PRÁCTICA 2

Para la solución del enunciado he implementado un ataque de fuerza bruta, algo que como es lógico no es nada eficiente. Realmente ha sido un problema de enfoque y de confusión, ya que en todo momento pensé que se exigía un script de este tipo y no he tenido tiempo de rectificarlo (de manera práctica).

Mi programa es muy sencillo, se ejecuta mediante la siguiente sintaxis:

python3 practica2_brute.py (-min "x") (-max "y") (-chr "z") (-out "file")

Si ejecutamos el comando con - -h / -help nos mostrará información extra sobre los parámetros de entrada (todos opcionales ya que hay valores por defecto).

Mi idea ha sido realizar un ataque de fuerza bruta para contraseñas de longitud y formatos variables, y almacenar todas las combinaciones probadas en un diccionario.

Los parametros -min y -max determinan la longitud mínima y máxima de las contraseñas a probar.

El parámetro -chr tiene distintas opciones. Si introducimos caracteres, realizará pruebas con la combinación de dichos caracteres (por ejemplo, si introducimos "*-min 3 – max 3 -chr abc"* como parámetro realizará pruebas con las combinaciones "aaa", "bbb", "ccc", "aab", "aac", "aba", "abb" , etc.). Sin embargo, y como indico en el panel de ayuda, si introducimos un dígito utilizará un formato predeterminado (como letras minúsculas, mayúsculas, dígitos, símbolos o combinados).

Buscaba así desarrollar un programa para un caso más general y no limitandome a contraseñas con letras minúsculas.

Sin embargo, el programa es prácticamente ineficiente ya que consume una gran cantidad de tiemo y recursos, a pesar de útilizar varios procesos. Esto generalmente creo que se debe a la escritura de las distintas contraseñas en el fichero y sobre todo al uso del comando gpg dónde creo que hay un cuello de botella. Básicamente, para contraseñas de 4 caracteres tarda más de 3 horas y no encuentra la contraseña ya que su longitud es mayor.

He probado tambień la librería de Python 'GnuPG' pero el resultado era prácticamente similar.

Para un ataque con diccionarios el tiempo que he conseguido es similar. Para compararlo con el caso anterior, para contraseñas de letras minúsculas de 4 caracteres tarda algo más de 2 horas.
Para ello he implementado un pequeño script para crear un diccionario de x caracteres (*make_dict.py)*.
Resultaría interesante utilizar colecciones de diccionarios utilizados en el ámbito del cracking como "rockyou.txt" o "passw12000.txt", que recopilan algunas de las contraseñas más utilizadas.

Debo matizar que tanto un ataque de fuerza bruta como un ataque mediante diccionarios son prácticamente ineficientes ante unas claves relativamente seguras, como hemos visto en clase.

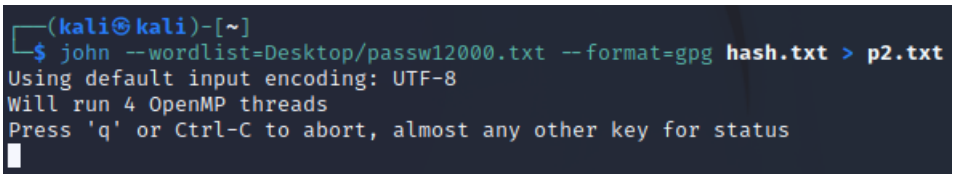
Seguramente paralelizando mediante hilos en lugar de procesos se consiga una mayor eficiencia, pero en mi caso personal, la librería Thread me causaba problemas al utilizar la librería GnuPG.
Otra opción interesante sería aprovechar la GPU para paralelizar las tareas de cómputo. Una posibilidad para combinar con Python es CUDA, pero este software es propio de Nvidia y por tanto queda limitado ordenadores con sus tarjetas gráficas.

En todo caso, dudo que se alcance una eficiencia parecida al de algunas aplicaciones de cracking como es ‘John he Ripper’ y de la que me gustaría hablar a continuación.

Esta herramienta escrita en C es capaz de romper los hashes MD5, SHA-1 entre otros. Está muy bien optimizado para una amplia gama de procesadores y arquitecturas.

En primer lugar, debemos ejecutar la herramienta *gpg2john*. Es una extensión de John the Ripper que nos permite tratar con cifrados GPG (existen también zip2john, rar2john, ssh2john, etc.). Esto redireccionará la salida hacia un archivo que contendrá el hash extraído.

Ahora, temenos la opción de ejecutar un ataque con diccionarios o con fuerza bruta (entre otros modelos de ataque). En mi caso, he usado un ataque con diccionario utilizando “passw12000”. También he incluido el formato de cifrado utilizado.



http://travisaltman.com/password-dictionary-generator/
https://github.com/agusmakmun/python-wordlist-generator/blob/master/wgen.py