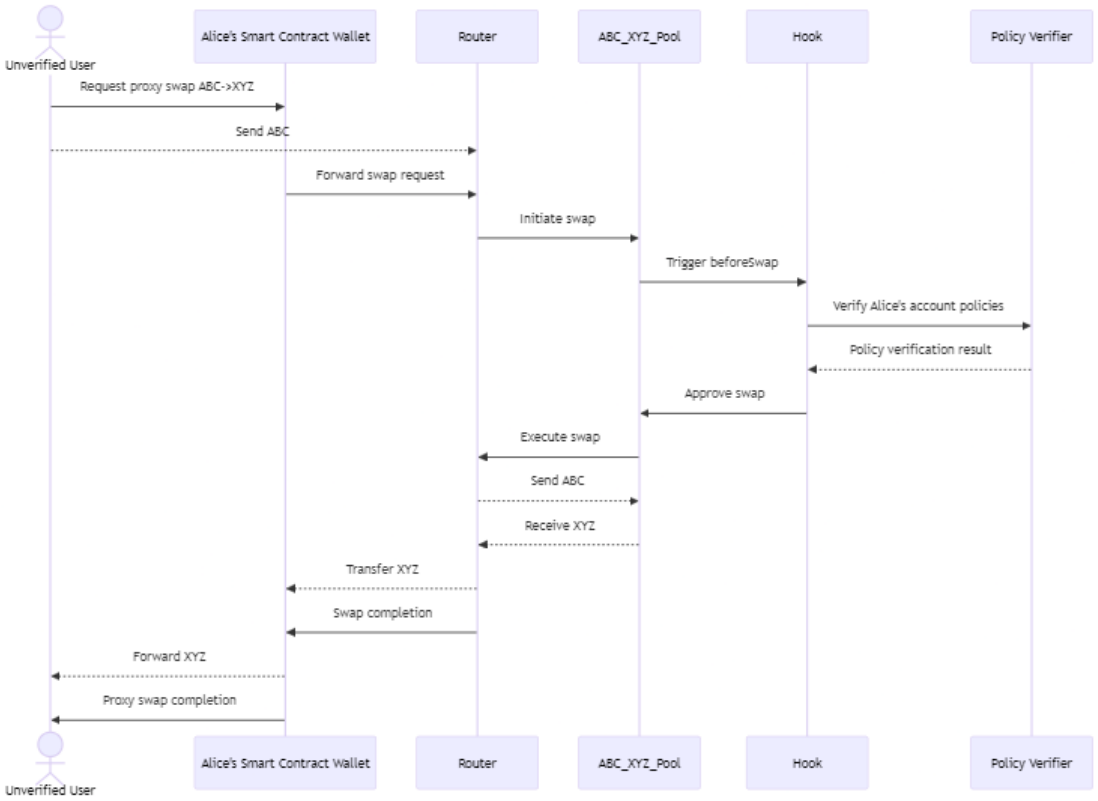
5.1.2 Verified smart wallet accounts can be used as proxies to facilitate swaps

Severity: Medium Risk

Context: VerifiedPoolsBasicHook.sol#L260-L344

Description: Smart contract wallets such as ERC-4337 contracts can receive attestations which allow interacting with Coinbase's verified pools. These wallets can be used as proxies executing swaps and creating wrapper contracts for managed positions.

Even if these wallets do not contain logic allowing arbitrary actors to bypass Coinbase's policies at the time of the attestation issuance, the required logic can be included at a later time as these are often upgradeable and include `delegatecall` functionality.



Impact: High: This scenario allows swaps to be performed by sanctioned entities bypassing the sanctions list and KYC verification. The verified contract does not require any upfront capital and is able to profit from this exchange.

Likelihood: Low/Medium: This requires a KYC verified user to violate Coinbase's ToS and to potentially interact with sanctioned funds. This exchange can be performed permissionlessly allowing any entity to perform swaps immediately. Wrapper contracts for liquidity positions only work until the attestation is revoked.

Recommendation: Consider only allowing a certain set of smart contract wallets to receive attestations. Otherwise ensure that such a service is in clear violation of the ToS.

Coinbase: Acknowledging the potential issue and noting that this would be a clear violation of our Terms of Service by the user.

Spearbit: Acknowledged.

5.1.3 Verified accounts can execute swaps as limit orders by sanctioned users

Severity: Medium Risk

Context: VerifiedPoolsBasicHook.sol#L316-L344

Description: Verified accounts are able to execute swaps placed as limit orders by sanctioned users.

When a sanctioned user wants to swap with the Coinbase verified pools, they can create an order in an order book contract waiting to be executed by a verified user. The verified user only provides the credentials for interacting with the verified pools contracts. They do not require any capital to execute the swap on behalf of a sanctioned user. The tainted funds always stay with the sanctioned user and are directly exchanged with the pools without touching the verified user. The verified user can receive a portion of the swap output as a commission as an incentive.

This requires the possibility of taking control of transaction execution when the pool manager is in an unlocked state. This step can depend on the functionality of the router serving as an entry point, however many possibilities exist. The proof of concept uses batched actions in the router that include a `TAKE` action using native currency and a zero value to enable an arbitrary call from `PoolManager`.

Note that there are many possibilities of actors gaining control of execution when the pool manager is unlocked. Some examples are:

- Routers allowing for arbitrary calls.
- Transferring native currency to arbitrary actors through `.call`.
- Transferring custom tokens.
- Interacting with unverified pools that include custom tokens.
- Unintended callbacks in routers when interacting with other protocols.
- Unlocking the router first from a smart contract account.
- Setting up subscriptions for liquidity positions that notify arbitrary actors.

The actor might not even require gaining explicit control as a limit order could also be set up through an unverified pool with a custom hook.

Impact: High: This scenario directly uses funds from sanctioned addresses to perform swaps, bypassing the sanctions list and KYC verification. Verified accounts can profit from this exchange without requiring upfront capital.

Likelihood: Low: This requires an external protocol being built on Base for the purpose of bypassing KYC. Swaps cannot be executed directly by sanctioned entities. These must wait for a corresponding verified user willing to engage with them for a commission. The verified user is likely to violate Coinbase's ToS in this process.

Proof of concept: As a final version of the router used in production was not finished, a `TestRouter` inheriting from `V4Router` was used for the proof of concept.

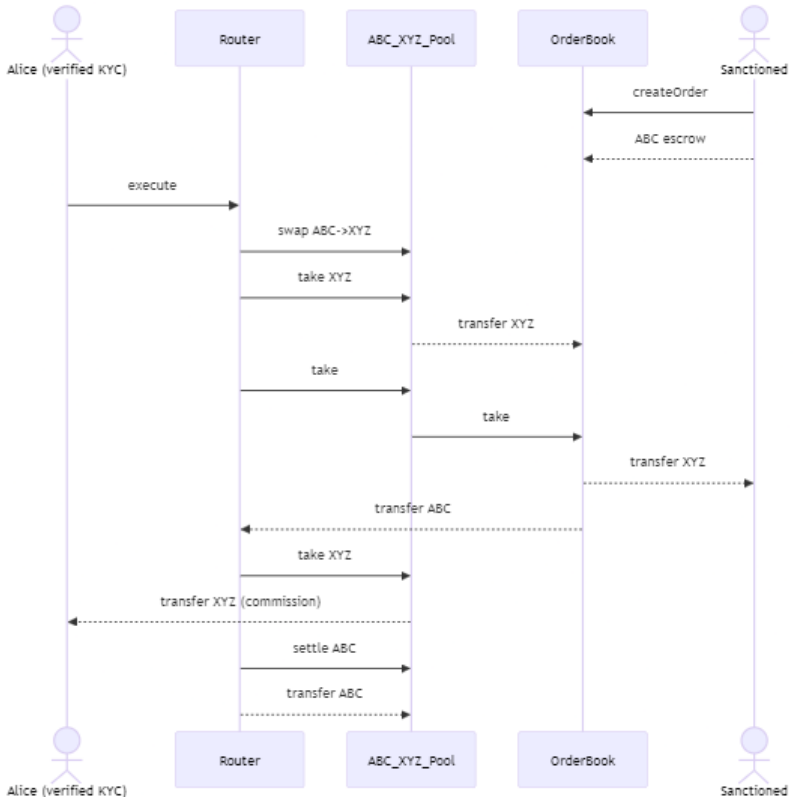
```
struct Order {
    Currency have;
    Currency want;
    uint256 haveAmount;
    uint256 wantAmount;
    address recipient;
}

contract OrderEscrow {
    Order order;
    IPoolManager immutable poolManager;
    address immutable router;

    constructor(IPoolManager poolManager_, address router_, Order memory order_) {
        poolManager = poolManager_;
        order = order_;
        router = router_;
    }
```

```
    function orderIsFilled() public view returns (bool) {
        return ERC20(Currency.unwrap(order.want)).balanceOf(address(this)) >= order.wantAmount;
    }

    function executeOrder() public {
        // Transfer maker
        ERC20(Currency.unwrap(order.want)).transfer(order.recipient, order.wantAmount);
        // Transfer taker
        ERC20(Currency.unwrap(order.have)).transfer(router, order.haveAmount);
    }

    receive() external payable {
        executeOrder();
    }
}

contract SanctionedOrderBook {
    IPoolManager immutable poolManager;
    address immutable router;

    constructor(IPoolManager poolManager_, address router_) {
        poolManager = poolManager_;
        router = router_;
    }

    function createOrder(Order calldata order) external returns (address orderEscrow) {
        bytes32 orderId = keccak256(abi.encode(order));
        orderEscrow = address(new OrderEscrow{salt: orderId}(poolManager, router, order));
        ERC20(Currency.unwrap(order.have)).transferFrom(msg.sender, orderEscrow, order.haveAmount);
    }
}

contract TestRouter is V4Router, ReentrancyLock {
    constructor(IPoolManager _poolManager) V4Router(_poolManager) {}

    modifier checkDeadline(uint256 deadline) {
        if (block.timestamp > deadline) revert("DeadlinePassed()");
        _;
    }

    function executeActions(bytes calldata unlockData, uint256 deadline)
        external
        payable
        isNotLocked
        checkDeadline(deadline)
    {
        _executeActions(unlockData);
    }

    function msgSender() public view override returns (address) {
        return _getLocker();
    }

    function _pay(Currency currency, address payer, uint256 amount) internal override {
        if (payer == address(this)) {
            currency.transfer(address(poolManager), amount);
        } else {
            ERC20(Currency.unwrap(currency)).transferFrom(payer, address(poolManager), uint160(amount));
        }
    }
}
```

```
contract OrderBookTest is EndToEndTest {
    using Planner for Plan;

    TestRouter router;

    function setUp() public override {
        super.setUp();

        router = new TestRouter(manager);
        vm.prank(_hookAdmin);
        _hook.setSwapRouter(address(router), true);
    }

    function test_limitOrderBySanctionedAccount() external {
        address alice = makeAddr("Alice");
        address sanctioned = makeAddr("Sanctioned");
        SanctionedOrderBook sanctionedOrderBook = new SanctionedOrderBook(manager, address(router));

        // Alice is has verified KYC
        _issueAttestations(alice);
        // Sanctioned does not have any valid attestations
        // and is sanctioned.
        MockSanctionsList(address(_sanctionsList)).addSanction(sanctioned);
        MockSanctionsList(address(_sanctionsList)).addSanction(address(sanctionedOrderBook));

        _token1.mint(sanctioned, 3 ether);

        vm.startPrank(sanctioned);
        // Sanctioned wants to swap `3.0 c1` for `2.0 c0`
        Order memory order = Order({
            have: Currency.wrap(address(_token1)),
            want: Currency.wrap(address(_token0)),
            haveAmount: 3 ether,
            wantAmount: 2 ether,
            recipient: sanctioned
        });

        // Sanctioned creates an order in `orderBook`
        ERC20(Currency.unwrap(order.have)).approve(address(sanctionedOrderBook), order.haveAmount);
        address orderEscrow = sanctionedOrderBook.createOrder(order);

        vm.stopPrank();

        vm.startPrank(alice);

        Plan memory planner = Planner.init();
        IV4Router.ExactInputSingleParams memory swapParams = IV4Router.ExactInputSingleParams({
            poolKey: _poolKey1,
            zeroForOne: false,
            amountIn: uint128(order.haveAmount),
            amountOutMinimum: uint128(order.wantAmount),
            sqrtPriceLimitX96: 0,
            hookData: ""
        });

        // Perform the swap to fill `order`
        planner.add(Actions.SWAP_EXACT_IN_SINGLE, abi.encode(swapParams));
        // Give what maker wants
        planner.add(Actions.TAKE, abi.encode(_token0, orderEscrow, order.wantAmount));
        // Take what maker has
        planner.add(Actions.TAKE, abi.encode(CurrencyLibrary.NATIVE, orderEscrow, 0));
```

```
        // Give alice the surplus as a commission
        planner.add(Actions.TAKE, abi.encode(_token0, alice, ActionConstants.OPEN_DELTA));
        // Settle maker's currency
        planner.add(Actions.SETTLE, abi.encode(_token1, ActionConstants.OPEN_DELTA, false));
        bytes memory actions = planner.encode();

        // Alice does not hold any tokens
        assertEq(_token0.balanceOf(alice), 0);
        assertEq(_token1.balanceOf(alice), 0);

        // Sanctioned does not hold any tokens
        assertEq(_token0.balanceOf(sanctioned), 0);
        assertEq(_token1.balanceOf(sanctioned), 0);

        // Escrow contract only holds `_token1`
        assertEq(_token0.balanceOf(orderEscrow), 0);
        assertEq(_token1.balanceOf(orderEscrow), order.haveAmount);

        router.executeActions(actions, block.timestamp);

        // Escrow does not hold any tokens
        assertEq(_token0.balanceOf(orderEscrow), 0);
        assertEq(_token1.balanceOf(orderEscrow), 0);

        // Sanctioned has filled their order and is holding `_token0`
        assertEq(_token0.balanceOf(sanctioned), order.wantAmount);
        assertEq(_token1.balanceOf(sanctioned), 0);

        // Alice has gained a commission in `_token0`
        assertGt(_token0.balanceOf(alice), 0);
        assertEq(_token1.balanceOf(alice), 0);
    }
}
```

Note: Alice, the verified user, requires no upfront capital. All funds used for the swap are provided by sanctioned entities. The diagram below highlights the main calls and the flow of funds.



Recommendation: Define clear rules for what actions violate Coinbase's ToS and warn users accordingly.

Coinbase: Acknowledging the potential issue and noting that this would be a clear violation of our Terms of Service by the user.

Spearbit: Acknowledged.

5.1.4 `PoolManager` allows funds from verified pools to be used by sanctioned accounts during flashloans

Severity: Medium Risk

Context: *(No context files were provided by the reviewer)*

Description: Tainted funds can be moved through `PoolManager` and sanctioned entities can borrow capital which stems from Coinbase's verified pools via flash loans. These could potentially be used to perform exploits which sometimes have large capital requirements.

Proof of concept:

```
contract TaintedFundsMover is SafeCallback {
    constructor(IPoolManager _poolManager) SafeCallback(_poolManager) {}

    function unlock(bytes calldata data) public {
        bytes memory unlockData = abi.encode(msg.sender, data);
        poolManager.unlock(unlockData);
    }

    function _unlockCallback(bytes calldata unlockData) internal override returns (bytes memory) {
        (address from, bytes memory data) = abi.decode(unlockData, (address, bytes));
        (Currency currency, address to, uint256 amount) = abi.decode(data, (Currency, address,
↪   uint256));
        // Take funds from `poolManager` and give to `to`
        poolManager.take(currency, to, amount);
        // Sync currency
        poolManager.sync(currency);
```

```
        // Move tainted funds from `from` to `poolManager`
        ERC20(Currency.unwrap(currency)).transferFrom(from, address(poolManager), amount);
        // Settle
        poolManager.settle();
        return "";
    }
}

contract TaintedFundsMoverTest is EndToEndTest {
    function test_moveTaintedFundsThroughManager() external {
        TaintedFundsMover mover = new TaintedFundsMover(manager);
        vm.label(address(mover), "Mover");

        address sanctioned = makeAddr("Sanctioned");
        address to = makeAddr("to");
        uint256 amount = 1 ether;

        _token0.mint(sanctioned, amount);

        // "Sanctioned" approves "Mover"
        vm.startPrank(sanctioned);
        _token0.approve(address(mover), amount);

        // "Mover" moves funds from "sanctioned" through `manager`
        bytes memory data = abi.encode(_token0, to, amount);
        mover.unlock(data);
    }
}
```

Recommendation: Consider implementing a wrapped version of tokens for Coinbase's verified pools which tightly controls transfers. The wrapping could be handled automatically by the pools, or could be performed by a custom router.

- Alice's proposal (modifications):

  Assumptions:

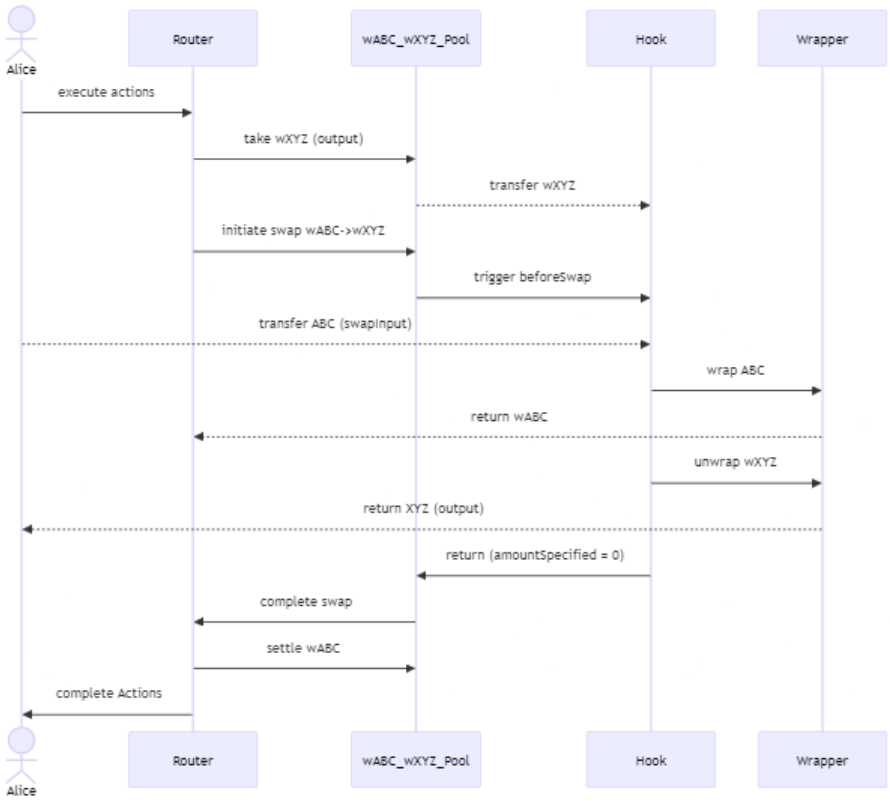
  1. Standard router.
  2. Requires router knowing `wXYZ swapOutput` amount before swap completion.
  3. `wABC`, `wXYZ` token restrictions:
     1. Only allow `wrap`/`unwrap` by `Hook`.
     2. Only allow `transfer` between `PoolManager` ↔ `Hook`.

  Standard router encoded actions:

  1. `TAKE wXYZ swapOutput` → `Hook`.
  2. `SWAP wABC` → `wXYZ`.
  3. `SETTLE wABC swapInput`.

  `beforeSwap` Hook actions:

  1. Transfer `ABC swapInput` from `Alice`.
  2. Wrap `ABC -> wABC` and forward to `Router` to cover `wABC` debt.
  3. Unwrap `wXYZ -> XYZ swapOutput` and forward to `Alice`.



- Custom router suggestion:

  Assumptions:

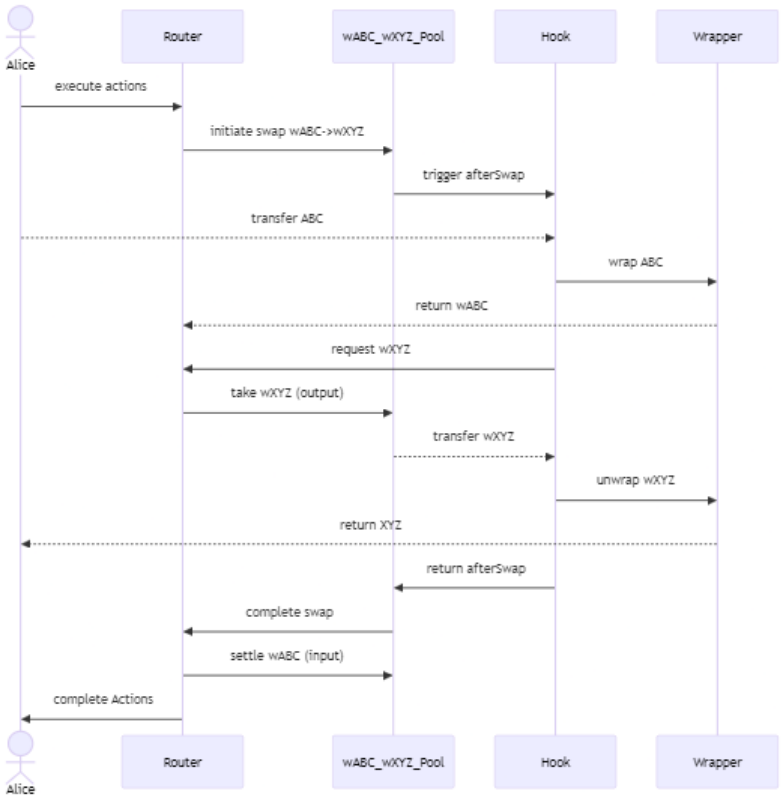
  1. Custom/modified router.
  2. Router can be reentered by `Hook` to allow for requesting tokens.
  3. `wABC`, `wXYZ` token restrictions:
     1. Only allow `wrap`/`unwrap` by `Hook`.
     2. Only allow `transfer` between `PoolManager` ↔ `Hook`.

  Custom router encoded actions:

  1. `SWAP` `wABC` → `wXYZ`.
  2. `SETTLE` `wABC` `swapInput`.

  `afterSwap` Hook actions:

  1. Transfer `ABC` `swapInput` from `Alice`.
  2. Wrap `ABC -> wABC` and forward to `Router` to cover `wABC` debt.
  3. Request router for `TAKE` action of `wXYZ` `swapOutput`.
  4. Unwrap `wXYZ -> XYZ` and forward back to `Alice`.



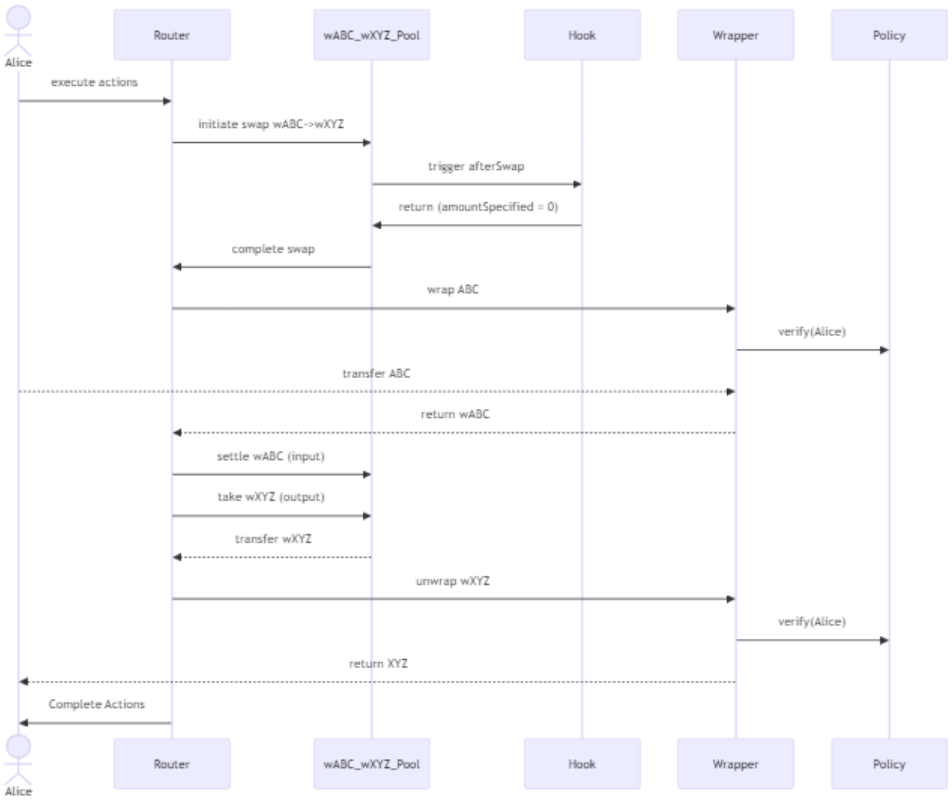
Pros:

- Wrap/Unwrap layer hidden in hook.
- Tight transfer restrictions (no flashloans possible).

Cons:

- More complex.
- Requires custom router (ideally).
- Possible composability issues:
  - Swapping with `wABC` → `wXYZ` pool requires `ABC` tokens.
  - Or (Alice's original suggestion): `ABC` → `XYZ` pool shows 0 liquidity.
- Brock's proposal

Assumptions: 1. Custom/modified router 2. `wABC`, `wXYZ` token restrictions: 1. `wrap`/`unwrap`/`transfer` requires policy checks. 2. `Hook` can be exempt from checks.



Pros:

- Simple solution.

Cons:

- Requires custom router.
- Requires multiple policy checks (optimization possible).
- Possible composability issues:
  - Wrapping needs to be done explicitly.
  - Liquidity fragmentation `ABC` $\leftrightarrow$ `wABC`.
  - Transfer restrictions on wrapped assets could hurt composability.

Coinbase: Acknowledging this issue. This requires a bigger conversation on our side and we will not be submitting a fix to address this as part of the audit report.

Spearbit: Acknowledged.

5.2 Low Risk

5.2.1 Individual pools cannot have zero-fee hooks if `DEFAULT_POOL` hook fee is set to non-zero

Severity: Low Risk

Context: VerifiedPoolsBasicHook.sol#L357-L360

Description: In the case where `hookFees[DEFAULT_POOL]` is set greater than zero, it is impossible for specific pools to allow for a `hookFee = 0`. Take the following scenario:

- Coinbase sets DEFAULT_POOL hook fee to 100 bips (1%) → `hookFees[DEFAULT_POOL] = 100`.
- Coinbase wants to incentivize use of their ETH-USDC pool with a 0% hook fee → `hookFees[poolId] = 0`.

The hookFee that will be used will be that of DEFAULT_POOL.

Recommendation: Create another mapping to declare intent to use specific pool hookFee, as opposed to keying off of the distinction between zero/non-zero.

Coinbase: Fixed in commit c272eb12.

Spearbit: Fixed.

5.2.2 `sweep` performs no policy checks

Severity: Low Risk

Context: VerifiedPoolsPositionManager.sol#L71

Description: `sweep` allows funds from unverified accounts to move funds in and out of the `PositionManager`.

These funds do not enter pools and only serve to leave a flow of funds through the `PositionManager` with an undesired appearance (`sender` and `recipient` being unverified).

Recommendation: Consider adding policy checks to `sweep` to avoid the appearance of interacting with unverified accounts.

Incoming funds from blind transfers cannot be restricted but the `sweep to` address can be.

Coinbase: Fixed in commits 38eea28c and cd5072ec.

Spearbit: Fixed.

5.2.3 `VerifiedPoolsPositionManager` does not override `tokenURI`

Severity: Low Risk

Context: VerifiedPoolsPositionManager.sol#L28-L32, VerifiedPoolsPositionManager.sol#L116-L130, ERC721Permit_v4.sol#L104-L107

Description: The `VerifiedPoolsPositionManager` does not override the `tokenURI` function from UniswapV4's `PositionManager` and `ERC721Permit_v4` contract. It currently returns "`https://example.com%22` for every token ID.

Any third party integration (e.g. opensea) will display a user's NFTs with data returned by `tokenURI`. If this is a dummy address, then there won't be any valid metadata for the NFT. The metadata info is optional per EIP-721 spec, however, as the "name" and "symbol" variables are overridden and the `tokenURI` function is defined, it is recommended to use a valid metadata URI.

Impact: Low/Medium: There are no loss of funds, however, nearly every NFT uses metadata to be displayed to the user. There is also potentially a reputational damage associated with a non-functioning feature which cannot be fixed.

Likelihood: Medium: Most NFTs use the metadata extension and aggregators fetch the token URI data automatically.

Proof of concept:

```
    function test_tokenURI() external {
        console.log(_posm.tokenURI(1)); // https://example.com
    }
```

Recommendation: Override the `tokenURI` function to point to a valid URI displaying Coinbase's managed positions. The functionality can be delegated to a "renderer" contract.

Coinbase: Acknowledged. We'll address this issue prior to launch and use a similar approach to Uniswap.

Spearbit: Acknowledged.

5.2.4 Sanctioned accounts can pay for pool interactions of verified users

Severity: Low Risk

Context: *(No context files were provided by the reviewer)*

Description: Both `UniversalRouter` (used for swaps) and `VerifiedPoolsPositionManager` (used for modifying liquidity) can use their contract balance to settle pool interactions:

```
// UniversalRouter
function payOrPermit2Transfer(address token, address payer, address recipient, uint256 amount) internal
↪  {
    if (payer == address(this)) pay(token, recipient, amount);
    else permit2TransferFrom(token, payer, recipient, amount.toUint160());
}

// periphery/PositionManager, inherited by VerifiedPoolsPositionManager
function _pay(Currency currency, address payer, uint256 amount) internal override {
    if (payer == address(this)) {
        // TODO: currency is guaranteed to not be eth so the native check in transfer is not optimal.
        currency.transfer(address(poolManager), amount);
    } else {
        permit2.transferFrom(payer, address(poolManager), uint160(amount), Currency.unwrap(currency));
    }
}
```

A sanctioned user can send funds to these router contracts and a verified account can use these funds to swap in or add liquidity to a verified pool.

Recommendation: The payer information is currently not available in the hooks. Completely preventing this finding would require using a custom swap router instead of the `UniversalRouter` and modifying these routers to either expose the payer or revert if the payer for `settle` actions is not the `msg.sender` (this would also limit router action composability).

Coinbase: Acknowledged.

Spearbit: Acknowledged.

5.2.5 Liquidity can be locked up if position manager is removed

Severity: Low Risk

Context: VerifiedPoolsBasicHook.sol#L717

Description: Liquidity to verified pools can only be managed through Coinbase's `VerifiedPoolsPositionManager`. The hooks check if the liquidity modification originated by an allow-listed position manager:

```
if (sender == ZERO_ADDRESS || !positionManagers[sender]) {
    revert PositionManagerNotAllowed(sender);
}
```

A position manager can be removed from the `positionManagers` mapping through a privileged `setPositionManager(address positionManager, bool allowed)` call. Users will be unable to remove their liquidity if a `PositionManager` is removed that they provided liquidity through. (The Coinbase `VerifiedPoolsPositionManager` also disallows transfers of the ERC721 liquidity position).

Furthermore, there is no proper migration path from one `PositionManger` to a new one in the current code as one cannot selectively disallow only adding liquidity operations while still allowing removals.

Recommendation: There are several approaches to address this finding. One is to define separate allow-lists for adding and removing liquidity, and disabling removals (only allowing additions) of `PositionManager`s from the remove-liquidity list. `setPositionManager(*, true)` would add to both mappings but `setPositionManager(*, false)` would only remove from the add-liquidity list.

Coinbase: Fixed in commit b4c61f7a.

Spearbit: Fixed.

5.2.6 `feeAmount` computation might fail because of intermediate overflow

Severity: Low Risk

Context: VerifiedPoolsBasicHook.sol#L376

Description: The `afterSwap` hook computes the `feeAmount` as:

```
uint256 feeAmount = uint128(swapAmount) * hookFee / TOTAL_FEE_BIPS;
```

Currently, the math is done in `uint128` types as all 3 variables involved are of type `uint128`. The cast to the `uint256` result type is only done after the computation is performed. The intermediate `swapAmount * hookFee` computation will overflow if it exceeds `type(uint128).max`, even if the final result would fit in a `uint128`.

Recommendation: Consider performing the entire computation in extended `uint256` types by casting `swapAmount` to `uint256`:

```
- uint256 feeAmount = uint128(swapAmount) * hookFee / TOTAL_FEE_BIPS;
+ uint256 feeAmount = uint256(uint128(swapAmount)) * hookFee / TOTAL_FEE_BIPS;
```

Coinbase: Fixed in commit 94b7e432.

Spearbit: Fixed.

5.2.7 Verified Account attestation checks ignore `data` field

Severity: Low Risk

Context: VerifiedAccountPolicy.sol#L80

Description: Checking if an account is a verified (business) account is performed by calling `AttestationVerifier.verifyAttestation(...)`. However, this function does not check the attribute's `data` field. Coinbase's `StaticAttester` encodes a `true` boolean in the data field in its `attestAccount` function.

```
AttestationRequest memory request = AttestationRequest({
    schema: verifiedAccountSchema,
    data: AttestationRequestData({
        recipient: recipient,
        expirationTime: 0,
        revocable: true,
        refUID: 0x0,
        // The Verified Account attestation schema is 'bool verifiedAccount'.
        // Bool fields are represented as a single uint256 and in our case is always set to true.
        data: abi.encode(true),
        value: 0
    })
});
```

Recommendation: While it's unluckily that Coinbase would call `StaticAttester.attest()` to create a verified account attestation with a `false` value (or any other data), consider being strict about the expected values of attributes and check that the `vaAttestation/vbaAttestation.data` ABI-encodes the `true` boolean value.

Coinbase: Acknowledged.

Spearbit: Acknowledged.

5.3 Gas Optimization

5.3.1 Policy verification results can be cached in transient storage

Severity: Gas Optimization

Context: *(No context files were provided by the reviewer)*

Description: Verifying if the sender has passed all the policy checks can be cached in `transient` storage allowing checks performed in the same transaction to be skipped. Transient storage automatically is reset once the transaction has executed. The attestation data and policy check results are likely not to change during the execution of a user initiated transaction and can therefore be cached for the duration of the transaction.

This can be useful when multiple swaps with verified pools are executed, or if wrapped tokens requiring policy checks on transfers are used in the future.

Recommendation: An example implementation of how the verified policy check result can be cached is given below. Beware that the current implementation does not account for any changes in the `data` parameter that might contribute to the policy verification result, as it is currently unused.

```
+ // The slot holding the cached account policy check result.
+ // bytes32(uint256(keccak256("BasicPolicy.accountPolicyCheckCache(address)")) - 1)
+ bytes32 internal constant ACCOUNT_POLICY_CHECK_CACHE_SLOT =
+     0xe9c5982aff7426f3bb108fa312da7becdfaa5fd56fd4f45ee2df008c9fae69bb;

+ function getAccountPolicyCheckCacheSlot(address sender) internal view returns (bytes32 slot) {
+     assembly ("memory-safe") {
+         mstore(0x14, sender)
+         mstore(0x00, ACCOUNT_POLICY_CHECK_CACHE_SLOT)
+         slot := keccak256(0x00, 0x34)
+     }
+ }
```

```
+ function readAccountPolicyCheckCache(address sender) internal view returns (bool verified) {
+     bytes32 slot = getAccountPolicyCheckCacheSlot(sender);
+     assembly ("memory-safe") {
+         verified := tload(slot)
+     }
+ }

+ function setAccountPolicyCheckCache(address sender, bool verified) internal {
+     bytes32 slot = getAccountPolicyCheckCacheSlot(sender);
+     assembly ("memory-safe") {
+         tstore(slot, verified)
+     }
+ }

  /// @dev Verify if the sender address has passed all the policy checks, including:
  ////      1) VerifiedAccountPolicy
  ///       2) VerifiedCountryPolicy
  ///       3) NonSanctionedPolicy
  ///
  /// @param sender The sender address to be verified
  /// @param data Arbitrary data to be passed into the policy
  ///
  /// @return A boolean indicating if the sender is verified
  function _verify(address sender, bytes calldata data)
      internal
-     view
      virtual
      override(VerifiedAccountPolicy, VerifiedCountryPolicy, NonSanctionedPolicy)
      returns (bool)
  {
-     return VerifiedAccountPolicy._verify(sender, data) && VerifiedCountryPolicy._verify(sender, data)
-         && NonSanctionedPolicy._verify(sender, data);
+     bool verified = readAccountPolicyCheckCache(sender);
+     if (verified) {
+         return true;
+     }
+     verified = VerifiedAccountPolicy._verify(sender, data) && VerifiedCountryPolicy._verify(sender,
↪  data)
+         && NonSanctionedPolicy._verify(sender, data);
+     if (verified) {
+         setAccountPolicyCheckCache(sender, true);
+     }
+     return verified;
  }
```

Coinbase: Acknowledged as something we can enable in the future by updating our policy contracts.

Spearbit: Acknowledged.

5.3.2 Gas savings by short circuiting the check on `vbaAttestation` if the first evaluates to `true`

Severity: Gas Optimization

Context: VerifiedAccountPolicy.sol#L72-L85, VerifiedCountryPolicy.sol#L99-L131

Description: Only one of `vaVerified` and `vbaVerified` is required to be true.

Recommendation: When `vaAttestation.uid != 0 && vaVerified` the finding (`_findAttestation`), and validating, of `vbaAttestation` can be skipped.

Coinbase: Fixed in commit 645d63f8

Spearbit: Fixed.

5.3.3 `_validCountry()` check is redundant

Severity: Gas Optimization

Context: VerifiedCountryPolicy.sol#L226-L228

Description: The `_isVerifiedCountry()` function contains a redundant check to `_validCountry()`. This function checks that the country string length is exactly 2 characters, however, the same `_validCountry()` check is performed when adding and removing a country from the `_countries` mapping. Therefore, there is no path for the country code input to be included in the `_countries` mapping while simultaneously not having a length of two characters. The extra call to `_validCountry()` occurs either once or twice upon each call to `verify()`:

- VerifiedCountryPolicy.sol#L118-L125:

```
if (vcValid) {
    string memory countryCode = _decodeCountryCode(vcAttestation.data);
    vcInVerifiedCountry = _isVerifiedCountry(countryCode);
}
if (vbcValid) {
    string memory countryCode = _decodeCountryCode(vbcAttestation.data);
    vbcInVerifiedCountry = _isVerifiedCountry(countryCode);
}
```

Recommendation: Perhaps this implementation was chosen since there is an external `isVerifiedCountry()` function that likely needs to check both string length and inclusion in the `_countries` mapping. It is recommended to move the `_validCountry` check to the external function and remove it from the internal `_isVerifiedCountry` to eliminate the redundant check.

Coinbase: Fixed in commit 5638c838

Spearbit: Fixed.

5.3.4 `_validCountry` can be constrained further

Severity: Gas Optimization

Context: *(No context files were provided by the reviewer)*

Description: As `_validCountry`'s comment states, the country codes use the Alpha-2 standard. Meaning country codes are represented by two uppercase ASCII characters.

The current code uses a UTF-8 string length library to properly count multi-byte characters.

Recommendation: As allowed characters for alpha-2 are single-byte characters, the `StringUtils` library can be removed and the check can be simplified to:

```
  function _validCountry(string memory countryCode) private view returns (bool) {
-     return StringUtils.strlen(countryCode) == 2;
+     // checking byte length is enough as valid Alpha-2 consists of two (single-byte) ASCII characters
+     return bytes(countryCode).length == 2;
  }
```

The check could be further constrained to check for uppercase `A-Z` characters (if the backend properly uppercases the country codes).

Coinbase: Fixed in commit cb8171d2.

Spearbit: Fixed.

5.4 Informational

5.4.1 Verified and unverified pool funds are likely to be mixed through arbitrage

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: Arbitrage between pools is a core mechanism in decentralized exchanges (DEXs) that helps maintain price synchronization with real market prices. This process has several benefits:

1. Reduced slippage for users.
2. More efficient trades.
3. Balanced prices across different pools.

Arbitrageurs are likely to interact with both verified and unverified pools to capitalize on price differences. This process occurs automatically as soon as arbitrage bots identify an opportunity.

As arbitrageurs keep prices synchronized between verified and unverified pools, liquidity providers in unverified pools may inadvertently benefit from swaps performed in verified pools. This creates a potential issue: funds could indirectly flow between Coinbase's verified pools and potentially sanctioned entities in unverified pools.

Recommendation: While it's possible to forbid verified users from interacting with unverified pools in the same transaction or swap action, this approach does not address the issue in its entirety. Arbitrage opportunities will eventually be balanced out indirectly, making it challenging to implement a simple mitigation strategy.

Coinbase: Acknowledged.

Spearbit: Acknowledged.

5.4.2 Swapping and modifying liquidity positions is not possible in a single transaction for EOAs

Severity: Informational

Context: VerifiedPoolsBasicHook.sol#L316-L344, VerifiedPoolsBasicHook.sol#L260-L313

Description: In the current implementation of the `VerifiedPoolsBasicHook`, it is not possible to both perform a swap and modify liquidity positions in a single transaction for externally owned accounts. Interacting with any of the current routers acquires the lock from `PoolManager` and allows performing batched actions from the router contract. If both actions (swapping and modifying liquidities) are to be done in one go, one router would need to call into the other router.

When modifying liquidity positions, the hook requires the sender to be an approved position manager (`positionManagers[sender]`) and it requires an account policy verification on the original sender `originalSender = IMessageSender(sender).msgSender()`. When performing a swap, the hook requires the sender to be an approved swap router (`swapRouters[sender]`) and likewise, an account policy verification on the original sender.

When one router calls into the other, then `originalSender` will be a router and not the user executing the transaction. Therefore the account policy check will always fail in its current implementation.

Recommendation: While a position manager could have extended functionality to also allow for swap actions and be an approved swap router, it could be better to keep the roles separate as currently is done to remain the separation of concerns.

Another option for allowing both actions to be performed in one go is to allow the swap router to call into the position manager. The position manager would then have to be aware of the call originating from the swap router and could implement an ERC-2771 `msgSender` forwarding mechanism to allow for position modifications for the corresponding owner. The hook would also require to be made aware of the caller forwarding by checking whether the original sender is an approved swap router (`swapRouters[originalSender]`) to correctly perform the policy check on the correct user in the hook when modifying liquidities.

These modifications, however, do not come without additional considerations and risks.

Coinbase: Fixed in commit 038d0f2e.

We also want to note that this will change will allow an approved router to call into another approved router for swapping (1-level depth max) which we want to enable for integrators who call into the universal-router.

Lastly, for the posm change to work here, we'd need to update the ensureOwner check in the current posm to support an approved router calling in. This change only enables the hook to allow this if/when we update the posm implementation to support it in the future, but we think the future flexibility is worth it (can always deploy a posm v2 and don't need to migrate hooks).

Spearbit:

> We also want to note that this will change will allow an approved router to call into another approved router for swapping (1-level depth max) which we want to enable for integrators who call into the universal-router.

Just a note: The universal router inherits the v4 swaprouter, it does not call into it. So the `PoolManager`'s caller (hook's `sender` parameter) should still be the universal router when doing a v4 swap through the universal router. (The universal router does however call into the position manager for managing liquidity - but this one is not *your* position manager and it shouldn't be an allowed position manager in your protocol as it doesn't have the transfer restrictions).

Maybe I'm misunderstanding, but wasn't this also possible before this change by just allowing the universal router as a swap router?

To do swap + liquidity in a single transaction you still need to deploy your own `NestingRouter`-style router and do the mentioned changes to `ensureOwner` in the current position manager, right?

Coinbase:

> We also want to note that this will change will allow an approved router to call into another approved router for swapping (1-level depth max) which we want to enable for integrators who call into the universal-router.

This is to potentially allow an integrators own smart contract to call into the universal-router where the integrators smart contract would implement the msgSender() interface that we can then pull the sender from.

5.4.3 Swapping can be used to allow funds to enter pools bypassing the donation hooks

Severity: Informational

Context: VerifiedPoolsBasicHook.sol#L383-L407

Description: Swapping large, possibly flash loaned amounts back and forth can allow actors to donate to liquidity providers through fees without triggering the `VerifiedPoolsBasicHook`'s `beforeDonate` hooks.

Recommendation: As the swap hook policy checks are likely to be just as strict or even stricter than the donation hooks this does not pose a practical issue at the moment. Be aware of this possibility in case the policy checks change in the future.

Coinbase: Acknowledged.

Spearbit: Acknowledged.

5.4.4 Verified accounts can use sanctioned 6909 claim tokens

Severity: Informational

Context: *(No context files were provided by the reviewer)*

Description: A sanctioned account can create an internal `PoolManager` claim balance through the `mint` and `sync`/`transfer`/`settle` combination. They can then set any account as an operator. This operator could `burn` the sanctioned account's balance to increase their own account's delta which can be used to pay for swaps or adding liquidity.

Note that currently the `MINT_6909` and `BURN_6909` actions are defined but not enabled on the `V4Router` (used by `UniversalRouter`) and the `PositionManager`.

However, a verified smart contract could unlock the pool manager to perform the burn prior to using the router's `modifyLiquiditiesWithoutUnlock` or similar functions. Tokens with callbacks or notification receivers through the `subscribe` functionality in the `PositionManager` could also hijack the control flow to `burn` on the pool manager.

Recommendation: The verified account would breach Coinbase's ToS.

Coinbase: Acknowledged.

Spearbit: Acknowledged.

5.4.5 Setting a new `schemaUid` effectively invalidates all existing verifications

Severity: Informational

Context: VerifiedAccountPolicy.sol#L97-L128

Description: For the purpose of verified pools, updating `schemaUid` effectively invalidates all existing verifications under the previous schema.

Recommendation: Be aware of above when migrating to new schemas, and be certain to not revert back to previous schemas without first revoking all that were revoked in a later schema version.

Coinbase: Acknowledged.

Spearbit: Acknowledged.

5.4.6 Caution when relying on `EAS` directly over the `indexer` to view attestation validity

Severity: Informational

Context: AttestationFinder.sol#L25

Description: Duplicate attestations in the same block result in two records, and `attestationUid`s. The indexer only tracks one `attestationUid` per `recipient/schemaUid` combination meaning the `EAS` contract may reflect a valid attestation when the latest has been revoked. There is no identified problem associated with this scenario as the indexer only references the latest attestation for a particular `recipient/schemaUid` combination.

Coinbase: Acknowledged.

Spearbit: Acknowledged.

5.4.7 Function argument shadows state variable

Severity: Informational

Context: VerifiedPoolsBasicHook.sol#L209, VerifiedPoolsBasicHook.sol#L539

Description: The `constructor` and `setQuoter` function have an argument that shadows the storage variable by the same name.

Recommendation: Avoid variable shadowing by considering the pattern used in the `_setQuoter(address quoterAddress)` function of adding a suffix `Address` or use the convention of `_` as the suffix. For more information see Shadowing State Variables.

Coinbase: Fixed in commit da597433.

Spearbit: Fixed.

5.4.8 Silence linter warning for unused function parameters

Severity: Informational

Context: NonSanctionedPolicy.sol#L49, VerifiedAccountPolicy.sol#L71, VerifiedCountryPolicy.sol#L98

Description: There are unused function parameters in some of the functions.

Recommendation:

- `VerifiedPoolsPositionManager.sol`:

```
- function transferFrom(address from, address to, uint256 id) public override {
+ function transferFrom(address, address, uint256) public override {
```

- `VerifiedCountryPolicy.sol`, `VerifiedAccountPolicy.sol`, `NonSanctionedPolicy.sol`:

```
- function _verify(address sender, bytes calldata data) internal view virtual returns (bool) {
+ function _verify(address sender, bytes calldata) internal view virtual returns (bool) {
```

Coinbase: Fixed in commit 1d99a4c1.

Spearbit: Fixed.

5.4.9 Consider adding a check to avoid user error and confirm `revocable == true`

Severity: Informational

Context: AttestationVerifier.sol#L25

Description: The `verifyAttestation` check verifies a similar set of checks to what is verified in the `indexer`. While gas could be saved by relying on the indexing the added safety is not harmful. The attestations for verified pools are also assumed to be revokable.

Recommendation: Consider adding an additional check confirming `revocable == true`.

Coinbase: Acknowledged.

Spearbit: Acknowledged.

5.4.10 `Quoter` swap simulations ignore policy checks

Severity: Informational

Context: VerifiedPoolsBasicHook.sol#L322

Description: The `Quoter` swap simulations trigger the hooks but the `beforeSwap` hook includes a special case to ignore the policy checks (this is because the `Quoter` does not provide the original sender).

Recommendation: Consider documenting in the natspec of `beforeSwap` that swap simulations using the `Quoter` will ignore the policy checks and performing the actual swap might fail.

Coinbase: Acknowledged. We could enhance the comment, and we will have to be diligent to only allowlist official quoters which explicity perform the revert at the end of their interaction.

Spearbit: Acknowledged.

5.4.11 `PolicyCheckFailed(originalSender)` error will not be used as policies already revert

Severity: Informational

Context: VerifiedPoolsBasicHook.sol#L281

Description: The hooks call the `policy` to check the sender:

```
bool verified = policy.verify(originalSender, hookData);
if (!verified) {
    revert PolicyCheckFailed(originalSender);
}
```

Even though the `policy.verify` returns a `boolean` on success, it'll always revert with its own error instead of returning `false`. (This is true for both `BasicPolicy` and `RemoveLiquidityPolicy`.)The error that will be shown is the internal error of the `policy` and not the `PolicyCheckFailed` error.

Recommendation: If there is no preference as to what error is ultimately thrown, the code can be left as is (and `verify` does not need to return a `boolean`). Otherwise, the policies should return `false` instead of reverting with their own error so the `Hook` can revert with the `PolicyCheckFailed(msg.sender)` error.

Coinbase: Fixed in commit cd5072ec.

Spearbit: Fixed.

5.4.12 `SanctionsListUpdated` parameter order is swapped

Severity: Informational

Context: NonSanctionedPolicy.sol#L21

Description: The `SanctionsListUpdated(address indexed sanctionsList, address indexed previousSanctionsList)` event's first parameter is the next sanctions list and the second parameter is the previous sanctions list. Usually, for these type of events the order is swapped with the previous value being the first. See also:

```
event VASchemaUpdated(bytes32 previousSchema, bytes32 currentSchema);
event VBASchemaUpdated(bytes32 previousSchema, bytes32 currentSchema);
event VCSchemaUpdated(bytes32 previousSchema, bytes32 currentSchema);
event VBCSchemaUpdated(bytes32 previousSchema, bytes32 currentSchema);
```

Recommendation: Consider swapping the order of the parameters in `SanctionsListUpdated` and update the `emit` with the new order.

```
- SanctionsListUpdated(address indexed sanctionsList, address indexed previousSanctionsList);
+ SanctionsListUpdated(address indexed previousSanctionsList, address indexed sanctionsList);

  function _setSanctionsList(ISanctionsList sanctionsList) internal {
      // ...

-     emit SanctionsListUpdated(address(sanctionsList), previousSanctionsList);
+     emit SanctionsListUpdated(previousSanctionsList, address(sanctionsList));
  }
```

Coinbase: Fixed in commit f74c78f8.

Spearbit: Fixed.

5.4.13 Typos & documentation errors

Severity: Informational

Context: BasicPolicy.sol#L24, IMessageSender.sol#L6, IMessageSender.sol#L7, ISanctionsList.sol#L4, NonSanctionedPolicy.sol#L14, VerifiedCountryPolicy.sol#L15, VerifiedCountryPolicy.sol#L212, VerifiedCountryPolicy.sol#L239, VerifiedCountryPolicy.sol#L55, VerifiedPoolsBasicHook.sol#L524

Description: There are typos and natspec/documentation errors in the code.

- Documentation:
  1. VerifiedCountryPolicy.sol#L55: "Error when both attestations are valid" → "Error when both attestations are invalid".
- Typos:
  1. IMessageSender.sol#L6-L8: "to is used" → "is used", "actua" → "actual", "having to using" → "having to use".
  2. ISanctionsList.sol#L4: "sactions" → "sanctions".
  3. BasicPolicy.sol#L24: "polcies" → "policies".
  4. NonSanctionedPolicy.sol#L14: "SactionsList" → "SanctionsList".
  5. VerifiedCountryPolicy.sol#L15: "Policy contract to verify if the caller in a verified country." → "Policy contract to verify if the caller is in a verified country."
  6. VerifiedCountryPolicy.sol#L212: "complys" → "complies".
  7. VerifiedCountryPolicy.sol#L239: "unint256" → "uint256".
  8. VerifiedPoolsBasicHook.sol#L524: "befreDonate" → "beforeDonate".

Recommendation: Consider addressing the mentioned issues.

Coinbase: Acknowledged.

Spearbit: Acknowledged. Some (if not all) of these have been fixed throughout the other PRs.