locked-out

- name: locked-out
- points: 736
- attachments
  - `locked-out.zip`

solution

Download https://github.com/kimci86/bkcrack.

Create a `flag.png` with the bytes `89 50 4E 47 0D 0A 1A 0A 00 00 00 0D` and zip it as `whatever.zip` (for reference, see `unlocked-out.zip`) without any compression.

Run the following command (with the expected output provided):

```
$ ./bkcrack-1.6.1-win64/bkcrack.exe -C locked-out.zip -c flag.png -P
unlocked-out.zip -p flag.png
bkcrack 1.6.1 - 2024-01-22
[01:55:01] Z reduction using 5 bytes of known plaintext
100.0 % (5 / 5)
[01:55:01] Attack on 1132321 Z values at index 6
Keys: 9c9d83e5 7915abee 58998fbc
4.3 % (48778 / 1132321)
Found a solution. Stopping.
You may resume the attack with the option: --continue-attack 48778
[01:55:33] Keys
9c9d83e5 7915abee 58998fbc
```

Change the password of `locked-out.zip` to empty (with the expected output provided):

```
$ ./bkcrack-1.6.1-win64/bkcrack.exe -C locked-out.zip -k 9c9d83e5 7915abee
58998fbc -U unlocked.zip ""
bkcrack 1.6.1 - 2024-01-22
[01:57:58] Writing unlocked archive unlocked.zip with password ""
100.0 % (2 / 2)
Wrote unlocked archive.
```

Unzip the with the empty password. Inside, there is a `flag.png`, which shows the flag.

Transcript the image to get the flag: `jctf{wh0_n33d5_p455w0rd5_4nyw4y5?}`.

process

The hints mention searching up ZipCrypto... So do use that hint. You should find that it is vulnerable to plaintext attack after searching online (Wikipedia).

Luckily, you do not need to implement the attack algorithm yourself, as somebody else already does: https://github.com/kimci86/bkcrack.

Furthermore, notice that the files inside the zip are not compressed at all. That means the attack is very easy, as long as we have some plaintext of a file inside.

Unfortunately, we do not have any of the plaintext. But what we could do is to guess the plaintext. Create an empty PNG image yourself using your favorite image editor, and you should see the first 12 bytes are `89 50 4E 47 0D 0A 1A 0A 00 00 00 0D`. So we are guessing the first 12 bytes of `flag.png` are the same, considering that the starting content of the file is usually magic bytes. This explains why we use those 12 specific bytes as the plaintext.

Then the rest is simply using the tools, as outlined in § solution. It does take some time to get the tool to work though, as it is not the most intuitive tool in the world...