Plutus

Public

add quick start to help
+2 −0

Isaacdelly authored on Dec 7, 2022

Watch 91 Fork 569 Star 1.4k

master Add file Code

Isaacdelly add quick start to help e791710 · 3 years ago 110 Commits

database	Update README.md	3 years ago
README.md	Add files via upload	3 years ago
plutus.py	add quick start to help	3 years ago
plutus.txt	Add files via upload	3 years ago
requirements.txt	Add files via upload	3 years ago

README

Plutus Bitcoin Brute Forcer

A Bitcoin wallet collider that brute forces random wallet addresses

Like This Project? Give It A Star

1.4k

Dependencies

Python 3.9 or higher

Python modules listed in the requirements.txt

If you have a **Linux** or **MacOS** operating system, libgmp3-dev is required. If you have **Windows** then this is not required. Install by running the command:

```
sudo apt-get install libgmp3-dev
```

Installation

```
git clone https://github.com/Isaacdelly/Plutus.git plutus
```

```
cd plutus && pip3 install -r requirements.txt
```

Quick Start

```
python3 plutus.py
```

Proof Of Concept

A private key is a secret number that allows Bitcoins to be spent. If a wallet has Bitcoins in it, then the private key will allow a person to control the wallet and spend whatever balance the wallet has. So this program attempts to find Bitcoin private keys that correlate to wallets with positive balances. However, because it is impossible to know which private keys control wallets with money and which private keys control empty wallets, we have to randomly look at every possible private key that exists and hope to find one that has a balance.

This program is essentially a brute forcing algorithm. It continuously generates random Bitcoin private keys, converts the private keys into their respective wallet addresses, then checks the balance of the addresses. If a wallet with a balance is found, then the private key, public key and wallet address are saved to the text file `plutus.txt` on the user's hard drive. The ultimate goal is to randomly find a wallet with a balance out of the 2^{160} possible wallets in existence.

How It Works

32 byte hexidecimal strings are generated randomly using `os.urandom()` and are used as our private keys.

The private keys are converted into their respective public keys using the `fastecdsa` python library. This is the fastest library to perform secp256k1 signing. If you run this on Windows then `fastecdsa` is not supported, so instead we use `starkbank-ecdsa` to generate public keys. The public keys are converted into their Bitcoin wallet addresses using the `binascii` and `hashlib` standard libraries.

A pre-calculated database of every funded P2PKH Bitcoin address is included in this project. The generated address is searched within the database, and if it is found that the address has a balance, then the private key, public key and wallet address are saved to the text file `plutus.txt` on the user's hard drive.

This program also utilizes multiprocessing through the `multiprocessing.Process()` function in order to make concurrent calculations.

Efficiency

It takes `0.002` seconds for this progam to brute force a **single** Bitcoin address.

However, through `multiprocessing.Process()` a concurrent process is created for every CPU your computer has. So this program can brute force a single address at a speed of `0.002 ÷ cpu_count()` seconds.

Database FAQ

An offline database is used to find the balance of generated Bitcoin addresses. Visit /database for information.

Parameters

This program has optional parameters to customize how it runs:

help: `python3 plutus.py help`
Prints a short explanation of the parameters and how they work

time: `python3 plutus.py time`
Brute forces a single address and takes a timestamp of how long it took - used for speed testing purposes

verbose: 0 or 1
`python3 plutus.py verbose=1` : When set to 1, then every bitcoin address that gets bruteforced will be printed to the terminal. This has the potential to slow the program down

`python3 plutus.py verbose=0` : When set to 0, the program will not print anything to the terminal and the bruteforcing will work silently. By default verbose is set to 0

substring: `python3 plutus.py substring=8` : To make the program memory efficient, the entire bitcoin address is not loaded from the database. Only the last <**substring**> characters are loaded. This significantly reduces the amount of RAM required to run the program. if you still get memory errors then try making this number smaller, by default it is set to 8. This opens us up to getting false positives (empty addresses mistaken as funded) with a probability of 1/(16^<**substring**>), however it does NOT leave us vulnerable to false negatives (funded addresses being mistaken as empty) so this is an acceptable compromise.

cpu_count: `python3 plutus.py cpu_count=1` : number of cores to run concurrently. More cores = more resource usage but faster bruteforcing. Omit this parameter to run with the maximum number of cores

By default the program runs using `python3 plutus.py verbose=0 substring=8` if nothing is passed.

Expected Output

If a wallet with a balance is found, then all necessary information about the wallet will be saved to the text file `plutus.txt` . An example is:

> hex private key: 5A4F3F1CAB44848B2C2C515AE74E9CC487A9982C9DD695810230EA48B1DCEADD
> WIF private key: 5JW4RCAXDbocFLK9bxqw5cbQwuSn86fpbmz2HhT9nvKMTh68hjm
> public key:
> 04393B30BC950F358326062FF28D194A5B28751C1FF2562C02CA4DFB2A864DE63280CC140D0D540EA
> 1A5711D1E519C842684F42445C41CB501B7EA00361699C320
> uncompressed address: 1Kz2CTvjzkZ3p2BQb5x5DX6GEoHX2jFS45

Recent Improvements & TODO

Create an issue so I can add more stuff to improve

About

An automated bitcoin wallet collider that brute forces random wallet addresses

python bitcoin multiprocessing address collider bruteforce brute-force-attacks brute-force wallet cracker btc cracking brute plutus crack bruteforce-attacks bruteforcing stealing stealer wallet-address

Readme
Activity
1.4k stars
91 watching
569 forks
Report repository

Releases

No releases published

Packages

No packages published

Contributors 2

Isaacdelly
p1r473

Languages

Python 100.0%