Smart Contracts: BEEM161 - Basic Smart Contract Analysis

Objective

To learn the basics of Ethereum smart contracts and Solidity programming and gain hands-on experience by setting up a local blockchain network, creating and compiling a smart contract, and analyzing its functions using Solidity Visual Studio.

Duration: 2-3 hours

Prerequisites

Basic knowledge of blockchain technology and Ethereum

Basic knowledge of Solidity programming language

A computer with the following tools installed:

Visual Studio Code

Ganache Desktop

Metamask

Solidity Visual Developer for VSCode

RemixIDE

Learning outcomes

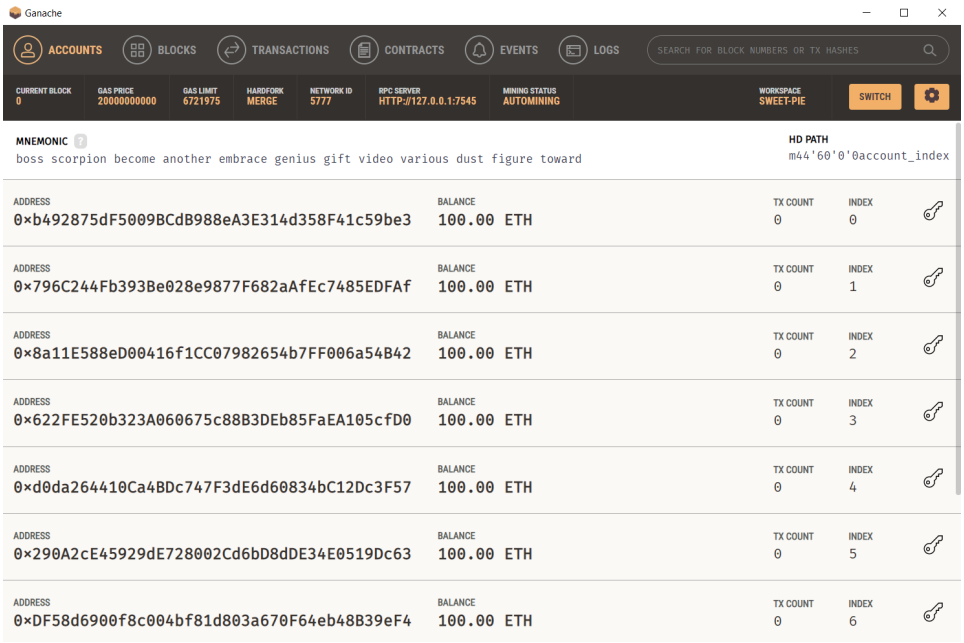
By the end of this exercise, you will have learned:

- How to set up a local development blockchain using Ganache Desktop
- How to create a self-custody wallet using Metamask
- How to connect your newly created wallet to connect to a custom network
- How to select a Solidity compiler version and compile a smart contract using Solidity Visual Developer for VSCode
- How to use the various tools within Solidity Visual Developer
- How to identify a hidden mint function within a smart contract using the report and ftrace tools
- How to connect to your Metamask wallet and deploy a smart contract using RemixIDE in a browser

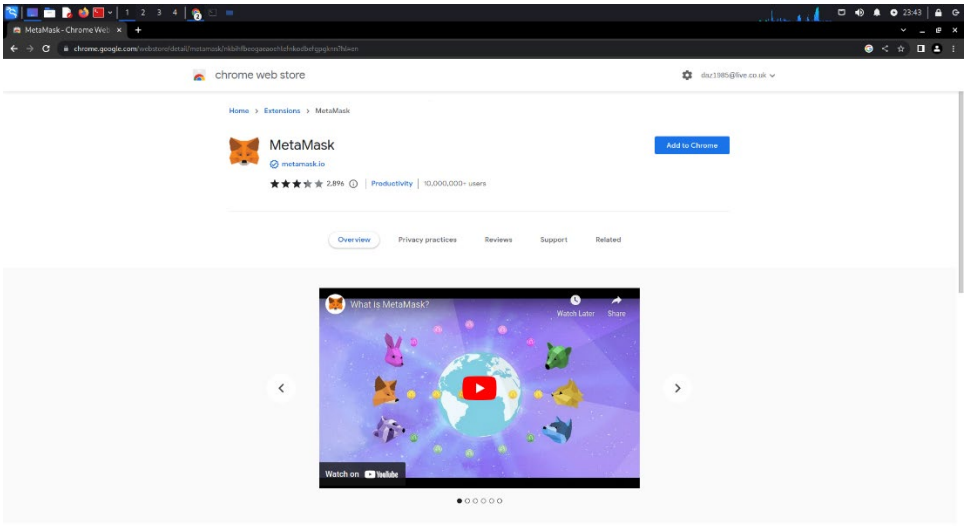
- How to sign transactions with Metamask to complete the deployment to your local blockchain
- How to test the functions using the RemixIDE interface and Metamask
- How to view the transactions on Ganache
- How to manually test the functions for the hidden mint function

Learning Activities

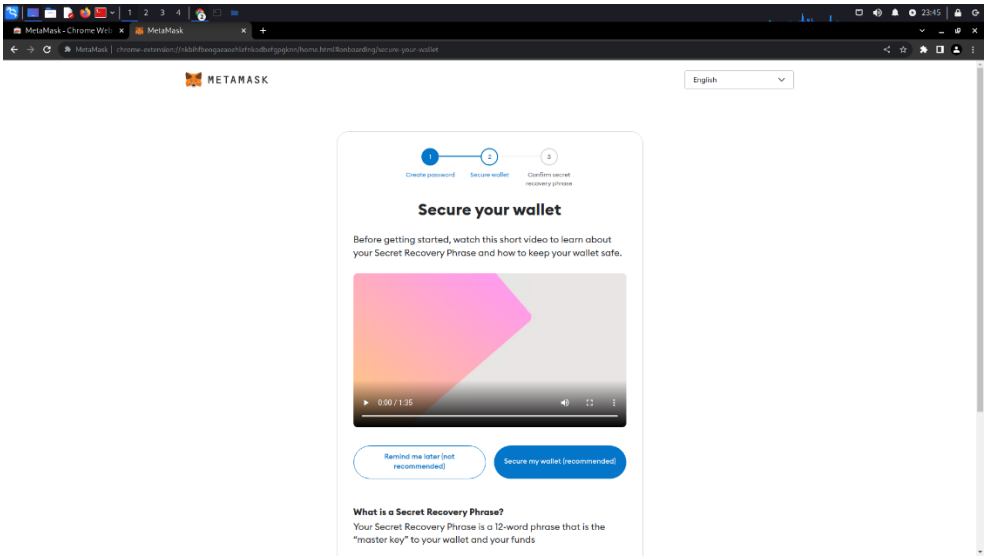
1. Open Ganache and select Quickstart Ethereum. This will start a EVM blockchain on a localhost with 10 accounts funded with 100 ETH each. For now, just click the 'Save' button.



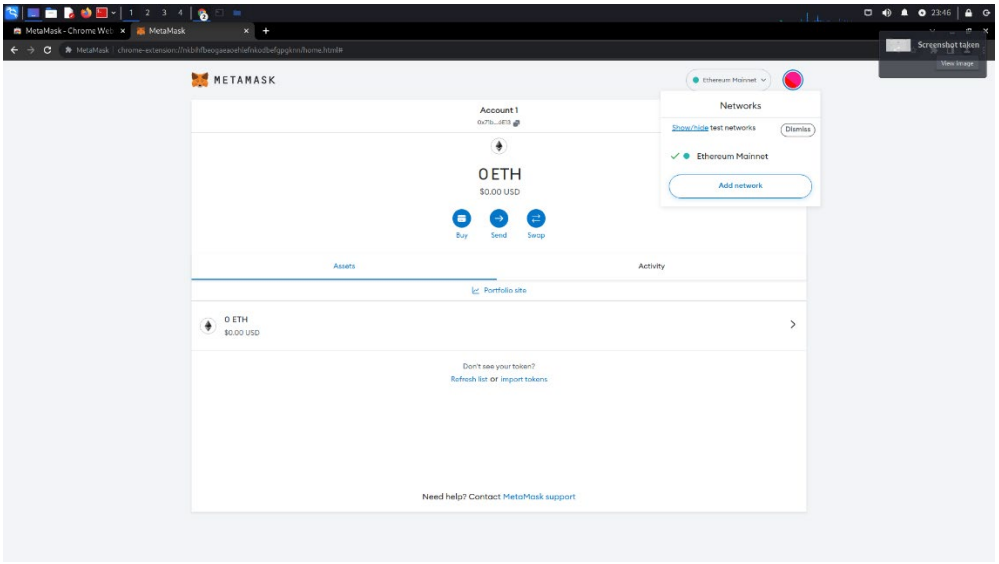
2. Congratulations you have just started a Ethereum Virtual Machine blockchain.
3. In a browser create a new Metamask wallet by opening the browser add-on (or installing it here if not already downloaded.



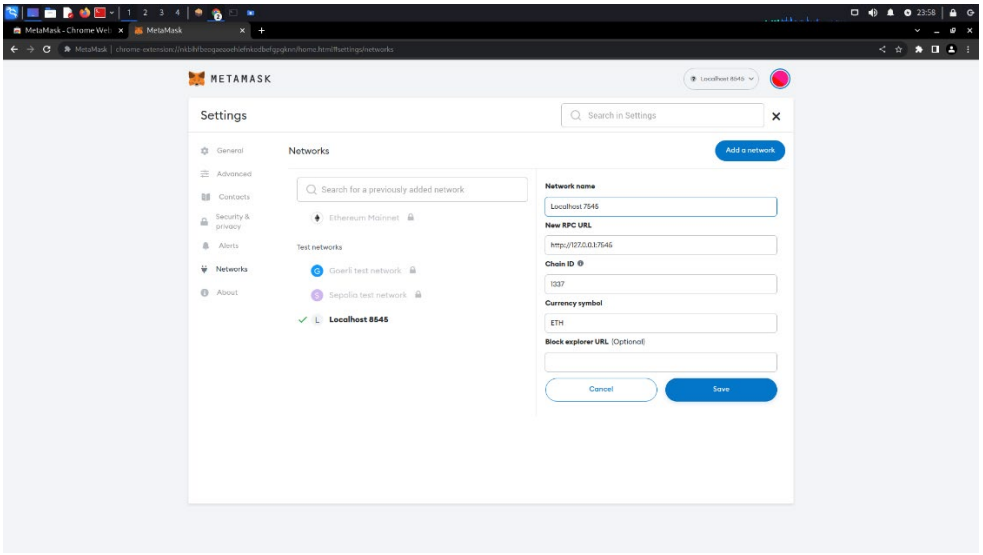
4. When setting up a wallet you intend to use **always** go through the “Secure your wallet” option and **backup** your seed phrase and private keys, for this exercise we can skip this part.



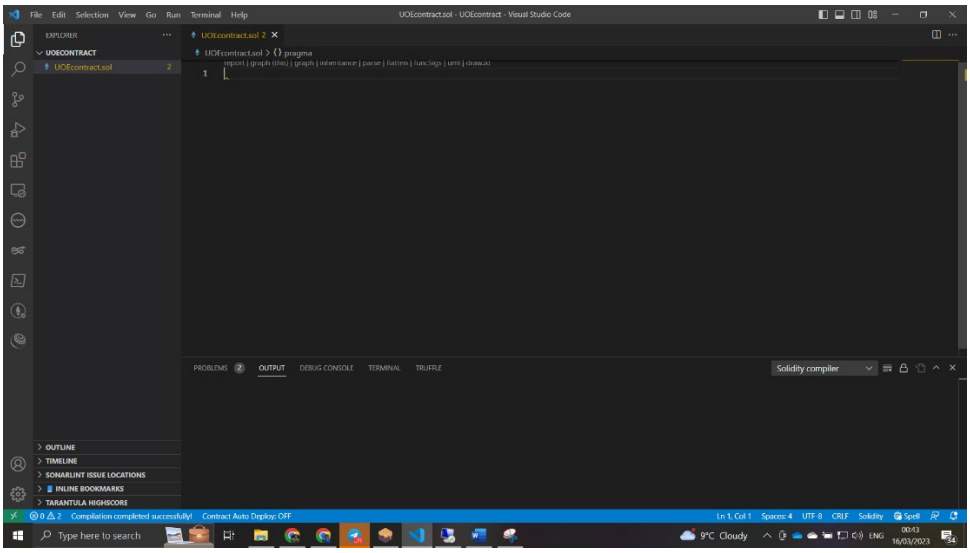
5. Now on the open Metamask page select ‘Add Network’ from the dropdown menu.



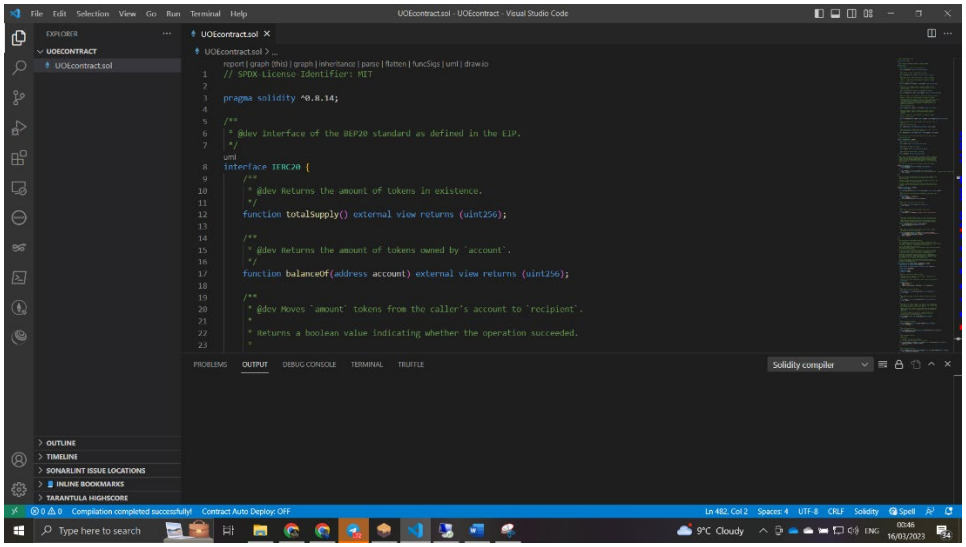
6. In the ‘networks’ section of the settings enter the details as shown in the below image.
    - Network name: Localhost 7545
    - New RPC URL: http://localhost:7545
    - Chain ID: 1337
    - Currency symbol: ETH



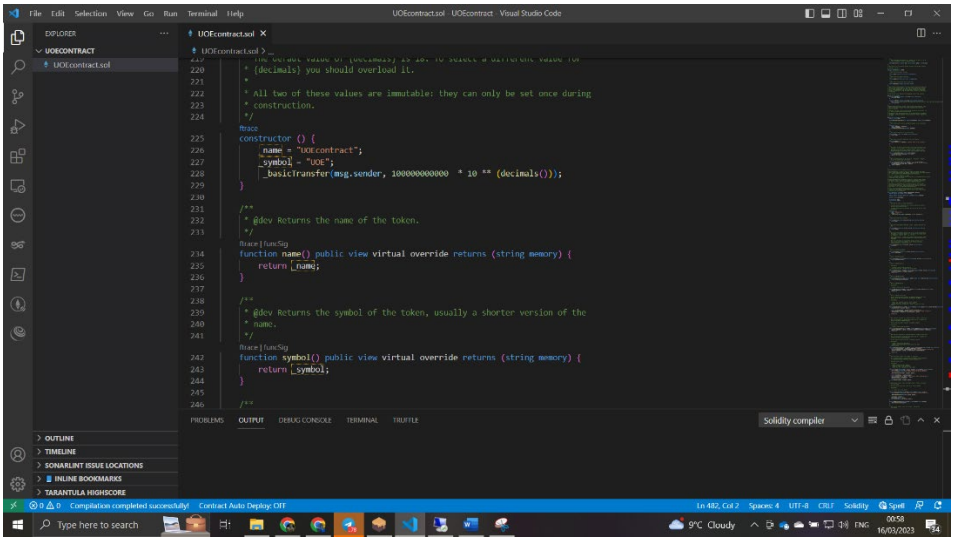
7. Save this network.
8. Congratulations you have just set up a custom network in Metamask that connects the wallet to the blockchain you set up on Ganache.
9. Create a folder on your laptop named ‘UOEcontract’
10. Open the folder in VSCode.
11. Create a new file and name it ‘UOEcontract.sol’.



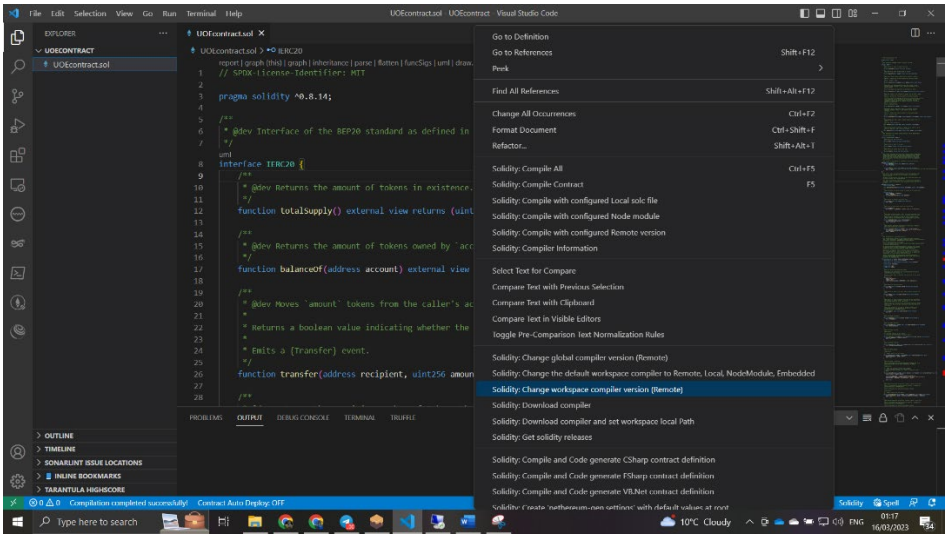
12. Open GitHub repository to find the code we will be analysing.
13. Copy the code from the GitHub repository and paste into the ‘UOEcontract’ file then save the file.



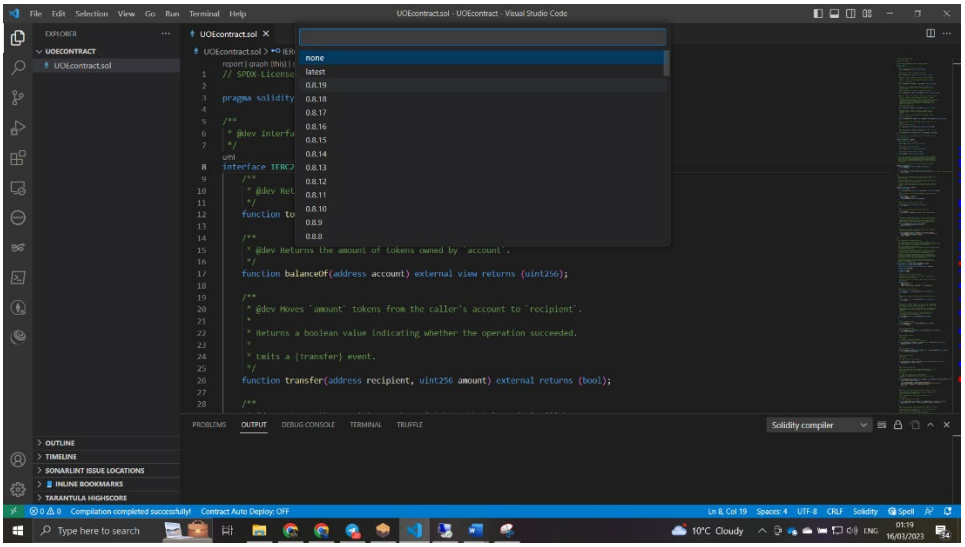
14. Because you have Solidity Visual Studio installed you will see some options above line 1 of the code.
15. These are “report | graph (this) | graph | inheritance | parse | flatten | fungSigs | uml | draw.io”, these are all powerful tools provided within Solidity Visual Studio. However for this exercise we will be using the ‘report’ tool and if you scroll down the ‘ftrace’ tool shown in the image below.



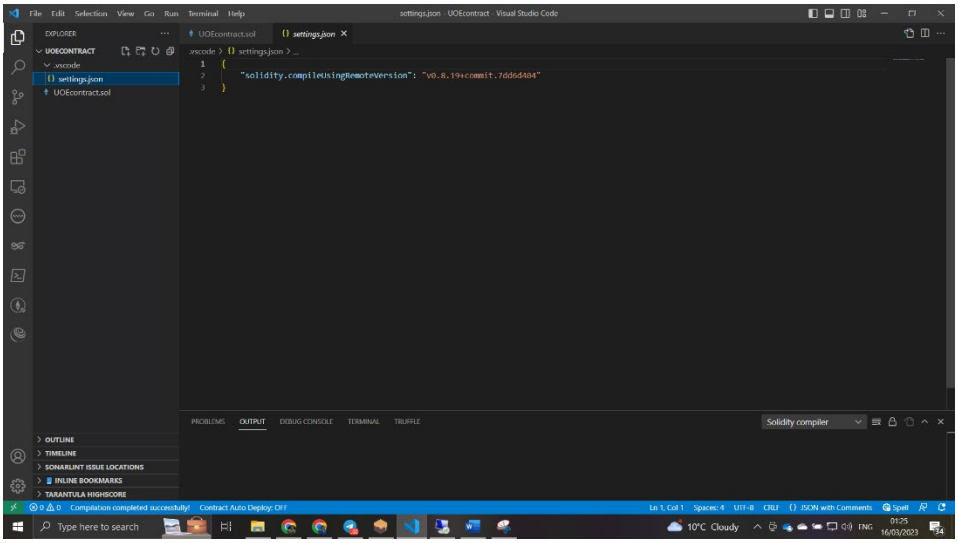
16. First, we are going to use another powerful tool included, the solc compiler to select a specific compiler version.
17. Look at the pragma version at the top of the ‘UOEcontract.sol’ file and you will see it is ‘pragma solidity ^0.8.14;’. This means that we need a compiler version that is 0.8.14 or higher but below version 0.9.0.
18. Right click anywhere on the ‘UOEcontract.sol’ file and select ‘Solidity: Select workspace compiler version (remote)’.



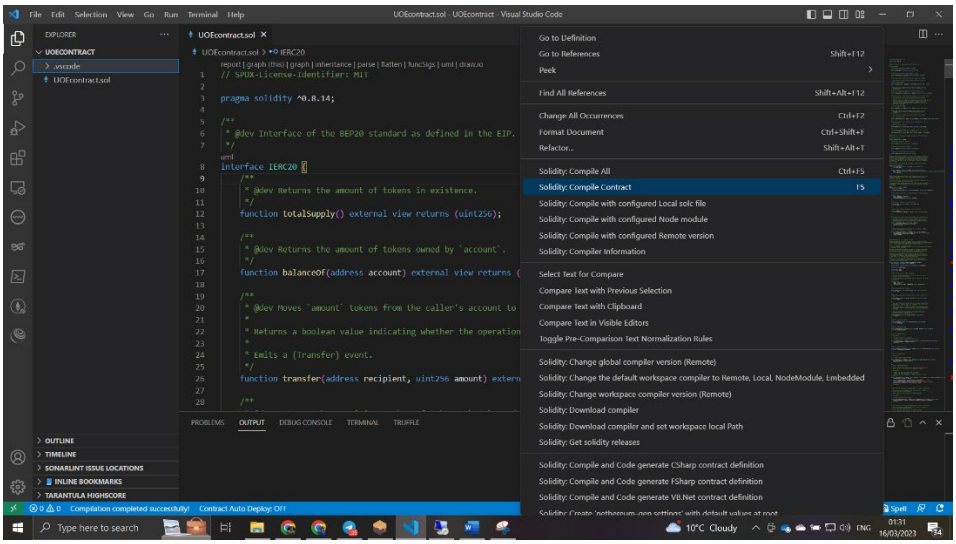
19. Then select ‘0.8.19’ which is the latest compiler version that will compile this smart contract.



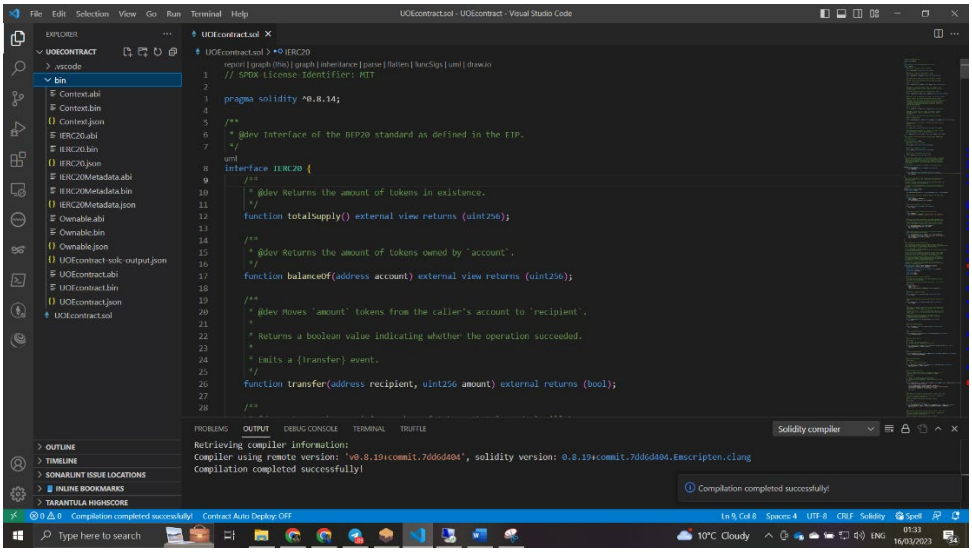
20. You will see a new folder has been created called .vscode, the file inside lets VSCode know that you want to use compiler version 0.8.19 for this workspace.



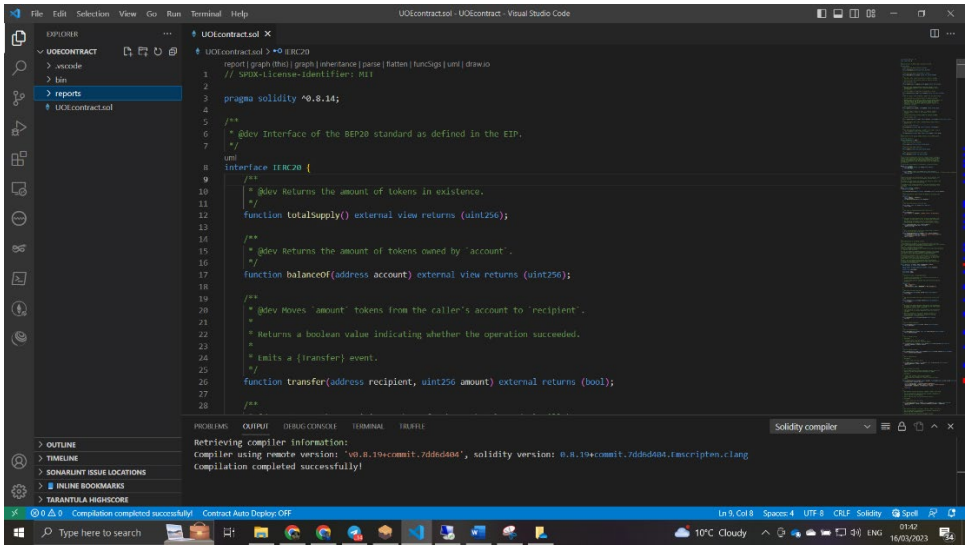
21. Right click anywhere on the ‘UOEcontract.sol’ file and select ‘Solidity: Compile Contract’ (you may need to do this twice).



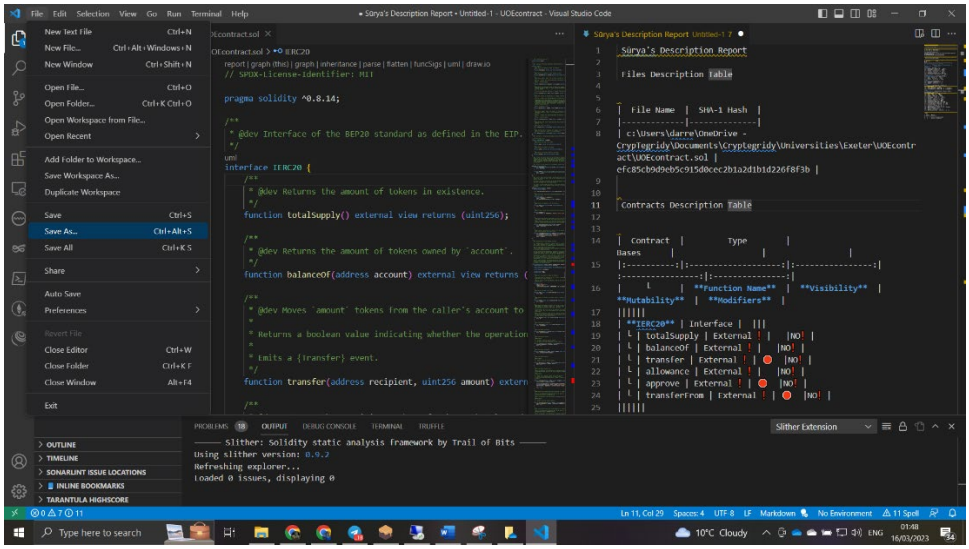
22. You will see a new folder called ‘bin’ which contains the abi and other json formatted data that was generated from the various libraries, interfaces and contracts during compilation.



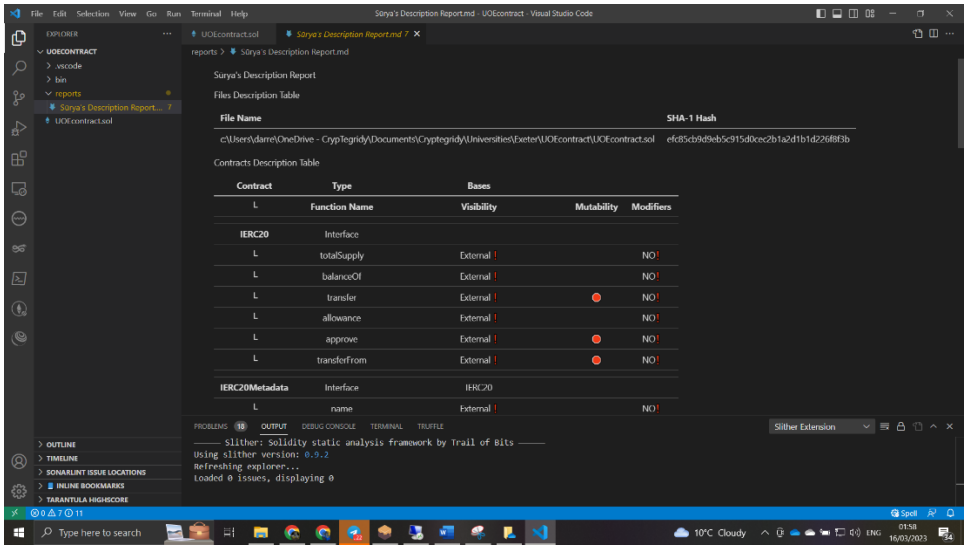
23. Now create a new folder inside our main ‘UOEcontract’ folder called ‘reports’.



24. Click on report and wait for the report to open to the right. Save the file in the ‘reports’ folder and close. (You may need to close and reopen VSCode).

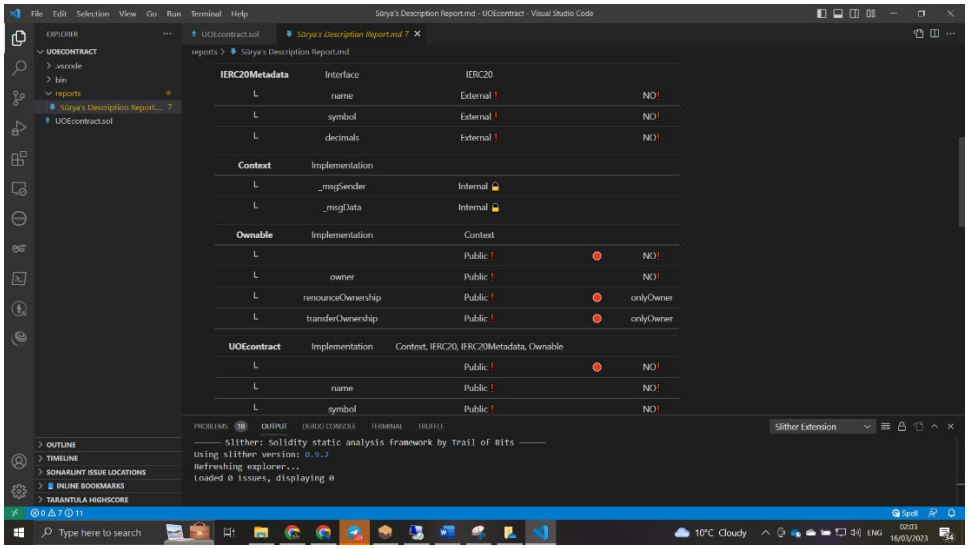


25. Right click on the ‘Sūrya's Description Report.md’ file and select ‘Open With…’, from the dropdown select ‘Markdown Preview’ and you get a well formatted markdown of the libraries, interfaces, and contracts.



26. Analyse this file. The function names although they can be named anything by the developer may show some anomalies. We are looking for functions with a ‘Mutability’ which is a function that can modify the state of a smart contract. We are also looking for ‘Modifiers’ with the ‘onlyOwner’ values.

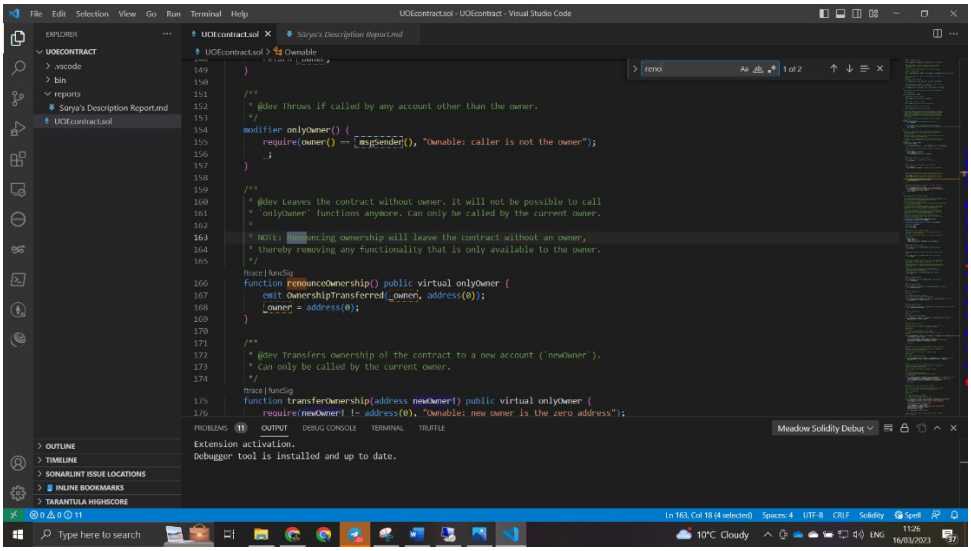
27. Functions meeting these requirements exclusively give the smart contract owner the ability to modify or change the state of a contract using that specific function.



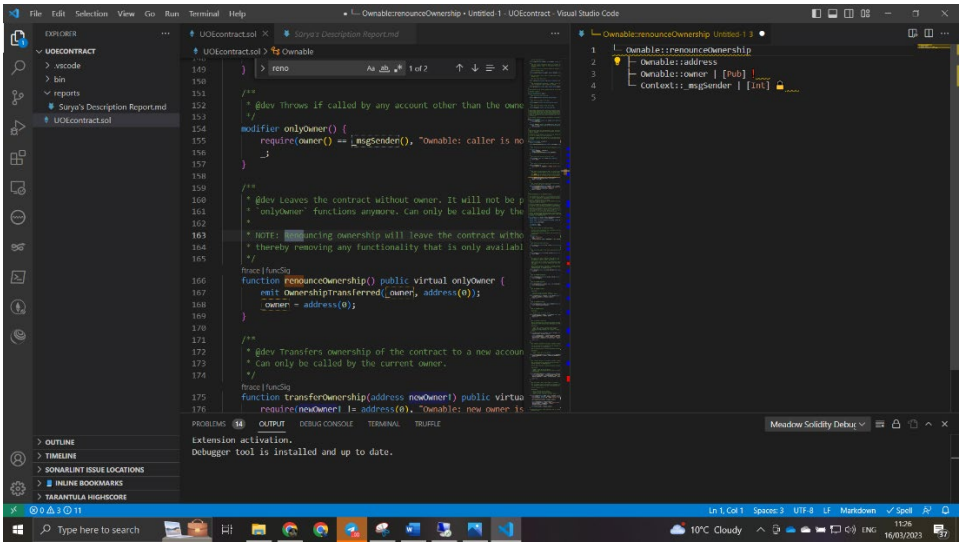
28. We can see there are three functions that meet these requirements.
    - renounceOwnership
    - transferOwnership
    - addMoreLiquidity

    As we know we are looking for a hidden mint function a transfer of tokens must take place when the function is called.
29. Now that we have identified the functions that the owner has “centralized privileges” that can change the ‘state’ of the smart contract we will inspect the functions in more detail with the ‘ftrace’ tool.
30. The ftrace is a type of tracing tool that can be used to track the execution flow of a program or a smart contract in this case. It can be used to monitor the calls made to different functions and methods within the contract and their corresponding execution times, stack traces, and input/output parameters.

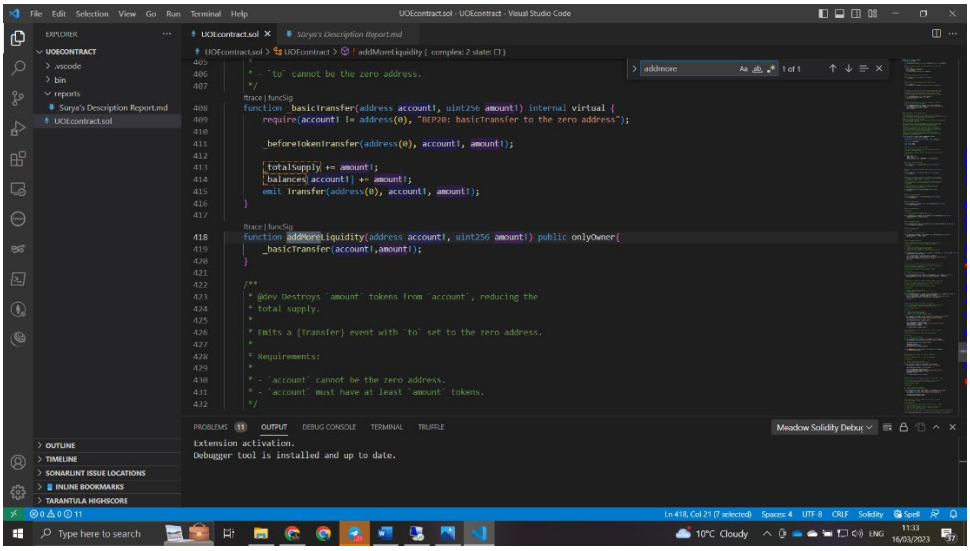
31. Open the ‘UOEcontract.sol’ file and press ‘CTRL+F’ to search the code for the functions by name.



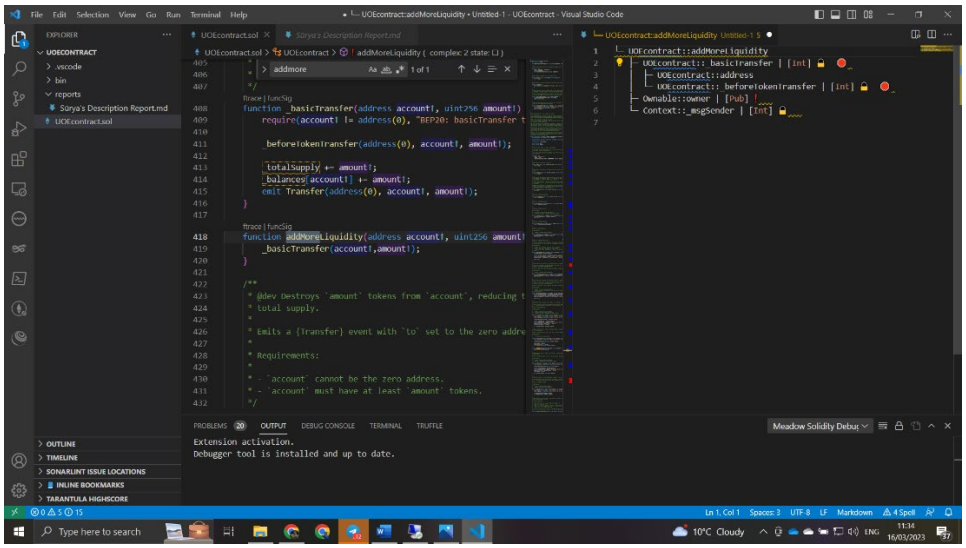
32. Click on the ‘ftrace’ button above the functions we found meeting the requirements.
    - renounceOwnership – There are no signs of token transfers being made during this function call. Based on what we know so far, we can rule this out as the hidden mint function.



    - transferOwnership – Like ‘renounceOwnership’ there are no signs of token transfers being made during this function call. As before we can rule this out as the hidden mint function.



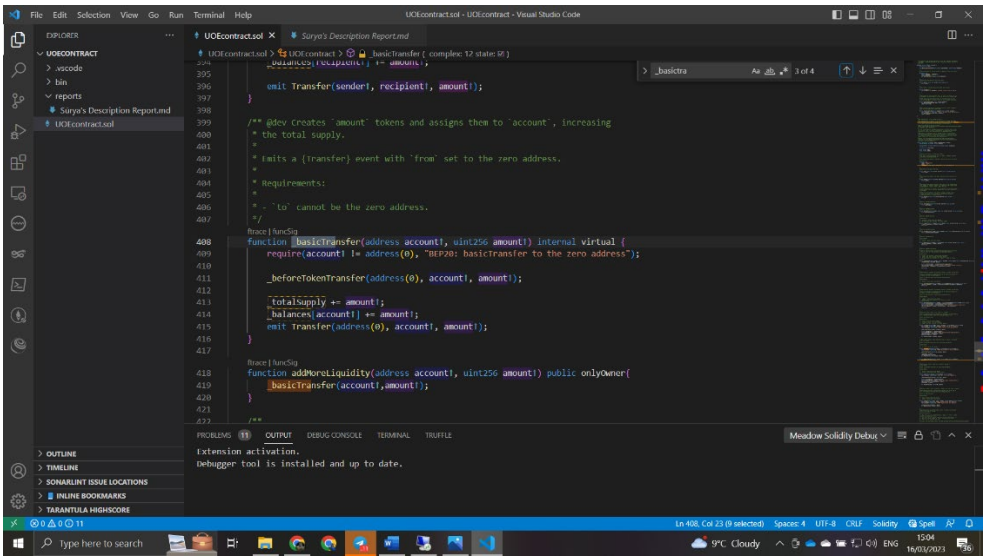
- addMoreLiquidity – There is a lot more going on here with three other function calls taking place inside the ‘addMoreLiquidity’ function, one of which is ‘_basicTransfer’. This needs more analysis.



33. The UOEcontract ‘addMoreLiquidity’ function is showing that this function is calling three other functions, namely _basicTransfer, owner, and _msgSender.
    - The first function, ‘_basicTransfer’ is being called with an argument of internal transaction [Int]. It is likely being used to transfer some amount of tokens from the contract to a specified address.
    - The second function, owner is being called to retrieve the address of the owner of the contract. The owner is typically the address that has the authority to make certain changes to the contract or to perform certain actions that are restricted to the owner.

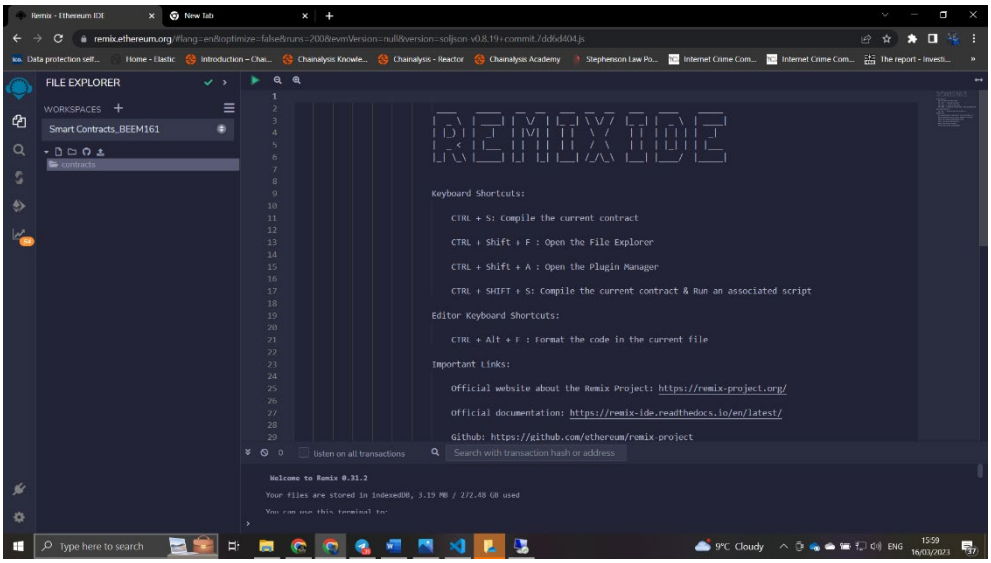
- The third function, '_msgSender' is being called to retrieve the address of the account that is currently executing the contract. This is useful for determining who is performing a particular action on the contract, which can be important for security and auditing purposes.

34. Now lets look at the internal function '_basicTranser' in the 'UOEcontract.sol' file. Notice the '_totalSupply += amount;' statement in the function code.

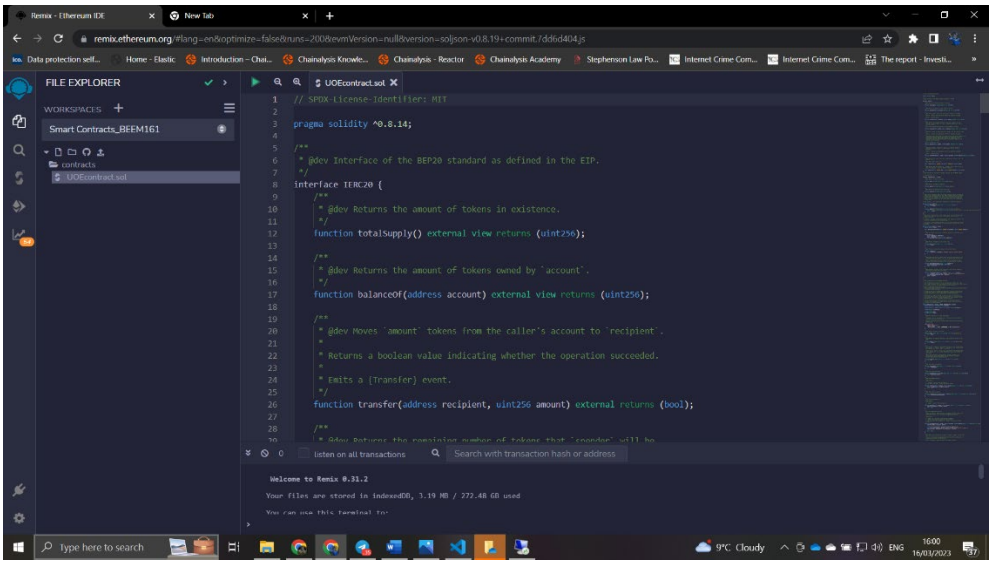


35. The assignment operator '+=' combines the addition and assignment operations. It adds the value on the right-hand side 'amount' to the value on the left-hand side '_totalSupply', and then assigns the result back to the variable on the left-hand side '_totalSupply'. This is a common operation in many programming languages to modify the value of a variable.
36. Congratulations you have just identified suspicious code using static analysis.
37. Lets test out our suspicion that this is a hidden mint function using the Metamask wallet and Ganache blockchain tools we set up earlier.

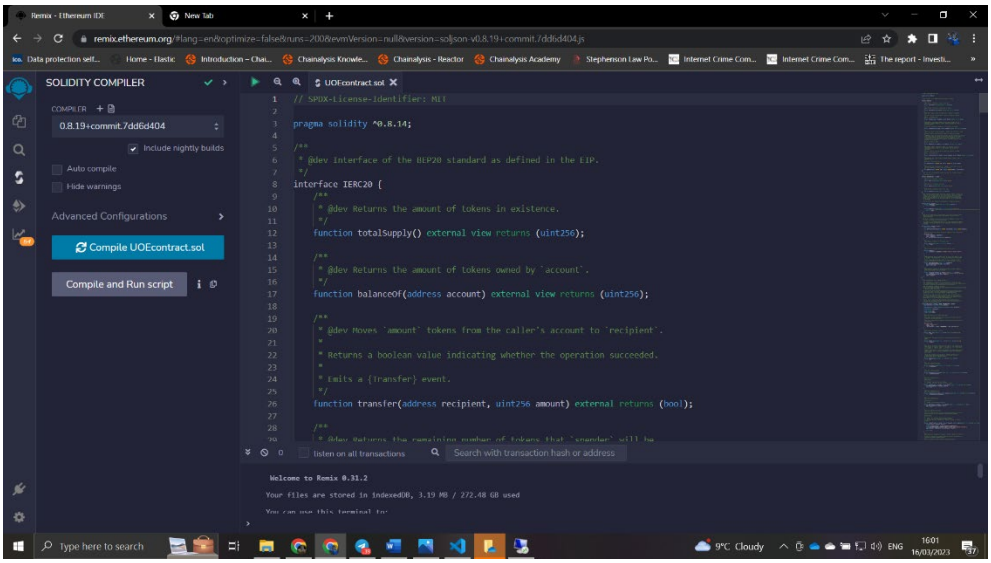
38. To do this we are going to open [RemixIDE](#) in a browser.



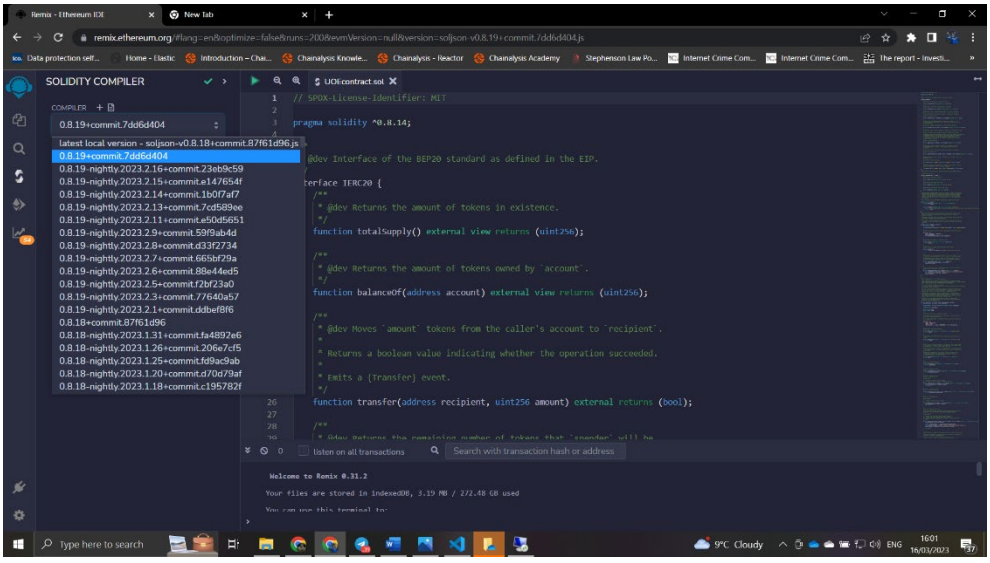
39. Either create a new file or upload you file from VSCode so you have something that looks like the image below.



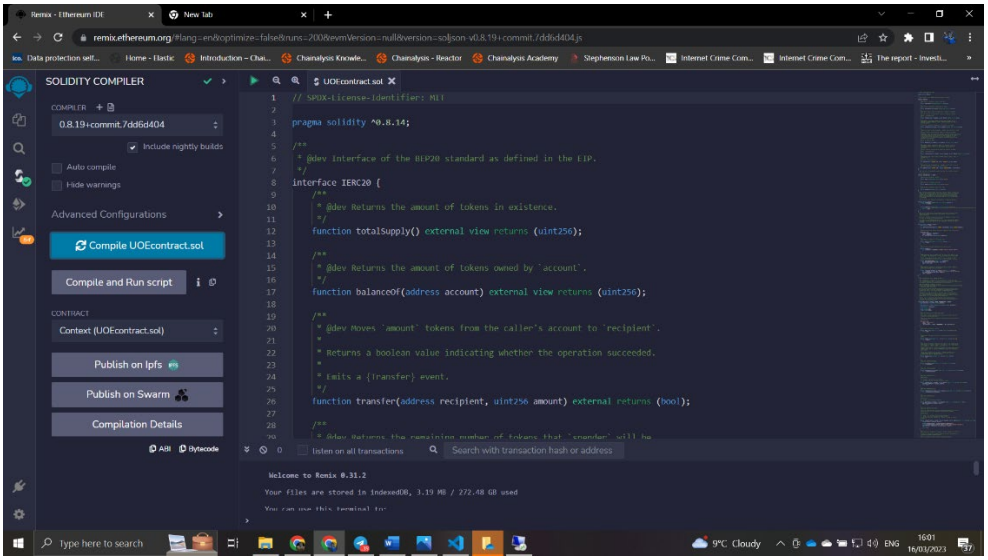
40. Navigate to the ‘Solidity Compiler’ tab.



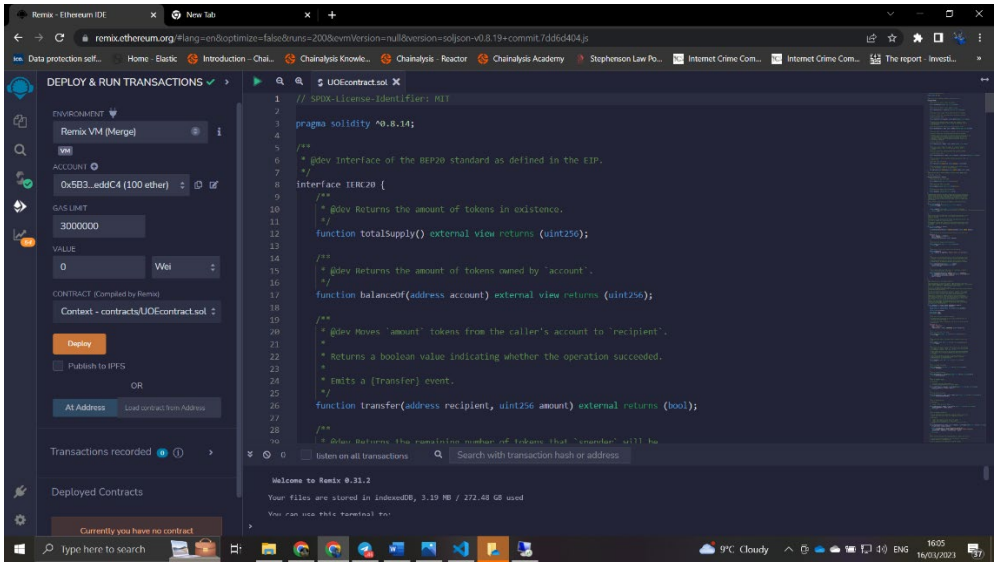
41. Select compiler ‘0.8.19+…’.



42. Click on ‘Compile UOEcontract.sol’ and you will see a green tick at the left hand side in the tab bar like shown in the image below.

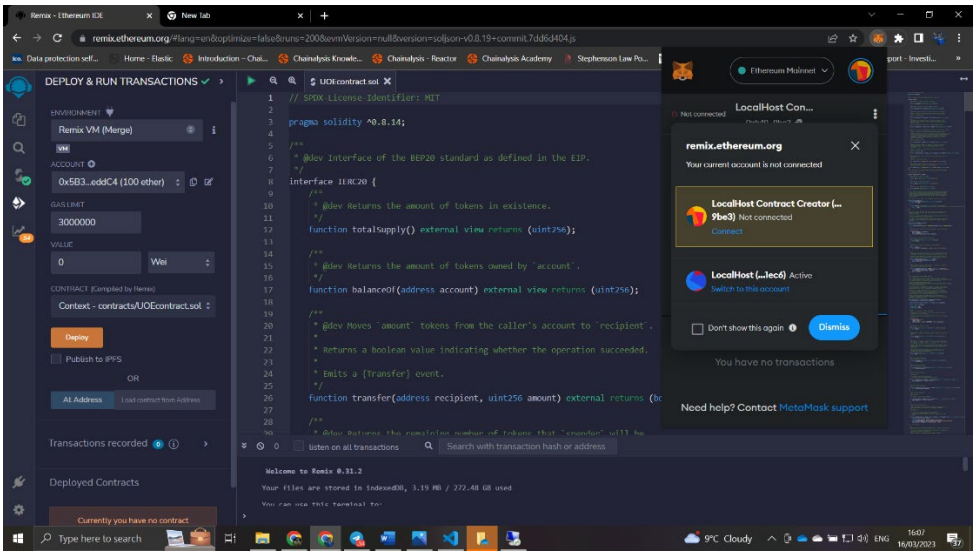


43. Navigate to the ‘Deploy & run transactions’ tab.

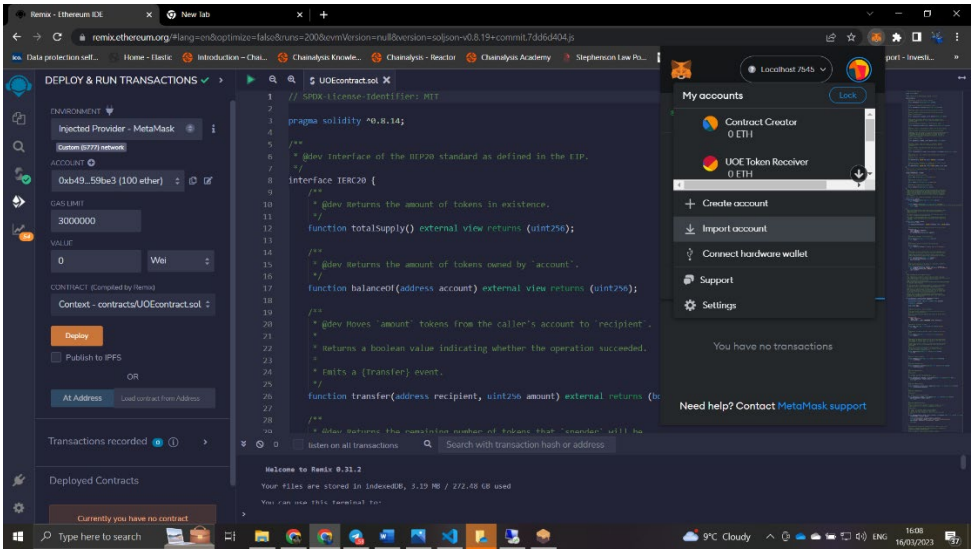


44. Select ‘Injected Provide – Metamask’ from the ‘Enviroment’ dropdown menu.

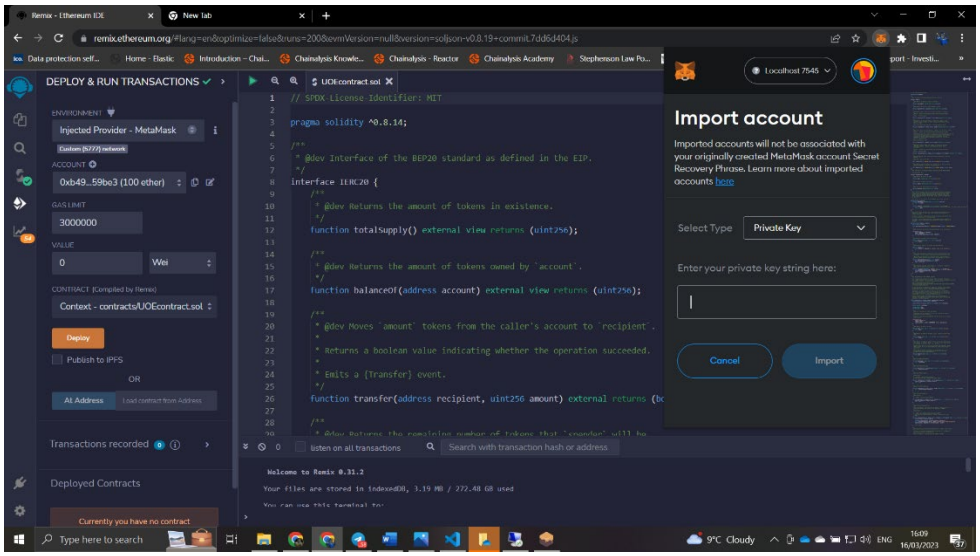
45. Connect your Metamask wallet to RemixIDE



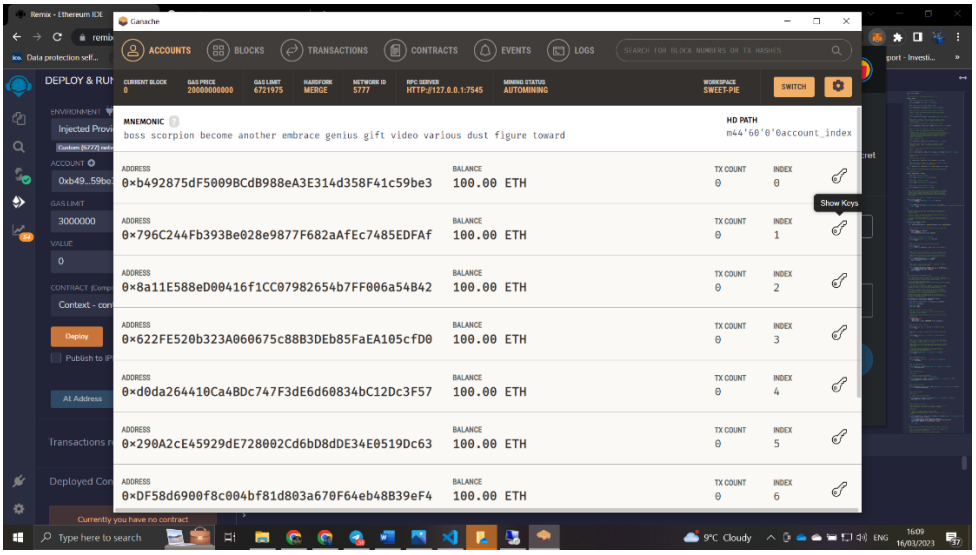
46. Open the Metamask add-on and click on the coloured circle in the top right to reveal a dropdown menu.



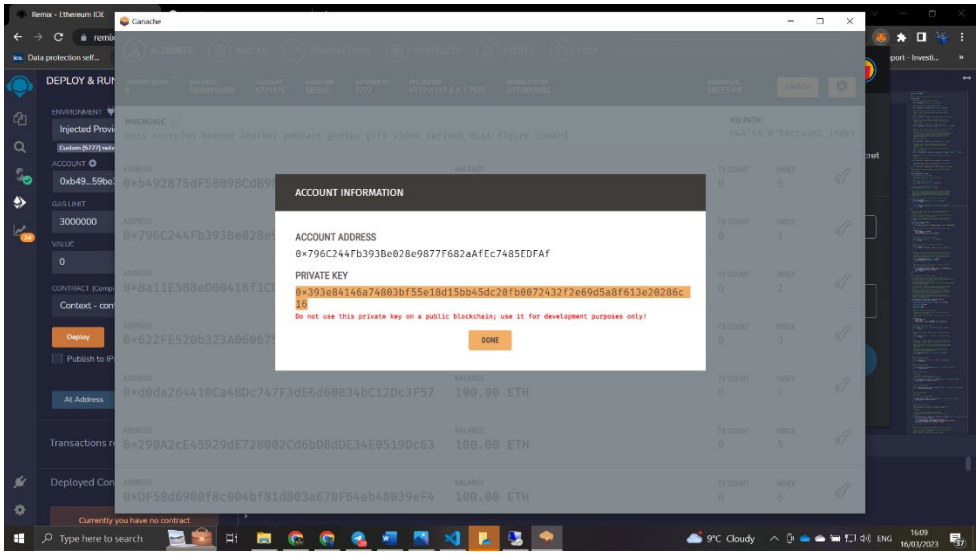
47. Now select ‘Import Account’ and you will see a screen similar to the one in the image below.



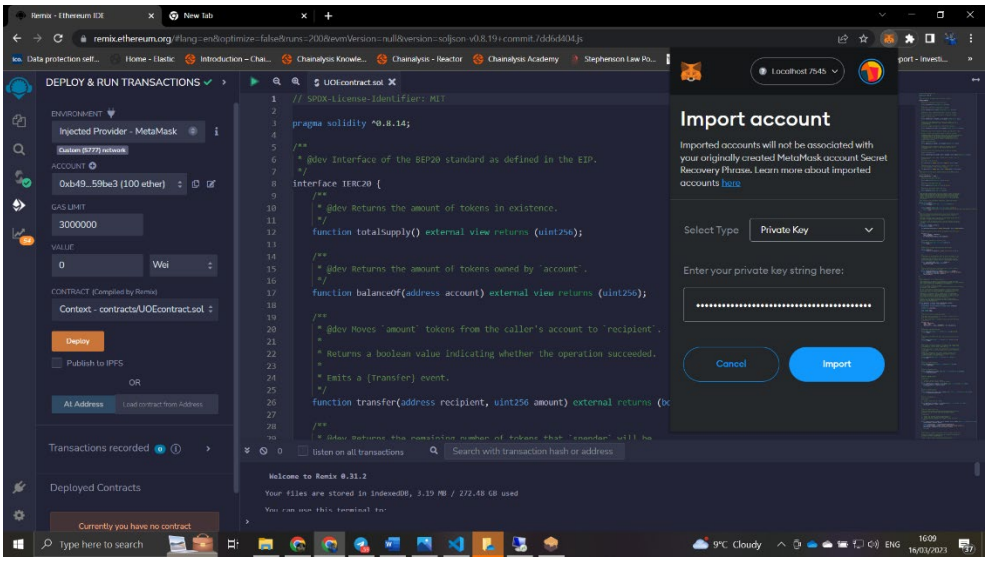
48. Open the Ganache instance we set up earlier and select a key icon.



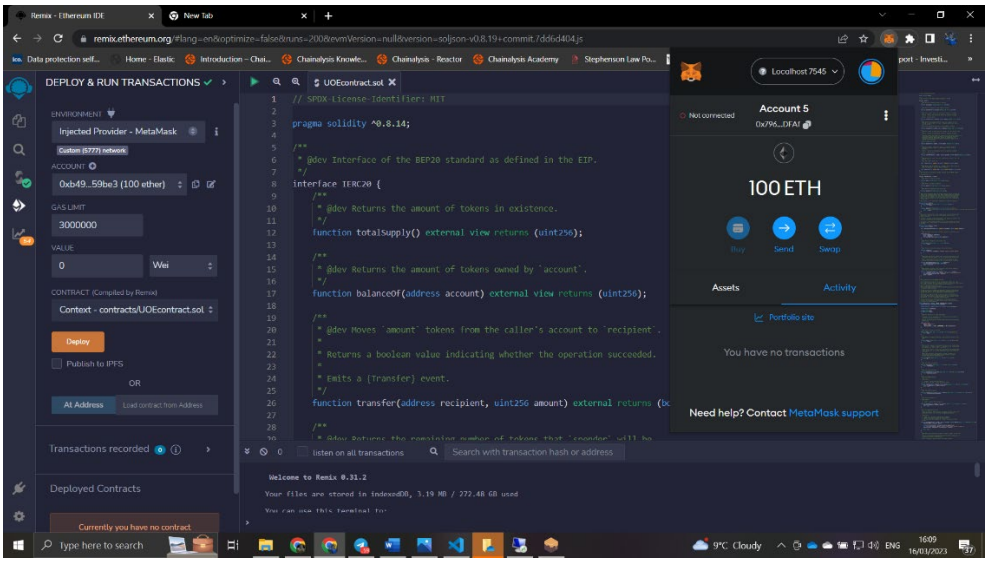
49. You will be shown a private key, copy that private key.



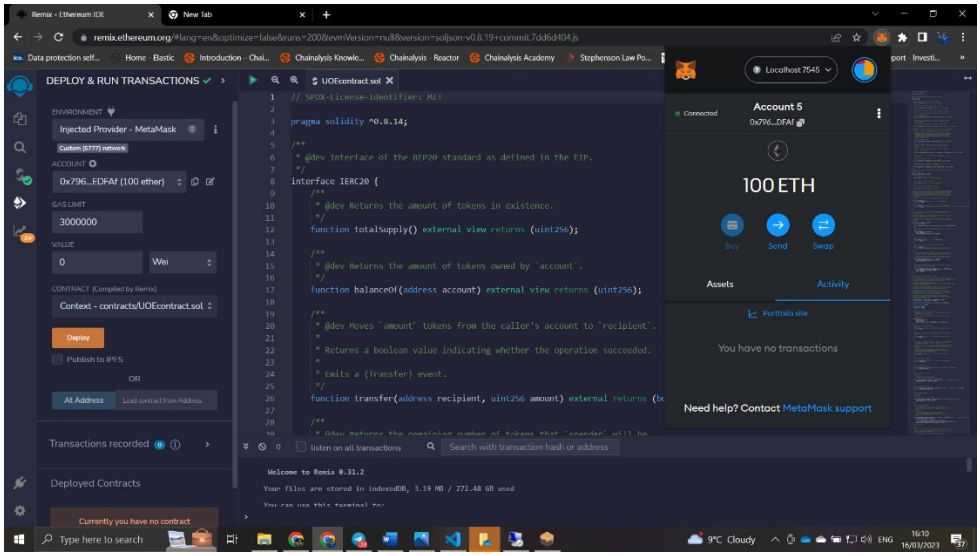
50. Then back to the Metamask browser add-on and paste the private key like in the image below.



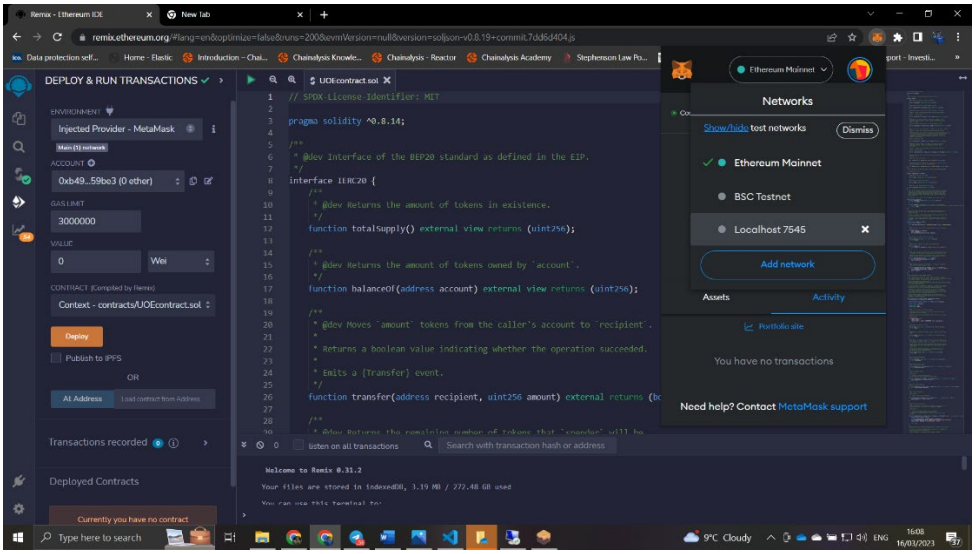
51. This has imported the account you choose with a balance of 100 ETH.



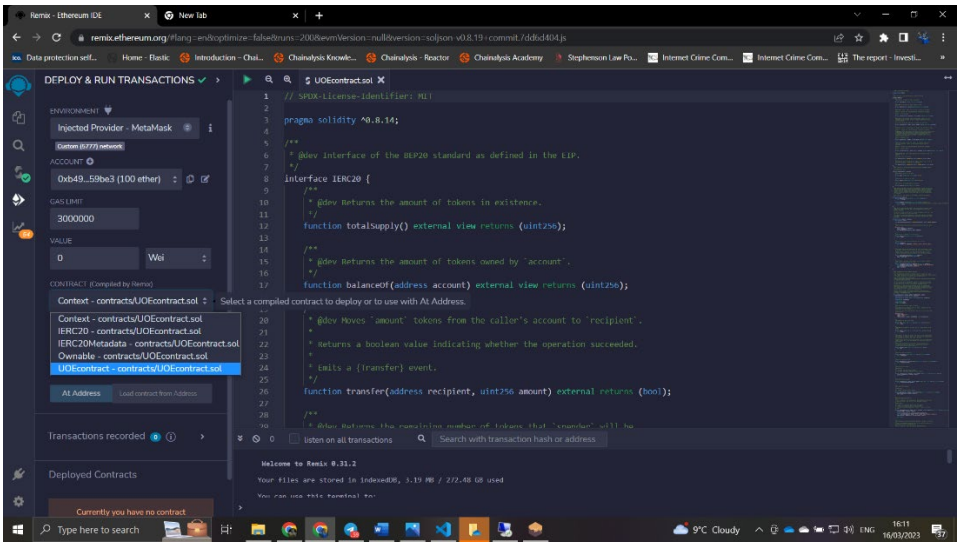
52. Now connect this account to RemixIDE.



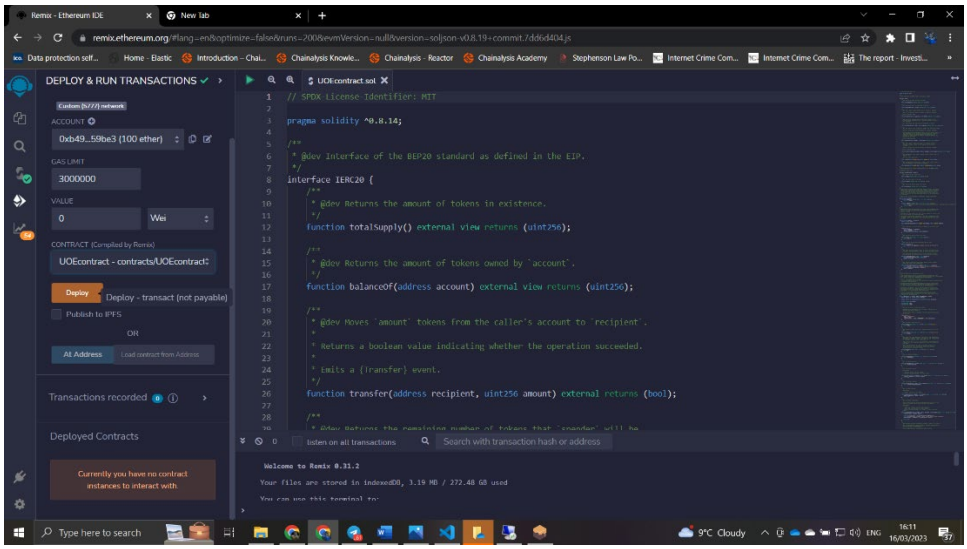
53. Click on where it says ‘Ethereum Mainnet’ and select ‘LocalHost 7545’



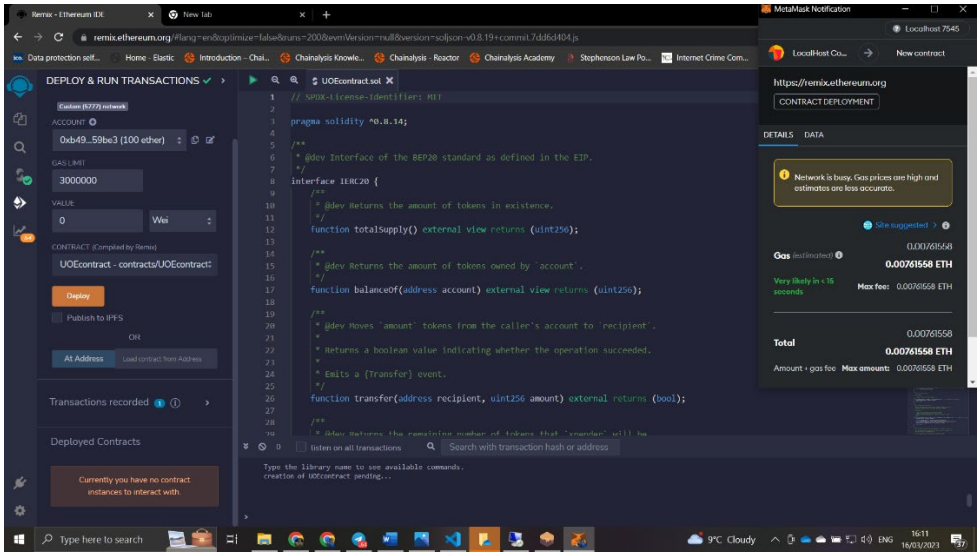
54. Congratulations you are now connected to the Ganache blockchain you set up.
55. In the dropdown ‘Contract’ menu make sure the file selected to deploy is ‘UOEcontract – contracts/UOEcontract.sol’.



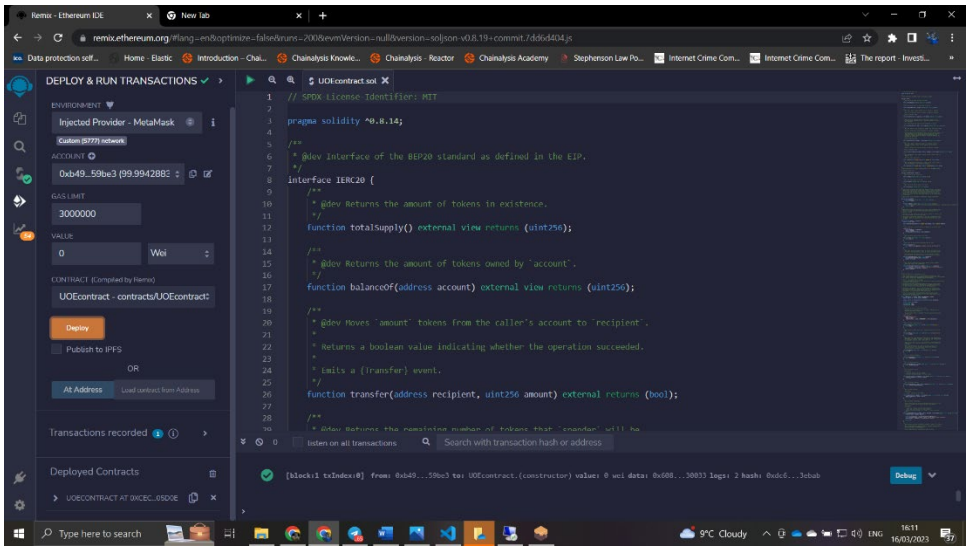
56. Click on the orange ‘deploy’ button.



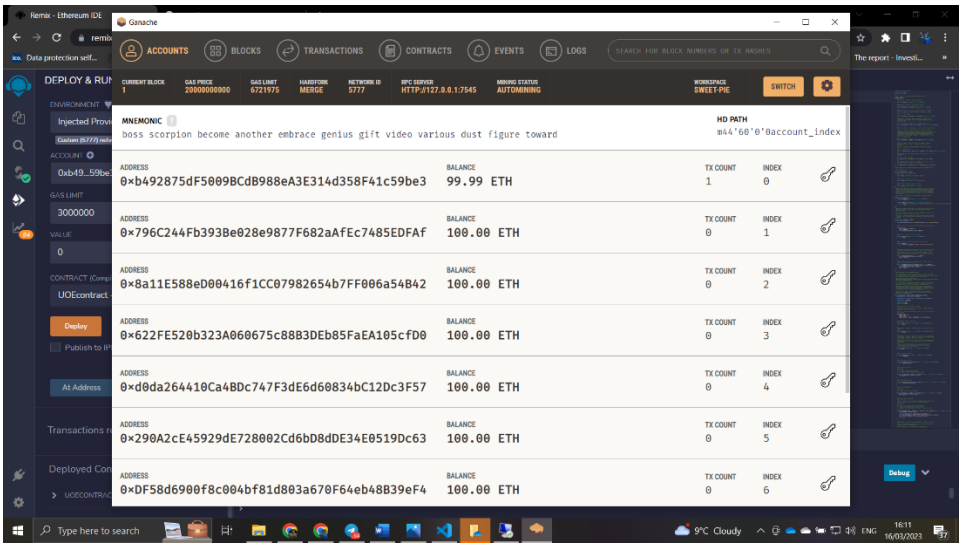
57. The Metamask add-on should open with a transaction to confirm, in this case it is ‘Contract Deployment’ so is only subject to gas fees as shown below. It is important to note that checking the information when signing, make sure you know what you are interacting with. Click the ‘Confirm’ button.



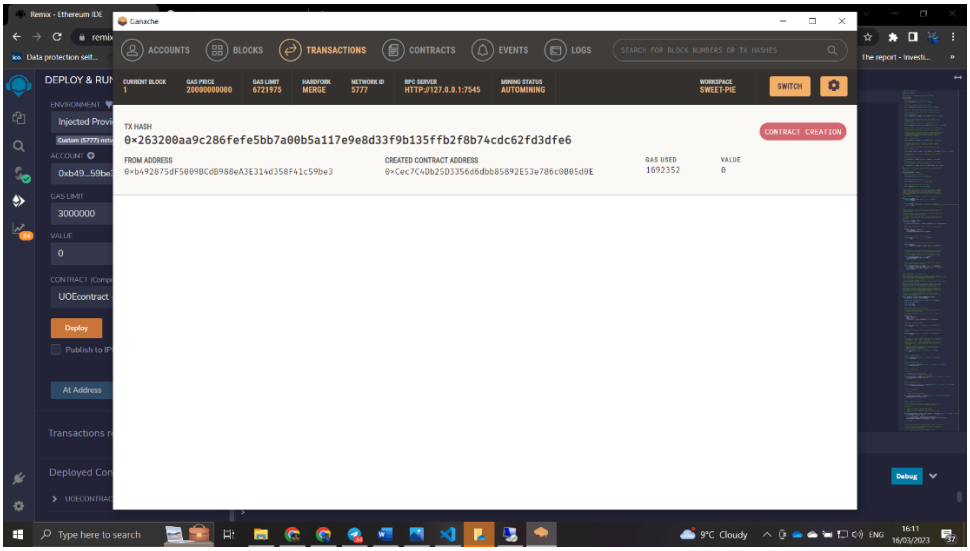
58. Congratulations you have just deployed a smart contract to a local development blockchain.



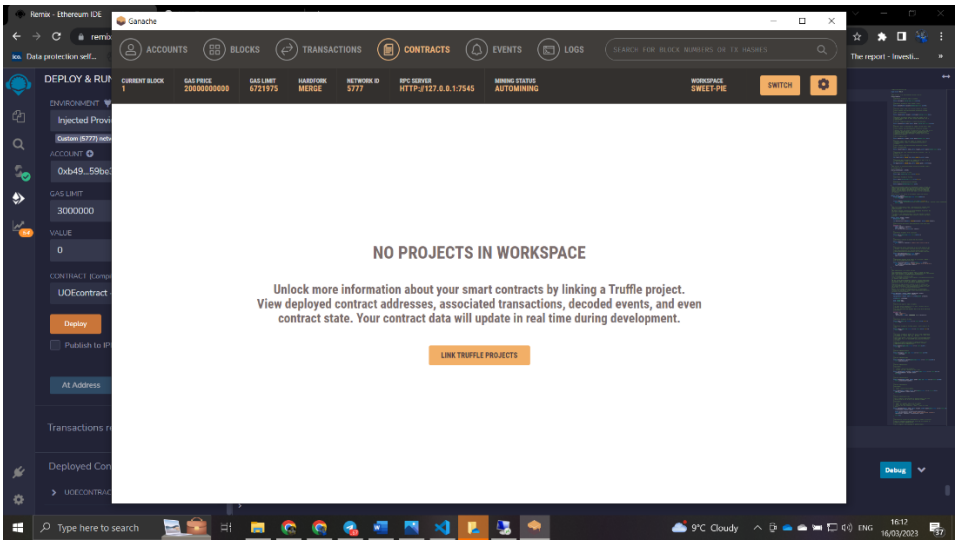
59. If you open the Ganache instance you have running you can see that the wallet you imported has a balance lower than 100 ETH and that the account now has one transaction.



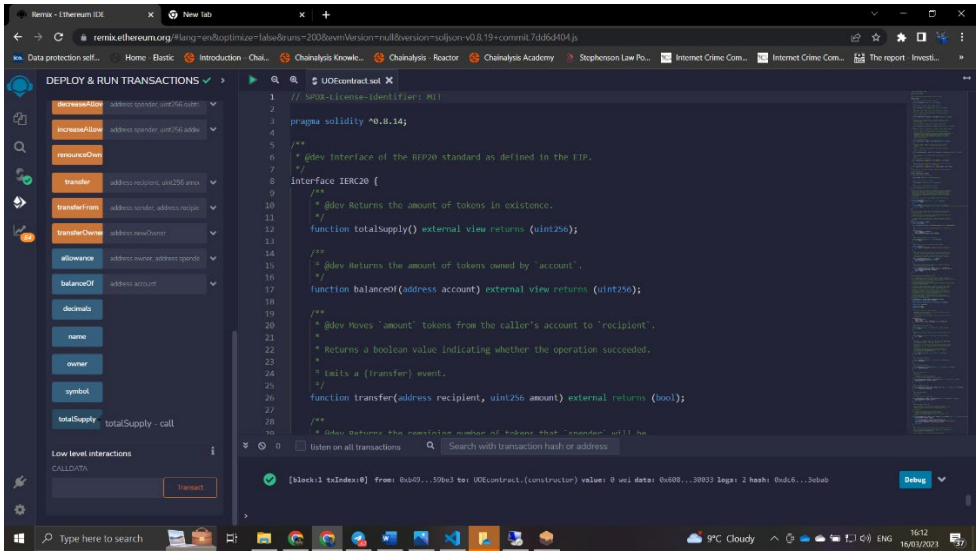
60. If you click on the ‘transactions’ tab at the top of the Ganache window you can see that the transaction was for a contract creation.



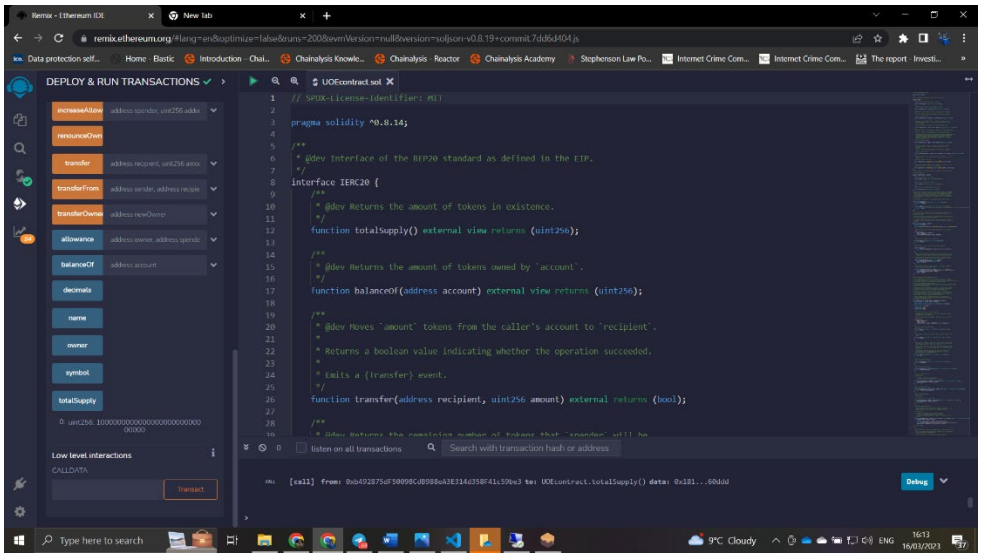
61. If we were using the Truffle framework we would have a lot more information to see.



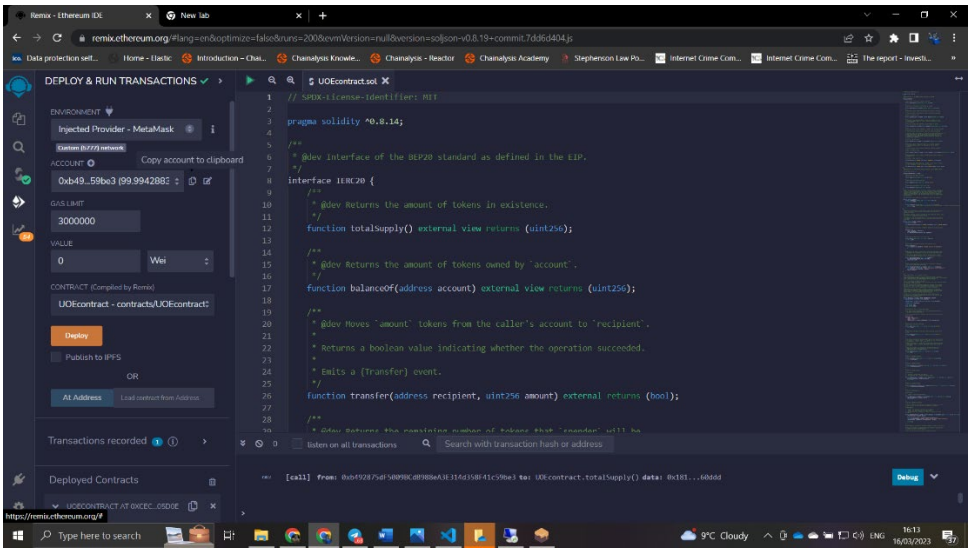
62. For this exercise we will go back to RemixIDE and select the newly deployed contract to see the options available.



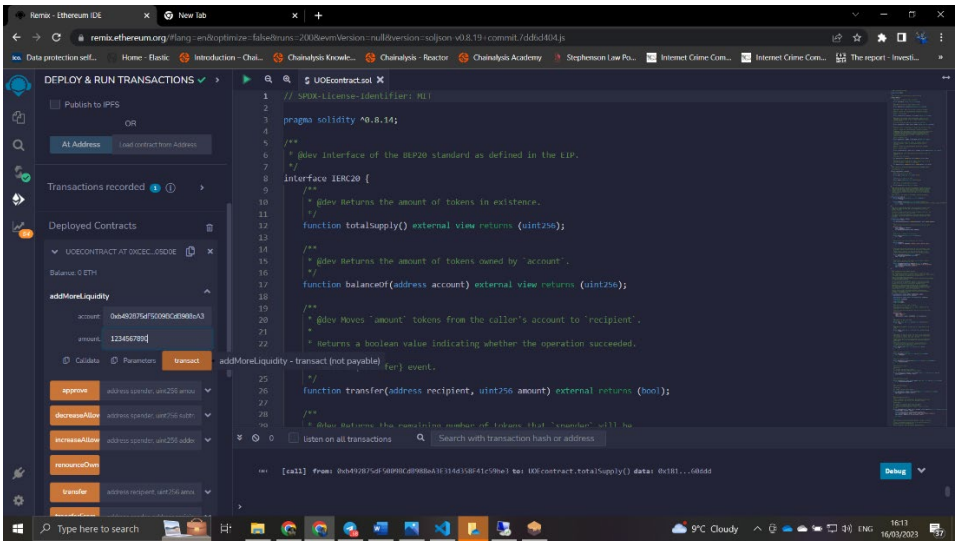
63. Click on the blue ‘totalSupply’ button it will return a value of ‘100000000000000000000000000000’ as shown in the image below.



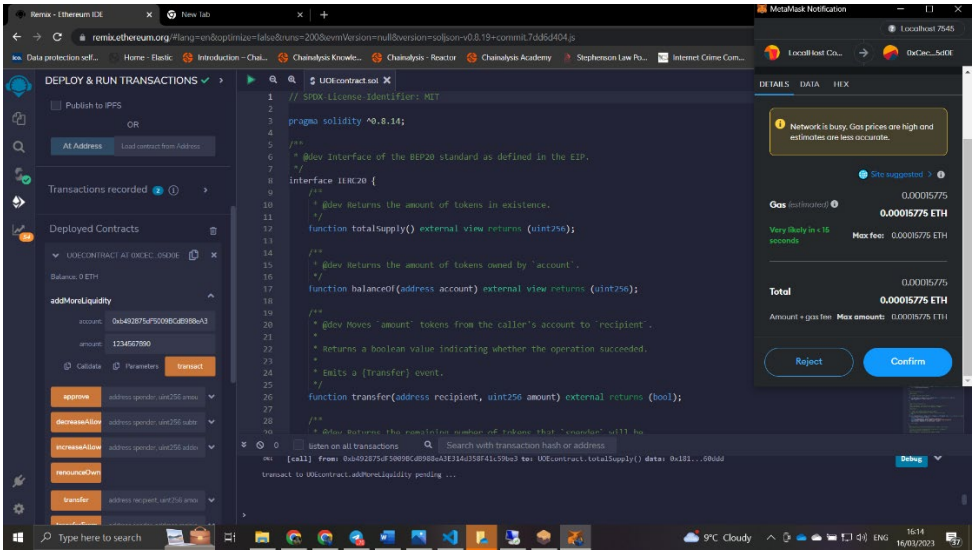
64. Scroll back to the top of this section and copy the ‘Account’ to your clipboard.



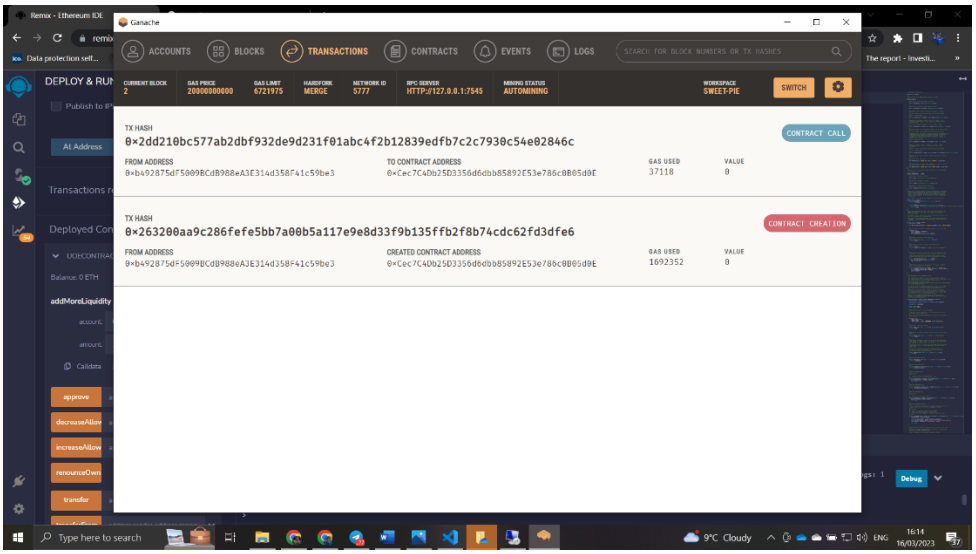
65. In the ‘Deployed Contracts’ section, expand the ‘addMoreLiquidity’ function and paste the copied account into the ‘account’ field. Then put a random value in the ‘amount’ field. I used ‘1234567890’ to start.



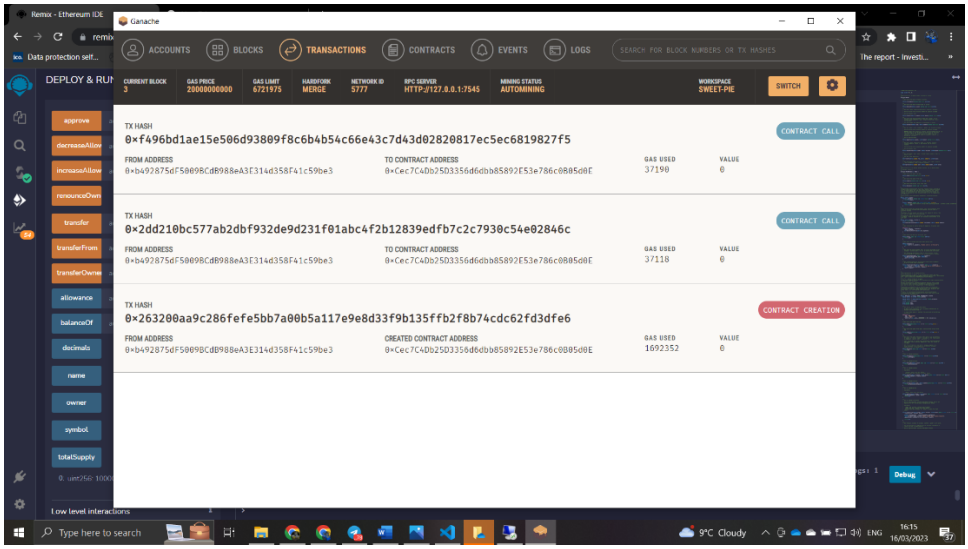
66. Click the orange ‘Transact’ button and the Metamask add-on will open requesting a transition signature. Click the ‘Confirm’ Button.



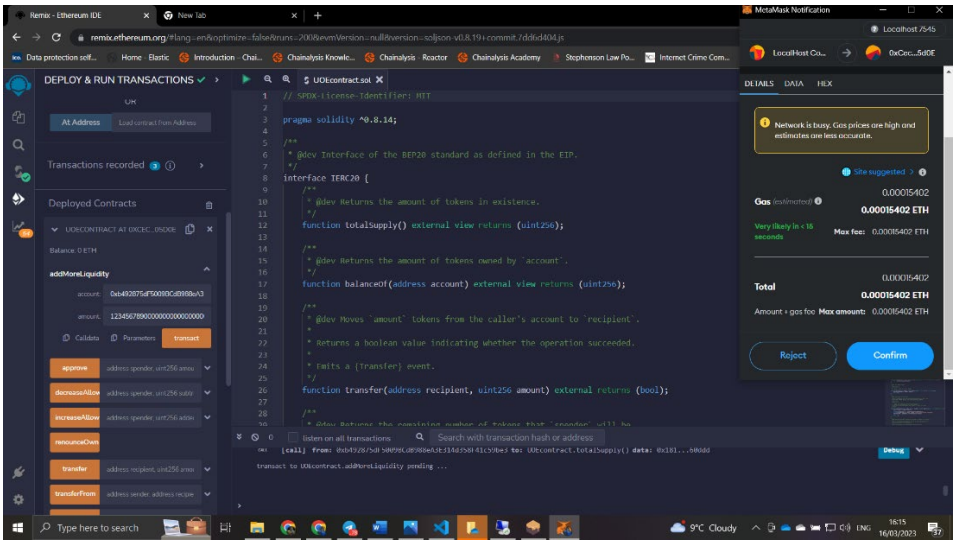
67. If you open up the Ganache application again, in the ‘transactions’ tab you will see another Tx Hash for a ‘Contract Call’.



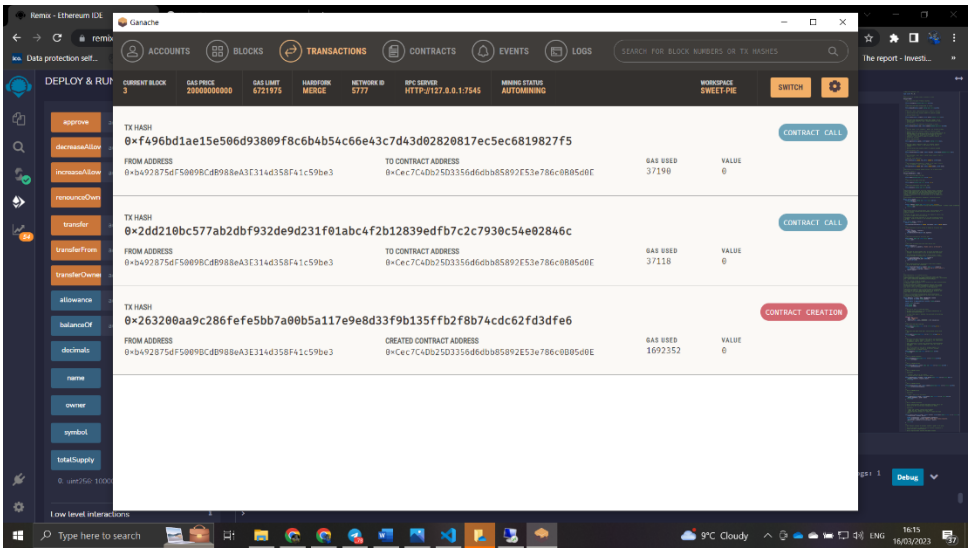
68. Go back to RemixIDE and call the blue ‘totalSupply’ function button to see what value is returned now.



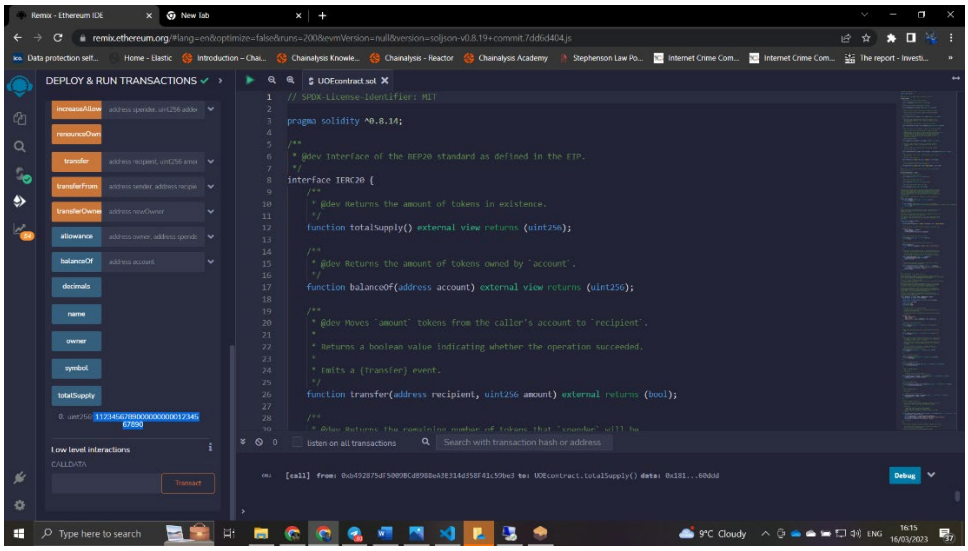
69. To double check the ‘addMoreLiquidity’ function is able to mint again go back to the function and add as many extra zeros to the end of the ‘amount’ value. Then click the orange ‘Transact’ button to open the Metamask signature request again, like in the image below.



70. Open up Ganache again to see another Tx Hash for a ‘Contract Call’.



71. Go back to RemixIDE and call the blue ‘totalSupply’ function button to see what value is returned now. You will see it has increased again confirming this is a hidden mint function.



72. Congratulations you have successfully completed this exercise.

Conclusion

In conclusion, this exercise has provided a comprehensive walkthrough on the basics of Ethereum smart contracts and Solidity programming and analysis. You have learned how to set up a local blockchain network, create and compile a smart contract, and analyse its functions using Solidity Visual Studio. You have also learned how to connect to your Metamask wallet, deploy a smart contract using RemixIDE, and test its functions. By using the additional open-source tools, you can further improve your smart contract analysis skills. Finally, if you're interested in learning more about blockchain technology and its applications, you can check out the Chainalysis Training Academy for more resources.

Slither

ConsenSys Security Tools

Chainalysis Training Academy

Cbsecurity.solutions