33AUDITS & CO.

UniswapV4 Split Fee Hook Audit Report

Security Audit Report

Introduction

A security review of the [Uniswap V3 LP Split Hook protocol] was done, focusing on the security aspects of the smart contracts. This audit was performed by 33Audits & Co with Radoslav Radev as Security Researchers.

Disclaimer

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

About 33 Audits & Company

33Audits is an independent smart contract security researcher company and development group. We conduct audits as a group of independent auditors with various experience and backgrounds. We have conducted over 15 audits with dozens of vulnerabilities found and are experienced in building and auditing smart contracts. We have over 4 years of Smart Contract development with a focus in Solidity, Rust and Move. Check our previous work here or reach out on X @33audits.

About

Uniswap V3 LP Split Hook protocol is a decentralized protocol featuring swap functionality, pool-based architecture, liquidity management, fee management.

Repository: kyzooghost/uniswapv3-lp-split-hook

Commit: 0c9f04731582a2ed0815ca766dc0ed57b21da474

Scope: kyzooghost/uniswapv3-lp-split-hook/*

Severity Definitions

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - The technical, economic, and reputation damage from a successful attack

Likelihood - The chance that a particular vulnerability gets discovered and exploited

Severity - The overall criticality of the risk

Informational - Findings in this category are recommended changes for improving the structure, usability, and overall effectiveness of the system.

Findings Summary

ID	Title	Severity	Status
[C-01]	Native ETH handled as `address(0)` and passed to Uniswap	Critical	Fixed
[H-01]	Swapped arguments in pool initialization price computation	High	Fixed
[H-02]	`_getLatestPositiveWeight` reverse-loop underflow causes revert / DoS	High	Fixed
[H-03]	Accumulation double-counts and inflates `accumulatedProjectTokens` state variable	High	Fixed
[L-01]	Missing validation for `FEE_PROJECT_ID` in constructor	Low	Fixed
[L-02]	Zero slippage protection on all Uniswap V3 mint / rebalance operations	Low	Fixed
[L-03]	Leftover tokens after Uniswap V3 mint are never handled	Low	Fixed
[L-04]	`claimFeeTokensFor` mutates state without emitting an event	Low	Fixed
[L-05]	`LPFeesRouted` event does not reflect actual fee split	Low	Fixed
[L-06]	Permissionless operational functions are undocumented	Low	Fixed
[I-01]	Hardcoded 10-ruleset lookback in `_getLatestPositiveWeight`	Informational	Fixed
[I-02]	Tick spacing is not enforced for Uniswap V3 1% fee tier	Informational	Fixed

Critical

[C-01] Native ETH handled as `address(0)` and passed to Uniswap `

Description

The `UniV3DeploymentSplitHook.sol` smart contract treats the “native terminal token” as `address(0)` and passes that into Uniswap V3 pool creation/minting (`createAndInitializePoolIfNecessary` /

`mint`). Uniswap V3 pools use ERC20 token addresses (WETH for ETH), not `address(0)`. This means any flow that ends up with `terminalToken == address(0)` will attempt to create/mint a pool with a zero token address.

Relevant code

```
// Converts JBConstants.NATIVE_TOKEN to address(0) for Uniswap
compatibility
terminalToken = acContext.token == JBConstants.NATIVE_TOKEN ? address(0) :
acContext.token;
```

```
// Uses terminalToken directly as an ERC20 token address for Uniswap calls
(address token0, address token1) = _sortTokens(projectToken,
terminalToken);

address newPool =
    INonfungiblePositionManager(UNISWAP_V3_NONFUNGIBLE_POSITION_MANAGER)
        .createAndInitializePoolIfNecessary(token0, token1,
UNISWAP_V3_POOL_FEE, sqrtPriceX96);

(uint256 tokenId,,,) =
    INonfungiblePositionManager(UNISWAP_V3_NONFUNGIBLE_POSITION_MANAGER)
        .mint{value: terminalToken == address(0) ? terminalTokenAmount :
0}(
            INonfungiblePositionManager.MintParams({
                token0: token0,
                token1: token1,
                ...
            })
        );
```

Impact

Pool deployment / liquidity minting will revert and be permanently unusable for projects whose terminal token is native ETH. That bricks Stage 2 deployment and any later LP actions (fees, rebalance) for those setups.

Recommendations

Stop using `address(0)` as a Uniswap token. Use the canonical WETH address for Uniswap-side operations. Only use `address(0)` on the Juicebox side if needed, and translate it to WETH before calling Uniswap.

Status

Fixed

High

[H-01] Swapped arguments in pool initialization price computation

Description

In `UniV3DeploymentSplitHook.sol` smart contract, `_createAndInitializeUniswapV3Pool` calls `_getSqrtPriceX96ForLatestPositiveWeight` with swapped arguments:

```
_getSqrtPriceX96ForLatestPositiveWeight(projectId, projectToken,
terminalToken);
```

but the function signature expects `(projectId, terminalToken, projectToken)`. This computes the initial sqrt price using the wrong token roles.

```
function _createAndInitializeUniswapV3Pool(uint256 projectId, address
projectToken, address terminalToken) internal {
    (address token0, address token1) = _sortTokens(projectToken,
terminalToken);
    uint160 sqrtPriceX96 =
_getSqrtPriceX96ForLatestPositiveWeight(projectId, projectToken,
terminalToken);
    ...
}
```

```
function _getSqrtPriceX96ForLatestPositiveWeight(
    uint256 projectId,
    address terminalToken,
    address projectToken
) internal view returns (uint160 sqrtPriceX96) { ... }
```

Impact

The pool can be initialized at an incorrect / inverted price (or revert depending on downstream conversions). If it initializes and liquidity is minted, external arbitrage can extract value from the LP position immediately due to mispricing.

Recommendations

Call the function with the correct argument order:

```
_getSqrtPriceX96ForLatestPositiveWeight(projectId, terminalToken,
projectToken);
```

Optionally, add sanity checks on computed price bounds before initializing.

Status

Fixed

[H-02] `_getLatestPositiveWeight` reverse-loop underflow causes revert / DoS

Description

`UniV3DeploymentSplitHook.sol#_getLatestPositiveWeight()` function uses an invalid reverse loop over a `uint256` index:

```
for (uint256 i = rulesets.length - 1; i >= 0; i--)
```

With unsigned integers, `i >= 0` is always true. When `i` reaches 0, `i--` underflows and the loop can revert / read out of bounds.

```
function _getLatestPositiveWeight(uint256 projectId) internal view returns
(uint256 weight) {
    ...
    JBRulesetWithMetadata[] memory rulesets =
IJBController(controller).allRulesetsOf(projectId, 0, 10);

    for (uint256 i = rulesets.length - 1; i >= 0; i--) {
        if (rulesets[i].ruleset.weight >= threshold) {
            return rulesets[i].ruleset.weight;
        }
    }

    return 0;
}
```

Impact

Any path that relies on `_getLatestPositiveWeight` can revert, including pool initialization via `_getSqrtPriceX96ForLatestPositiveWeight`. This can DoS pool deployment.

Recommendations

Rewrite the loop safely:

```
for (uint256 i = rulesets.length; i > 0; i--) {
    uint256 idx = i - 1;
    if (rulesets[idx].ruleset.weight >= threshold) return
rulesets[idx].ruleset.weight;
}
return 0;
```

Status

Fixed

[H-03] Accumulation double-counts and inflates `accumulatedProjectTokens` state variable

Description

Stage 1 accumulation is tracked incorrectly. `_accumulateTokens` adds the contract's full project-token balance on every call:

```
accumulatedProjectTokens[projectId] +=
IERC20(projectToken).balanceOf(address(this));
```

This double-counts across multiple split events and inflates `accumulatedProjectTokens[projectId]` beyond the real balance.

Relevant code

```
function _accumulateTokens(uint256 projectId, address projectToken)
internal {
    uint256 projectTokenBalance =
IERC20(projectToken).balanceOf(address(this));
    accumulatedProjectTokens[projectId] += projectTokenBalance;
}
```

Impact

Deployment uses `accumulatedProjectTokens[projectId]` as the source of truth (e.g., cashing out half). Once inflated, the contract can attempt to cash out / use more project tokens than it actually has, causing deployment to revert and potentially bricking Stage 2 deployment.

Recommendations

Track the incremental amount received per split instead of total balance. The simplest fix is to use the split amount from the hook context in `processSplitWith` (Stage 1 path) and do:

```
accumulatedProjectTokens[projectId] += context.amount;
```

Avoid using `balanceOf(this)` for accumulation tracking.

Status

Fixed

Low

[L-01] Missing validation for `FEE_PROJECT_ID` in constructor

Description

`FEE_PROJECT_ID` is immutable but never validated to correspond to an existing Juicebox project. If it is set to `0` or a non-existent project, `_routeFeesToProject` silently skips routing fees because `primaryTerminalOf` returns `address(0)`.

This does not revert and provides no signal that the fee split is misconfigured.

Recommendations

Validate that `FEE_PROJECT_ID` has a valid controller or terminal during construction, or explicitly document that misconfiguration disables fee routing.

Status

Fixed

[L-02] Zero slippage protection on all Uniswap V3 mint / rebalance operations

Description

All Uniswap V3 `mint` and `decreaseLiquidity` calls use:

```
amount0Min: 0,
amount1Min: 0
```

This accepts any execution outcome and provides no protection against unfavorable price movement during execution.

Recommendation

Add configurable or conservative minimum amounts to reduce value loss from adverse execution.

Status

Fixed

[L-03] Leftover tokens after Uniswap V3 mint are never handled

Description

The return values `amount0` and `amount1` from `mint()` are ignored. Any unused project or terminal tokens remain in the contract with no recovery, sweep, or accounting mechanism.

Over time, this can strand dust or non-trivial balances.

Recommendations

Track unused amounts after mint and either re-accumulate, burn, or provide a recovery path.

Status

Fixed

[L-04] `claimFeeTokensFor` mutates state without emitting an event

Description

`claimFeeTokensFor` resets `claimableFeeTokens[projectId]` and transfers tokens, but emits no event.

This makes off-chain accounting and monitoring of fee claims difficult.

Recommendations

Emit an event such as `FeeTokensClaimed(projectId, beneficiary, amount)` on successful claims.

Status

Fixed

[L-05] `LPFeesRouted` event does not reflect actual fee split

Description

`LPFeesRouted` emits only the total routed amount, but not:

- the fee portion sent to `FEE_PROJECT_ID`
- the remaining amount sent to the project
- the number of fee-project tokens minted

This prevents accurate off-chain reconstruction of fee flows.

Recommendations

Extend the event to include fee amount and remaining amount, or emit separate events per routing action.

Status

Fixed

[L-06] Permissionless operational functions are undocumented

Description

The following functions are permissionless by design:

- `deployPool`
- `collectAndRouteLPFees`
- `rebalanceLiquidity`

While not unsafe, this behavior is non-obvious and undocumented.

Recommendations

Explicitly document that these functions are intentionally permissionless and safe to be called by anyone.

Status

Fixed

Informational

[I-01] Hardcoded 10-ruleset lookback in `_getLatestPositiveWeight`

Description

`_getLatestPositiveWeight` only inspects the last 10 rulesets:

```
allRulesetsOf(projectId, 0, 10)
```

If a project has more than 10 historical rulesets, the intended reference weight may be skipped.

Recommendations

Document this limitation or make the lookback configurable.

Status

Fixed

[I-02] Tick spacing is not enforced for Uniswap V3 1% fee tier

Description

Ticks derived via `TickMath.getTickAtSqrtRatio` are used directly without alignment to the pool's tick spacing (200 for the 1% fee tier).

Unaligned ticks can cause `mint()` or rebalance operations to revert.

Recommendations

Align `tickLower` and `tickUpper` to the pool's tick spacing before minting.

Status

Fixed

Conclusion

This security audit of the [Uniswap V3 LP Split Hook protocol] identified 12 vulnerabilities across 1 Critical, 3 High, 6 Low, and 2 Informational. Issues have been addressed by the development team.

Key findings addressed include:

- **Critical**: 1 critical finding was identified and addressed
- **High**: 3 high findings were identified and addressed
- **Low**: 6 low findings were identified and addressed
- **Informational**: 2 Informational findings were identified and reported

Key improvements implemented include:

- Enhanced input validation and access controls
- Improved error handling and edge case coverage
- Strengthened security patterns and best practices

The development team's commitment to addressing these findings demonstrates a strong security-focused approach to smart contract development. The protocol has undergone significant improvements based on the audit findings, with all reported issues resolved. Continued monitoring and periodic security reviews are recommended as the protocol evolves.

This report was prepared by 33Audits & Co and represents our independent security assessment of the [Uniswap V3 LP Split Hook protocol] project.