

Working with Crowdfunding on Etherscan Using Brownie and Solidity

No comments



ARASHTAD
Design and Development

WORKING WITH CROWDFUNDING ON ETHERSCAN

(USING BROWNIE AND SOLIDITY)

ARASHTAD.COM



*In this tutorial, we are going to implement the **crowdfunding project folder** and compile it. Then we are going to interact with the compiled and deployed crowdfunding contract on the Etherscan. There are a number of scripts that we are going to work on. They are helpful_scripts.py, deploy.py, brownie_config.yaml, Fundme.sol, and .env file.*

Crowdfunding on Etherscan: the Essentials

If you have read the [solidity smart contracts tutorial](#), you can remember how we wrote the FundMe.sol contract and how we deployed it using Remix IDE. As you know, Remix IDE is just for learning solidity and beginners and we need some other deployment tools such as Node.js or Python web3, or brownie to be able to run it in a real-world application. To set up the Fundme.sol contract inside Brownie, we take the following steps:

1. Create a directory folder for the project:

```
mkdir brownie_fund_me
```

2. In the terminal:

```
brownie init
```

3. Inside the contracts folder, create a file called FundMe.sol and paste the FundMe smart contract that we wrote earlier in it.

Fundme.sol:

```
// SPDX-License-Identifier: MIT

pragma solidity >= 0.6.6 < 0.7.0;

import
  "@chainlink/contracts/src/v0.6/interfaces/AggregatorV3Interface.sol";
import "@chainlink/contracts/src/v0.6/vendor/SafeMathChainlink.sol";

contract FundMe {
  using SafeMathChainlink for uint256;
  mapping(address => uint256) public addressToAmountFunded;
  address[] public funders;
  address public owner;

  constructor() public {
    owner = msg.sender;
  }

  function fund() public payable {
    uint256 minimumUSD 50 * 10 ** 18;
    require(getConversionRate(msg.value)
) >= minimumUSD, "You need to spend more ETH!");
    addressToAmountFunded[msg.sender] += msg.value;
    funders.push(msg.sender);
  }

  function getVersion() public view returns (uint256){
    AggregatorV3Interface priceFeed =
    AggregatorV3Interface(0x8A753747A1Fa494EC906cE90E9f37563A8AF630e);
    return priceFeed.version();
  }

  function getPrice() public view returns(uint256){
    AggregatorV3Interface priceFeed = AggregatorV3Interface
    0x8A753747A1Fa494EC906cE90E9f37563A8AF630e);
    (,int256 answer,,) = priceFeed.latestRoundData();
    return uint256(answer * 10000000000);
  }

  function getConversionRate(uint256 ethAmount) public view returns
```

```
(uint256){
    uint256 ethPrice = getPrice();
    uint256 ethAmountInUsd = (ethPrice * ethAmount) 1000000000000000000;
return ethAmountInUsd;
}

modifier onlyOwner {
    require(msg.sender ==owner);
    _;
}

function withdraw() payable onlyOwner public {
    msg.sender.transfer(address(this).balance);
for (uint256 funderIndex=0
; funderIndex < funders.length; funderIndex++){
    address funder = funders[funderIndex];
    addressToAmountFunded[funder]0#
    }
    funders new address[](0);
}
}
```

Deploy.py

4. In the scripts folder of the directory, create a deploy.py file with the following code , so that you can interact with the smart contract:

```
from brownie import FundMe
from scripts.helpful_scripts import get_account

def deploy_fund_me():
    account = get_account()
    fund_me = FundMe.deploy({"from":account})
    print(f"Contract deployed to {fund_me.address}")

def main():
    deploy_fund_me()
```

Helpful_scripts.py

5. Create another file named helpful_scripts.py and paste the below codes in it:

```
from brownie import network, config, accounts
```

```
def get_account():
if network.show_active() == "development":
return accounts[0]
else:
return accounts.add(config["wallets"]["from_key"])
```

This piece of code helps deploy.py find the account it should connect to. In order to make it possible for deploy.py to import this file, you should create another file called `__init__.py`.

Brownie-config.yaml

6. In the main directory create a file named `brownie-config.yaml` and paste the below scripts in it.

```
dependencies:
# - @
- smartcontractkit/chainlink-brownie-contracts@1.1.1
compiler:
solc:
remappings:
- '@chainlink=smartcontractkit/chainlink-brownie-contracts@1.1.1'

dotenv: .env
wallets:
from_key: ${PRIVATE_KEY}
```

We have defined the chainlink smart contract kit as a dependency to be able to use it afterwards. We have also defined using environment variables and reading from the private key.

.env file

7. Create a `.env` file and paste your private key and Infura Rinkeby id inside of it.

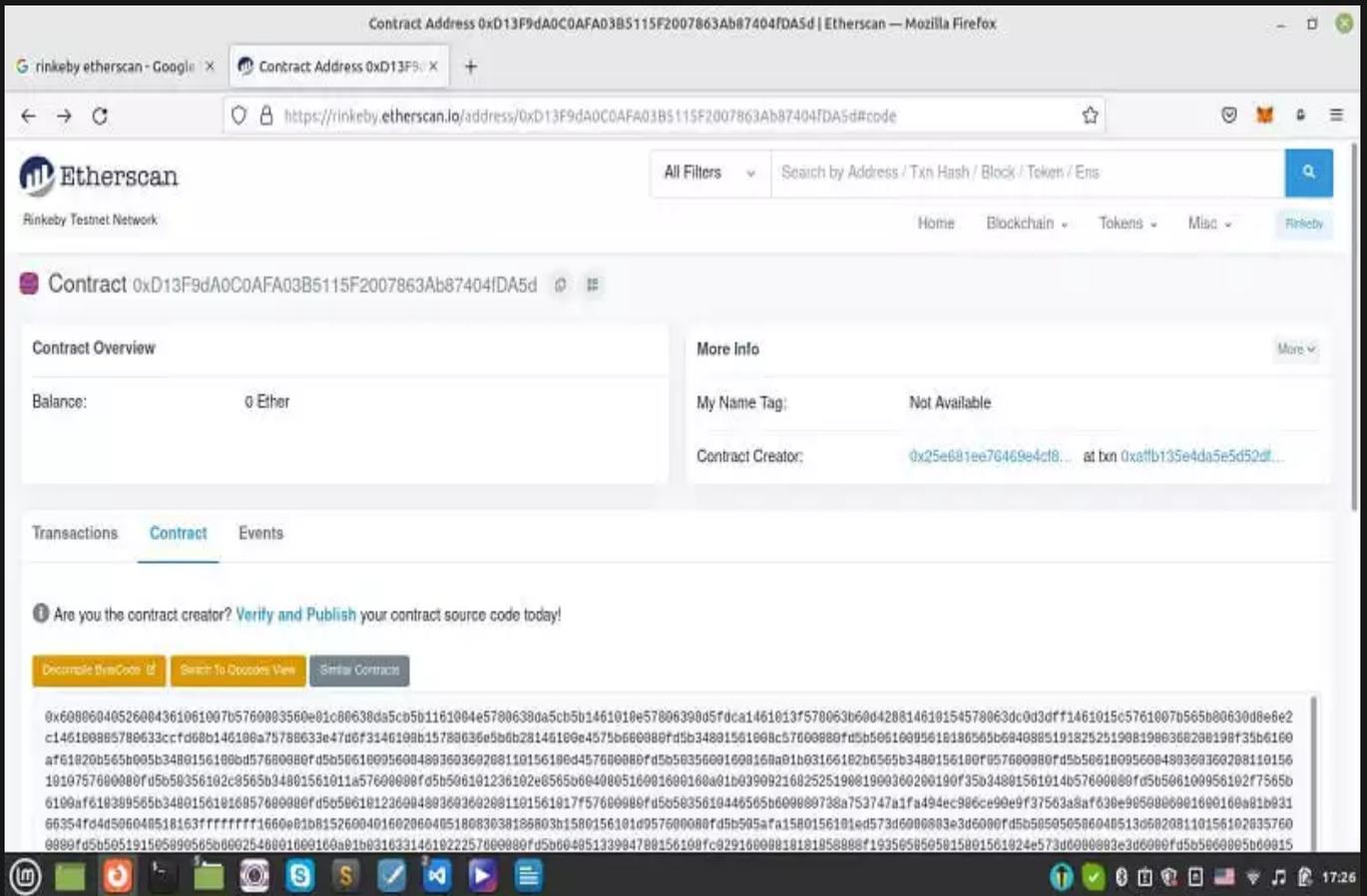
```
export PRIVATE_KEY=
export WEB3_INFURA_PROJECT_ID=
```

8. Now that all files are set up, in your terminal write:

```
brownie compile
```

Result:

```
Brownie v1.18.1 - Python development framework for EthereumDownloading
```

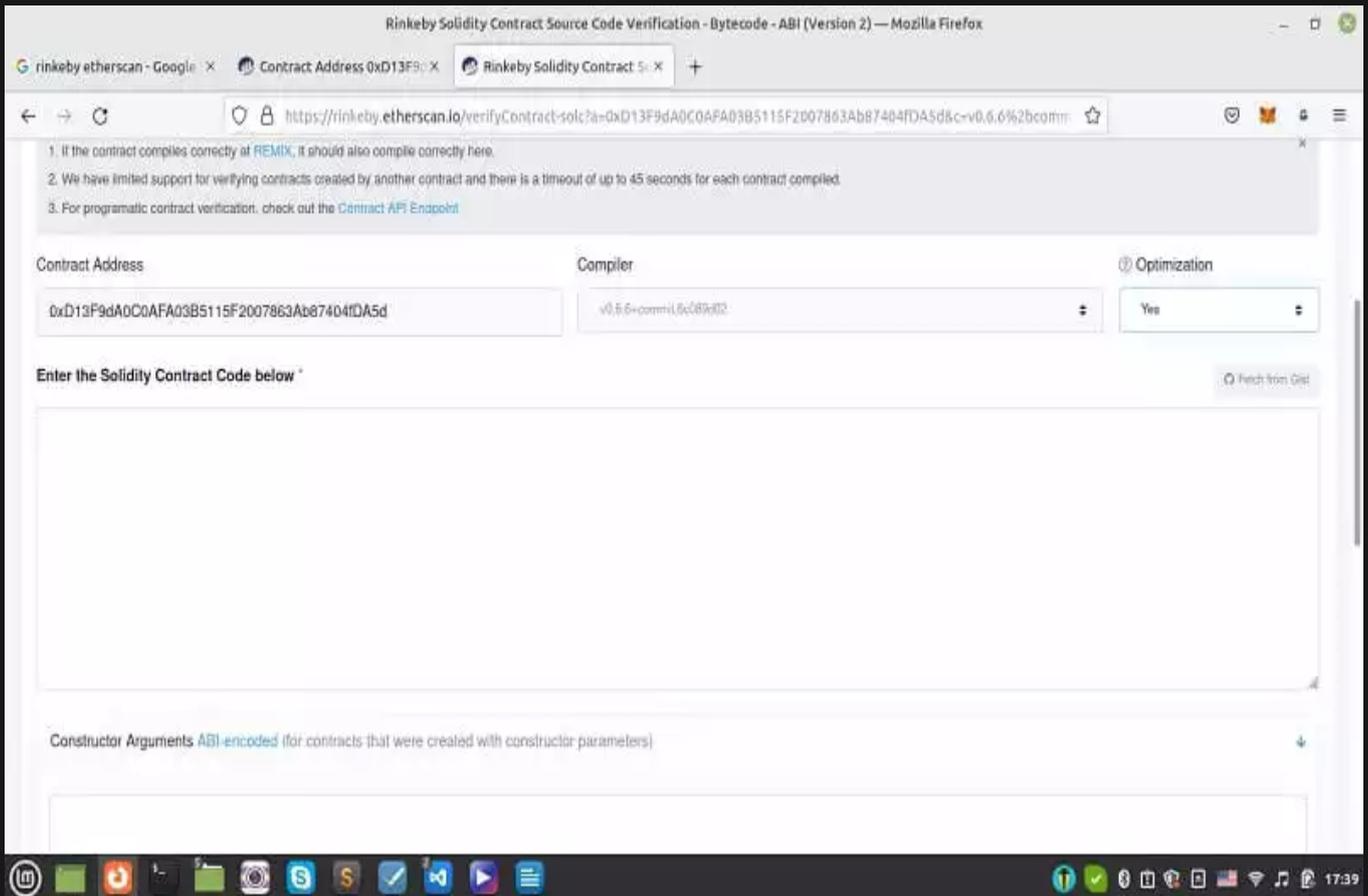
The screenshot shows the Etherscan website interface. At the top, the browser address bar displays the URL: `https://rinkeby.etherscan.io/address/0xD13F9dA0C0AFA03B5115F2007863Ab87404fDA5d#code`. The page title is "Contract Address 0xD13F9dA0C0AFA03B5115F2007863Ab87404fDA5d | Etherscan — Mozilla Firefox".

The main content area displays the contract details for `0xD13F9dA0C0AFA03B5115F2007863Ab87404fDA5d`. Under the "Contract Overview" section, the balance is shown as `0 Ether`. The "More Info" section indicates that the "My Name Tag" is "Not Available" and the "Contract Creator" is `0x25e681ee76469e4cf8...` at txn `0xa1b135e4da5e5d52df...`.

Below the overview, there are tabs for "Transactions", "Contract", and "Events". A message prompts the user: "Are you the contract creator? Verify and Publish your contract source code today!". Below this message are buttons for "Decompile this Code", "Search to Obsolete Verbs", and "Similar Contracts".

The bottom section of the page displays a long hexadecimal string representing the contract source code, starting with `0x60006040526004361061007b5760003560e01c00638da5cb5b1161004e5700639da5cb5b1461010e5700639005fdca1461013f570063b60d420814610154570063d0d3dff1461015c5761007b565b00630d8e62c146100885780633ccfd06b146100a75780633e47d0f3146100b15780636e5b0b28146100e4575b600080fd5b34801561008c57600080fd5b5061000956180505b68408851918252519081906360200198f35b6160af61020b565b005b3480156100bd57600080fd5b50610095600480360369208110156100d457600080fd5b503560010901600a01b03166102b6565b3480156100f057600080fd5b506100956004803603602081101561010757600080fd5b50356102c0565b34801561011a57600080fd5b506101236102e0565b604000516001600a01b039092160252519001900360200190f35b34801561014b57600080fd5b506100956102f7565b6100af610309565b34801561016857600080fd5b506101236004803603602081101561017f57600080fd5b5035610446565b000080738a753747a1fa494ec906ce90e9f37503a8af630e905080001600160001b03166354fd4d506040518163ffffffffff1660e01b015260040160206040518083038180803b15001561010957600080fd5b505afa1500156101ed573d6000803e3d6000fd5b5050505080405130602081101561020357600080fd5b505191505090565b000254800160016001b0316331461022257000080fd5b00405133904780156100f0c0291600010181058000f193565050515001561024e573d6000802e3d6000fd5b5060005b00015`.

So far so good! But we have a problem here. We are not yet able to interact with the smart contract. To do this manually, you can click on verify and publish. On the new page, you enter the contract address, compiler type and its license and press continue.

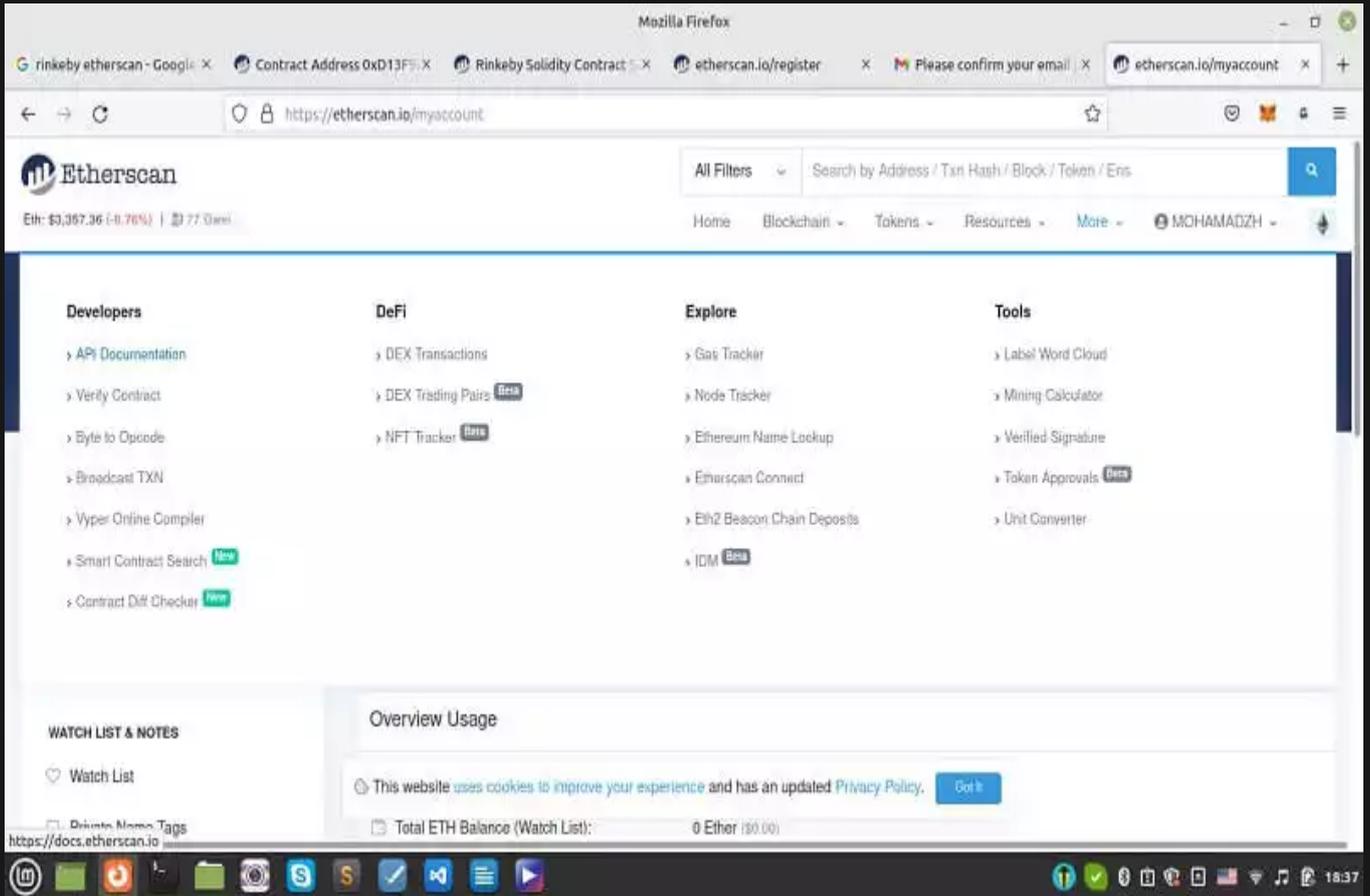


We can copy and paste our smart contract. But there is a problem, Etherscan will not be able to identify chainlink AggregatorV3Interface.sol. We have a solution for this in the next section of our tutorial.

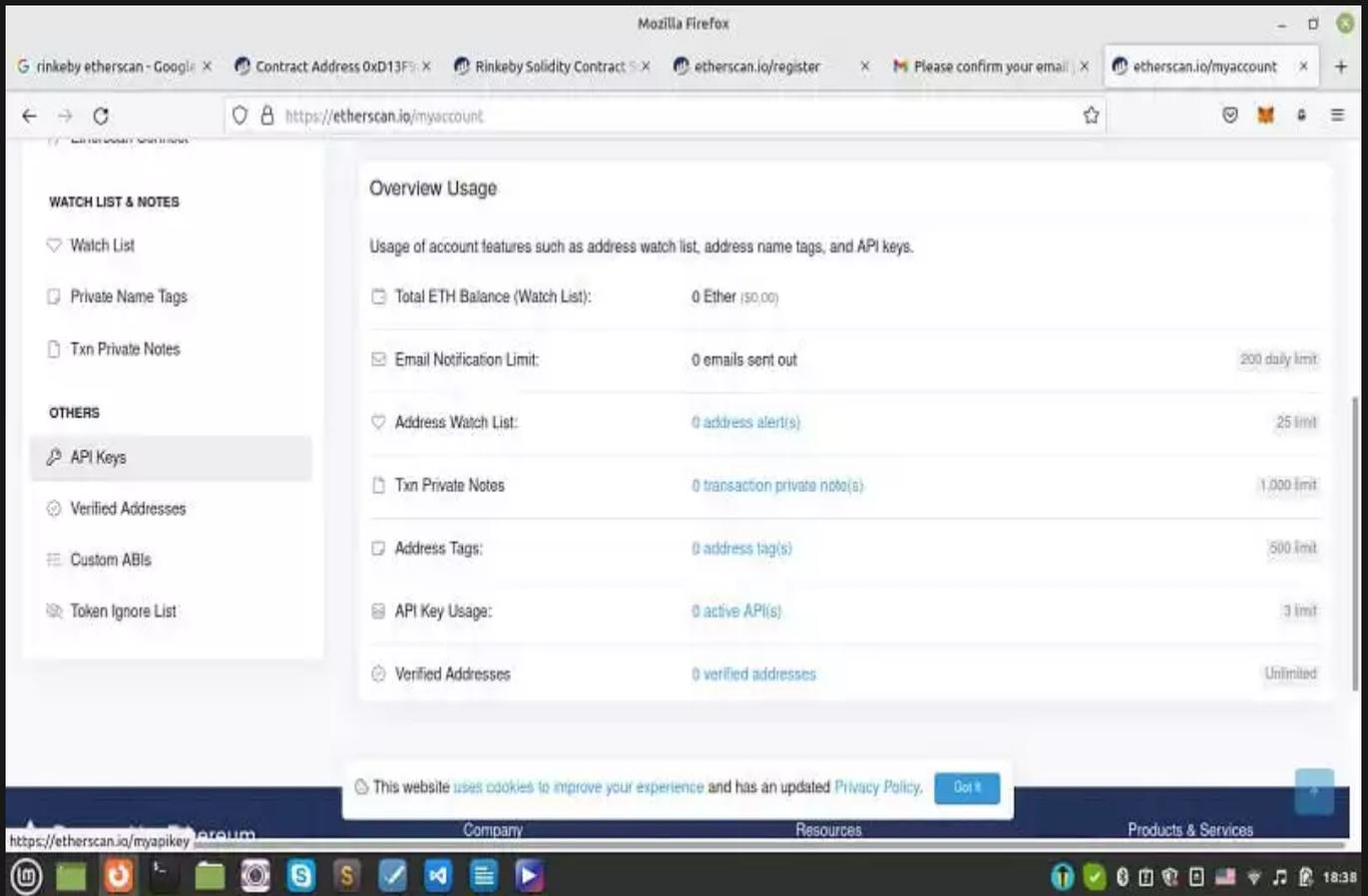
Crowdfuning on Etherscan: Getting An API Key

In this section, we are going to get an API key from etherscan.io and paste it into the .env file to be able to keep track of our crowdfunding smart contract on the Rinkeby chain Etherscan. Also, we're going to interact with the fund me smart contract with a simple-to-use interface. As you recall, we had a problem and it was that we couldn't paste our code inside the box related to smart contracts because Etherscan could not identify the chainlink Aggregator. But, there is a solution for that.

First, we should head over to etherscan.io and sign up for an account and then in the More tab Developers section click on API documentation:



Then, on the left hand side bar, click on API keys:

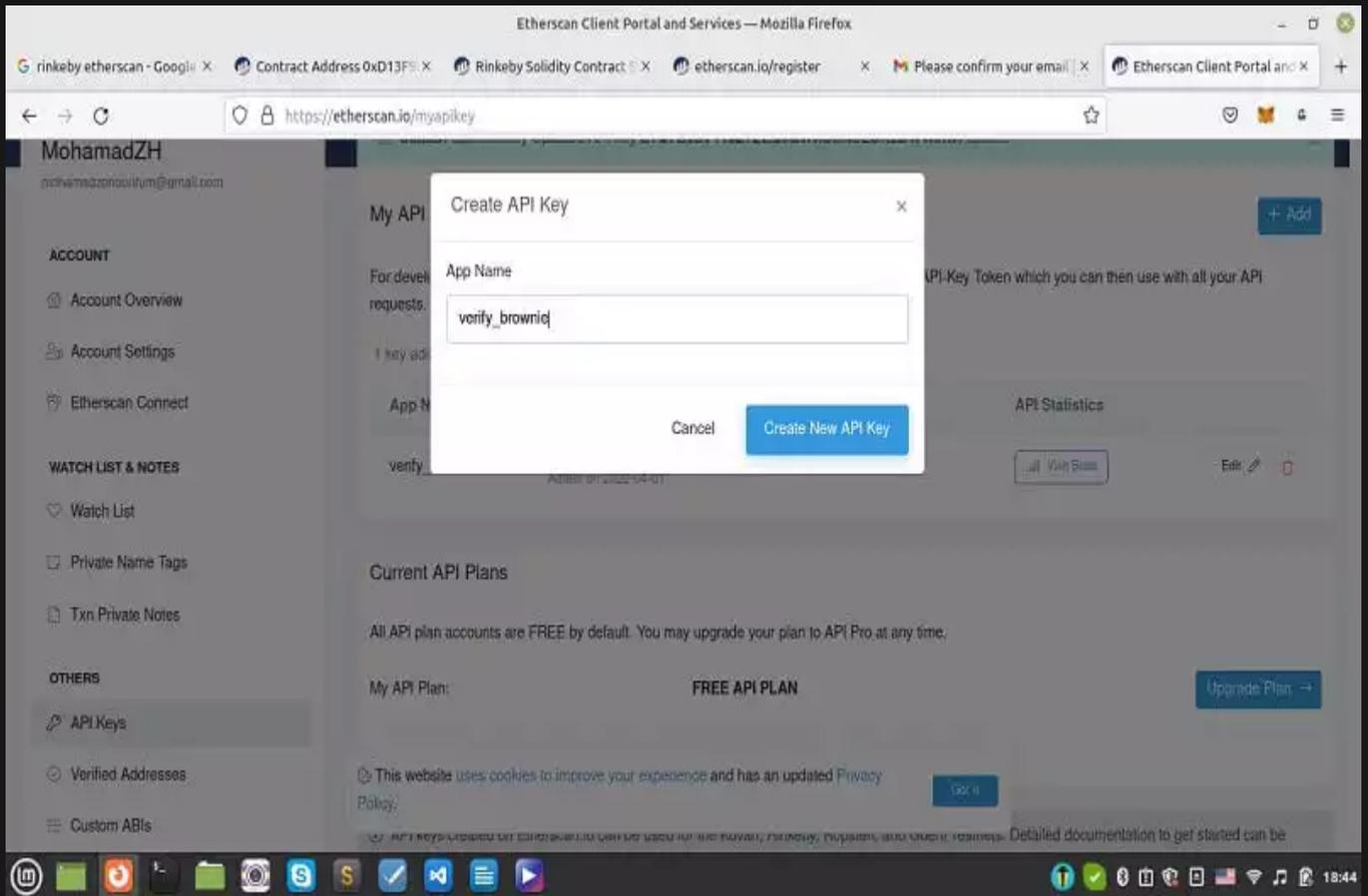


The screenshot shows the etherscan.io/myaccount page in a Mozilla Firefox browser. The page displays account usage statistics under the heading "Overview Usage". The statistics are as follows:

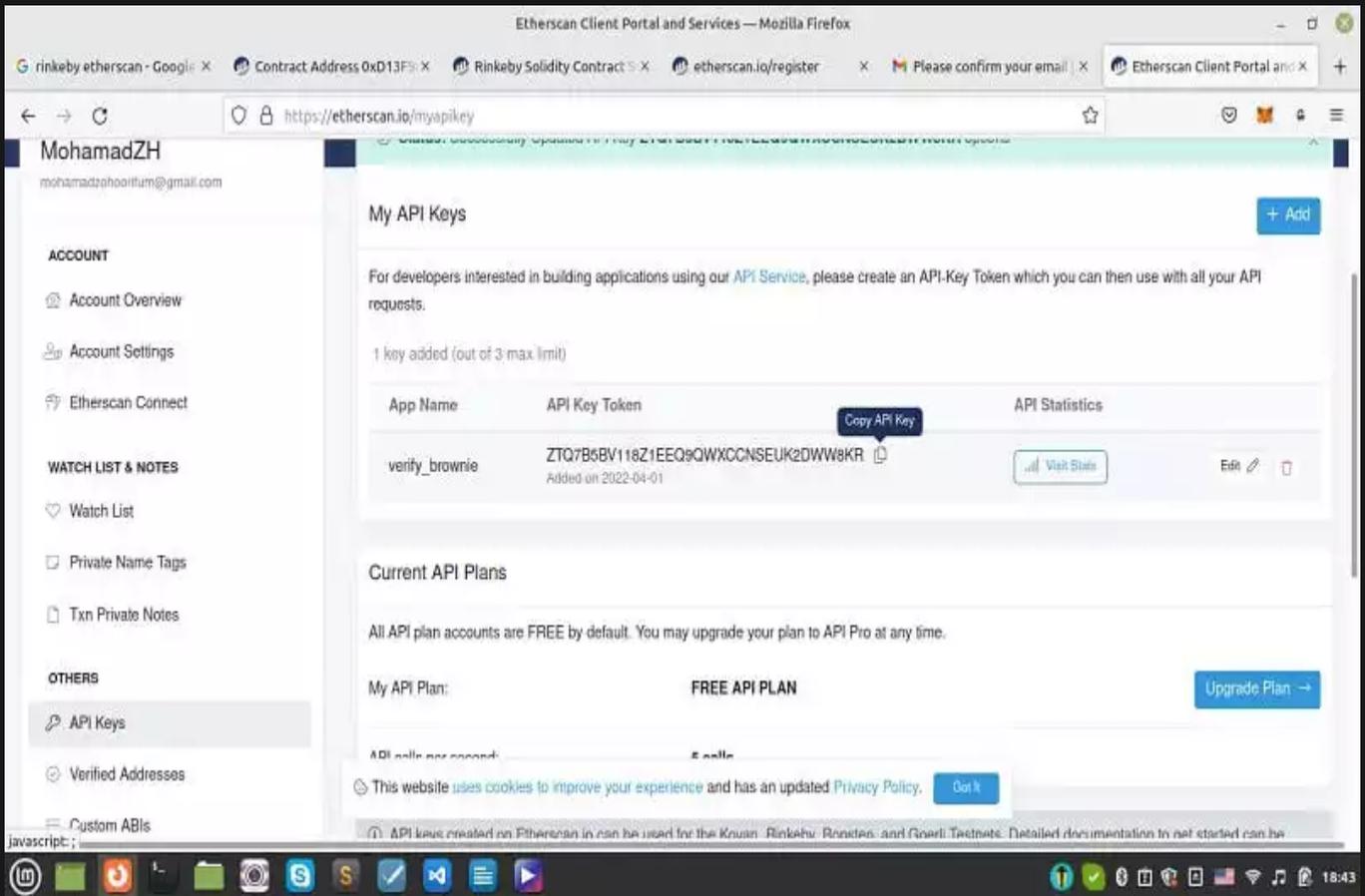
Feature	Current Usage	Limit
Total ETH Balance (Watch List)	0 Ether (\$0.00)	-
Email Notification Limit	0 emails sent out	200 daily limit
Address Watch List	0 address alert(s)	25 limit
Txn Private Notes	0 transaction private note(s)	1,000 limit
Address Tags	0 address tag(s)	500 limit
API Key Usage	0 active API(s)	3 limit
Verified Addresses	0 verified addresses	Unlimited

A cookie notice is visible at the bottom of the page: "This website uses cookies to improve your experience and has an updated Privacy Policy." with a "Got it" button.

And add an API with the name verify_brownie:



After adding an API key, copy it:



And paste it into the .env file like this:

```
export ETHERSCAN_TOKEN=
```

Also, in the deploy.py file, change:

```
fund_me = FundMe.deploy({"from":account})
```

To

```
fund_me = FundMe.deploy({"from":account},publish_source=True)
```

To be able to publish our code on Rinkeby Etherscan.

Now, it is time to run our code once more:

```
brownie run scripts/deploy.py --network rinkeby
```

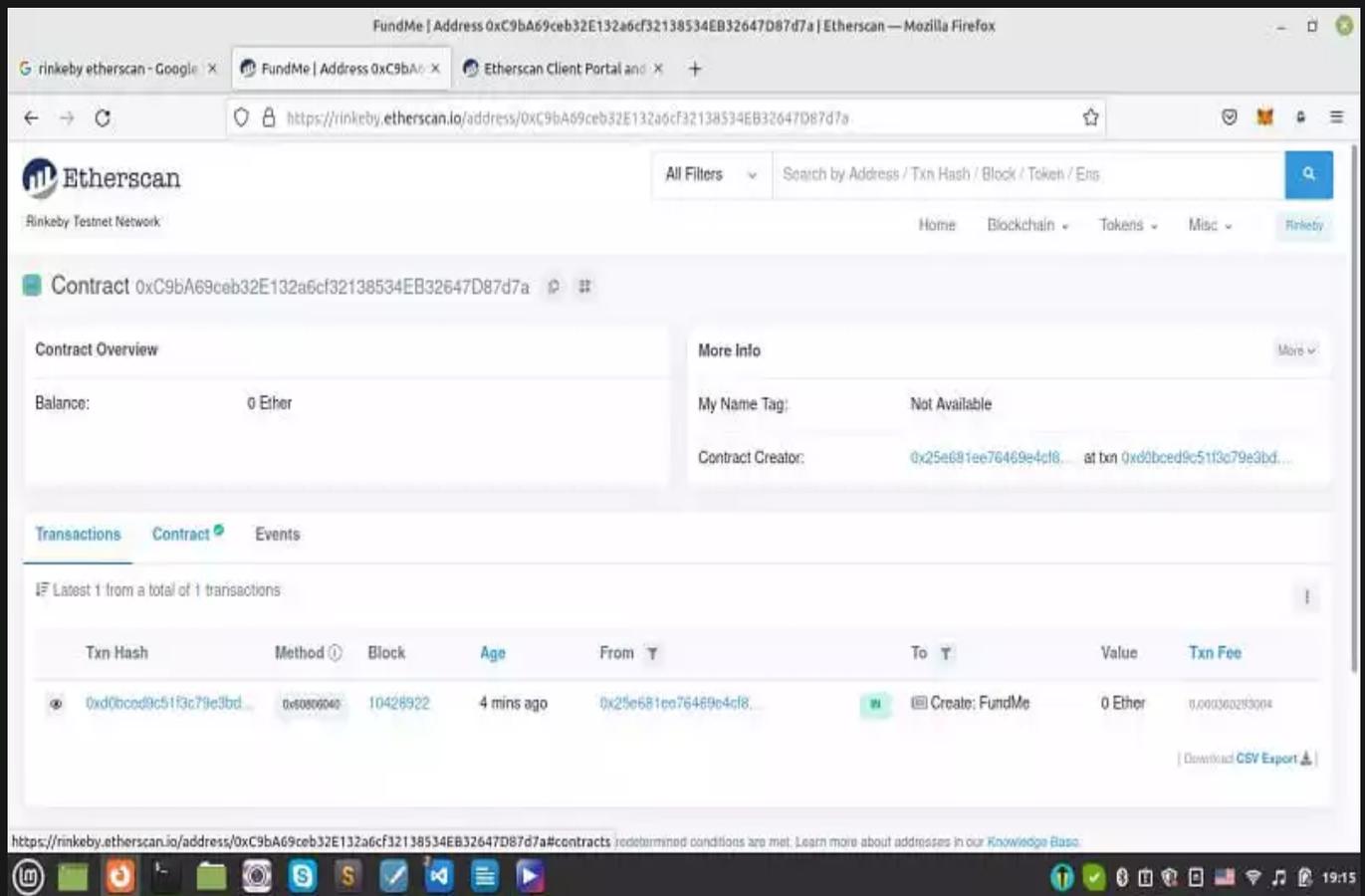
Result:

```

Brownie v1.18.1 - Python development framework for
EthereumBrownieFundMeProject is the active project.Running
'scripts/deploy.py::main'... Transaction sent:
0xd0bced9c51f3c79e3bda4a150600159480c49ab8c7e7fb57b92112b3ee4efc80 Gas
price: 1.000000013 gwei Gas limit: 396322 Nonce: 51 FundMe.constructor
confirmed Block: 10428922 Gas used: 360293 (90.91%) FundMe deployed
at: 0xC9bA69ceb32E132a6cf32138534EB32647D87d7aWaiting for https://api-
rinkeby.etherscan.io/api to process contract... Verification submitted
successfully. Waiting for result... Verification complete. Result:
Pass - Verified Contract deployed to
0xC9bA69ceb32E132a6cf32138534EB32647D87d7a
    
```

Interacting on Rinkeby Etherscan:

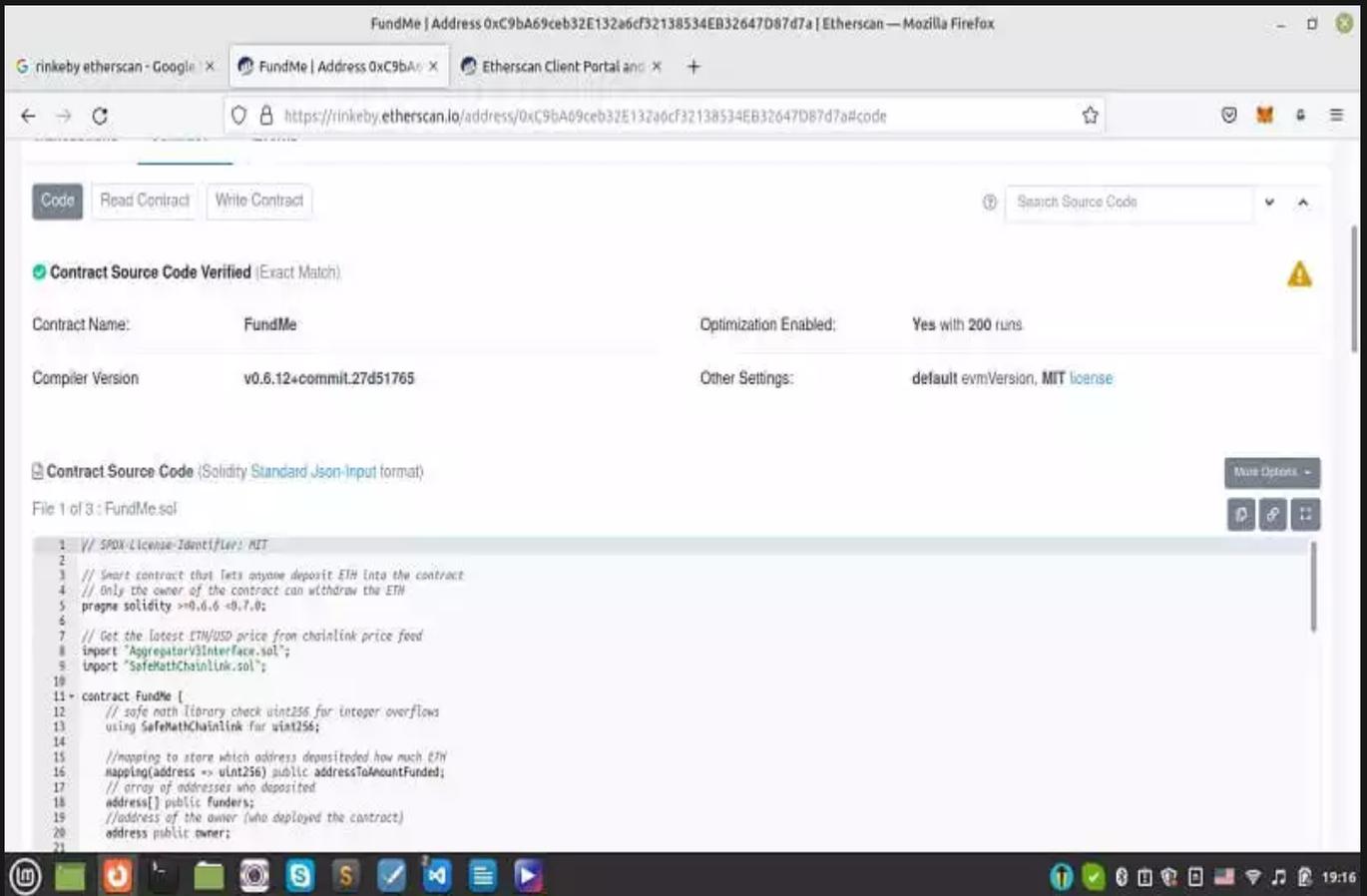
Now, if we go to Rinkeby Etherscan and paste the address of the contract (given in the terminal) in the search bar, we will be able to see that the contract tab is checked:



The screenshot shows the Etherscan interface for the Rinkeby Testnet Network. The contract address `0xC9bA69ceb32E132a6cf32138534EB32647D87d7a` is entered in the search bar. The 'Contract' tab is active, displaying the contract overview and a list of transactions. The transaction table shows a single transaction for the 'Create: FundMe' operation.

Txn Hash	Method	Block	Age	From	To	Value	Txn Fee
0xd0bced9c51f3c79e3bd...	<code>0x0806040</code>	10428922	4 mins ago	0x25e681ee76469e4cf8...	0xC9bA69ceb32E132a6cf32138534EB32647D87d7a	0 Ether	0.000360293004

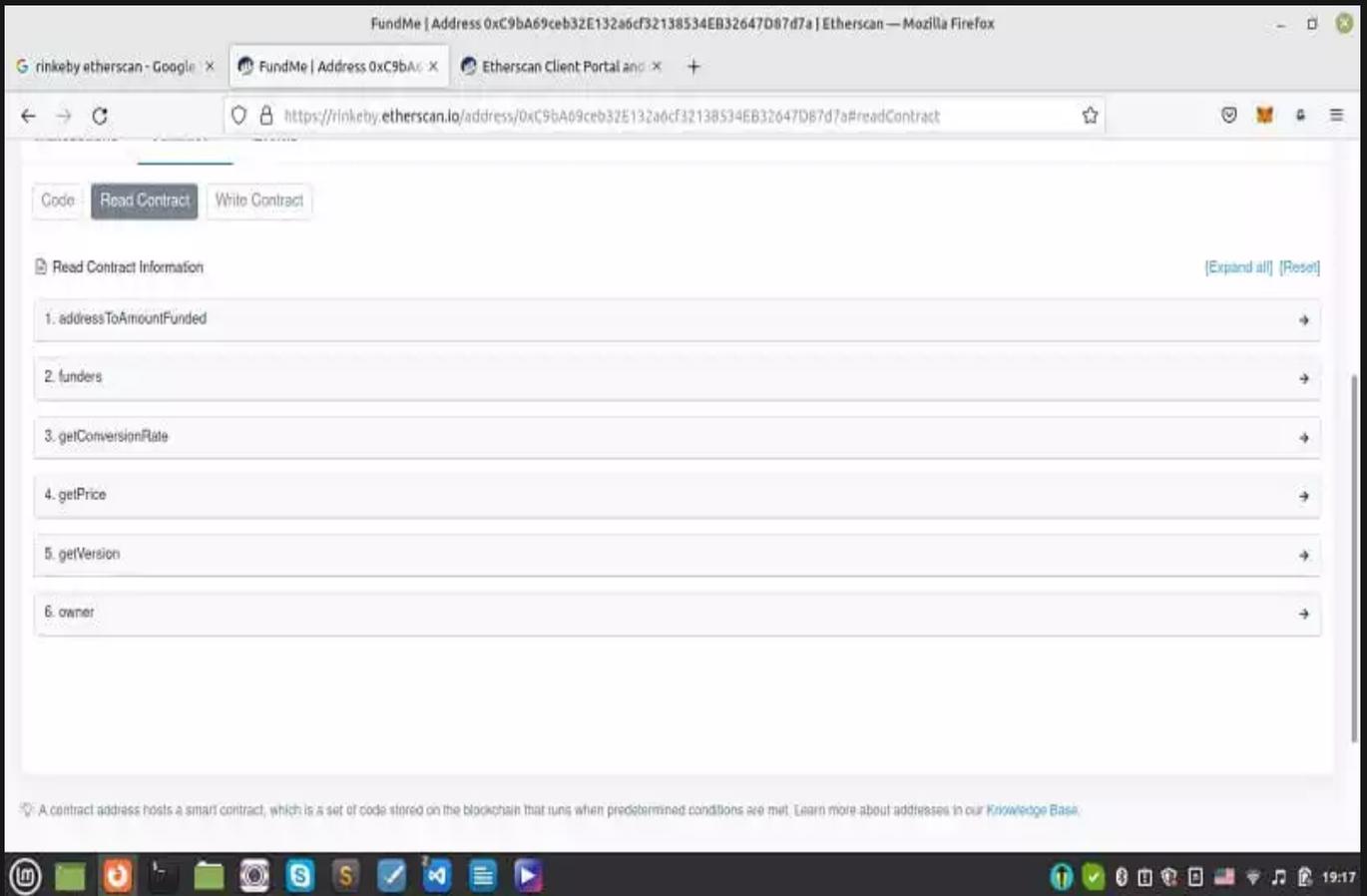
If you go to the contracts section, you will be able to see that the FundMe.sol is published with its chainlink dependency code. Also, ABI of the contract is published in another box.



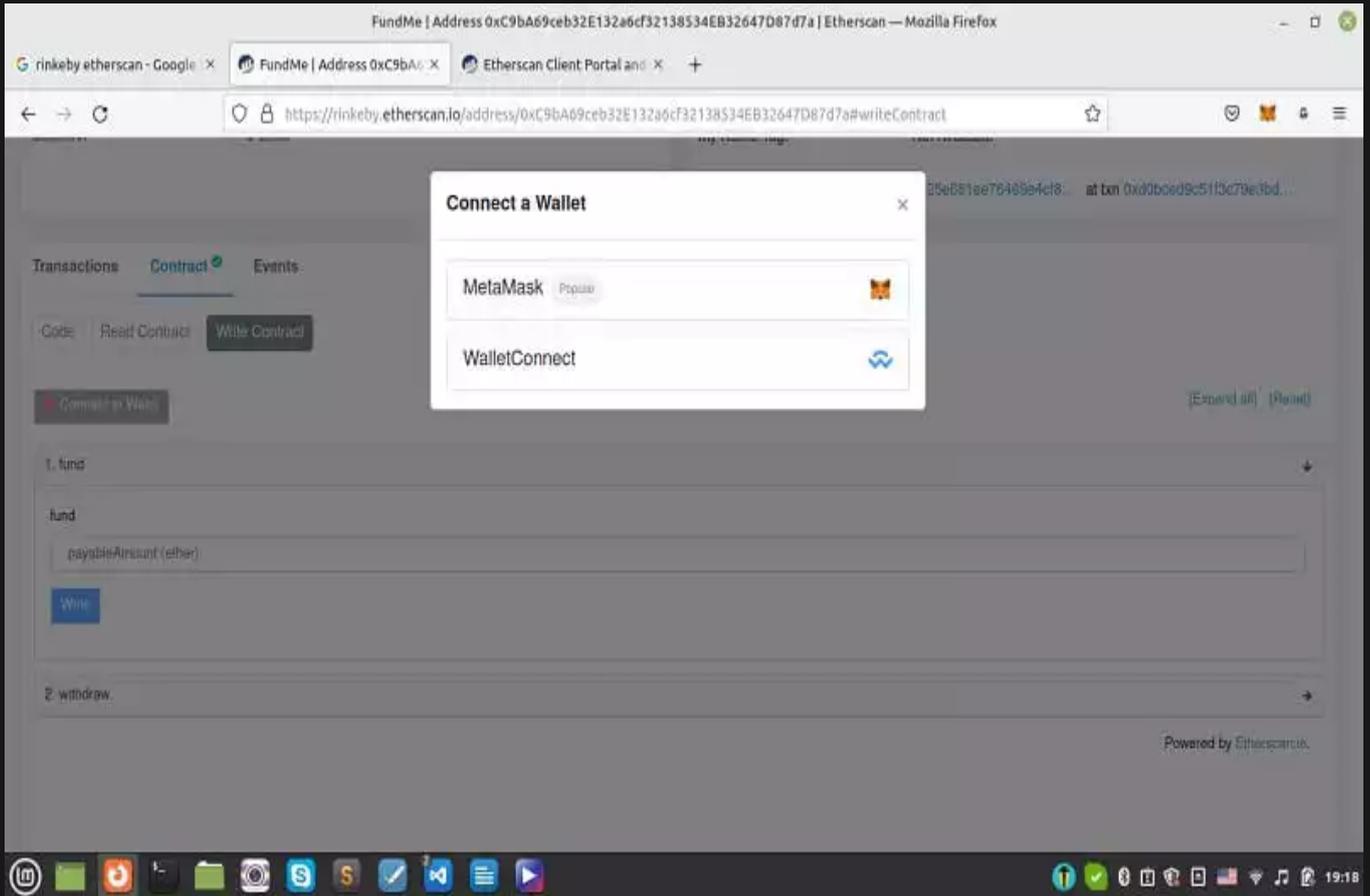
The screenshot shows the Etherscan interface for a contract named 'FundMe' at address 0xC9bA69ceb32E132a6cF32138534EB32647D87d7a. The 'Contract Source Code Verified' status is confirmed with an 'Exact Match'. The contract name is 'FundMe', and optimization is enabled with 200 runs. The compiler version is v0.6.12+commit.27d51765. The source code is displayed in Solidity Standard Json-Input format, showing a contract that allows anyone to deposit ETH and only the owner to withdraw it, using Chainlink for price feeds.

```
1 // SPDX-License-Identifier: MIT
2
3 // Smart contract that lets anyone deposit ETH into the contract
4 // Only the owner of the contract can withdraw the ETH
5 pragma solidity >=0.6.6 <0.7.0;
6
7 // Get the latest ETH/USD price from chainlink price feed
8 import "AggregatorV3Interface.sol";
9 import "SafeMathChainlink.sol";
10
11 contract FundMe {
12     // safe math library check uint256 for integer overflow
13     using SafeMathChainlink for uint256;
14
15     //mapping to store which address deposited how much ETH
16     mapping(address => uint256) public addressToAmountFunded;
17     // array of addresses who deposited
18     address[] public funders;
19     //address of the owner (who deployed the contract)
20     address public owner;
21 }
```

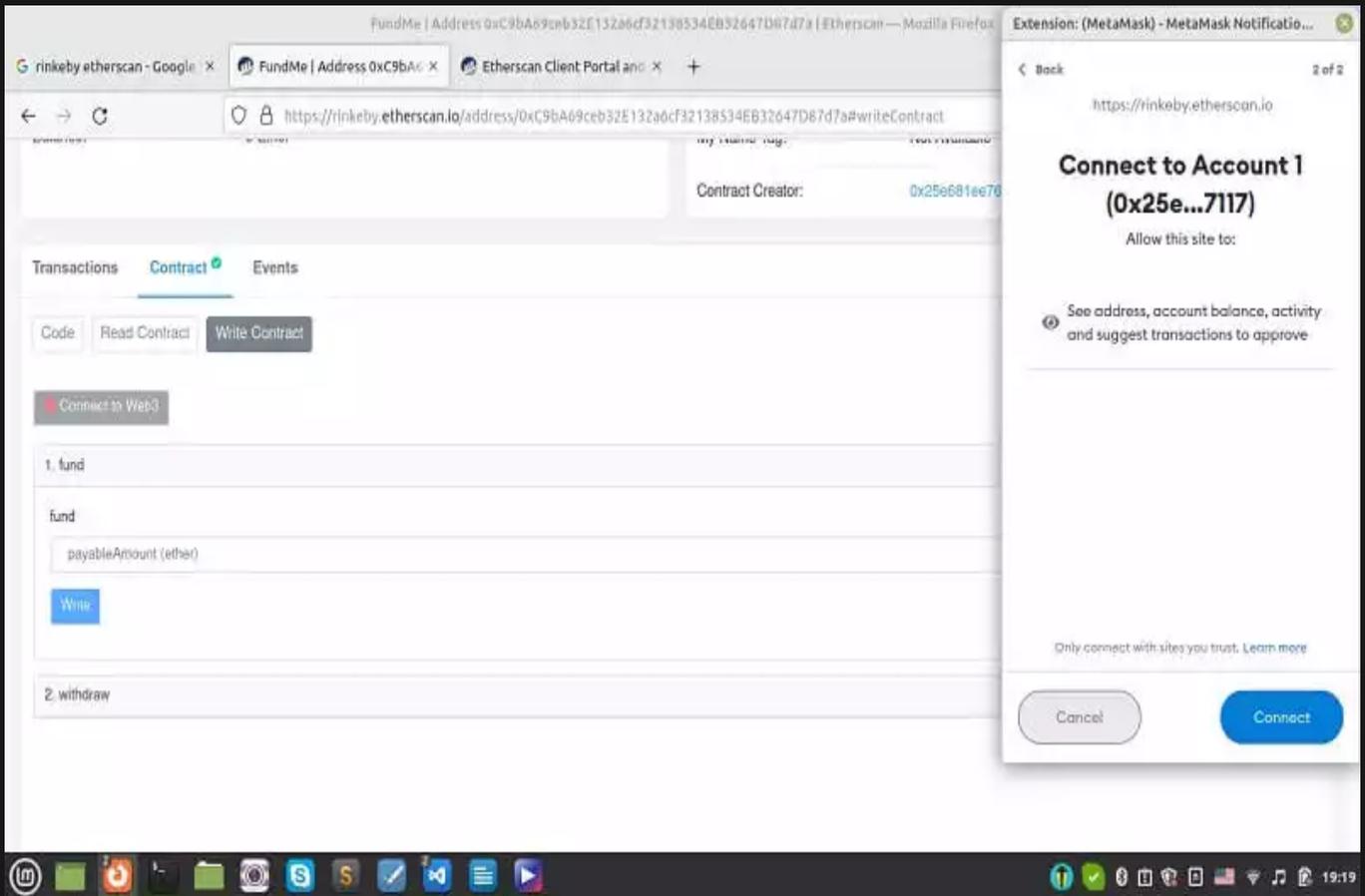
If we try posting the contract and press any of the keys related to public data, we will be able to retrieve them.



Also, if we go to write the contract and click on connect to web3, we can connect our Metamask wallet to Etherscan and fund the project using the test ethers in the Rinkeby account.

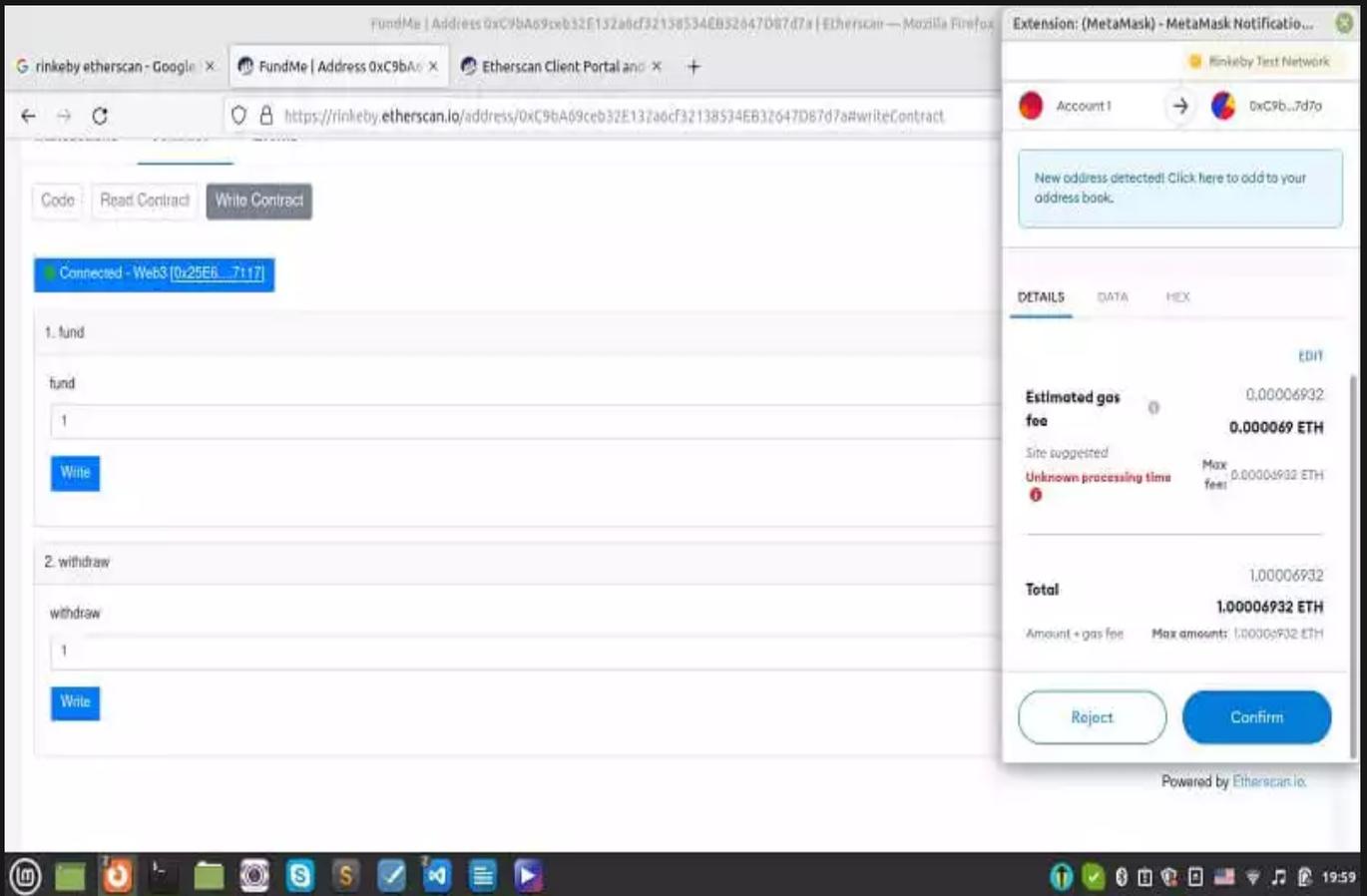


The screenshot shows a web browser window with the URL `https://rinkeby.etherscan.io/address/0xC9bA69ceb32E132a6cF32138534EB32647D87d7a#writeContract`. The page is titled "FundMe | Address 0xC9bA69ceb32E132a6cF32138534EB32647D87d7a | Etherscan — Mozilla Firefox". A modal window titled "Connect a Wallet" is open, showing two options: "MetaMask" (Popular) and "WalletConnect". The background page shows a "Contract" tab with a "Write Contract" button and a code editor containing the function `fund payableAmount ether`. The interface also includes tabs for "Transactions", "Contract", and "Events", and a "Contract" section with a "Write" button. The bottom of the browser shows a taskbar with various application icons and a system tray with the time 19:18.



The screenshot shows a web browser window with the URL `https://rinkeby.etherscan.io/address/0xC9bA69ceb32E132a6cF32138534EB32647D87d7a#writeContract`. The page displays a "Contract Creator" field with the address `0x25e681ee70...`. Below this, there are tabs for "Transactions", "Contract", and "Events". Under the "Contract" tab, there are buttons for "Code", "Read Contract", and "Write Contract". A "Connect to Web3" button is also visible. The "Write Contract" section has a "fund" event type selected, with a "payableAmount (ether)" input field containing the number "1" and a "Write" button. A "withdraw" event type is also listed below. A MetaMask confirmation pop-up is overlaid on the right side of the browser window, titled "Connect to Account 1 (0x25e...7117)". The pop-up asks for permission to "See address, account balance, activity and suggest transactions to approve" and provides "Cancel" and "Connect" buttons.

You can enter 1 in the box under fund and press the Write button and confirm the Metamask pop-up:



rinkeby etherscan - Google | FundMe | Address 0xC9bA... | Etherscan Client Portal and +

https://rinkeby.etherscan.io/address/0xC9bA69ceb32E132a6cF32138534EB32647D87d7a#writeContract

Code | Read Contract | Write Contract

Connected - Web3 [0x25E6...7117]

1. fund

fund

1

Write

2. withdraw

withdraw

1

Write

Extension: (MetaMask) - MetaMask Notificatio...

Rinkeby Test Network

Account 1 | 0xC9b...7d7a

New address detected! Click here to add to your address book.

DETAILS | DATA | HEX

EDIT

Estimated gas fee 0.00006932
0.000069 ETH

Site suggested
Unknown processing time Max fee: 0.00006932 ETH

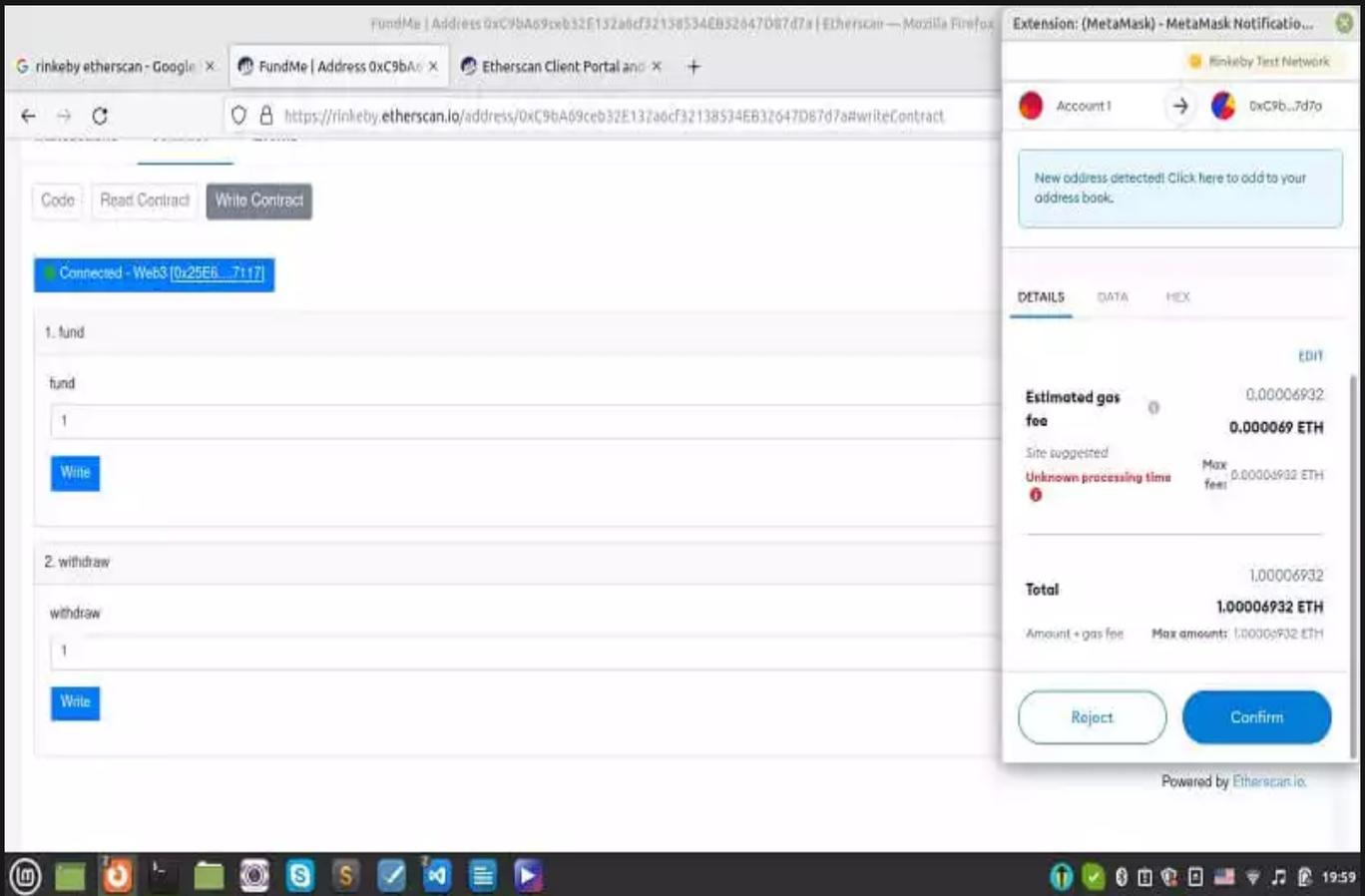
Total 1.00006932
1.00006932 ETH

Amount + gas fee Max amount: 1.00006932 ETH

Reject Confirm

Powered by Etherscan.io.

Once, the transaction has been confirmed, we can withdraw that 1 ether back into our account:



By doing the exact same process for the Mainnet account, we will be able to write and deploy a real-world smart contract!

Conclusion

In this article, we have managed to create and organize the crowdfunding project folder and compile it. Then, we started the interaction with the compiled and deployed the crowdfunding contract using the Etherscan. There are a number of scripts that we have worked on such as helpful_scripts.py, deploy.py, and brownie_config.yaml, Fundme.sol and .env file.

Finally, we have managed to completely interact with the Fundme.sol smart contract using the address we have got in the terminal at the time we de-ployed the Fundme.sol contract. In this interaction which is on the Rinkeby chain (Ethereum test network), we can fund the contract and withdraw the funds as the admin.

Join Arashtad Community

Follow Arashtad on Social Media

We provide variety of content, products, services, tools, tutorials, etc. Each social profile according to its features and purpose can cover only one or few parts of our updates. We can not upload our videos on SoundCloud or provide our eBooks on Youtube. So, for not missing any high quality original content that we provide on various social networks, make sure you follow us on as many social networks as you're active in. You can find out Arashtad's profiles on different social media services.



Get Even Closer!

Did you know that only one universal Arashtad account makes you able to log into all Arashtad network at once? Creating an Arashtad account is free. Why not to try it? Also, we have regular updates on our newsletter and feed entries. Use all these beneficial free features to get more involved with the community and enjoy the many products, services, tools, tutorials, etc. that we provide frequently.

[SIGN UP](#)[NEWSLETTER](#)[RSS FEED](#)