



Feast

Security Audit Report

October 9, 2026



Contents

1 Introduction

[1.1 About Feast](#)

[1.2 Source Code](#)

[1.3 Revision History](#)

2 Overall Assessment

3 Vulnerability Summary

[3.1 Overview](#)

[3.2 Security Level Reference](#)

[3.3 Vulnerability Details](#)

4 Appendix

[4.1 About AstraSec](#)

[4.2 Disclaimer](#)

[4.3 Contact](#)

1 Introduction

1.1 About Feast

Feast is a token launch protocol on BNB Chain, built on PancakeSwap Infinity. A creator mints a fixed-supply token and opens a concentrated-liquidity pool in one transaction, with the liquidity locked permanently. A hook takes 1% of every swap and splits it among the platform, the creator, a referrer, and a buy-and-burn.



1.2 Source Code

The following source code was reviewed during the audit:

- ▶ <https://github.com/alanmagpie/feast-contracts>
- ▶ CommitID: e5c0a77

And this is the final version representing all fixes implemented for the issues identified in the audit:

- ▶ <https://github.com/alanmagpie/feast-contracts>
- ▶ CommitID: 44f36b8a

1.3 Revision History

Version	Date	Description
v1.0	Oct 9, 2026	Initial Audit

2 Overall Assessment

This report has been compiled to identify issues and vulnerabilities within the Feast protocol. Throughout this audit, we identified a total of 8 issues spanning various severity levels. By employing auxiliary tool techniques to supplement our thorough manual code review, we have discovered the following findings.

Severity	Count	Acknowledged	Won't Do	Addressed
Critical	—	—	—	—
High	—	—	—	—
Medium	4	—	—	4
Low	4	2	—	2
Informational	—	—	—	—
Total	8	2	—	6

3 Vulnerability Summary

3.1 Overview

Click on an issue to jump to it, or scroll down to see them all.

~~M-1~~

[Potential Sandwich Attack on FeastHook::claimAsBnb\(\)](#)

~~M-2~~

[Permanent FEAST Lock in FeastLaunchWindow::settle\(\)](#)

~~M-3~~

[START Display with Descent Curve Kept outside the Pricing Band in FeastPlatformSink](#)

~~M-4~~

[Permanent Referrer Binding of an Unbound Bundler](#)

L-1

[Anti-bot Cap Bypass via Multiple Wallets](#)

~~L-2~~

[Silent TWAP Window Shortening in FeastOracle::consult\(\)](#)

L-3

[Potential Risks Associated with Centralization](#)

~~L-4~~

[Potential Emissions Capture through a Shallow Pool Pinned in FeastPricer](#)

3.2 Security Level Reference

In web3 smart contract audits, vulnerabilities are typically classified into different severity levels based on the potential impact they can have on the security and functionality of the contract. Here are the definitions for critical-severity, high-severity, medium-severity, and low-severity vulnerabilities:

Severity	Acknowledged
C-X (Critical)	A severe security flaw with immediate and significant negative consequences. It poses high risks, such as unauthorized access, financial losses, or complete disruption of functionality. Requires immediate attention and remediation.
H-X (High)	Significant security issues that can lead to substantial risks. Although not as severe as critical vulnerabilities, they can still result in unauthorized access, manipulation of contract state, or financial losses. Prompt remediation is necessary.
M-X (Medium)	Moderately impactful security weaknesses that require attention and remediation. They may lead to limited unauthorized access, minor financial losses, or potential disruptions to functionality.
L-X (Low)	Minor security issues with limited impact. While they may not pose significant risks, it is still recommended to address them to maintain a robust and secure smart contract.
I-X (Informational)	Warnings and things to keep in mind when operating the protocol. No immediate action required.
U-X (Undetermined)	Identified security flaw requiring further investigation. Severity and impact need to be determined. Additional assessment and analysis are necessary.

3.3 Vulnerability Details

3.3.1 [M-1] Potential Sandwich Attack on FeastHook::claimAsBnb()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
FeastHook.sol	Time and State	Medium	Medium	Addressed

`claimAsBnb()` lets the beneficiary convert fees to `wrappedNative` along a self-chosen route, but there is no beneficiary-specified `minOut` — any actual output must be accepted. The only on-chain protections are: (1) a per-hop admission requiring spot to be within 300 ticks ($\approx 3.045\%$) of TWAP; (2) a final route-level floor of 3%. An attacker can first push the spot price to the lower band edge on each hop pool, deflating the expectation itself, and then execute at the floor: worst case across three hops $\approx 11.35\%$ slippage.

```

1413 function claimAsBnb(PoolKey calldata key, address account, FeastConverter.Hop[] calldata route)
1414     external
1415     returns (uint256 out)
1416 {
1417     ...
1418     out = converter.convert(Currency.unwrap(quote), amount, route, account);
1419 }
```

```

185 function convert(address tokenIn, uint256 amountIn, Hop[] calldata route, address recipient)
186     external
187     returns (uint256 amountOut)
188 {
189     ...
190     address[] memory pools = new address[](route.length);
191     uint160[] memory sqrtPrices = new uint160[](route.length);
192     {
193         address token = tokenIn;
194         for (uint256 i = 0; i < route.length; i++) {
195             (pools[i], sqrtPrices[i]) = _validateHop(token, route[i].tokenOut, route[i].fee);
196             token = route[i].tokenOut;
197         }
198     }
199     ...
200     uint256 floor_ =
201         Math.mulDiv(expectedOut, FeastConstants.BPS - MAX_ROUTE_DEVIATION_BPS, FeastConstants.BPS);
202     if (currentAmount < floor_) revert RouteReturnedTooLittle(currentAmount, expectedOut, floor_);
203     ...
204     ...
205     ...
206 }
```

```

209 function _validateHop(address tokenIn, address tokenOut, uint24 fee)
210     internal
211     view
212     returns (address pool, uint160 sqrtPriceX96)
213 {
214     ...
215     uint32 age = _oldestObservationAge(pool, index, cardinality);
216     if (cardinality < MIN_ORACLE_CARDINALITY || age < FeastConstants.MIN_OBSERVATION_AGE) {
217         revert OracleUnusable(pool, cardinality, age);
218     }
219
220     int24 averageTick = _averageTick(pool, age);
221     int24 deviation = spotTick > averageTick ? spotTick - averageTick : averageTick - spotTick;
222     if (deviation > MAX_TICK_DEVIATION) revert SpotTooFarFromAverage(pool, spotTick, averageTick);
223 }

```

Remediation Add a beneficiary-specified `minOut` to `claimAsBnb()`, pass it into `convert()`, and enforce it at the final check.

3.3.2 [M-2] Permanent FEAST Lock in FeastLaunchWindow::settle()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
FeastLaunchWindow.sol	Business Logic	High	Low	Addressed

`settle()` is the only call that creates the platform trading pool, and it is allowed only for one hour after the auction ends. If it does not succeed in that hour, the pool is never created and the launch window keeps 60% of the FEAST supply with no market.

When FEAST is currency1, a small amount of WBNB raised makes the clearing price too large for the pool to store, and `settle()` reverts (line 230). An auction with no bids would still open the pool. A bid of 1 wei blocks that path and causes the same revert. The lock also follows if nobody calls `settle()` during the hour.

```

207 function settle() external nonReentrant {
208     ...
209
210     if (total == 0) {
211         int24 startTick = factory.platformOpening();
212         uint160 sqrtOpening = TickMath.getSqrtPriceAtTick(startTick);
213         poolManager.initialize(_key, sqrtOpening);
214         uint128 liquidity = coinIsCurrency0
215             ? _amount0Liquidity(coinHeld, sqrtOpening)
216             : _amount1Liquidity(coinHeld, sqrtOpening);
217         IERC20(platformCoin).safeTransfer(address(hook), coinHeld);
218         (int24 lower, int24 upper) =
219             coinIsCurrency0 ? (startTick, _maxUsableTick()) : (_minUsableTick(), startTick);
220         factory.seedWindowOpening(lower, upper, liquidity);
221         tranche = 0;
222         settled = true;
223         emit Settled(0, 0, 0);
224         return;
225     }
226     ...
227
228     (uint256 num, uint256 den) = coinIsCurrency0 ? (net, sold) : (sold, net);
229
230     if (num / den >= 1 << 64) revert ClearingPriceUnrepresentable(num, den);
231
232     ...
233 }
```

Remediation When `net` is below the representable floor, treat the round as no participation and refund the dust. Keep `settle()` available after the hour, so a missed call cannot lock the supply.

3.3.3 [M-3] START Display with Descent Curve Kept outside the Pricing Band

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
FeastPlatformSink.sol	Business Logic	High	Low	Mitigated

The `lot` is priced on its descending curve only while `offered <= held < 2 * offered`. Outside the band, `priceQ128()` returns START (line 311) but `anchor/anchoredAt` are not reset and the descent clock keeps running. When the balance re-enters the band (via a transfer in, or via `take()` reducing it from above), the already-decayed curve applies immediately. An attacker can wait for the curve to decay, then clear the inventory far below START — with nearly no cost if the balance can be pushed below `offered` first (e.g. deflationary token) and 1 wei transferred back in.

```
feast-contracts-feat-v1.6.10-fixes - FeastPlatformSink.sol
304 function priceQ128(address asset) public view returns (uint256 price) {
305     return _price(asset, _lots[asset], IERC20(asset).balanceOf(address(this)));
306 }
307
308 function _price(address asset, Lot storage lot, uint256 held) private view returns (uint256 price) {
309     if (lot.anchoredAt == 0) revert NotOpened(asset);
310     if (!pricesThisLot(lot, held)) return _atLeastTheFloor(lot, FeastGenesis.SINK_START_PRICE_Q128);
311
312     uint256 frozen = guardian.frozenSeconds() - lot.frozenAtAnchor;
313     uint256 wall = block.timestamp - lot.anchoredAt;
314     uint256 elapsed = wall > frozen ? wall - frozen : 0;
315     uint256 halvings = elapsed / FeastGenesis.SINK_HALF_LIFE;
316     price = halvings >= 255 ? 0 : lot.anchor >> halvings;
317     uint256 into = elapsed % FeastGenesis.SINK_HALF_LIFE;
318
319     price -= Math.mulDiv(price, into, 2 * FeastGenesis.SINK_HALF_LIFE);
320     price = _atLeastTheFloor(lot, price);
321 }
322
323 function _pricesThisLot(Lot storage lot, uint256 held) private view returns (bool) {
324     return held >= lot.offered && held / FeastGenesis.SINK_REOPEN_MULTIPLE < lot.offered;
325 }
```

```
feast-contracts-feat-v1.6.10-fixes - FeastPlatformSink.sol
393 function take(address asset, uint256 amountOut, uint256 maxCoinIn) external nonReentrant returns (uint256 cost) {
394     _requireUnfrozen();
395     if (amountOut == 0) revert NothingToTake();
396     uint256 held = IERC20(asset).balanceOf(address(this));
397     if (amountOut > held) revert MoreThanHeld(amountOut, held);
398     Lot storage lot = _lots[asset];
399
400     uint256 price = _price(asset, lot, held);
401     ...
402 }
```

Remediation Reset the descent when the balance leaves the band, so a later return starts a new curve instead of the one that ran while the quote showed START.

3.3.4 [M-4] Permanent Referrer Binding of an Unbound Bundler

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
FeastHook.sol	Business Logic	Medium	Medium	Addressed

Referrers are stored at `_referrerOf[tx.origin]` with lifetime binding and no way to clear or replace the entry. Under ERC-4337, `tx.origin` is the bundler (a public address). While a bundler is unbound, any swap it submits can carry `abi.encode(attacker, bundler)` as `hookData`; the equality holds and the bundler becomes permanently bound to the attacker (line 567). All later trades that bundler relays — on every pool, for every account — credit the referrer fee to the attacker.

```
feast-contracts-feat-v1.6.10-fixes - FeastHook.sol
546
547 function _referrerFor(PoolId poolId, Pool storage p, address sender, bytes calldata hookData)
548     private
549     returns (address)
550 {
551     ...
552     address origin = tx.origin;
553
554     address bound = _referrerOf[origin];
555     if (bound != address(0)) return bound;
556
557     (address named, address wallet) = _referrerFrom(hookData);
558     if (named == address(0)) return address(0);
559
560     if (named == origin) return address(0);
561
562     // The 64-byte form, the only one that can bind.
563     if (wallet != address(0)) {
564         ...
565         if (wallet != origin) return named;
566
567         _referrerOf[origin] = named;
568         emit ReferrerBound(origin, named, poolId);
569         return named;
570     }
571
572     return named;
573 }
```

Remediation Remove the automatic bind from the 64-byte `hookData`; instead add a function to change `_referrerOf[msg.sender]`, so a referrer can only be set explicitly through that function.

3.3.5 [L-1] Anti-bot Cap Bypass via Multiple Wallets

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
FeastHook.sol	Business Logic	Low	Low	Acknowledged

During the first 2 blocks after opening, the buy cap is accumulated per `tx.origin` (EOA identity). The attacker (or bot) only needs multiple independent wallets, each consuming its own cap, to bypass the 0.5%-per-wallet anti-bot limit, weakening opening fairness.

```
920
921 function _afterSwap(
922     ...
923 ) internal override returns (int128) {
924     ...
925     if (p.servedBlocks <= FeastConstants.CAP_BLOCKS && !(sender == factory && _prebuyMarked(poolId))) {
926         int128 coinLeg = p.coinIsCurrency0 ? delta.amount0() : delta.amount1();
927         // Only purchases. Capping sales would trap the earliest buyers in a
928         // coin they are not allowed to leave – the opposite of the intent.
929         if (coinLeg > 0) {
930             uint256 bought = _purchased[poolId][tx.origin] + uint256(uint128(coinLeg));
931             if (bought > p.perWalletCap) revert PurchaseCapExceeded();
932             _purchased[poolId][tx.origin] = bought;
933         }
934     }
935     ...
936     ...
937 }
```

Remediation Improve the anti-bot mechanism. For example, charge a higher fee on earlier purchases, so splitting a buy across several wallets is less profitable.

Response By Team It has been accepted by the team.

3.3.6 [L-2] Silent TWAP Window Shortening in FeastOracle::consult()

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
FeastOracle.sol	Business Logic	Medium	Low	Addressed

`consult()` silently shrinks [effective](#) to [age](#) when the pool's observation history is shorter than the requested window (only guaranteed not to fall below [MIN_OBSERVATION_AGE](#) = 300s). This leads to a mismatch between the window the caller expects and the window actually used, with no way to notice (the function neither reverts nor returns the actual window). This directly weakens manipulation resistance: the shorter the window, the faster the average follows price manipulation — an attacker only needs to keep the price deviated for a short while to drag the average anchor.

```
feast-contracts-feat-v1.6.10-fixes - FeastOracle.sol
106 function consult(PoolId poolId, uint32 window) public view returns (int24) {
107     uint256 n = _count[poolId];
108     if (n == 0) revert NoObservations();
109
110     uint32 nowTs = uint32(block.timestamp);
111     Observation memory oldest = _oldest(poolId, n);
112     uint32 age = _elapsedSince(oldest.timestamp);
113     if (age < FeastConstants.MIN_OBSERVATION_AGE) {
114         revert OracleTooYoung(age, FeastConstants.MIN_OBSERVATION_AGE);
115     }
116
117     uint32 effective = window > age ? age : window;
118     if (effective < FeastConstants.MIN_OBSERVATION_AGE) effective = FeastConstants.MIN_OBSERVATION_AGE;
119     uint32 target = uint32(block.timestamp - effective);
120
121     int56 cumulativeAtTarget = _cumulativeAt(poolId, n, target);
122
123     int56 cumulativeNow = _cumulativeAt(poolId, n, nowTs);
124     return int24((cumulativeNow - cumulativeAtTarget) / int56(uint56(effective)));
125 }
```

Remediation Revert when the price history is shorter than the requested window.

3.3.7 [L-3] Potential Risks Associated with Centralization

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
Multiple Contracts	Security	Medium	Low	Acknowledged

The Feast protocol relies on multiple privileged accounts that possess extensive control over critical operations, introducing notable centralization risks. These accounts, assigned distinct roles, can unilaterally influence the protocol's functionality and integrity. Below are key examples of privileged functions and their associated roles:

- Guardian Role: Can pause trading and outflows for the whole system ($\leq 72\text{h}$ at a time, $\leq 168\text{h}$ lifetime), and permanently stop creation of new coins.
- Grants Role: Can fill an empty grant slot once within 90 days; filled addresses cannot be replaced.
- Emissions Role: Can set the value-cap ratio for later weeks (5%–20%), redirect up to 50% of a later week's emissions to an arbitrary escrow, and replace the pricer used by later settlements.

This concentration of power in privileged accounts creates a dependency on their security and trustworthiness.

```
feast-contracts-feat-v1.6.10-fixes - FeastEmissions.sol
223
224 /// @notice Price the next settlements with a new pricer (new routes).
225 function setPricer(address pricer_) external onlyTimelock {
226     if (pricer_.code.length == 0) revert PricerHasNoCode(pricer_);
227     pricer = FeastPricer(pricer_);
228     emit PricerReplaced(pricer_);
229 }
```

Remediation To mitigate the identified issue, it is recommended to introduce multi-sig mechanism to undertake the role of the privileged accounts. Moreover, it is advisable to implement timelocks to govern all modifications to the privileged operations.

Response By Team This issue has been acknowledged by the team.

3.3.8 [L-4] Potential Emissions Capture through a Shallow Pool Pinned in FeastPricer

TARGET	CATEGORY	IMPACT	LIKELIHOOD	STATUS
FeastPricer.sol	Business Logic	Medium	Low	Mitigated

A shallow pool pinned in FeastPricer is easy to manipulate, inflating the quote price. `_vouch()` accepts spot whenever it is within 300 ticks of the pool's own TWAP; on a thin pool the spot can be pushed up and held ~600 seconds until the TWAP catches up, so `priceQ128()` still returns 'Ok'. `settle()` then values that quote's fees in WBNB at the inflated spot, capturing extra emissions.

```

212 function _vouch(IPancakeV3Pool pool) internal view returns (Status, uint160) {
213     (uint160 sqrtPriceX96, int24 spotTick, uint16 index, uint16 cardinality,,,) = pool.slot0();
214     if (!_inVenueRange(sqrtPriceX96)) return (Status.Unpriceable, 0);
215     if (cardinality < 2) return (Status.Unpriceable, 0);
216     uint32 age = _oldestObservationAge(pool, index, cardinality);
217     if (age < FeastConstants.MIN_OBSERVATION_AGE) return (Status.Unpriceable, 0);
218     uint32 window = age < FeastConstants.TWAP_WINDOW ? age : FeastConstants.TWAP_WINDOW;
219     uint32[] memory secondsAgos = new uint32[](2);
220     secondsAgos[0] = window;
221     try pool.observe(secondsAgos) returns (int56[] memory cumulatives, uint160[] memory) {
222         int24 averageTick = int24((cumulatives[1] - cumulatives[0]) / int56(uint56(window)));
223         int24 deviation = spotTick > averageTick ? spotTick - averageTick : averageTick - spotTick;
224         if (deviation > FeastConstants.MAX_TICK_DEVIATION) return (Status.Moved, 0);
225         return (Status.Ok, sqrtPriceX96);
226     } catch {
227         return (Status.Unpriceable, 0);
228     }
229 }
```

Remediation Every pool configured in FeastPricer must have sufficient depth and trading activity so the TWAP price cannot be pulled up.

4 Appendix

4.1 About AstraSec

AstraSec is a premier blockchain security firm dedicated to delivering high-quality auditing services for blockchain-based protocols. Composed of veteran security researchers with extensive experience across the DeFi landscape, our team maintains an unwavering commitment to excellence and precision. AstraSec’s comprehensive methodology and deep understanding of blockchain architecture enable us to ensure protocol resilience, making us a trusted partner for our clients.

4.2 Disclaimer

The content of this audit report is for informational purposes only and does not constitute legal, financial, or investment advice. The findings, views, and conclusions presented herein are based on the code and documentation provided at the specific time of the audit and may be subject to risks and uncertainties not detected during the assessment.

While AstraSec exerts every effort to ensure the accuracy and completeness of this report, the information is provided on an "as is" basis. We assume no responsibility for any errors, omissions, or inaccuracies. Users should conduct their own independent due diligence and consult with professional advisors before making any investment or deployment decisions. AstraSec shall not be held liable for any direct or indirect losses, damages, or consequences arising from reliance on this report.

4.3 Contact

Phone	+86 156 0639 2692
Email	contact@astrasec.ai
Twitter	https://x.com/AstraSecAI