BitcoinPizzaV2 Smart Contract Audit Report

Overview

This report reviews the provided `BitcoinPizzaV2` Solidity contract targeting BNB Chain and assesses security, correctness, governance behavior, and operational risks based on the code supplied for review.[cite:1]

The contract implements a BEP-20 token with a fixed maximum supply of 21 billion PIZZA, a 10 percent LP reserve minted to the contract at deployment, daily paid minting into 100-day escrow, a one-time PancakeSwap LP seeding path, and a vote-gated emergency withdrawal path for the reserve and accumulated BNB.[cite:1]

Scope

The review covered the full contract logic visible in the submitted source, including supply accounting, escrow minting and claims, transfer restrictions, LP seeding, governance voting, emergency withdrawal, and owner permissions.[cite:1]

This report is a source-level review only. It does not include deployment verification, bytecode matching, proxy analysis, external dependency audits of PancakeSwap or Chainlink, or adversarial economic simulation beyond what can be inferred from the contract logic.[cite:1]

Executive assessment

Based on the reviewed code, no clear critical exploit was identified that allows arbitrary theft of user escrowed PIZZA through the `emergencyWithdraw()` path.[cite:1]

The strongest earlier concern was that reserve tokens and escrowed tokens are both held in the contract balance, but under the observed state transitions the contract balance remains equal to `AUTO_LP_RESERVE + outstanding locked escrow`, and the emergency withdrawal amount is capped to `min(balanceOf(address(this)), AUTO_LP_RESERVE)`, which preserves user escrow backing under the present implementation.[cite:1]

The largest practical risks are not reserve theft but operational and design issues: claim gas growth over time, governance being Sybil-prone, dependency on oracle freshness for mint availability, and implicit accounting that is safe today but easy to misunderstand in future revisions.[cite:1]

Architecture summary

At deployment, the constructor mints the full 10 percent LP reserve to the contract itself and nothing to the deployer, while setting the owner and initial token URI.[cite:1]

Users may call `mint()` once per UTC day, paying a Chainlink-priced BNB fee to mint 10,000 PIZZA into escrow, subject to a 21,000-wallet daily cap and a total 90 percent user-pool cap.[cite:1]

Escrowed balances are represented by `Pending` entries plus `_lockedOf`, and users later call `claimUnlocked()` to move matured tokens from the contract balance to their wallet after the 100-day lock period.[cite:1]

Before LP is seeded, transfers are intentionally blocked for normal holders by `_transfer()`, making the token non-tradeable until `seedLp()` succeeds after the full user pool has been minted.[cite:1]

If that threshold is never reached, the owner may open a vote round, token holders can vote with one vote per 100,000 PIZZA based on wallet balance plus locked escrow, and after a successful round the owner may withdraw the BNB plus reserve tokens one time through `emergencyWithdraw()`.[cite:1]

Findings

Severity table

Severity	Count	Notes
Critical	0	No clear critical drain or escrow theft issue found in the reviewed logic.[cite:1]
High	2	Claim gas growth and governance Sybil risk are the main high-severity concerns.[cite:1]
Medium	4	Mostly robustness, availability, and maintainability issues.[cite:1]
Low	4	Naming, event consistency, portability, and hygiene concerns.[cite:1]

High: Unbounded claim loop can create gas pressure

`claimUnlocked()` iterates over matured `Pending` entries in a `while` loop until it reaches a still-locked entry or the end of the queue.[cite:1]

A user who mints for many days can build up a long pending queue, and eventually a single claim transaction may become gas-heavy or inconvenient. This is primarily a self-DoS or UX scalability issue rather than a theft vector, but it is still material because the contract is designed for daily participation over long periods.[cite:1]

Recommendation: add a bounded claim function such as `claimUnlocked(uint256 maxEntries)` so users can process matured escrow in chunks.[cite:1]

High: Governance is Sybil-prone

Voting power is computed as `(wallet balance + locked escrow) / 100,000 PIZZA`, and voting is recorded per address for the active round.[cite:1]

This makes governance easy to split across multiple addresses from an identity perspective, and power is evaluated at vote time rather than by a block snapshot, so the mechanism is governance-light rather than governance-hard. This is not a direct smart contract exploit, but it weakens the social trust assumptions of the emergency path.[cite:1]

Recommendation: if stronger governance guarantees are desired, use explicit snapshot logic or a more formal vote token / checkpoint model.[cite:1]

Medium: Oracle freshness can halt minting

`currentMintFeeWei()` depends on Chainlink `latestRoundData()` and reverts if the answer is non-positive or older than `PRICE_STALE_AFTER`.[cite:1]

That behavior is defensible and safety-oriented, but it also means minting availability depends on timely oracle updates. In an oracle outage or stale-data event, users cannot mint even though no contract-side fault exists.[cite:1]

Recommendation: keep this behavior if liveness loss is acceptable for price safety; otherwise document it clearly in user-facing materials.[cite:1]

Medium: `pendingCount()` depends on invariant integrity

`pendingCount(address u)` returns `_pending[u].length - _nextClaimIndex[u]` directly.[cite:1]

Under normal operation this is fine, but if invariants were ever corrupted by future edits, migration mistakes, or storage bugs, the subtraction could revert due to underflow. This is not exploitable in the current flow, but it is fragile design for a public view function.[cite:1]

Recommendation: guard with a conditional and return zero if the next index is already at or beyond length.[cite:1]

Medium: Reserve accounting is implicit rather than explicit

The reviewed code keeps reserve tokens and locked escrow within the same on-chain token balance at `address(this)`, relying on the invariant that contract balance equals reserve plus outstanding escrow.[cite:1]

That invariant appears sufficient for safety in the current version, but it is implicit rather than self-documenting. Future maintainers may misread the logic, and later code changes could break the invariant more easily than if reserve accounting were explicit.[cite:1]

Recommendation: consider adding a dedicated reserve tracker even though the present code path appears safe.[cite:1]

Medium: Day boundary uses timestamp bucketing

The minting rule uses `block.timestamp / 1 days` to determine the current UTC day bucket.[cite:1]

This is standard and likely acceptable here, but validators have small influence over timestamps near boundaries, so exact edge-of-day behavior is not perfectly rigid. The practical impact is low, but the rule should be described as day-bucket based rather than wall-clock exactness.[cite:1]

Low: Confusing total supply variable name

The token uses `override_totalSupply` as the backing supply variable.[cite:1]

The name is unusual and can mislead reviewers into thinking it relates to inheritance override mechanics rather than token supply storage. This is a readability issue only.[cite:1]

Recommendation: rename to `_totalSupply` for clarity.[cite:1]

Low: `transferFrom()` does not emit `Approval` on allowance reduction

The function decreases allowance unless the allowance is `type(uint256).max`, but it does not emit a new `Approval` event after updating the stored allowance.[cite:1]

This pattern exists in some tokens and is not inherently incorrect, but some tooling and indexers assume allowance changes are always mirrored by an `Approval` event.[cite:1]

Recommendation: emit `Approval(from, msg.sender, newAllowance)` after reducing allowance for better ecosystem compatibility.[cite:1]

Low: `approve()` lacks zero-address spender check

`approve()` writes the allowance without rejecting `spender == address(0)`.[cite:1]

That is generally permitted by token implementations and is not a security issue by itself, but some teams reject zero-address approvals as a hygiene measure.[cite:1]

Low: Mainnet-specific hardcoded dependencies

The contract hardcodes the BNB/USD Chainlink feed and PancakeSwap V2 router addresses for BNB mainnet.[cite:1]

That is appropriate if the deployment target is fixed, but it reduces portability and makes testnet or alternate-chain reuse less straightforward.[cite:1]

Correctness notes

The emergency withdrawal path appears correctly bounded with respect to escrow under the current implementation. Since deploy mints the reserve to the contract, `mint()` adds equal amounts to both contract balance and user escrow liability, and `claimUnlocked()` removes equal amounts from both, the contract token balance remains at reserve plus outstanding escrow before `emergencyWithdraw()` is used.[cite:1]

As a result, the expression `min(_balances[address(this)], AUTO_LP_RESERVE)` resolves to exactly the reserve amount while the emergency path is still open, and after withdrawal the contract retains the escrow backing needed for future claims.[cite:1]

The pre-LP transfer lock is also intentionally enforced. `_transfer()` reverts for all senders except the contract itself while `lpSeeded` is false, which keeps the token non-transferable until LP creation succeeds.[cite:1]

Setting `lpSeeded = true` before the external router call in `seedLp()` is acceptable because a revert from the router unwinds the entire transaction, while pre-setting the flag reduces reentrancy surface during the liquidity add flow.[cite:1]

Recommended improvements

- Add a bounded claim method such as `claimUnlocked(uint256 maxEntries)` to prevent large claim loops from becoming too expensive over time.[cite:1]
- Add a safer implementation of `pendingCount()` that returns zero if `_nextClaimIndex[u] >= _pending[u].length`.[cite:1]
- Rename `override_totalSupply` to `_totalSupply` for maintainability.[cite:1]
- Emit `Approval` on allowance decreases inside `transferFrom()` for better indexer and wallet compatibility.[cite:1]
- Consider explicit reserve tracking even if the current invariant already protects escrow, because explicit accounting improves auditability and reduces future maintenance risk. [cite:1]
- Consider stronger vote snapshot logic if the emergency vote is intended to carry stronger governance guarantees.[cite:1]

Final conclusion

The reviewed `BitcoinPizzaV2` contract does not show an obvious critical exploit in the supplied code, and the previously disputed emergency-withdrawal concern is not supported once the reserve-plus-escrow invariant is traced through constructor, mint, and claim flows. [cite:1]

The contract's current risk profile is dominated by gas-scalability, governance design, oracle liveness dependence, and code maintainability concerns rather than direct theft paths. With modest hardening changes, especially around claims and clarity, the design can be made easier to operate and defend publicly.[cite:1]