Preliminary Notes: Doxed

Brewlabs Services Hub

Summary: Doxed is an BEP20 token smart contract for the Doxed project. It is written in Solidity and compiled using build 0.8.11. The smart contract supports regenerative tokenomics.
Audit package: Standard security, with logic check
Deployment Status: Deployed to BSC Mainnet
Files Audited: Single file deployment on Mainnet
Audit Result: **Security PASSED, logic FAILED**

Disclaimer: *This audit documentation is for discussion purposes only. The scope of this audit was to analyze and document Doxed smart contract codebase for quality, security, and correctness. This audit guarantees that your code has been revised by an expert.*

Overview of Audit
No critical security issues identified.
0 critical logic issue identified.
6 major security issue identified.
3 major logic issue identified.
Several minor and informational issues identified across security and logic.

Security issues

Major level security issues detected:

The following issues violate the ethics of Decentralised Finance and present a centralisation risk to the community. To mitigate these risks, ensure the Development team is doxxed, improve privileges and roles within the team to help decentralise, or consider (Know Your Customer) KYC services.

1. Contract allows owners to enable/disable liquidity deposits (swapEnabled) which can impact those adding to the liquidity pool
2. Contract allows owners to alter transactions fees.
3. Contract allows owners to exclude wallets (potentially permanently) from rewards, and to do so with no visibility to public (no event emitted).
4. No upper limit when setting fee state variables leading to potential risk if contract owner sets the fee to 100%. Recommendation is to embed an upper limit or communicate to community when relevant changes are committed to improve transparency.
5. Liquidity pool tokens generated by the contract are sent to the owner address, which means a significant portion of tokens will be held by the owner over time. This poses a centralisation risk for the project.
6. Contract allows owner to withdraw BNB from contract at anytime, and to do so with no visibility to public (no event emitted).

Minor level security issues detected:

1. Missing events for changes to state variables in various functions (eg. Line 288, 297, 311, 315, 599, 605, 611 etc). This means only the contract owner can alter state variables without the community being aware of changes. Recommendation is to add events for all changes to state variables or communicate to community when relevant changes are committed to improve transparency.
2. Lack of input validation on fee changes which could lead to centralisation risk (for example, if the dev = 100) or loss in liquidity pool deposits (for example, if the amount is set to zero).
3. Lack of input validation for wallet changes (eg. Line 599, 605, 611) which could lead to unexpected loss if an invalid address is set, for example, address(0).

Informational issues detected:

1. Floating pragma set. If relying on byte-code level verification, best practice is to lock the compiler version prior to deployment.

2. Third party dependency on Pancake Swap protocol (line 568, 588). Recommendation is for the Dev team to monitor updates made to third party contracts to ensure changes to not impact the receiving contract or are mitigated/managed accordingly.

Logic issues:

Major level logic issues detected:

1. Reflected burn amount (rBurn) is based on the reflected burn amount and not transferred burn amount (tBurn) (line 423). This impacts the rTransferAmount which is used to determine the reflected amount for the holder. Recommendation is for the Development team to confirm with contract developer.
2. Mismatch in the order of return values declared in the _getRValues() function, the values returned by the function, and the implementation on line 392. This means the incorrect fees are being applied. For example, the _getRValues() declares the variables return in the following order:

   rRfi, rDev, rLiquidity, rBurn, rMarketing, rGiveback

   In the function, variables are returned in the following order

   rRfi, rDev, rLiquidity, rBurn, rGiveback, rMarketing

   In the implementation on line 392, variables are returned in the following order

   rMarketing, rGiveback, rRfi, rDev, rLiquidity, rBurn

   The result of this mapping inconsistency means the fees for each entity are not being applied as intended. In its current form, the fees are applied as so:

   - Marketing wallet is receiving the holders fee
   - Giveback wallet is receiving the dev fee
   - Holders are receiving the liquidity fee
   - Dev wallet is receiving the burn fee (which is suspected to be zero based on the first issue)
   - Liquidity wallet is receiving the giveback fee
   - Burn address is receiving the marketing fee

   Recommendation is for the Development team to confirm with contract developer.
3. Calculation used to determine values of rTransferAmount (Line 426) does not account for rMarketing or rGiveback fees which means the reflectedAmount of holders is greater than intended. Recommendation is for the Development team to confirm with contract developer.

Minor level logic issues detected:

1. Transfer event emitted on line 515 for the Marketing wallet transfer refers to the deadAddress as the recipient, when its contract address. Transfer event could be removed and reduced to single event on 523.
2. Transfer event emitted on line 519 for the Giveaway wallet transfer refers to the deadAddress as the recipient, when its contract address. Transfer event could be removed and reduced to single event on 523
3. Transfer event emitted on line 523 does not reflect the true amount sent to the contract address and should be updated to include the tMarketing and tGiveaway amounts.

Code Style:

1. Layout of the file does not follow Solidity guidelines.
2. Low use of comments. Comments help to improve readability and is encouraged.
3. Opportunity to improve use of camelCase style across contract (eg. Line 599, 605, 611)

Conclusion: The Brewlabs team thank you for the opportunity to review and audit your smart contact code. The data from this report will be formalised in the audit publication for your community. Keep in touch as we offer discounts for repeat business and on a range of other services!
The Brewlabs Team.