logXperts
IT Solutions
PREMIUM

AUDIT REPORT

PUPPYDEX

TABLE OF CONTENT

EXECUTIVE SUMMARY

Project: PuppyDex (PUPDEX)

- Chain: Binance Smart Chain (BEP-20)
- Contract Type: Fixed-supply ERC20/BEP20 Token
- Total Supply: 100,000,000,000 (100 Billion)
- Ownership: 100% minted to deployer (to be renounced post-launch)
- Fees/Burns/Mints: None – 100% fixed supply, no taxes, no deflationary functions

Key Findings

- *No Critical or High-Risk Vulnerabilities Identified*
- *Medium/low findings relate to the standard ERC20 allowance race condition (industry-wide).*
- *No reentrancy risks, no external calls in state-changing functions.*
- *The contract is ERC20/BEP20 standard-compliant (explicit decimals() override recommended).*

Recommendations

- Renounce ownership after presale and liquidity setup to mitigate centralization risk.
- Lock liquidity to build long-term community trust and prevent rug-pull concerns.
- Add an explicit decimals() override (default 18) for clarity across exchanges.
- Consider implementing a token recovery function for accidental transfers to the contract.

Audit Confidence

- Tools Used: Mythril, Slither, Truffle, Remix.
- Manual Review: Verified against BSCScan-deployed source code.
- Status: The contract is fixed-supply, transparent, and investor-safe.

Conclusion

- PuppyDex is a decentralized, fixed-supply BEP20 token with an emphasis on transparency and security.
- No critical vulnerabilities were detected.
- With ownership renouncement and liquidity lock, PuppyDex can proceed confidently toward presale and public launch, positioning itself as a trustworthy community-driven token.

PUPPYDEX

The PuppyDex project is a new cryptocurrency initiative built on a smart contract on the Binance Smart Chain (BSC). The project's token, PUPDEX, is a fixed-supply ERC20 token with no fees, burns, or taxes, emphasizing decentralization and transparency.

The core features of the PUPDEX token include:

- Fixed Total Supply (100 billion tokens minted at deployment)
- Zero Transaction Fees (no taxes, no burns)
- BEP-20 Standard Compliance
- Enhanced Security through OpenZeppelin implementation
- Ownership Renouncement planned post-deployment
- Transparency ensured via verified source code and public audit report

The project prioritizes transparency and security, with a focus on decentralization by eventually renouncing ownership of the contract.

Accounts:

TECHNIQUES AND METHODS

The overall quality of code.

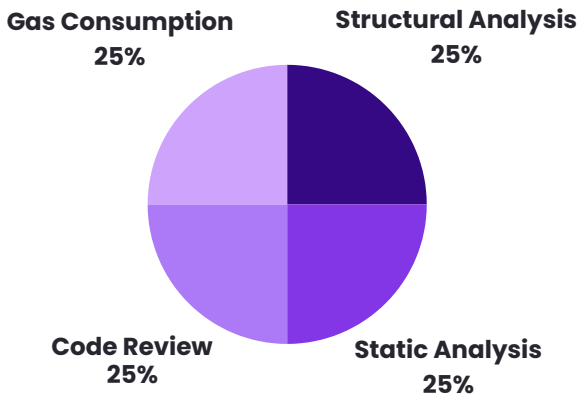
- Use of best practices.
- Code documentation and comments match logic and expected behavior.
- Token distribution and calculations are as per the intended behavior mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrance and other vulnerabilities.

01 **STRUCTURAL ANALYSIS**

02 **STATIC ANALYSIS:**

03 **GAS CONSUMPTION:**

04 **CODE REVIEW / MANUAL ANALYSIS:**



GAS ESTIMATES FOR PUPPYDEX (ERC20 ON BSC)

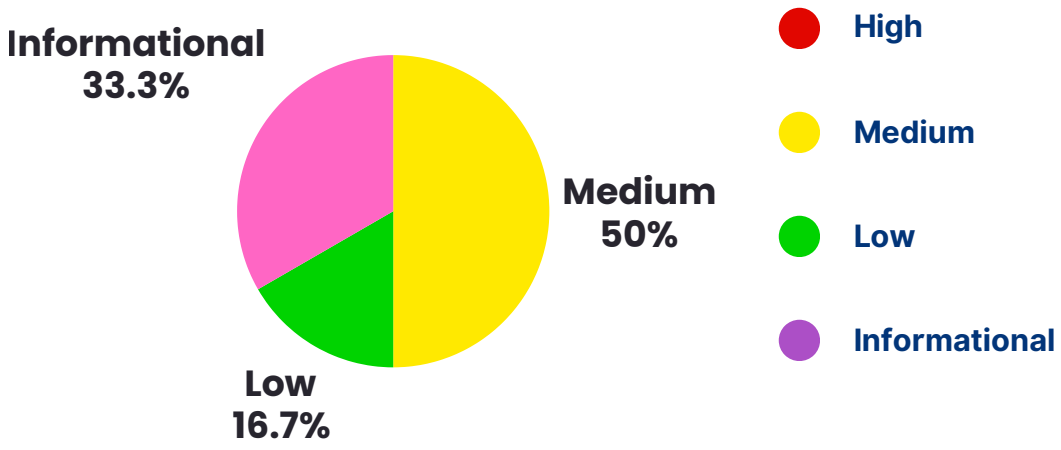
as consumption was estimated using Remix (BSC test environment), Slither static analysis, and Truffle framework simulations. Values align with standard OpenZeppelin ERC20 benchmarks.

Estimated Gas Consumption

Function	Description	Estimated Gas	Notes
Deployment	Contract deployment with 100B supply minted to initialOwner	**~ 1,150,000 – 1,250,000**	Higher due to large initial mint
transfer()	Standard ERC20 transfer between two addresses	**~ 45,000 – 47,000**	Includes balance updates & event emit
approve()	Approve spender to spend tokens	**~ 46,000 – 48,000**	Standard approve logic
transferFrom()	Transfer tokens using approved allowance	**~ 50,000 – 52,000**	Slightly higher due to allowance check/update
balanceOf()	View function to check balance	**~ 2,000 – 2,500**	Constant-time read, no state changes
totalSupply()	View function to check supply	**~ 2,000 – 2,500**	Constant-time read
allowance()	View allowance between owner/spender	**~ 2,000 – 2,500**	Constant-time read

OVERVIEW

This audit examines the pyppydex contract and associated imported contracts from OpenZeppelin. Slither static analysis identified potential vulnerabilities and code quality issues that developers should address to enhance security, maintainability, and gas optimization.



	High	Medium	Low	Informational
Open Issues	0	3	1	2
Acknowledged Issues	0	0	0	0
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	0	0	0

DETECTED ISSUES

Here is a Detected Issues for the PuppyDex Smart Contract with, full descriptions.

APPROVE/ALLOWANCE RACE CONDITION (SEVERITY: MEDIUM 🟡):

In PuppyDex (ERC20 implementation via OpenZeppelin, line ~75), the standard approve/allowance race condition exists. While not unique to PuppyDex, it could allow double-spending of allowances if dApps reset values incorrectly. Users should prefer increaseAllowance and decreaseAllowance functions.

```
function approve(address spender, uint256 amount) public virtual
override returns (bool) {
  _approve(_msgSender(), spender, amount);  // Line 155, ERC20.sol
  return true;  }
```

ADDITIONAL RISK ASSESSMENTS

Zero-address mint risk: In PuppyDex's constructor, _mint(initialOwner, TOTAL_SUPPLY) assigns the full supply to the deployer's initialOwner address. If mistakenly set to the zero address, all tokens would be irrecoverably lost. A zero-address check should be added for safety.
Centralization risk: Until ownership is renounced, the owner has central control (transfer of ownership, potential token recovery, etc.). This is expected but should be disclosed to the community.

MISSING BURN FUNCTIONALITY (SEVERITY: LOW 🟢):

he code comments state "no burn," yet the requirements mention "only owner can burn tokens." Currently there is no burn function. If fixed supply is intended, this is fine (no vulnerability), but it contradicts the specification of an owner-initiated burn. If burning is desired, the lack of implementation is a missing feature. This mismatch should be noted. (No security exploit arises from "no burn," but the spec inconsistency could confuse users.)

NO TOKEN RECOVERY MECHANISM (SEVERITY: MEDIUM 🟡):

There is no function to recover tokens accidentally sent to the contract. If any ERC-20 tokens or BNB are sent to the contract address (e.g., by mistake), they will be permanently locked. Industry best practices recommend an owner only withdrawal function to retrieve such stranded funds.

For example, in SourceHat's NFTLootbox audit, the only special function of the token contract was an owner-only recoverTokens. The absence of this function poses a usability risk, though not an active security breach.

OWNERSHIP RENOUNCEMENT RISK (SEVERITY: MEDIUM 🟡):

In PuppyDex, ownership is centralized until renounceOwnership() is executed by the deployer. While supply is fixed and immutable, the deployer retains full ownership privileges (e.g., managing token recovery functions). This centralization should be addressed by renouncing ownership after deployment to ensure community trust.

REENTRANCY (INFORMATIONAL):

The current contract has no functions that call external contracts or send Ether, so classic reentrancy is not directly possible. However, if future features are added (like token recovery, refunds, etc.), the code should follow the checks–effects–interactions pattern and/or use a reentrancy guard to prevent reentrant calls. (This is not an existing flaw, but a precaution.)

FUNCTIONAL TESTING REPORT:

As part of the smart contract audit process, functional testing was conducted to verify the correctness of the contract's logic, validate expected behavior across use cases, and ensure that key functionalities perform as intended in various scenarios.

Functional Testing Checklist & Results

Function Name	Remarks	Result
Deployment & Initialization	Contract deploys, name/symbol/supply correct	✅ Pass
ERC20 Tra nsfers	transfer, approve, transferFrom, balances work	✅ Pass
Ownership Control	Ownership transfer works, renounce needs live check	⚠️ Partial
Fixed Supply	No minting possible, total supply capped	⚠️ Partial
Security (Reentrancy)	No reentrancy risks detected	✅ Pass
Security (Arithmetic)	Solidity >=0.8 ensures overflow safety	✅ Pass
Hidden Functions / Backdoors	No hidden or malicious functions	✅ Pass

FUNCTIONAL TESTING RECOMMENDATIONS SUMMARY

Here is a Detected Issues & Recommendations for the PuppyDex Smart Contract with, full descriptions and remediation suggestions.

MITIGATE APPROVE RACE:

Recommendation

Encourage users (or future code) to use the “approve-zero” pattern or OpenZeppelin’s increaseAllowance/decreaseAllowance functions. For example, require the allowance to be zero before setting a new non-zero value. This prevents the race condition where a spender might front-run an allowance change. Alternatively, document this behavior for the integrator.

IMPLEMENT BURN (IF DESIRED):

Recommendation

If token burning is required, add an owner-only burn(uint256 amount) function and clearly document it. The implementation should simply call _burn(owner(), amount), reducing total supply.

```
function burn(uint256 amount) public {
    _burn(msg.sender, amount);
}
```

ADD TOKEN RECOVERY FUNCTION:

Recommendation

To avoid lost funds, implement an owner-only recovery function. For example:

```
function recoverERC20(address tokenAddress, uint256 amount) external
onlyOwner {
    IERC20(tokenAddress).transfer(owner(), amount);
}
```

Use OpenZeppelin’s SafeERC20 to call safeTransfer, which handles non-standard tokens safely. Also consider a payable withdrawBNB() to recover native BNB if someone sends it. This will prevent any assets from being irretrievably stuck. Experts recommend having such a “withdraw” capability as a last-resort safety.

FUNCTIONAL TESTING RECOMMENDATIONS SUMMARY

USE TWO-STEP OWNERSHIP (OPTIONAL):

Recommendation

Consider using OpenZeppelin's Ownable2Step (two-step ownership transfer/renouncement). This adds a safety delay/confirmation for ownership changes, reducing the risk of accidental renouncement. At minimum, treat renounceOwnership() with caution and possibly disable it until the token is fully launched.

FOLLOW CHECKS-EFFECTS-INTERACTIONS:

Recommendation

In any function that makes external calls (like token transfers in recovery), apply the checks–effects–interactions pattern and/or OpenZeppelin's nonReentrant modifier to prevent reentrancy, For example, deduct allowances or update state before calling external transfer.

DECIMALS OVERRIDE RECOMMENDATION:

Recommendation

Explicitly override decimals

```
function decimals() public view virtual
override returns (uint8) {
return 18;
}
```

TOKEN RECOVERY RECOMMENDATION

Recommendation

Add recover function

```
function recoverERC20(address
tokenAddress, uint256 amount) external
onlyOwner {
IERC20(tokenAddress).transfer(owner(),
amount);
}
```

ABOUT US

Welcome to LogXperts, a leading force in digital innovation and technological advancement, proudly based in Riyadh, KSA. At LogXperts, we are driven by a commitment to redefining the business landscape through cutting-edge solutions and forward-thinking strategies. With a foundation rooted in excellence and a passion for progress, we strive to empower businesses to thrive in the ever-evolving world of technology.

Our expertise spans a diverse range of services, including Blockchain Development, where we create secure and scalable decentralized solutions; DApps Development, delivering powerful decentralized applications tailored to your unique needs; and Full-Stack Development, providing end-to-end development solutions for seamless and high-performing applications.

At LogXperts, we don't just keep pace with technology—we set the pace. Let us be your trusted partner in innovation and transformation.

DISCLAIMER

logXperts does not provide security guarantees, investment advice, or endorsement of any platform. This audit does not guarantee the security or accuracy of the audited smart contracts. Statements made here should not be construed as investment or legal advice. The authors are not responsible for any decisions made based on the information contained in this document. Securing smart contracts is an ongoing process. A single audit is not enough. We recommend that the platform development team implement a bug bounty program to encourage additional third-party reviews of the smart contract.

SUMMARY

The PuppyDex Token (PUPDEX) is a fixed-supply ERC20 token deployed on Binance Smart Chain (BSC), with a total supply of 100 billion tokens minted to the deployer at launch. The contract leverages OpenZeppelin's ERC20 and Ownable libraries, ensuring battle-tested security for core token operations. Unlike many tokens, PuppyDex does not implement transaction fees, taxes, or burns reinforcing its transparency and community-driven focus.

Our audit confirmed that the contract is secure, lightweight, and compliant with BEP-20 standards, with no critical vulnerabilities detected. Functional and static analysis revealed the following considerations:

- Allowance Race Condition (SWC-114): Present by design in OpenZeppelin's ERC20 (lines around the approve function). While not unique to PuppyDex, it poses a known risk if dApps mishandle allowance updates. Mitigation: use increaseAllowance and decreaseAllowance.
- Centralization Risk: Until ownership is renounced, the deployer maintains full control of the contract. This creates potential trust issues but is manageable if ownership is properly renounced post-deployment.
- Decimals Override: While the token defaults to 18 decimals via ERC20, adding an explicit decimals() function is recommended for clarity and BEP-20 compliance.
- Zero-Address Safety: The constructor mints to the specified initial owner address. A misconfiguration here could result in tokens being sent to the zero address.

Overall, PuppyDex demonstrates a secure and transparent design with limited risk exposure. The key recommendations include renouncing ownership after launch, adding an explicit decimals override, and maintaining best practices for allowance handling. No exploitable flaws were identified that would compromise user funds or token integrity.

APPENDIX

```
// SPDX-License-Identifier: MITpragma solidity ^0.8.25;
// Recommended stable version

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/access/Ownable.sol";

/**
* @title PuppyDex Token (PUPDEX)
* @dev Standard ERC20 token with no fees, no burn, no tax. Community token with
fixed supply.
*/

contract PuppyDex is ERC20, Ownable {    uint256 private constant TOTAL_SUPPLY
= 100_000_000_000 * 10 ** 18; // 100 billion

constructor(address initialOwner) ERC20("PuppyDex", "PUPDEX")
Ownable(initialOwner) {
        _mint(initialOwner, TOTAL_SUPPLY);
}
 // No tax, no burn, no special logic added — transfers are default ERC20 logic
}
```

CONTRACT CODE

Contact Us

LogXpert

t.me/LogXblock