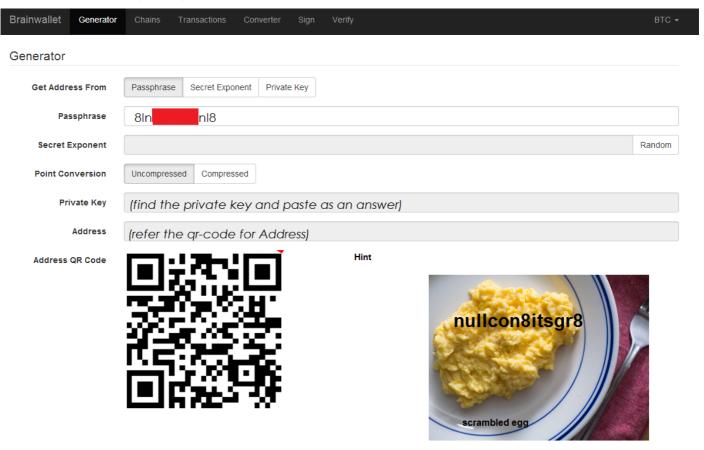
◀ home (/)

## nullcon HackIM Crypto 1 writeup

This weekend was nullcon HackIM CTF (https://ctftime.org/event/421) and I wanted to post my writeup for the Crypto 1 challenge, because I found it particularly interesting.

Although crypto challenges aren't usually my favorite category, I often find them to be quite interesting and a good way to learn something new.

This particular challenge caught my attention, because it was about bitcoins:



From looking at the picture we learn that:

- It has something to do with BrainWallets
- We have a public bitcoin address in the QRcode
- · We have to find the correct private key
- · We have some parts of the password
- · It has a hint which does not make sense yet

The first step I did was to extract the QRcode from the screenshot and use an online QRcode reader to the get public key:



The resuling bitcoin address is:

17iUnGoZbFrGS7uU9z2d2yRT9BKgVqnKnn

But then I was confused how the partial password should reveal the private key or if we're supposed to bruteforce the private key for the given public key. The latter, however, wouldn't make sense, because otherwise bitcoin would be broken.

After a bit of googling we learn that a brainwallet's private key is the sha256 of the password (https://filippo.io/brainwallets-from-the-password-to-the-address/):

private\_key = sha256(password)

That means we only have to find the correct password (the letters behind the red bar) to restore the private key and solve the challenge!

I've found a repo on GitHub (https://github.com/dan-v/bruteforce-bitcoin-brainwallet/) which automatically bruteforces brainwallets and looks if there are unspent bitcoins on them. *Make sure to have a long long unique brainwallet password otherwise your bitcoins might get stolen!*  For my solution I've adapted their Wallet-class (https://github.com/dan-v/bruteforce-bitcoin-

brainwallet/blob/master/lib/wallet.py) because it takes care of the password-to-publickey computation and the final idea was to generate possible password combinations until the resulting public key matches our bitcoin address.

This is the point where the hint became helpful.

I first started the brute-force with combinations of length 1-7 with a charset made of all characters of scrambled egg and nullcon8itsgr8.

After a while my teammate Alex suggested to only use the characters from nullcon8itsgr8 and a guessed length of 8 characters. It turns out that he was right...

Here's the final script:

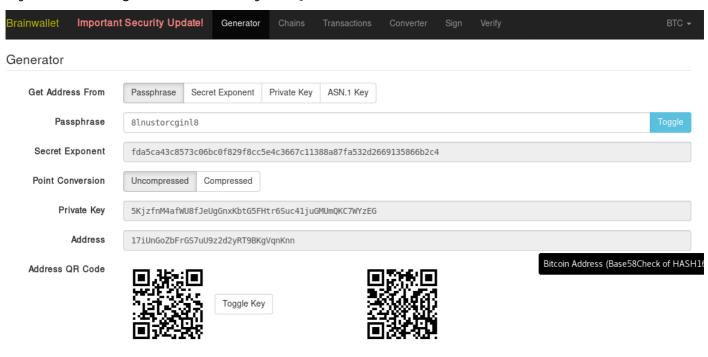
```
#!/usr/bin/python2
from coinkit import BitcoinKeypair
import logging
import itertools
PASSWORD = "8ln{{X}}n18"
TARGETADDR = "17iUnGoZbFrGS7uU9z2d2yRT9BKgVqnKnn'
charset = []
for hint in ["nullcon8itsgr8"]: #,"scrambled egg"]:
    for c in hint:
        if not c in charset:
            charset.append(c)
#Those characters probably won't be in the password again
charset.remove("n")
charset.remove("1")
charset.remove("8")
# Source: https://github.com/dan-v/bruteforce-bitcoin-brainwallet/blob/master/lib/wallet.py
    def __init__(self, passphrase, is_private_key = False):
        self.passphrase = passphrase
        self.address = None
        self.public_key = None
        self.private_key = None
        try:
            if is private kev:
                keypair = BitcoinKeypair.from_private_key(self.passphrase.encode('ascii'))
                keypair = BitcoinKeypair.from_passphrase(self.passphrase)
            self.address = keypair.address()
            self.public key = keypair.public key()
            self.private_key = keypair.private_key()
        except Exception as e:
            logging.warning(u"Failed to generate keypair for passphrase '{}'. Error: {}".format(passphrase, e.args))
print "Charset: {}".format(charset)
for 1 in range(8,9):
    print "Testing...length {}".format(1)
    for comb in itertools.permutations(charset, 1):
        pw = PASSWORD.replace("{{X}}}", ''.join(comb))
        wallet = Wallet(pw)
        if wallet.address == TARGETADDR:
            print pw
            print wallet.address
            print wallet.public_key
            print wallet.private_key
            break
```

After a couple of minutes we get a result:

```
$> python2 solver.py
Charset: ['u', 'c', 'o', 'i', 't', 's', 'g', 'r']
Testing...length 8
8lnustorcgin18
17iUnGoZbFrGS7uU9z2d2yRT9BKgVqnKnn
047663d087dd1da8315644e0800b7651e0763b5fbf9a2388834db0bbb282d5a1761fd8c993dd3ea7fa5cdb616b591fa391dc00bafce7e70feb1a7002a10e9ca152
fda5ca43c8573c06bc0f829f8cc5e4c3667c11388a87fa532d2669135866b2c4
```

With a bit of googling I found the brainwallet website used in the screenshot: https://brainwalletx.github.io/ (https://brainwalletx.github.io/)

Entering the password reveals the WIF-encoded (https://en.bitcoin.it/wiki/WIF) private key: 5KjzfnM4afWU8fJeUgGnxKbtG5FHtr6Suc41juGMUmQKC7WYzEG



Flag: flag{5KjzfnM4afWU8fJeUgGnxKbtG5FHtr6Suc41juGMUmQKC7WYzEG}

This was quite cool and I most certainly know now why I'm not using a brainwallet:)

0 Comments 0dav.work ■ Login ▼ Sort by Best -C Recommend 1 Share Start the discussion...

Be the first to comment.

ALSO ON ODAY.WORK

## Boston Key Party CTF 2016 | Sebastian Neef

10 comments • a year ago •



ZorGO — I didn't find a decryption key that looks like this 24445b8df60b73bc3133323435363738 but I was able to partially decipher the text and get the key

## **TROOPERS 2016 Challenge & PacketWars writeups**

4 comments • a year ago•



Odaywork — +1Thanks for the details I have missed!

Subscribe Add Disgus to your site Add Disgus Add Privacy

iCTF 2017 flasking unicorns writeup - Or how we might have rooted your iCTF VM

8 comments • 14 days ago •



Odaywork — Hey, yeah, I got the same error. The reason why that happens is that the special characters are encoded into their html entities during the first rendering step. That's also the reason ...

0CTF 2016 Quals writeups | Sebastian Neef

10 comments • a year ago•



0daywork — Hi, oh, without dp it's hard to solve. But using 0x10001 wasn't the right way either, and this was just a 'guess' from my side.

© 2017 Sebastian Neef - Oday.work (https://0day.work) All rights reserved. Theme created by Milos Bejda (https://www.mbejda.com)