Lite Paper

DEVIL TOKEN & ECOSYSTEM

Disclamer

This document and its contents are not intended implicitly or explicitly to provide for financial or legal advice. Consult with your financial advisor and/or attorney for any financial or legal questions. Cryptocurrencies are fiscally volatile and high-risk. By acquiring or disposing of any cryptocurrency, token, or digital asset you are doing so on your own accord and are responsible for complying with all the laws and regulations of any jurisdiction in which you reside or that pertains to the possession or custody of your digital currencies and/or assets.

Project Links & Contact

Website
devilcrypto.io

Telegram
twitter.com/DevilTheCrypto

Instagram
instagram.com/devilthecrypto/

Telegram
https://t.me/DevilTheCrypto

Reddit
https://www.reddit.com/user/DevilTheCrypto

Github
github.com/DevilTheCrypto/

Certifications*

KYC Certificatied from InterFi

*Audit Pending

INTRODUCTION

I. What is Devil (DEVL)?

DEVL is a reward and utility token deployed on the Binance Smart Chain Network (BEP-20). The token pairs proven incentive mechanisms with robust economics, including a fee-on-transfer structure that creates a sustainable revenue source for it's decentralized applications (dApps). The multiple reward mechanisms exist to encourage holders to maintain and accumulate their balances, contributing to the growth of the ecosystem.

II. What is the Devil Portal?

The Devil Portal is home to DEVL's series of dApps. The fuel to the fire of the Devil ecosystem, holders of DEVL will have the keys to a suite of decentralized finance (DeFi) products which aim to help grow and manage wealth, including bridging tokens cross-chain, fiat-to-crypto onramps, and a wallet with personal usages statistics and exportable transaction history.

At launch, the Portal will include a dApp with pricing statistics and the ability to swap DEVL/BNB without leaving the dApp, a single-staking yield farm rewarding users in BUSD, and a governance dApp which controls the DEVL contract's permissions.

III. Why Devil?

Like all cryptocurrency tokens, to own DEVL is to contribute towards and benefit from the growth of the token's market capitalization. Unlike many tokens however, DEVL is launching with fully developed v1 dApps, designed to kick-start the token's use, economics, and utility.

From the very beginning, holders of DEVL will have the ability to contribute to the token's price stability, access the Pancake Swap decentralized exchange in-dApp, and have a say in any adjustments the development team would like to make.

The development team is not seeking to assume trust, but to earn it. By releasing at-launch, the team is hoping the community will see it as a sign of commitment and dedication to growth.

1. THE DEVIL TOKEN

1.1 About the Token (DEVL)

Name: Devil Token

Symbol: DEVL

Total Supply: 666,666,666 DEVL

Presale: 329,999,940 DEVL

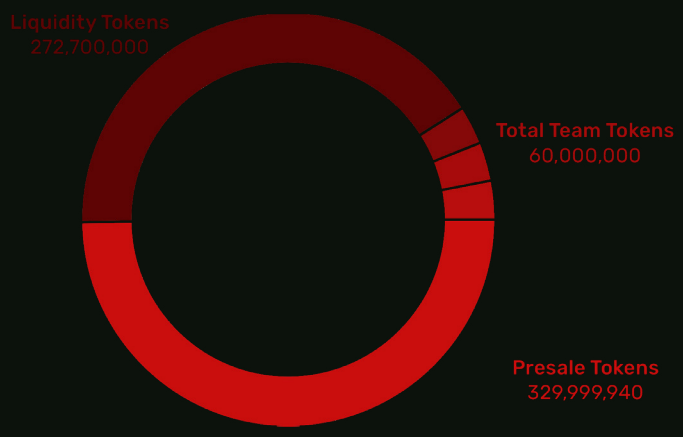
Liquidity: 272,700,000 DEVL

Marketing: 20,000,000 DEVL (3%)

Business Operations: 20,000,000 DEVL (3%)

Development: 20,000,000 DEVL (3%)

Total Team: 60,000,000 (9%)



DEVL currently has three fee-on-transfer functions: a fee that goes into liquidity within the Pancsake Swap v2 liquidity pool, decreasing volatility and price swings; a fee that is reflected to holders; and a fee that funds the Devil's Vault dApp.

To ensure the project is forward thinking and future-proof, DEVL is a widely adjustable contract, with everything from the fee rates being adjustable, addresses where funds are set, to a toggling on or off of any of the incentive mechanisms or fee functions. For this reason, the contract is locked by a governance dApp, which requires a vote from token holders to trigger an unlocking of the contract, allowing holders to decide when they would trust the contract's custodians (development team) to make adjustments to the contract if needed.

1.2 Tokenomics

DeFi may be one of the greatest advancements in financial technology (FinTech) since the advent of blockchain. With endless capabilities, particularly in market equalization, it is no

surprise that the DeFi space has exploded, particularly over the last few years.

Ultimately, this large increase in projects and users can be seen as a good for the space, it is not without its downsides including a high amount of poor quality or outright scam/theft oriented projects. Paired with a great amount of competition amongst valid projects, it is apparent that new token projects in this climate must match the benefits of its competitors - both legitimate and illegitimate - while providing a novel and useful product.

DEVL's approach as a DeFi product has been crafted to address this reality of the market. The token has been designed to incorporate existing features that have been proven to work as incentive mechanisms for holders, while maintaining adaptability and modularity so that it can grow into its proposed ecosystem and beyond. For this reason, DEVL takes two primary incentive mechanisms from popular recent DeFi projects, with changes in place to lay the foundation for future functions.

Automated Liquidity is one of those features. Liquidity is one of the single greatest challenges in DeFi, behind access and adoption. Without the presence of sentient market makers, people or organizations who provide capital or an asset to set with the token in a marketplace, DeFi's automated market makers (such aor AMMs, such as Pancakeswap) rely on the token's team or general public to deposit liquidity into the marketplace. While there are some incentives to do this, such as a share of transaction fees, the rewards are generally less than can be acquired from other utilizations of cryptocurrencies or assets. There can also arise detrimental impacts on price, should a person with a large amount of liquidity pull out of that pool. Higher liquidity provides for a more stable price and more importantly, allows the market to exist.

For this reason, DEVL charges a fee (3% at launch), which is taken from all transactions and activated after certain metrics have been hit, half the token balance from this fee is swapped into BNB using the core AMM, and then deposited with the other half of the balance. This means that roughly 3% of every transaction is added to liquidity, making the marketplace more stable and self-sufficient based on the volume of the token's trading.

Token Reflection is the second common mechanism that DEVL adopts, due to its track-record

in the DeFi space. Token reflection, also referred to as “frictionless, static, reflection rewards” accrue naturally simply by holding tokens in your wallet. In the case of DEVL, a fee is taken out of every transaction and is immediately applied to this method. This incentive mechanism, while now widely used, is complex compared to other mechanisms.

Frictionless yield generation emerged around 2020 as a technological development beyond manual yield-farming, or the process of depositing tokens into a dApp to earn a share of rewards, or similar reward structures such as airdrops delivered to token holders. It is widely adopted as its approach is currently unparalleled in the DeFi space for its minute impact on transaction costs while rewarding those who hold tokens against those who sell the tokens.

Devil’s Vault is an incentive mechanism unique to DEVL. For this function, a fee is charged on transfers, which after certain metrics are achieved is then swapped into BNB and sent out of the contract to a second contract, which then takes that BNB and swaps it for the stablecoin BUSD. For efficiency and to save on gas/transaction costs, this mechanism is built into the Automated Liquidity function. The liquidity fee and vault fees are mushed together into one amount, divided up proportionally to their use, and an amount of DEVL tokens is swapped for BNB which is then subsequently used for both the liquidity deposit and sent out of the contract to be swapped into BUSD.

The BUSD can be obtained by DEVL holders through depositing their tokens in the Devil’s Vault yield farm dApp, which rewards BUSD generated through this fee in direct proportion to teach user’s amount of DEVL deposited in the dApp. This function was designed as a hedge against price downturns, and to furth incentives the holding of tokens thus rewarding those who use their tokens to contribute to price stability. It should be noted that while 3% of each transaction being turned into BUSD during a price increase technically removes that 3% from participating in the price increase, the BUSD generated by the fee can be worth more if the price of DEVL drops below its price at that time of transfer.

Through this mechanism, users are given the option to diversify their holdings effortlessly and are rewarded with a token that can be used to either reinvest in DEVL, withdraw for fiat profits, or be contributed to other DeFi products. This mechanism is the first example of DEVL’s focus on facing market realities, providing financial tools through DeFi, and creating a token that has a use and value through all market climates.

2. THE DEVIL PORTAL

2.1 About the Devil Portal

The Devil Portal is the primary hub of the DEVL Ecosystem of dApps. Through this one superseding dApp, all the other dApps can be accessed. At launch, the portal includes the Gateway, Vault, and Lock dApps, which allow users to: purchase DEVL with BNB and see basic price statistics and personal balances; deposit tokens to participate in the BUSD yield farming mechanism; and vote to allow the DEVL team to lock and unlock the contract to make changes to the DEVL's function parameters.

The only requirement to use the portal at this time is to have a browser wallet installed and enabled on your device.

Subsequent versions of the portal may also require the possession of DEVL tokens in your wallet, but that is not a necessity at launch, however, almost all the functions of the dApps do require DEVL to utilize.

2.2 Devil's Gateway

One of the primary goals of DEVL is to increase accessibility to the world of DeFi and through that, bring new users into the DeFi space by lessening the learning curve. One of the principal ways this will be achieved will be with the Devil's Gateway. At launch, the Gateway serves as a place to purchase DEVL with BNB and have access to basic price statistics. In subsequent releases, the Gateway will serve as a means to on and offramp fiat currencies into and out of DEVL. The vision of the Devil's Gateway is to provide a place where all a user would need to acquire DEVL is a cryptocurrency wallet and either BNB or a fiat credit card, with minimal roadblocks.

Features of the Gateway v1

In Gateway v1, users will be able to see their connected wallet's DEVL and BNB balances, as well as the current price in DEVL relative to BNB and estimates of those prices relative to the USD.

Users have the option of inputting the the amount of BNB they wish to use to buy DEVL and then pressing "Buy DEVL" in order to make the purchase through the tokens original market-place, Pancake Swap.

To sell DEVL back into BNB, users can input the amount of DEVL they wish to sell, click "Sell DEVL", and that amount will be exchanged through Pancake Swap returning an amount of BNB to the user's account.

2.3 Devil's Vault

The Vault facilitates the Devil's Vault feature of the DEVL token contract, as described in the Devil Token - Devil's Vault section of this document. It allows holders of DEVL to deposit their DEVL tokens in order to have their share of the BUSD rewards generated by the DEVL contract.

Features of the Vault v1

In the Vault, users can input the amount of DEVL tokens they wish to deposit or withdraw into the vault. The vault also gives the connect account's balance of DEVL tokens above the input form to help indicate how many tokens can be deposited. This can be done by inputting the de-sired amount of DEVL tokens into the input form and selecting either "Deposit" or "Withdraw".

Users can also claim their awarded BUSD tokens by selecting “Claim”. Claiming tokens transfers the entire balance of tokens currently awarded to the user into the users wallet, there is no need or ability to withdraw a specific amount of BUSD tokens.

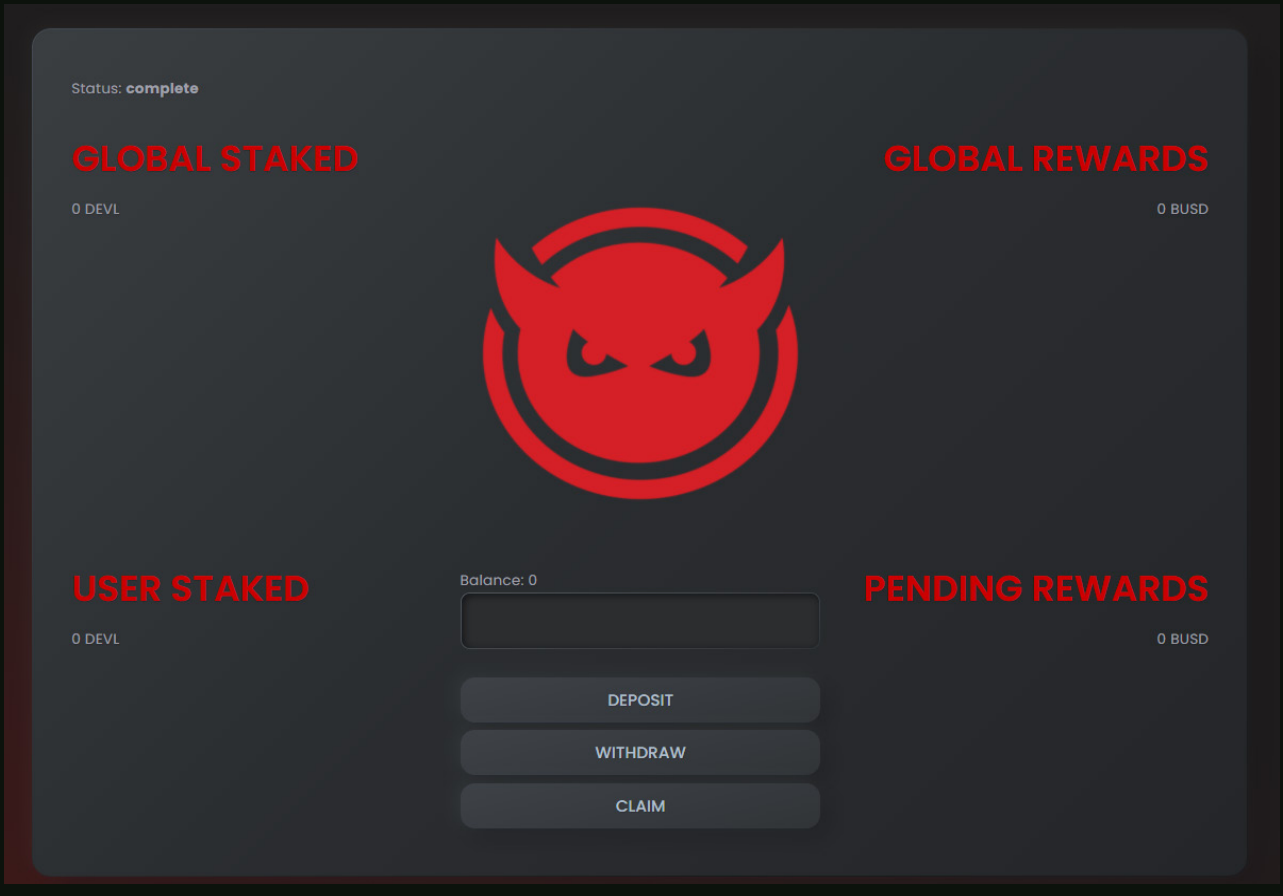
Description of User Interface

Global Staked - The total amount of DEVL deposited into the vault by all users

Global Rewards - The total amount of BUSD rewarded by the vault to all users throughout its lifetime

User Staked - The amount of DEVL tokens the individual user (connected account) currently has deposited into the Vault

Pending Rewards - The amount of BUSD allotted to the individual user (connected account) that has yet been claimed by that user. Upon claiming, the balance is set to 0



2.4 Devil Lock

Devil Lock is the dApp that interfaces with DEVL's governance contract. In this first version of the Devil's planned governance structure, DEVL holders are given the ability to vote on requests to lock or unlock the DEVL token's contract by the development team, so that they can change various parameters of the smart contracts.

Due to DEVL's modular design as well as its planned upgrades and implementations, there will be times where the development team needs to lock and unlock the token's contract in order to finalize those upgrades. With that requirement comes an acknowledged risk to token holders, by adjusting various parameters, the development team could perform nefarious or hostile acts, such as changing address where fees are sent to. For this reason, the development team will always focus on building trust with the community by tying their hands to perform those changes only with the voluntary permission of its holders.

While Devil Lock v1 only pertains to the locking and unlocking of the DEVL contract, its subsequent versions will include a far more stringent governance structure that only allows for the functions of DEVL and all its related dApps to have parameters changed with the authorization of the community, which also includes pre-approving the proposed parameters to change.

For example in the proposed v2 version of Devil Lock, the development team would like to change the fee taken to fund the Devil's Vault from its launch value of 3% to 4%. In order to make that change, the team would have to request that an election be opened to earn an allowance to call that change of parameter from 3 to 4%. The 4% would be hardcoded into the proposal that users vote for, thus eliminating any need for trust of the development team.

Should the vote be successful, the development team would be able to call the function to change the vault fee only once, and the governance contract itself would input the 4% value.

Features of Lock v1

In Lock v1, the amount of votes per user is 1:1 with DEVL tokens in their possession. There is no need to manually input the amount of votes you wish to cast, your voting power is automatically calculated by the smart contract and deducted by the smart contract once you have already voted. You cannot vote for the same proposal twice.

To vote or oppose a current proposal, users simply need to press “Support” to allow cast their yes votes, or press “Oppose” to reject the proposal. The rest of the work is done by the smart contract itself.

Description of User Interface

Contract/Election Status: Indicates whether the contract is currently locked or unlocked and if there is an active election. (An unlocked contract means the development team has the ability to change any parameters, locked means a vote must take place to allow contract changes to be made).

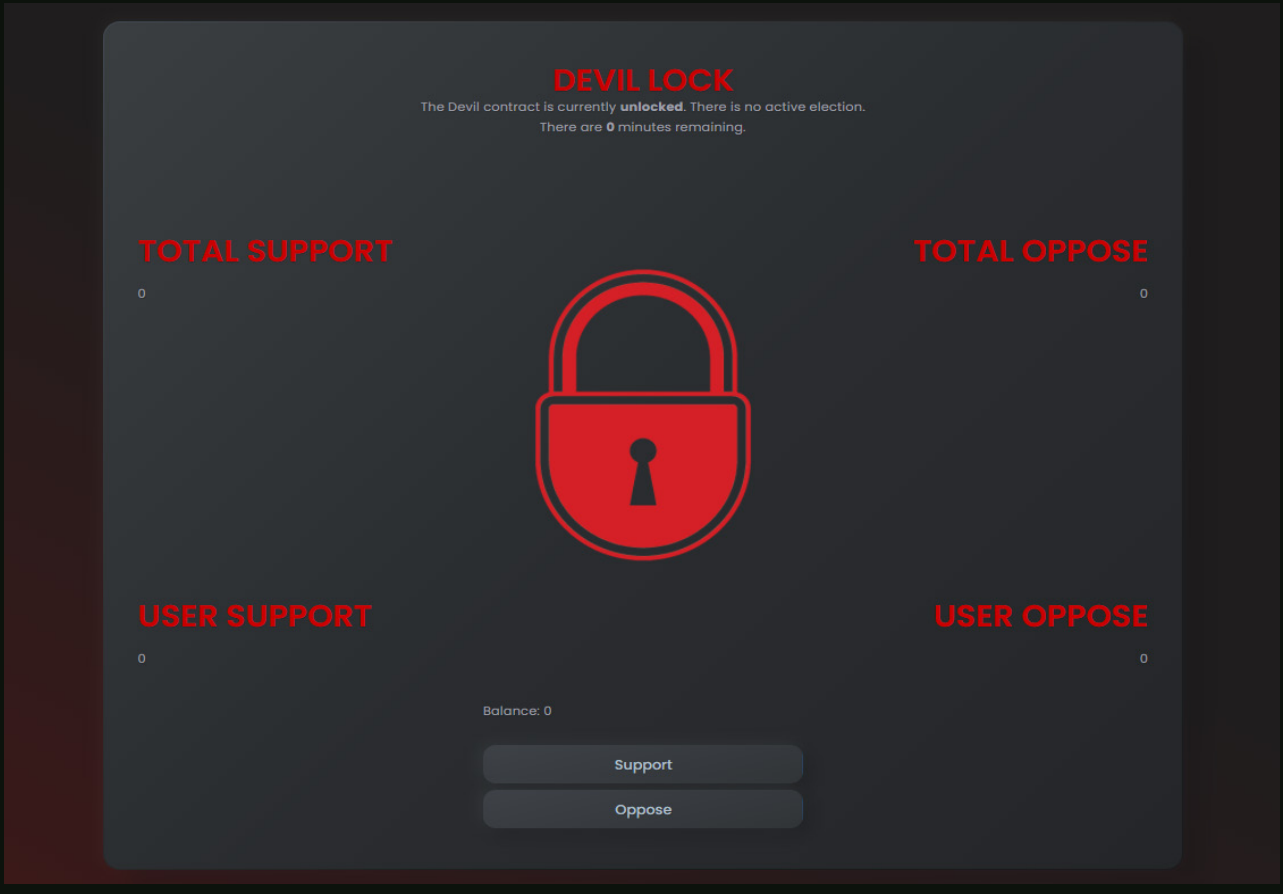
Election Time: By default 0, lists the amount of minutes remaining for users to vote if there is a current election.

Total Support: The total votes in support of the current proposal among all participating users

Total Oppose: The total votes in opposition to the current proposal among all participating users

User Support: The amount of votes in support cast by the current user (connected account)

User Oppose: The amount of votes in opposition cast by the current user (connected account)



3. Technical Details

3.1 The Devil Token

The Devil Token utilizes existing codebases, libraries, and new code most often in a hybrid format.

Pragma:

Solidity 0.7.6

Dependencies:

IERC20 Interface - Open Zeppelin

SafeMath.sol - Open Zeppelin

Context (abstract)

Address (library)

Ownable - Open Zeppelin

IUniswapV2Factory (interface) - Uniswap SDK

IUniswapV2Pair (interface) - Uniswap SDK

IUniswapV2Router01 (inteface) - Uniswap SDK

IUniswapV2Router02 (interface) - Uniswap SDK

Functions:

The following is a list of primary functions within the contract with their input type and values written in (input type input value) format. For a full list of functions, or to read the contract yourself, visit the verified contract on BSC Scan:

Function Name	*Input Parameters*	*Permissions*	*Description*
owner	*()*	*Public View Returns (address)*	*Shows the contract owner*
transferOwnership	*(address newOwner)*	*Public Virtual Owner*	*Transfers contract ownership*
geUnlockTime	*()*	*Public View Returns (uint256)*	*Provides the amount of time before the contract unlocks, if locked*
lock	*(uint256 time)*	*Public Owner*	*Locks the contract for an amount of time in seconds*
unlock	*()*	*Public*	*Unlocks the contract if the lock time has elapsed.*
isExcludedFromReward	*(address account)*	*Public View Returns (bool)*	*Provides bool if an address is excluded from rewards*
totalFees	*()*	*Public View Returns (uint256)*	*Lists total active fees levied by contract, in whole uint256 representing percentage of fees*
reflectionFromToken	*(uint256 tAmount, bool deductTransferFee)*	*Public View Returns (uint256)*	*How many rewards generated by amount of tokens in supply*
tokenFromReflection	*(uint256 rAmount)*	*Public View Returns (uint256)*	*Amount of tokens generated from reflection*
excludeFromFee	*(address account)*	*External Owner*	*Excludes an address from fees*
excludeFromReward	*(address account)*	*External Owner*	*Excludes an address from rewards*
includeInFee	*(address account)*	*External Owner*	*Includes an address in fees*

includeInReward	*(address account)*	*External Owner*	*Includes an address in rewards*
setcommunityFundWallet	(address newWallet)	External Owner	Sets destination of a fee not set at launch
setDevFeePercent	(uint256 devFee)	External Owner	A tax that goes to the communityFundWallet, set 0 at launch
setLiquidityFeePercent	(uint256 liquidityFee)	External Owner	Sets the tax that goes to the locked liquidity AND devils vault feature
setMaxTxPercent	(uint256 maxTxPercent)	External Owner	
setNumTokensSellToAddToLiquidity	(uint256 _newNumTokens)	External Owner	Sets the minimum token contract balance that would trigger the swap and liquify and devils vault feature
setRouterAddress	(address newRouter)	External Owner	Allows pancake swap (uniswap in code) router to be changed if necessary
setVaultAddress	(address payable _vaultAddress)	Public Owner	Sets the address where the swapped BNB will go to fund the Devils Vault Feature

Description Key Functions

Ownership Features

The Devil Token contract has the codebase from OpenZepplin's ownable.sol contract in order to provide a better way for Devil Lock to interact with the Devil Token contract.

Devil Lock, described in detail later in this document, is set as the owner of the Devil Token contract and uses an interface of the token contract to call functions such as lock, transfer ownership, and unlock.

It should be noted that the unlock feature has been left as publicly callable, this has to do with access and future Devil Lock features. This does not pose a security risk, as none of the high-risk features are callable by anyone other than the owner using the onlyOwner modifier.

Token Reflection

Devil Token uses the proven incentive mechanism of token reflection, popularized by Safemoon and RFI. Token reflection (or frictionless yield generation) was developed to solve key issues with rewarding holders: gas fees and gas limits from distributing tokens from an external wallet to holder wallets, the difficulty of computing all holders on blockchain given gas limits, security concerns regarding approve functions, and more.

Since it's advent, this feature has been particularly successful at encouraging accumulation and retainment behavior, with the necessary tax levied as acting as a disincentive to selling.

Put simply, token reflection works by taking a fee of transfers and distributing that amount to all holders.

This works through a complex process that includes keeping two balances, a traditional token supply (tSupply) and a reflected token balance (rSupply). These balances exist so that the ratio of the total supply and reflection supply can be used to calculate the amount of tokens allocated to an individual account based on their share of the tokens total supply without any transfers or iterations. Two supply balances are kept so that the actual supply is not variable, which can cause a considerable amount of issues in dealing with other contracts and dApps.

Devil's Vault & Locked Liquidity

The Devils Vault (or vault) is a feature unique to DEVL. Please see the subsequent section on Devil's Vault for more details. The Devil Token contract provides the funding mechanism for the Devil's Vault feature. This has been designed in a way to minimize the feature's impact on gas fees and complexity.

Within the Devil Token contract, an amount of Devil is turned into BNB and sent outside of the contract to then be swapped into BUSD. This modular approach reinforces security, upgradability, and reduces the potential gas fees that could be added if handled entirely within the token contract.

This feature is buried within the Swap and Liquify feature, which takes an amount of DEVL and swaps it for BNB and then deposits equal parts BNB/DEVL into the token's original liquidity pool on Pancake Swap.

Swap and Liquify (swapAndLiquify) works by first taking the balance of DEVL tokens in the contract from collected fees. It divides half of the DEVL for the Devil's Vault, one quarter for half of the liquidity deposit that will be turned into BNB, then the final quarter for the amount of DEVL that will be paired with the swapped amount of BNB for the liquidity deposit.

The total amount of DEVL to swap for BNB (listed as for Eth in contract), both for vault and liquidity, are temporarily added together. An initial balance is tracked, to prevent math errors, then the amount of tokens to swap for BNB is swapped through the Pancake Swap router.

The balance of tokens swapped to BNB is then calculated and split up, with half going to the vault and the other half going to liquidity paired with DEVL.

3.2 The Devil Portal

The Devil Portal is the primary dApps that houses the other dApps. It's responsible for managing the account credentials, setting state variables that pertain to the web3 connection, and providing the UI framework for the other dApps. The following information pertains to the portal and its component dApps: Gateway, Vault, and Lock.

Languages:
Javascript
HTML
CSS
ReactJs (Framework)
Typescript

Methods:

React Hooks

Key Dependencies:

useWalletModal

useAuth (from Devil Portal ../hooks)

PancakeSwap-libs/uikit

3.3 DEVIL GATEWAY

Devil's Gateway (or "Gateway") is the first component loaded into the Devil Portal. It provides users with the ability of purchasing BSC BNB (BEP-20) with fiat currencies through a 3rd party aggregator, onramper. It also allows users to buy DEVL with BNB and sell DEVL for BNB using an interface wired to the Pancake Swap router.

Pragma:

Solidity 0.7.6

Dependencies:

SafeERC20 - OpenZepplin

IUniswapV2Factory (interface) - Uniswap SDK

IUniswapV2Pair (interface) - Uniswap SDK

IUniswapV2Router01 (inteface) - Uniswap SDK

IUniswapV2Router02 (interface) - Uniswap SDK

Functions:

The following is a list of primary functions within the contract with their input type and values written in (input type input value) format.

For a full list of functions, or to read the contract yourself, visit the verified contract on BSC Scan:

Function Name	Input Paramters	Permissions	Description
getQuoteToken	()	Public View Returns (address)	Shows the contract address of the token being used to trade with BNB
setQuoteToken	(address quoteToken)	External Owner	Sets the contract address of the token being used to trade with BNB
buyDevl	()	Public Payable	Allows contract to receive BNB, swaps BNB for DEVL using pancake swap router
sellDevl	(uint256 _ amount)	Public	Users can call to swap a certain amount of DEVL for BNB using pancakeswap router

Information on the Fiat On Ramp:

Users can purchase and sell BNB using the Gateway's integration with the 3rd party application Onramper. Onramper is a fiat gateway aggregator which selects the best fiat gateway option for each user based on the fiat they are using and the current BNB price. KYC may be required by the end point gateway that is connected through onramper, however, the entire process takes place within the widget itself.

3.4 THE DEVIL'S VAULT

Devil Vault is a collection of two smart contracts, Devils Vault contract and VaultSwapContract, and a reactjs component written in javascript and html. It allows holders to deposit DEVL tokens for their share of BUSD staking rewards.

DevilVault.sol

This contract handles the staking balances, rewards balances, and distributing rewards to stakers.

Pragma:

Solidity 0.7.6

Devil's Vault Contract Dependencies:

IERC20 Interface - Open Zeppelin

IBEP20

SafeBEP20

SafeERC20

ReentrancyGuard

SafeMath.sol - Open Zeppelin

Context (abstract)

Address (library)

Pausable

Ownable - Open Zeppelin

Functions:

The following is a list of primary functions within the contract with their input type and values written in (input type input value) format. For a full list of functions, or to read the contract yourself, visit the verified contract on BSC Scan:

Function Name	Input Paramters	Permissions	Description
setBaseToken	(address _stakingToken)	External Owner	Sets the token used for staking
setRewardToken	(address _rewardToken)	External Owner	Sets the token used for rewards

addUser	(address _address)	Internal	Adds user to the stakers array and tracks place in array if they're not in it
removeUser	(address _address)	Internal	If user removes entirety of staking token, they are moved to the end of the stakers array then deleted along with their place in the staking map.
getUserStakingBalance	()	Public View Returns (uint)	Returns DEVL staking balance of address who sends the request
getUserPendingRewardBalance	(address _address)	Public View Returns (uint256)	Returns reward balance of address who sends the request
getNumberOfActiveHolders	()	Public View Returns (uint)	Returns the number of stakers currently staking (length of staker aray)
getRewardTokenBalanceOfContract	()	Public View Returns (uint256)	Returns current balance of reward tokens in contract
getAllocatedRewards	()	Public View Returns (uint256)	Returns amount of reward tokens already distributed to holders but not yet claimed.
getGlobalStakingTokenBalance	()	Public View Returns (uint256)	Gets the number of total DEVL tokens staking in contract by all stakers
getStakerIndezx	(address _address)	Public View Returns (uint)	Returns the place in the array that a staker is held in, for testing and the bump mechanism to remove holders after entire DEVL amount withdrawn from contract.
stake	(uint256 _amount)	Public	Allows users to stake DEVL tokens, updates global and user balances, calls addUser if needed. Sets the isStaking bool.

withdraw	(uint256 _ amount)	Public	Transfers staking tokens back to user in requested amount, removes them from array and sets isStaking to false if staking balance = 0.
claim	()	Public	Transfers number of reward tokens allocated to the user, updates balances, removes amount from allocatedRewards state variable.
distributeReward	()	External	Distributes the reward tokens in contract to holders proportional to their staking share. Uses array iteration.

Description Key Functions (DevilVault.sol)

Adding/Removing Users (addUser & removeUser functions)

Stakers are tracked through an array to make iteration possible. Because there is no minting mechanism and it would be inefficient to automatically transfer reward tokens to holders, the Devil's Vault contract has to keep track of the total balance of reward tokens while allocating a portion of that balance to holders.

For these reasons, a combination of array and mapping is used.

Stakers' addresses are added to the staking map then added to the array. Their place in the array is tracked via the mapping as a uint (stakerIndex), making the removal process possible.

To save on gas through iterations, removeUser is called if the staking balance goes to 0 as a result of a withdrawal of staking tokens. The function removes the user by moving them to the

end of the array through nested mapping/reassigning their staker index to the end, then using pop to remove their address.

Tracking Staking/Reward Token Balances

Staking tokens are tracked through several mappings of address to value and using state variables for global values. Staking Tokens are only tracked using the globalStakingTokenBalance, or the entirety of staking tokens staked by all users, and the amountStaked of each individual user.

Reward tokens are tracked using several state and local variables: currentReward, captures the contract's balance of reward tokens; unallocatedReward, is the amount of tokens in the contract that have not yet been allocated/assigned to users; allocatedReward, the amount of reward tokens assigned to individual users and thus cannot be claimed by anyone other than the user they're assigned to.

Claiming reward tokens removes them from the allocated rewards balance to avoid mathematical errors.

Distributing Rewards

This function is called by the VaultSwapContract. It works by first capturing the number of reward tokens in the contract, subtracting the amount of tokens that have already been allocated, then calculating how many reward tokens the user should get based on the amount of tokens they have staked.

Because it only uses fixed-point numbers, otherwise known as not being able to use decimals, the userRewardPerToken feature takes the globalStakingBalance and turns it into a multiplier (remove wei value) and divides the available rewards by it to find rewards in wei per token staked.

This number is then used in the for loop/iteration through the stakers array which calculates the amount of rewards token to be allocated to the user based on their share of the globalStakingTokenBalance then updates the rewards they have available to claim (rewardsPending mapping).

Once the loop is completed, the number of reward tokens that were just distributed to stakers is added to the amount of allocated rewards, so that the next time the distribute function is called, no one token is assigned to multiple people.

VaultSwapContract.sol

This contract receives the BNB sent out by the Devil Token contract, checks if it's balance is over a certain amount, swaps that BNB for BUSD, sends it out of the contract and calls the distribute rewards function in the Devil's Vault contract.

Pragma:

Solidity 0.7.6

Experimental ABIEncoderV2

Dependencies:

ERC20 Interface - Open Zeppelin

SafeMath.sol - Open Zeppelin

Context (abstract)

Address (library)

Ownable - Open Zeppelin

IUniswapV2Factory (interface) - Uniswap SDK

IUniswapV2Pair (interface) - Uniswap SDK

IUniswapV2Router01 (inteface) - Uniswap SDK

IUniswapV2Router02 (interface) - Uniswap SDK

Functions:

The following is a list of primary functions within the contract with their input type and values written in (input type input value) format. For a full list of functions, or to read the contract

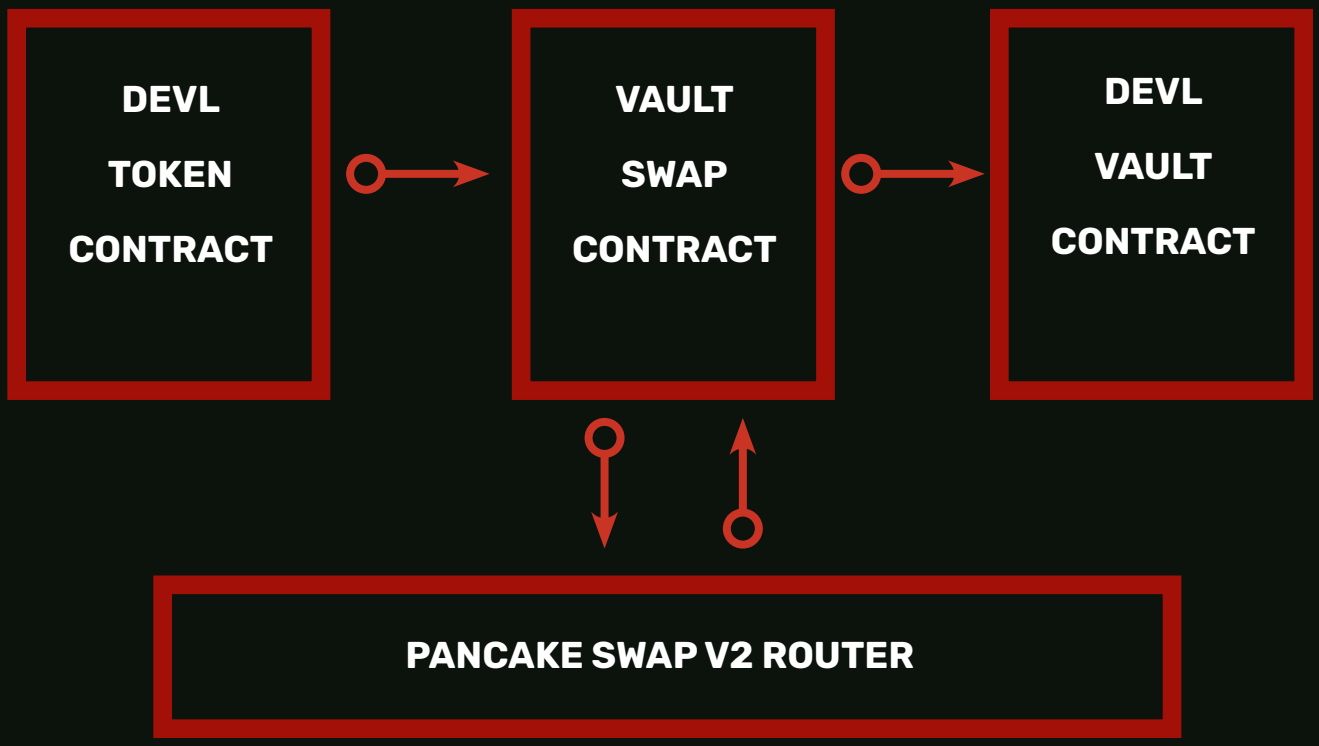
yourself, visit the verified contract on BSC Scan:

Function Name	Input Parameters	Permissions	Description
getQuoteToken	()	Public View Returns (address)	Provides the address of the current “quote” token, or reward token that we are swapping from BNB.
setQuoteToken	(address quoteToken)	External Owner	Sets the quote (reward) token, exists only for modularity, & upgradability should there be issues with reward token.
getWithdrawAddress	()	Public View Returns (address)	The address where tokens are to be withdrawn from contract balance.
setWithdrawAddress	(address _withdrawAddress)	External Owner	Sets the address where tokens are to be withdrawn. Exists for modularity and upgradability.
withdrawQuoteToken	()	Public	Calls for the amount of reward tokens in contract to be withdrawn to DevilVault.sol, calls rewards to be distributed.
setminNumVaultEthtoSwap	(uint256 _minNumVaultEthToSwap)	External Owner	Sets the balance of eth (BNB) needed to trigger the swap feature, save on gas & prevent overuse.
getMinNumVaultEthToSwap	()	Public View Returns (uint256)	Returns the current amount of eth (BNB) needed to trigger the swap feature.

vaultSwap	()	Public Payable	Swaps eth (BNB) for quote token (reward token) through Pancake Swap router, then calls withdrawQuoteToken.

Explanation of Mechanisms & the Modal Structure of Smart Contracts

The Devil's Vault feature relies on three smart contracts - the Devil Token, which provides funding; the VaultSwapContract, which swaps BNB to BUSD and calls the distribute rewards function in the Devil's Vault Contract; and the Devil's Vault Contract, which is the yield farm allowing holders to stake DEVL for their share of BUSD rewards.



3.5 DEVIL LOCK

Devil Lock is the first version of the token's governance contract. In this iteration, it controls the Devil Token itself. Future versions will control the entire ecosystem, have advanced creation proposals mechanisms, and data tracking for all elections.

Pragma:

Solidity 0.7.6

Dependencies:

IERC20 Interface - Open Zeppelin

SafeMath.sol - Open Zeppelin

Context (abstract)

Address (library)

Ownable - Open Zeppelin

Functions:

The following is a list of primary functions within the contract with their input type and values written in (input type input value) format. Getter functions are not listed here for this contract. For a full list of functions, or to read the contract yourself, visit the verified contract on BSC Scan:

Function Name	Input Parameters	Permissions	Description
addConvener	(address _address)	Public Convener	Allows a convener permission to be added to an address.
removeConvener	(address _address)	Public Convener	Removes convener permissions from an address.
setDevilTokenAndContract	(address _address)	Public Convener	Sets the Devil Token contract address as the linked contract and token interface.

electionsDuration	(uint256 _newVot-ingPeriod)	External Convener	Adjusts the period of time that an election has to take place before a convener can close the election.
startVoting	()	External Convener	Starts an election to allow a transfer to take place for the input address.
endVoting	()	External Convener	Closes the election, tabu-lates results, resets state variables pertaining to election.
callLock	()	External Convener	Locks the contract again
callUnlock	()	External Convener	Unlocks the contract
callTransferOwnership	()	External Convener	Allows the contract own-ership to be transferred to the address specified in the election.

Description Key Functions In Devil Lock

Conveners

Conveners are the authorized individuals (the development team), who are able to request an election, start an election, close an election (after the election duration has elapsed), and then unlock the contract/transfer ownership to themselves to call necessary functions if given the allowance to do so by the community.

Voting & the Lock System

Elections can be initiated by a convener who is requesting access to make changes to the contract. Elections must take place for at least 24 hours; conveners are locked from being able to end the election until 24 hours have passed. Voting power is currently determined by ownership of DEVL. Once you vote in favor or in opposition to a proposed unlock of the contract, you can no longer vote.

Because unlocking the contract requires ownership of the contract to be temporarily transferred to another address, that address must be provided by the convener at the beginning of the election. Only that pre-agreed upon address can be used to transfer ownership, increasing transparency and accessibility.

At the end of an election, a convener calls to close the election and the smart contract tabulates the results. If successful, the convener may then call to unlock the contract and/or transfer ownership in order to make necessary changes.

After changes have been made, the contract is locked again.

4. THE FUTURE

4.1 The DEVL Plan

The ultimate goal of the Devil Token is to forge new ways for DeFi users to utilize the outstanding innovations of the space, it's aim is to be less in competition with projects in the space as much as it seeks to append the growing ecosystem.

DEVL and its ecosystem sees its usage from two distinct groups: the DeFi enthusiasts who could be aided by various tools to assist them in organizing, tracking, and transferring DeFi assets - and users both new to DeFi or even cryptocurrencies in general.

To accomplish this goal, the DEVL team will constantly be looking for new ways to enhance the DeFi user experience such as: improving the process for fiat/crypto interactions, assisting in the management and tracking of all crypto assets, developing multiple cross-chain methods of transferring tokens, and even wallet development.

The final significant aim of DEVL's ecosystem is to embrace the trustlessness of smart contract technology, by gradually developing the token's governance system to the point where it becomes wholly and entirely decentralized, meaning all aspects of the token's development will ultimately be handled by a DAO, with developers accessing files in accordance with the token's governance.

4.2 Philosophy

DEVL is a long-term project with a dedicated team behind it. While market price is an important focus and target to fuel the token's ecosystem, development and collaborations with other projects are an equally valued priority.

The aim of the DEVL project is to breed radical financial independence for each of its users, empowering them with the tools to manage, access, and utilize their crypoassets.

The DEVL team sees themselves less as rulers or owners of the token, and more as custodians or conveners who are tasked with helping facilitate the token and ecosystem's growth. From this perspective, community comes first. Whatever challenges, significant decisions, or developments come the DEVL's way, it will be the entire community that has an opportunity to guide the DEVL's path.

Through DEVL we are seeking to breed that radical independence together.