

“NCJM”

Nmap – Crunch – John The Ripper - Medusa

Kevin Morillo Gallego

Ivan Chirmici Coeva

22/05/2019



INS Joaquim MIR



RESUMEN

El objetivo de este trabajo ha sido el desarrollo de una **herramienta** de gestión de los siguientes **programas**: (**Nmap**, **Crunch**, **John The Ripper** y **Medusa**) y la **Integración** entre ellas. La cual logra un **funcionamiento liviano** entre todos ellos. Se pretende por tanto que la **herramienta** nos facilite el trabajo y a la par, no generar problemas al no conocer las líneas de comandos de los **programas integrados**.

Como resultado del trabajo realizado, cumple nuestro objetivo y tiene las **funcionalidades** propuestas inicialmente.

Palabras Clave

Integración, Nmap, Crunch, John The Ripper, Medusa, herramienta.

ABSTRACT

The objective of this work has been the development of a management tool of the following programs: (**Nmap**, **Crunch**, **John The Ripper** and **Medusa**) and the integration between them. Which achieves a light operation between all of them. It is therefore intended that the tool facilitates the work and at the same time do not generate problems by not knowing the command lines of the integrated programs.

As a result of the work done, it fulfils our objective and has the functions proposed initially.

Keywords

Integration, Nmap, Crunch, John The Ripper, Medusa, tool.

ÍNDICE DE CONTENIDOS

INTRODUCCIÓN	6
Introducción al Proyecto	6
¿Por qué “NCJM”?	7
Planificación	8
TECNOLOGÍAS UTILIZADAS	9
Lenguaje de programación	9
Programas	10
Oracle VirtualBox	10
Atom Editor	11
Page	12
DISEÑO LÓGICO	13
Diseño lógico entre programas	13
Nmap y Medusa	13
Crunch y John The Ripper	16
Estructura	18
Estructura de funcionamiento	18
Estructura de directorios	19
CREACION DE UNA INTERFAZ	21
Paso 1: Diseño lógico	21
Paso 2: Diseño gráfico	21
Paso 3: Asignación de métodos gráficamente	22
Paso 4: Generar los archivos necesarios para la interfaz	23
Paso 5.1: Creación de métodos	25
Paso 5.2: Funcionalidades adicionales	29
FUNCIONALIDADES DE NCJM	35
Nmap	35
Crunch	37
John The Ripper	39

Medusa	41
CONCLUSIONES	42
Objetivos cumplidos	42
Objetivos no cumplidos	43
Viabilidad y aplicabilidad de NCJM	44
CONCLUSIONS	45
Objectives not met	45
Viability and applicability of NCJM	46
PRESUPUESTO	47
REFERENCIAS Y BIBLIOGRAFIA	49

1 INTRODUCCIÓN

1.1 Introducción al Proyecto

Nuestro **proyecto** es una **plataforma** para sistemas Linux, la cual integra las **funcionalidades** de varios **programas**, que ya tenían una fácil implementación en la última versión estable de Ubuntu, la 18.04.

En la documentación de nuestro **proyecto** podrás encontrar:

- El porqué del nombre de nuestra **plataforma**
- Los objetivos seguidos para su creación.
- Una tabla resumen de la planificación.
- Las **herramientas** y lenguajes usados.
- Un mapa del diseño lógico.
- La estructura seguida y sus **funcionalidades**.
- Nuestras conclusiones post creación.
- El “Valor de mercado” que tendría nuestra **plataforma**
- La aplicabilidad del **proyecto** a afectos reales.

1.2 ¿Por qué “NCJM”?

Porque nuestro **programa** es una mezcla de las siguientes **herramientas** ya existentes e integradas en los sistemas Linux.

- **Nmap** es una **herramienta** de código abierto para la exploración y la auditoria de seguridad.
- **Crunch** es un **herramienta** que genera “**Wordlist**” con una serie de condiciones introducidas por el usuario.
- **Jhon de Ripper** es una **herramienta** de crackeo de contraseñas.
- **Medusa** es una **herramienta** pensada para atacar por fuerza bruta a un **servicio** y **host** específico.

El nombre de nuestro **proyecto** refleja la unión entre los diversos aplicativos de forma gráfica y simple.

1.3 Planificación

A continuación se detalla la planificación tomada para la realización del trabajo fin de grado en cuatro semanas. Se calcula el trabajo realizado entre los períodos 23/04/2019 y 21/05/2019.

Tarea	Fecha de Inicio	Duración	Fecha Final
Estudio y análisis previo general sobre el rumbo a tomar del proyecto ¹	23/04/2019	6 días	29/04/2019
Estudio sobre la herramienta definitiva	29/04/2019	4 días	03/05/2019
Desarrollo de las interfaces	03/05/2019	7 días	10/05/2019
Integración de las distintas interfaces	10/05/2019	4 días	14/05/2019
Testing & Debugging	14/05/2019	2 días	16/05/2019
Generación de documentación, presentación y exposición oral.	16/05/2019	5 días	21/05/2019

¹ El primer día partimos de nuestra idea principal, la cual en ese instante era generar un auditor de **red**. Pero poco después nos percatamos de nuestro **proyecto** nos motivaba más bien, poco. Entonces empezamos un largo y tendido debate, sobre cuál sería nuestra decisión final como **proyecto**,. Durante esos días, hicimos varias pruebas descartando posibles **proyectos**, editores y lenguajes.

2 TECNOLOGÍAS UTILIZADAS

1.1 Lenguaje de programación

El lenguaje usado en este **proyecto** es Python. Han sido utilizados tanto python2.7 como Python 3. Esto es debido a que hemos usado la librería tkinter o Tkinter según la versión de la que hablemos, como observarás en la fotografía.

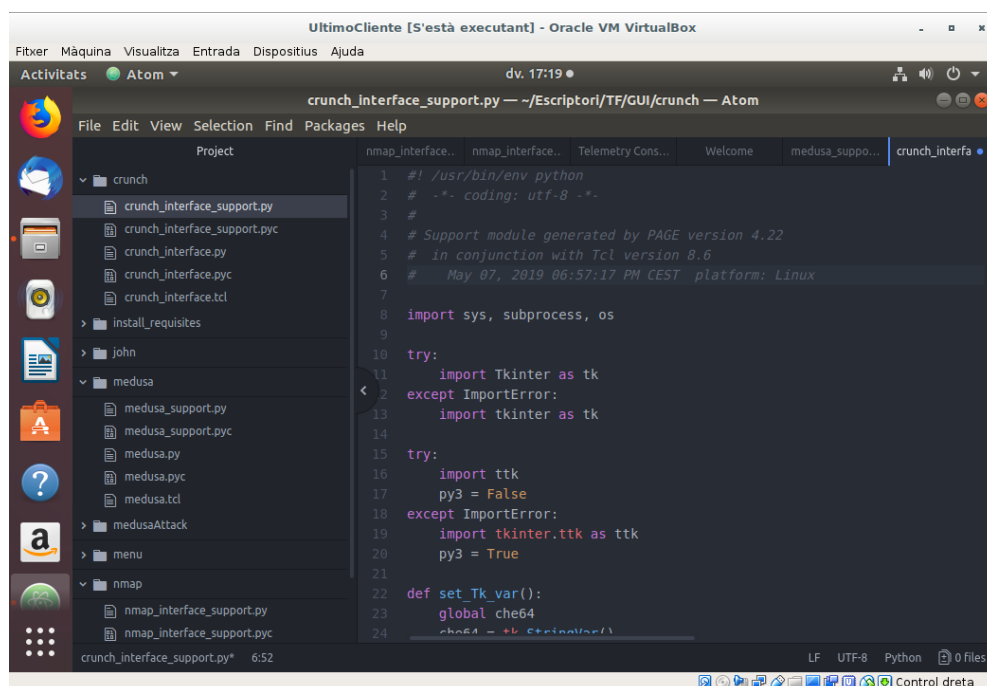
```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  #
4  # Support module generated by PAGE version 4.22
5  # in conjunction with Tcl version 8.6
6  # May 07, 2019 06:57:17 PM CEST platform: Lin
7
8  import sys, subprocess, os
9
10 try:
11     import Tkinter as tk
12 except ImportError:
13     import tkinter as tk
14
15 try:
16     import ttk
17     py3 = False
18 except ImportError:
19     import tkinter.ttk as ttk
20     py3 = True
```

En la foto podemos observar como llamamos a python2.7 pero más abajo vemos un control de errores (**Try/Catch**) que este **funciona** en base a las clases/**métodos** que usemos, ya que si falla Tkinter renombrado como "tk" nuestro except hará que pase a ser tkinter, que es la versión de python2.7 renombrado de la misma forma.

1.2 Programas

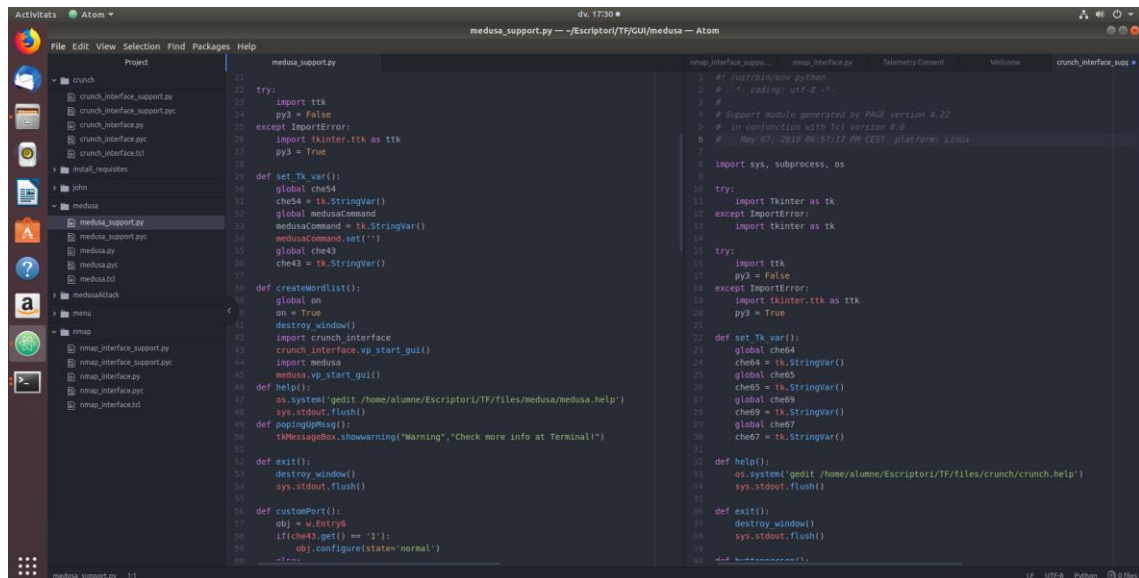
1.2.1 Oracle VirtualBox

El **programa** Oracle Virtualbox ha sido utilizado para generar y usar una máquina virtual por motivos de seguridad y falta de permisos.

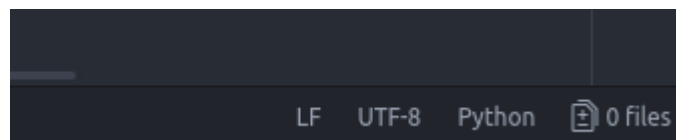


1.2.2 Atom Editor

Hemos usado Atom Editor como editor ya que es fácil de usar y **liviano**. Además de ser gratuito y otras cualidades como el duplicado de **pantallas** o la navegación entre carpetas y ficheros.



Atom Editor posee una librería interna para Python:



1.2.3 Page

Page es un editor gráfico el cual usa la librería de Python Tkinter o tkinter.

Después de varios días consultando en internet e incluso con profesores, fue lo mejor que encontramos a pesar de que es bastante “básico”, pero lo suficientemente útil como para poder generar nuestras interfaces gráficas de cara al **programa**.

TODO

De hecho como curiosidad el mismo page está hecho con la librería Tkinter de Python.

3 DISEÑO LÓGICO

En este capítulo se describe el diseño lógico y estructural que seguíamos para el desarrollo de nuestra **herramienta**.

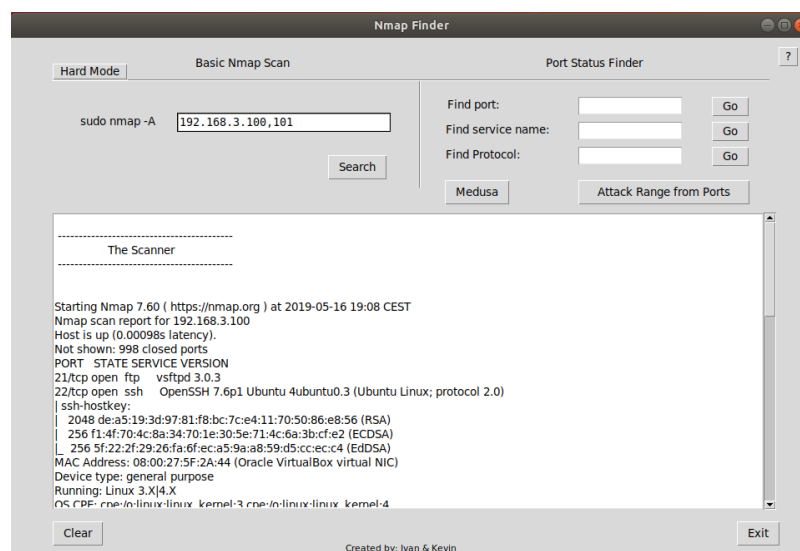
2.1 Diseño lógico entre programas

Primero de todo, hemos de distinguir que la idea principal era interconectar los diferentes **software** entre sí.

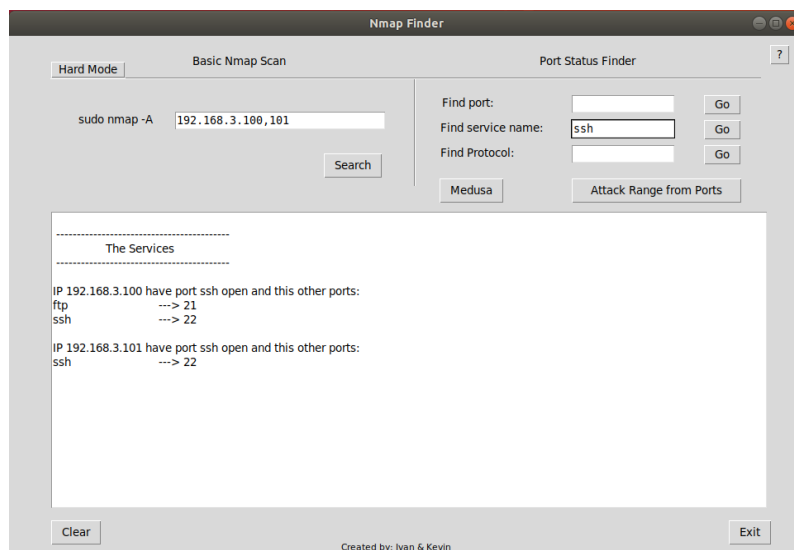
Podemos destacar nuestro primer ejemplo de ello, el trabajo ligado entre **Nmap** y **Medusa**.

Nmap escanea una **red** o un **host** específico, y nos muestra los **servicios** existentes en cada uno de estos **host**. Y tenemos la opción del mismo **Nmap** de acceder a **Medusa** para hacer el **ataque** de manera automática, sin tener que recordar o copiar la **ip** y el **servicio**.

2.1.1 Nmap y Medusa

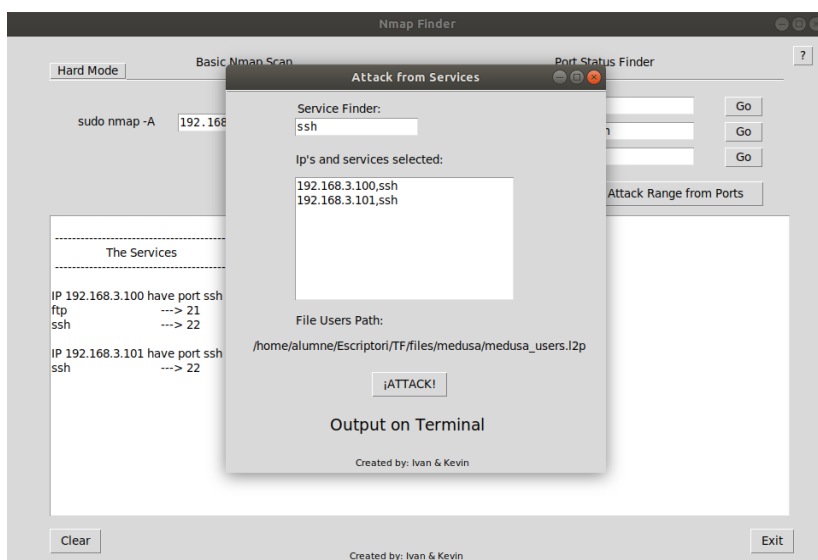


Podemos observar en la imagen como **Nmap** escanea dos **hosts** (192.168.3.100 y 192.168.3.100) y obtenemos un gran texto, que a simple vista es complejo de entender.

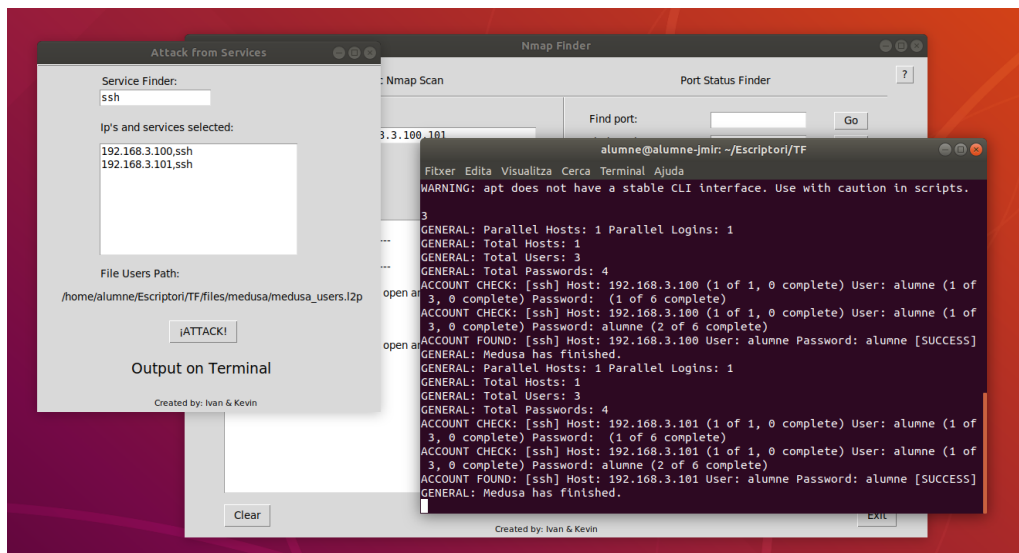


Una vez filtramos por el **servicio 'ssh'** por ejemplo (como en la imagen), la **herramienta** gestiona todo ese texto ininteligible anterior y nos muestra el resultado por el filtro aplicado.

A continuación le damos clic a “Attack Range from **Ports**” y nos abrirá una ventana como la siguiente:



Ahora es donde se ve entiende el título del apartado: se han transferido los datos de **Nmap** a una ventana nueva que ejecuta comando de **Medusa** en bucle.

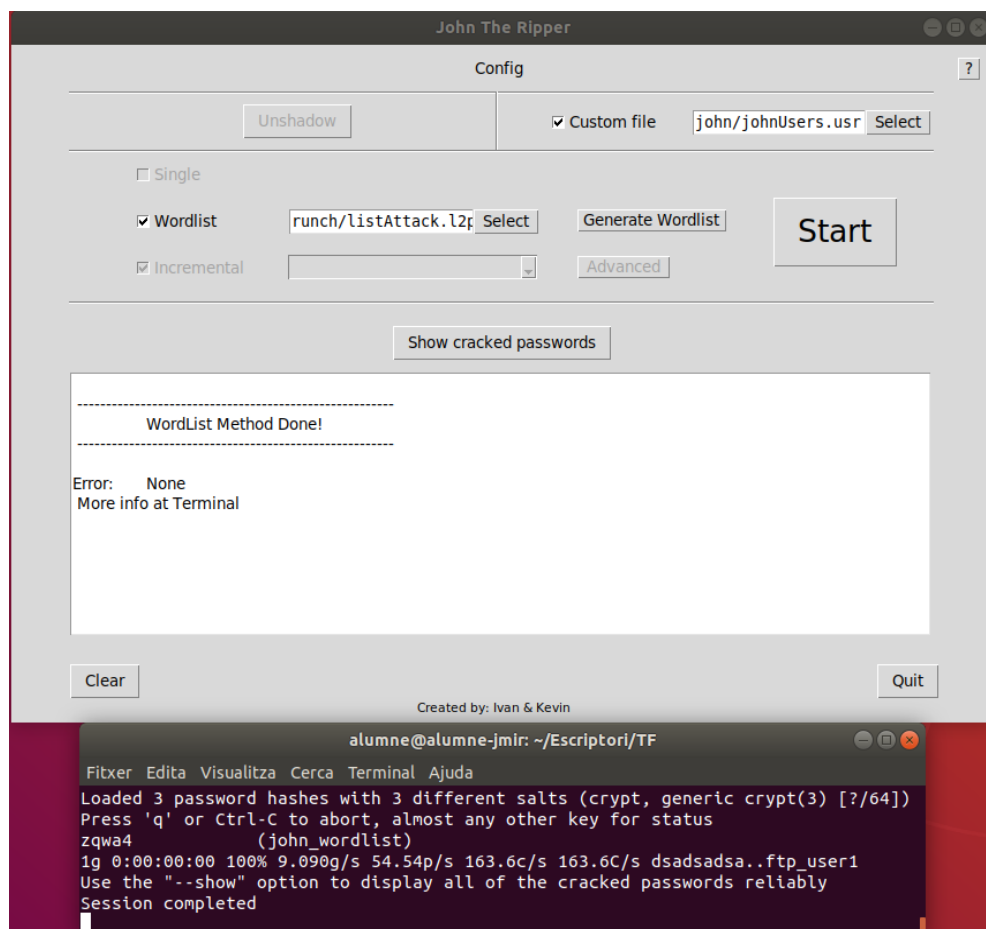


Como en nuestro ejemplo hemos detectado dos **redes** con **ssh**, el **Medusa** hace el **ataque** a los dos **hosts** (primero uno y cuando acaba éste, empieza el siguiente).

2.1.2 Crunch y John The Ripper

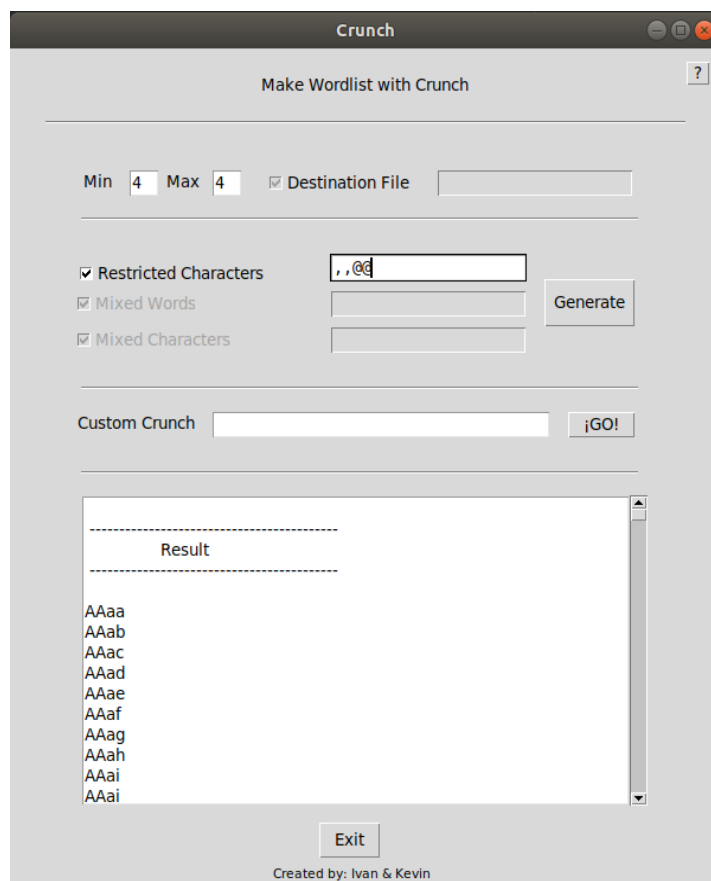
Otro ejemplo más de esta idea:

Para encontrar una contraseña con **JTR (John The Ripper)**, es necesario un diccionario de palabras (**Wordlist**). Para eso utilizamos la **herramienta Crunch** para generar esa **lista** de palabras y pasársela a **JTR** para el **ataque**.



Observamos en la imagen que ya se ha seleccionado la opción de **Wordlist** y utilizado esta **funcionalidad** del **JTR**. A la par se ha seleccionado automáticamente la ruta de nuestro diccionario generado con el botón “Generate **Wordlist**” el cual abre el **programa Crunch**.

Además un poco más abajo enseñamos brevemente el **funcionamiento** del **programa JTR**, del cual hablaremos más adelante.



E aquí la **pantalla** de **Crunch**, en el cual le indicamos el formato de generación de la contraseña para el **JTR**.

2.2 Estructura

2.2.1 Estructura de funcionamiento

En este apartado queremos enseñar cómo se comporta nuestro **programa** en base lógica.



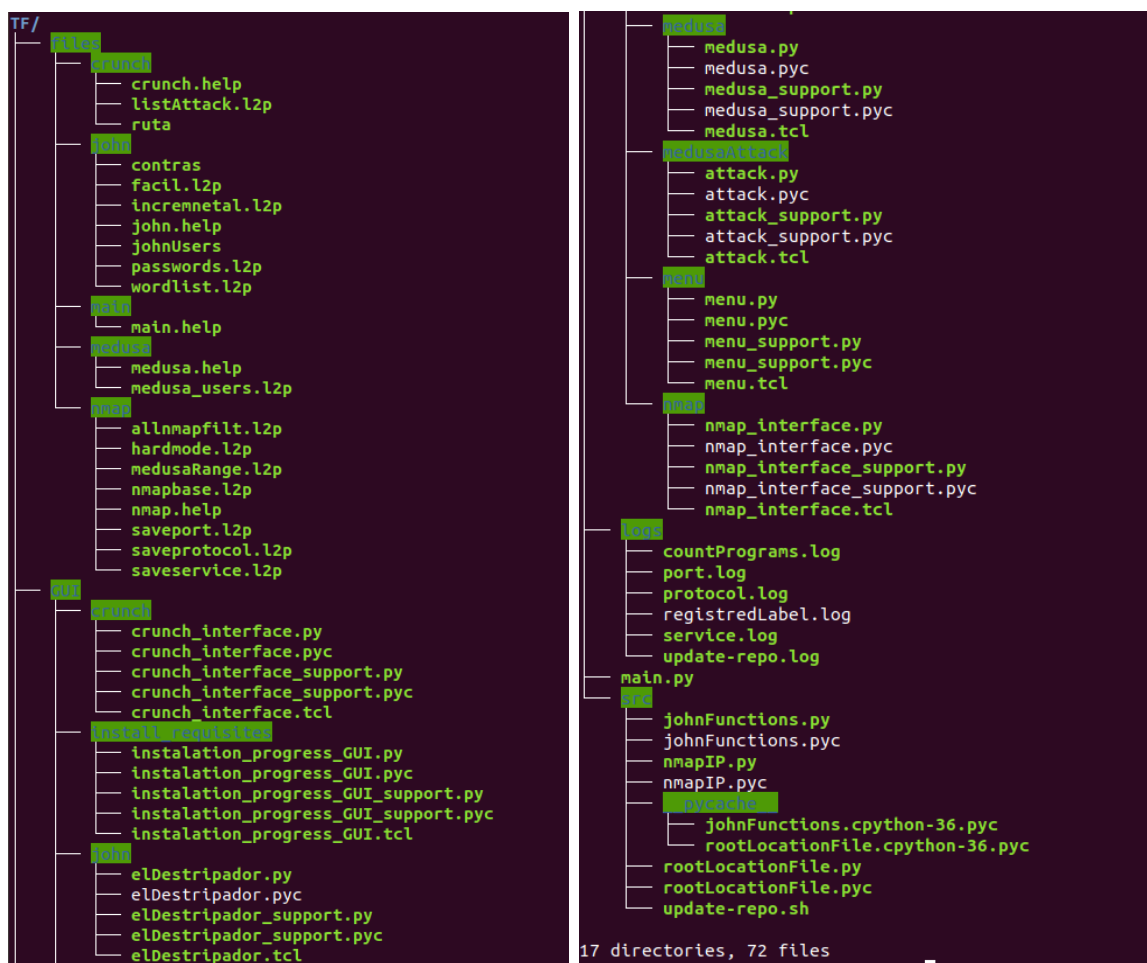
Como podemos observar en la **imagen**, todas las **herramientas** están interconectadas sin tener en cuenta el menú principal. La imagen refleja además las **funcionalidades** principales de cada una de las **pantallas** que se han creado.

Con **pantalla** en púrpura, “**Medusa Aux**”: se ve diferente al resto, queremos reflejar que ésta conforma la unión entre las **herramientas Nmap, Medusa y Crunch**.

2.2.2 Estructura de directorios

Queremos reflejar con este apartado, que cuando desarrollas un **programa/software**, la clave principal de éste es un orden y organización tanto de documentos como de la sintaxis utilizada en todo el camino recorrido hasta obtener algo **funcional**.

Más adelante distinguimos la estructura utilizada y una breve descripción de cada directorio:



- TF: directorio principal en el cual se almacena toda la lógica de **negocio** de nuestra **herramienta**.

- Files: este directorio tiene subdirectorios separados por los **programas** utilizados, en los cuales almacenamos todos los archivos generados por los scripts y los helpers.
- GUI: directorio con las interfaces gráficas generadas por Pages y **programadas** por nosotros. Además también reside parte del modelo de **negocio** de la **aplicación** ya que utiliza pequeños *scripts* con **funcionalidades**, los cuales no era necesario reubicarlos en “src” según nuestro criterio.
- Logs: como bien indica su nombre, aquí almacenamos todos los *logs* generados por la **herramienta** en el momento de existir un error.
- Src: se establece la lógica de **negocio** en este directorio, ya que contiene todos los scripts necesarios para el **funcionamiento** de la **herramienta**.

4 CREACION DE UNA INTERFAZ

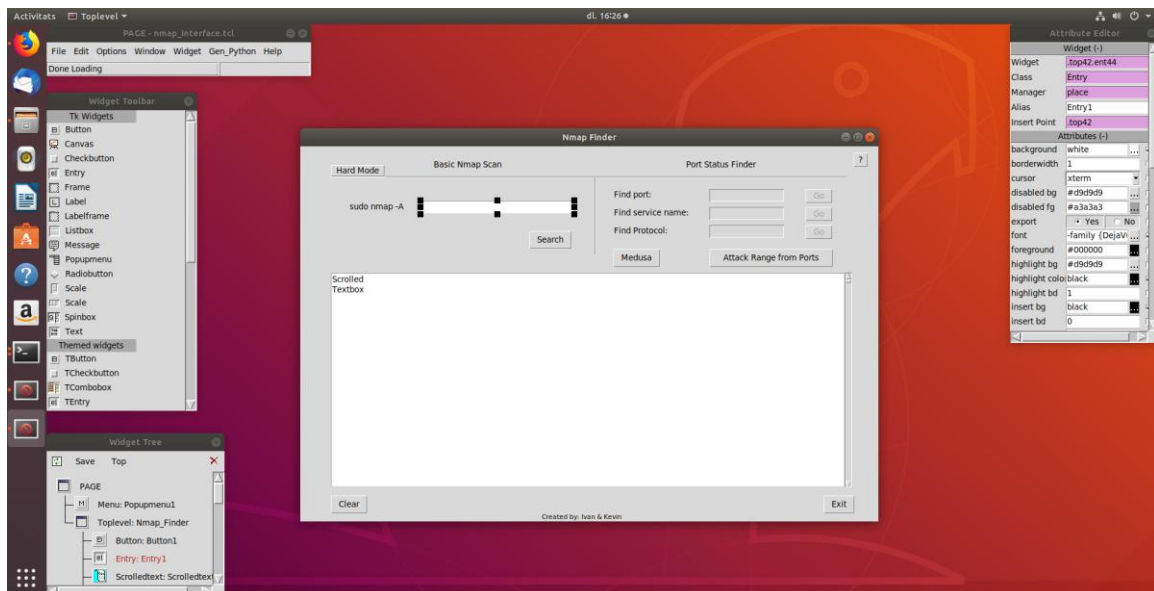
El ejemplo explicado más adelante, es aplicable a todo el **proyecto** realizado.

Paso 1: Diseño lógico

Comenzaríamos por plantearnos qué queremos y cómo lo queremos. La forma más adecuada consideramos que sería tomar nota de las **funcionalidades** que necesitamos y crear en base a ellas nuestra **interfaz**.

Paso 2: Diseño gráfico

El segundo paso una vez acabado el diseño lógico, deberíamos tomar papel y lápiz. Y empezar a construir gráficamente nuestra **interfaz**.

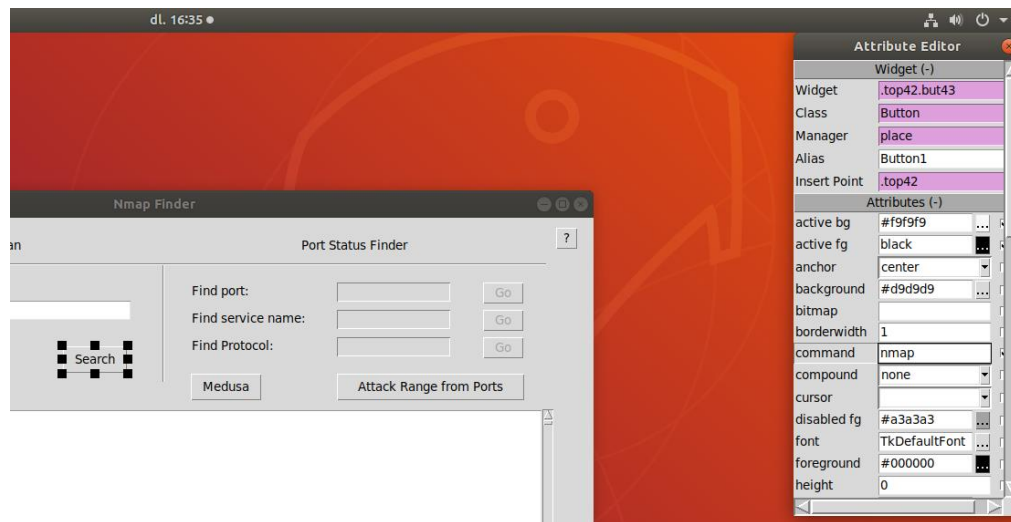


Paso 3: Asignación de métodos gráficamente

Cada objeto posee una serie de **propiedades**, la gran mayoría de ellas son todas de entorno gráfico pero algunos de los **atributos** son de entorno lógico.

La **propiedad** *alias* contiene el nombre que usaremos más tarde para la llamada del objeto.

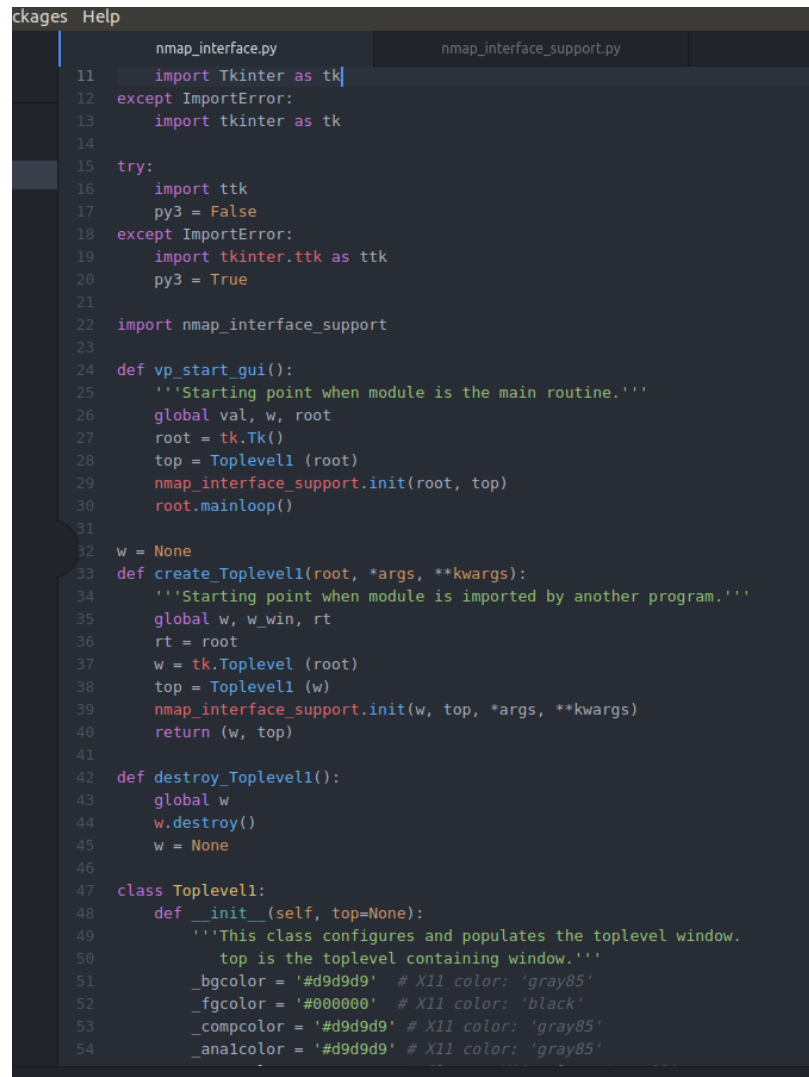
En este caso la **propiedad** *command* aloja el nombre del **método** al cual llamará nuestro botón.



Paso 4: Generar los archivos necesarios para la interfaz

Page genera dos archivos un nombre.py y un nombre_support.py.

Ejemplo nombre.py



```
11 import Tkinter as tk
12 except ImportError:
13     import tkinter as tk
14
15 try:
16     import ttk
17     py3 = False
18 except ImportError:
19     import tkinter.ttk as ttk
20     py3 = True
21
22 import nmap_interface_support
23
24 def vp_start_gui():
25     '''Starting point when module is the main routine.'''
26     global val, w, root
27     root = tk.Tk()
28     top = Toplevel1 (root)
29     nmap_interface_support.init(root, top)
30     root.mainloop()
31
32 w = None
33 def create_Toplevel1(root, *args, **kwargs):
34     '''Starting point when module is imported by another program.'''
35     global w, w_win, rt
36     rt = root
37     w = tk.Toplevel (root)
38     top = Toplevel1 (w)
39     nmap_interface_support.init(w, top, *args, **kwargs)
40     return (w, top)
41
42 def destroy_Toplevel1():
43     global w
44     w.destroy()
45     w = None
46
47 class Toplevel1:
48     def __init__(self, top=None):
49         '''This class configures and populates the toplevel window.
50         top is the toplevel containing window.'''
51         _bgcolor = '#d9d9d9' # X11 color: 'gray85'
52         _fgcolor = '#000000' # X11 color: 'black'
53         _compcolor = '#d9d9d9' # X11 color: 'gray85'
54         _anacolor = '#d9d9d9' # X11 color: 'gray85'
```

Ejemplo nombre_support.py

```
nmap_interface.py | nmap_interface_support.py

import sys, subprocess, os
sys.path.append(os.getcwd() + '/src')
from nmapIP import IPfinder_all_f as find
from nmapIP import ipForMedusen as ipFM
sys.path.append(os.getcwd() + '/GUI/medusaAttack')
from attack import vp_start_gui as startAttack

try:
    import Tkinter as tk
except ImportError:
    import tkinter as tk

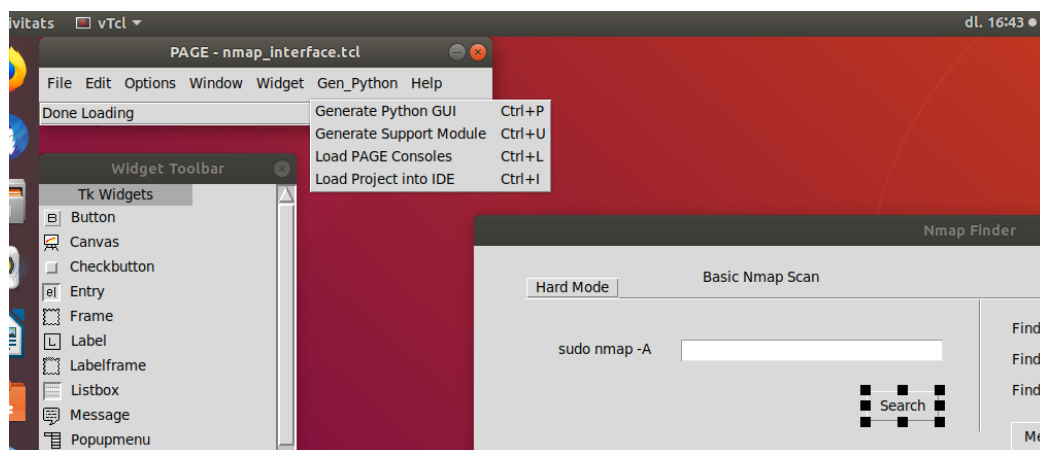
try:
    import ttk
    py3 = False
except ImportError:
    import tkinter.ttk as ttk
    py3 = True

def hardMode():
    global w
    obj = w.Scrolledtext1
    obj2 = w.Entry1
    obj.delete(1.0, tk.END)
    os.system("cat " + os.getcwd() + "/files/nmap/hardmode.l2p > " + os.getcwd() + "/files/nmap/nmapbase.l2p")
    proc = subprocess.Popen("cat " + os.getcwd() + "/files/nmap/hardmode.l2p", stdout=subprocess.PIPE, shell=True)
    (out, err) = proc.communicate()
    obj.insert(tk.END, '\n -----\n \tThe Scanner \n -----')
    w.Entry2.configure(state='normal')
    w.Entry3.configure(state='normal')
    w.Entry4.configure(state='normal')
    w.Button2.configure(state='normal')
    w.Button3.configure(state='normal')
    w.Button4.configure(state='normal')
    sys.stdout.flush()

def attackMedusaCompo():
    global w
    startAttack()
    sys.stdout.flush()

def attackMedusaSimple():
```

El primero genera toda la base del código gráfico y aloja todas las **propiedades** de los objetos, el segundo contiene los **métodos** usados en los botones del principal.



Paso 5.1: Creación de métodos

Aquí tenemos dos **ejemplos** de función los cuales son llamados por un botón:

- El botón "clear"

```
def clear():  
    global w  
    obj = w.Scrolledtext1  
    obj.delete(1.0, tk.END)  
    sys.stdout.flush()
```

Este sería uno de los **ejemplos** más básicos de botón.

Lo único que hace sería llamar a la variable **global** "w" que es quien contiene todos los objetos.

Luego llamaremos al objeto "Scrolledtext1" básicamente es donde recibimos el texto, luego vaciamos el objeto por si tuviera contenido. Y por último, se ejecuta en todas las **funcione** un sys.stdout.flush(), para limpiar cualquier out que genere el **programa**.

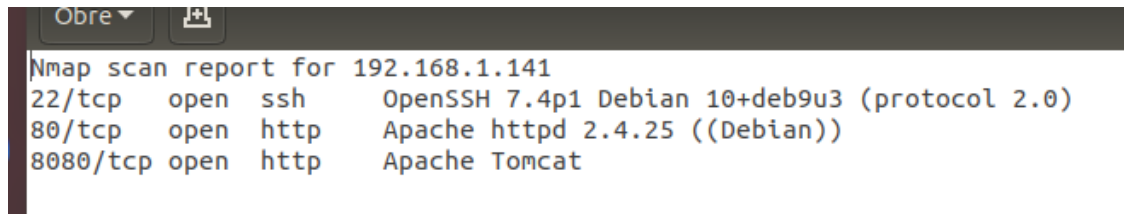
- El botón "Search"

El cual llama al **método Nmap** que hace lo siguiente:

```
def nmap():  
    global w  
    obj = w.Scrolledtext1  
    obj2 = w.Entry1  
    obj.delete(1.0, tk.END)  
    #proc = subprocess.Popen("cat /home/alumne/Escriptori/nmap.txt", stdout=subprocess.PIPE)  
    procc = subprocess.Popen("nmap -A " + obj2.get(), stdout=subprocess.PIPE, shell=True)  
    proc = subprocess.Popen("nmap -A " + obj2.get() + "| egrep -E '(Nmap scan)|tcp' > "  
    + os.getcwd() + "/files/nmap/nmapbase.l2p", stdout=subprocess.PIPE, shell=True)  
    (out, err) = procc.communicate()  
    obj.insert(tk.END, '\n ----- \n \tThe Scanner \n')  
    w.Entry2.configure(state='normal')  
    w.Entry3.configure(state='normal')  
    w.Entry4.configure(state='normal')  
    w.Button2.configure(state='normal')  
    w.Button3.configure(state='normal')  
    w.Button4.configure(state='normal')  
    sys.stdout.flush()
```

Llamamos a los objetos necesarios y ejecutamos dos *subprocess* (ejecutan ordenes de sistema).

El principal ejecuta un comando en shell que nos devolverá un fichero de este tipo:

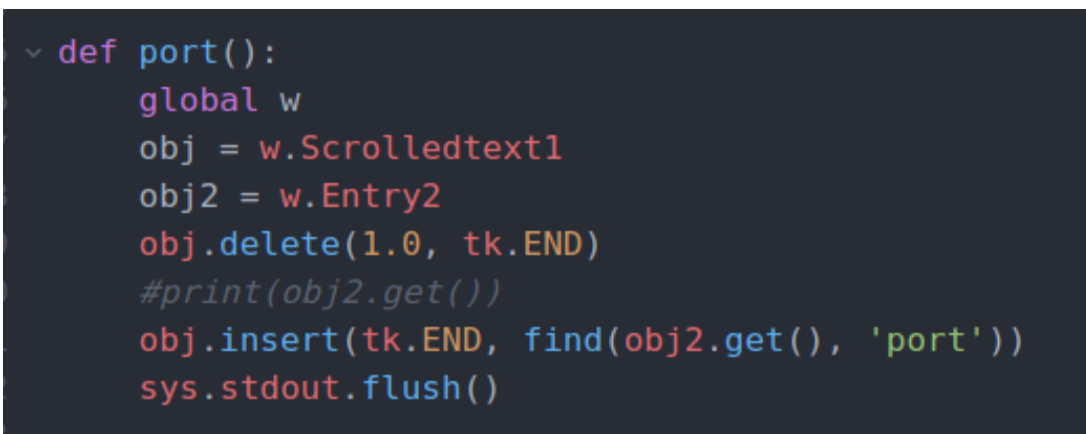


```
Nmap scan report for 192.168.1.141
22/tcp    open  ssh      OpenSSH 7.4p1 Debian 10+deb9u3 (protocol 2.0)
80/tcp    open  http     Apache httpd 2.4.25 ((Debian))
8080/tcp  open  http     Apache Tomcat
```

Ya que este último es mucho más fácil de tratar que el original.

También podemos observar en las fotos muchas llamadas a un **método** configure, el cual aplicamos para activar o desactivar los botones a necesidad de nuestras restricciones de **programa**.

- El botón GO, filtro por Puerto



```
def port():
    global w
    obj = w.Scrolledtext1
    obj2 = w.Entry2
    obj.delete(1.0, tk.END)
    #print(obj2.get())
    obj.insert(tk.END, find(obj2.get(), 'port'))
    sys.stdout.flush()
```

Este botón en apariencia es parecido a los anteriores, pero si observamos bien estamos llamando a una función llamada find() que proviene del siguiente **import**.

```
import sys, subprocess, os
sys.path.append(os.getcwd() + '/src')
from nmapIP import IPfinder_all_f as find
from nmapIP import ipForMedusen as ipFM
sys.path.append(os.getcwd() + '/GUI/medusaAttack')
from attack import vp_start_gui as startAttack
```

Ahora que ya sabemos de donde proviene echaremos un vistazo al script al que llama.

Haremos un breve repaso de algunas partes del script las relacionadas con el puerto:

En esta parte podemos observar cómo abrimos un fichero y generamos dos **string**: uno el cual será el visto por **pantalla** y otro que irá escrito en otro fichero.

En la siguiente línea en base al argumento escribiremos una salida por **pantalla** u otra.

```
def IPfinder_all(n):
    f = open(os.getcwd() + '/files/nmap/nmapbase.l2p', 'r')
    string = ''
    string_w = ''
    if n == 'port':
        string += '\n ----- \n \tThe Ports \n'
    if n == 'service':
```

Seguidamente procedemos a filtrar el fichero en base a los campos que nos interesan y concatenarlos para poder tratarlos.

```
string += '\n' + line[0] + '\n'
for line in f:
    l_1 = line.split()
    if l_1[0] == '|':
        continue
    if 'Nmap' in line:
        if len(l_1) == 6:
            string_w += '\n' + l_1[5][1:-1]
            string += '\n' + 'IP ' + l_1[5][1:-1] + ' have this open ' + n + ': \n'
        else:
            string_w += '\n' + l_1[4]
            string += '\n' + 'IP ' + l_1[4] + ' have this open ' + n + ': \n'
    elif n == 'port':
        string_w += ',' + str(l_1[0]).split('/')[0]
        string += str(l_1[0]).split('/')[0] + '\n'
```

Por último registramos y grabamos la salida en un fichero y lo que devolveríamos por **pantalla** lo metemos en un *return* para imprimirlo gráficamente.

```
if n == 'protocol':
    string_w += '\n'
    f2 = open(os.getcwd() + '/files/nmap/save' + n + '.l2p', 'w')
    f2.write(string_w[1:])

return string
```

Ahora llegamos por fin a la función la cual es llamada por el botón.

Podemos observar que recibimos dos argumentos, qué tipo de dato es y el dato. En resumen, en la siguiente imagen vemos como si el argumento que recibimos es nulo o es un "*" aplicaremos el script aplicado anteriormente. y sino también ya que es esencial para el siguiente paso. De lo contrario entraremos en esta función.

```
def IPfinder_all_f(port,type):
    string = ''
    port = str(port)
    if port == ''.strip() or port == '*':
        return IPfinder_all(type)
    else:
        IPfinder_all(type)
        f2 = open(os.getcwd() + '/files/nmap/save' + type + '.l2p','r')
        contador1 = 0
```

Leemos un fichero dónde está almacenada nuestra información del script primitivo.

```
IPfinder_all(type)
f2 = open(os.getcwd() + '/files/nmap/save' + type + '.l2p','r')
contador1 = 0
```

Comprobamos si lo que estamos buscando existe en algunas de las líneas del fichero, si es así lo mostramos.

```
if type == 'port':
    string+='\n -----\n \tThe Ports \n ----- \n\n'
    for line in f2:
        l1 = line.split(',')
        if port in l1:
            string += 'IP ' + l1[0] + ' have port ' + port + ' open and this other ports: \n'
            for i in l1[1:]:
                string+=i + '\n'
            contador1+=1
    if (contador1 == 0):
        return ('\n ----- \n\n' + 'This ' + type + ' ' + port + ' ' + " doesn't exist" + '

```

Por último registraremos la salida de este script en un log.

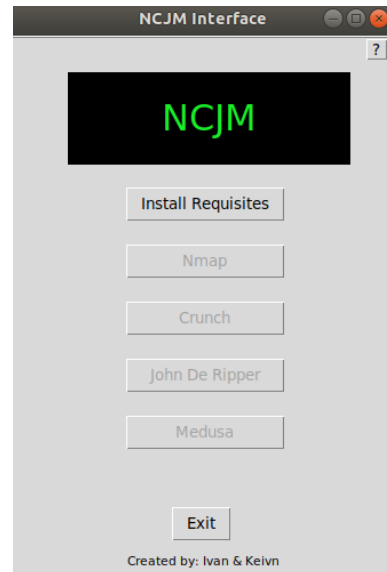
```
f3 = open('/home/alumne/Escriptori/TF/logs/' + type + '.log','w')
f3.write(string[:-1])
return string[:-1]
```

Paso 5.2: Funcionalidades adicionales

Explicaremos un par de **ejemplos** de **funcionalidades** que por necesidad hemos tenido que desarrollar.

- Restricción en caso de no tener los **programas** necesarios instalados:

Así es como luce nuestra **aplicación** cuando no se cumplen los requisitos en cuanto a instalaciones:



¿Cómo hemos conseguido esto?

Pues lo que hacemos es, en el init(el inicializador) de nuestro menú, llamamos a una línea que cuenta y recoge cuántos de los **programas** necesarios tenemos. Si este número es igual a 4, es que están todos los **programas** contabilizados y pasamos a deshabilitar el botón de instalación que ya no tendría sentido si ya tenemos todo, por lo contrario bloqueamos todo el menú, ya que es necesario tenerlo instalado.

```
def init(top, gui, *args, **kwargs):
    global w, top_level, root
    w = gui
    top_level = top
    root = top
    paco = os.system('apt list --installed | grep -e crunch/ -e john/ -e medusa/ -e nmap/ | wc -l > ' +
        os.getcwd() + '/logs/countPrograms.log')
    f = open(os.getcwd() + "/logs/countPrograms.log", "r")
    r = f.read()
    print(r.rstrip())
    if (r.rstrip() == "4"):
        w.Button1.configure(state="disabled")
    else:
        w.Button1.configure(state="normal")
        w.Button2.configure(state="disabled")
        w.Button3.configure(state="disabled")
        w.Button6.configure(state="disabled")
```

¿Qué contiene nuestro botón *install*?

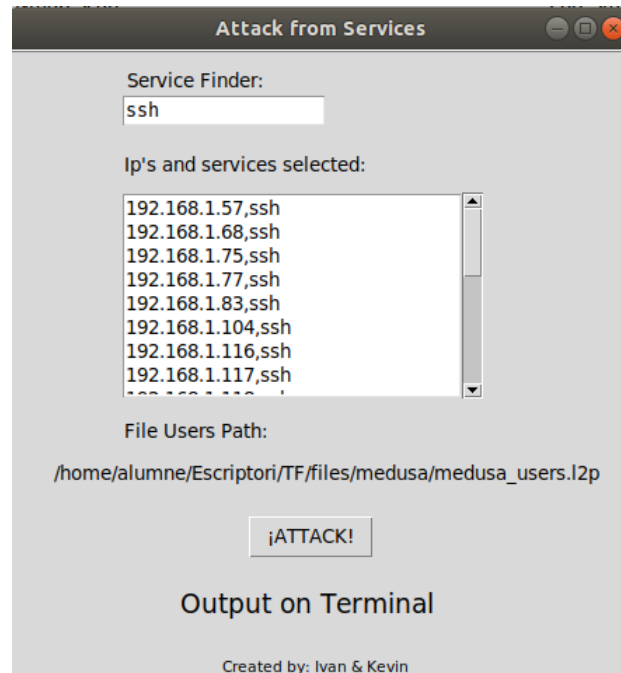
Simplemente cuando le damos, cerramos la ventana, luego llamamos a un script en *bash* donde está todo lo que deseamos instalar o actualizar. Una vez acabado este, volvemos a llamar a menú, ya que automáticamente se cargará de la forma correcta.

```
def install():  
    destroy_window()  
    #os.system('./GUI/install_re  
    iniciarInstalacion()  
    import menu  
    menu.vp_start_gui()  
    sys.stdout.flush()
```

Bash Sript Instalation:

```
# Actualizamos repositorios  
echo 'deb http://archive.ubuntu.com/ubuntu bionic main restricted universe' > /etc/apt/sources.list  
echo 'deb http://archive.ubuntu.com/ubuntu bionic-security main' >> /etc/apt/sources.list  
echo 'deb http://archive.ubuntu.com/ubuntu bionic-updates main restricted universe multiverse' >> /etc/apt/sources.list  
echo 'deb http://archive.ubuntu.com/ubuntu bionic multiverse' >> /etc/apt/sources.list  
# Actualizamos la maquina  
apt update  
# Instalacion del JohnTheReapper  
apt-get -y install john  
# Instalacion de nmap  
apt-get -y install nmap  
# Instalacion del medusa  
apt-get -y install medusa  
# Instalacion del crunch  
apt-get -y install crunch
```

Por último explicaremos nuestra **Interfaz** “Attack from Services”:



Esta **interfaz** es de las más curiosas que tenemos ya que recoge los datos de la **Interfaz nmap**, los rescata y luego los usa para generar un **ataque** en bucle con **Medusa** sobre un range de Ip's.

Pero... ¿Cómo **funciona**?

Para empezar simplemente llamaremos la **funciona** que llama a la nueva **interfaz**.

```
def attackMedusaCompo():  
    global w  
    startAttack()  
    sys.stdout.flush()
```


Este es el **import**:

```
5 from nmapIP import ipForMedusen as ipFM
6 sys.path.append(os.getcwd() + '/GUI/medusaAttack')
7 from attack import vp_start_gui as startAttack
8
```

En el init de **attack_support** encontraremos las siguientes líneas.

```
def init(top, gui, *args, **kwargs):
    global w, top_level, root
    w = gui
    top_level = top
    root = top
    obj2 = w.Entry1
    file = open(os.getcwd() + "/logs/registredLabel.log", "r")
    a = file.read()
    obj2.insert(tk.END, a)
    obj = w.Scrolledtext1
    proc = subprocess.Popen("cat " + os.getcwd() + "/files/nmap/medusaRange.l2p", stdout=subprocess.PIPE, shell=True)
    (out, err) = proc.communicate()
    obj.insert(tk.END, out)
    sys.stdout.flush()
```

Las cuales en resumen, recogen el campo requerido de **nmap** ya guardado en un fichero y ejecutamos un cat sobre el ScrollText, que apunta a un fichero producido por el siguiente script:

```
def ipForMedusen(cosa):
    contador1 = 1
    string=""
    f2 = open(os.getcwd() + '/files/nmap/saveservice.l2p', 'r')
    for line in f2:
        l1 = line.split(',')
        if cosa in l1:
            string+=l1[0] + "," + l1[l1.index(cosa)] + "\n"
    f3 = open(os.getcwd() + '/files/nmap/medusaRange.l2p', 'w')
    f3.write(string)
    return string[:-1]
```

El cual filtra un archivo producido por **nmap**, sobre la búsqueda por puerto y lo filtra para devolver lo mostrado en la primera captura.

Por ultimo tenemos el siguiente botón.

```
import sys, subprocess, os
sys.path.append(os.getcwd() + '/src')
from nmapIP import attackRange as aRange
```

```
def attack():
    aRange()
    sys.stdout.flush()
```

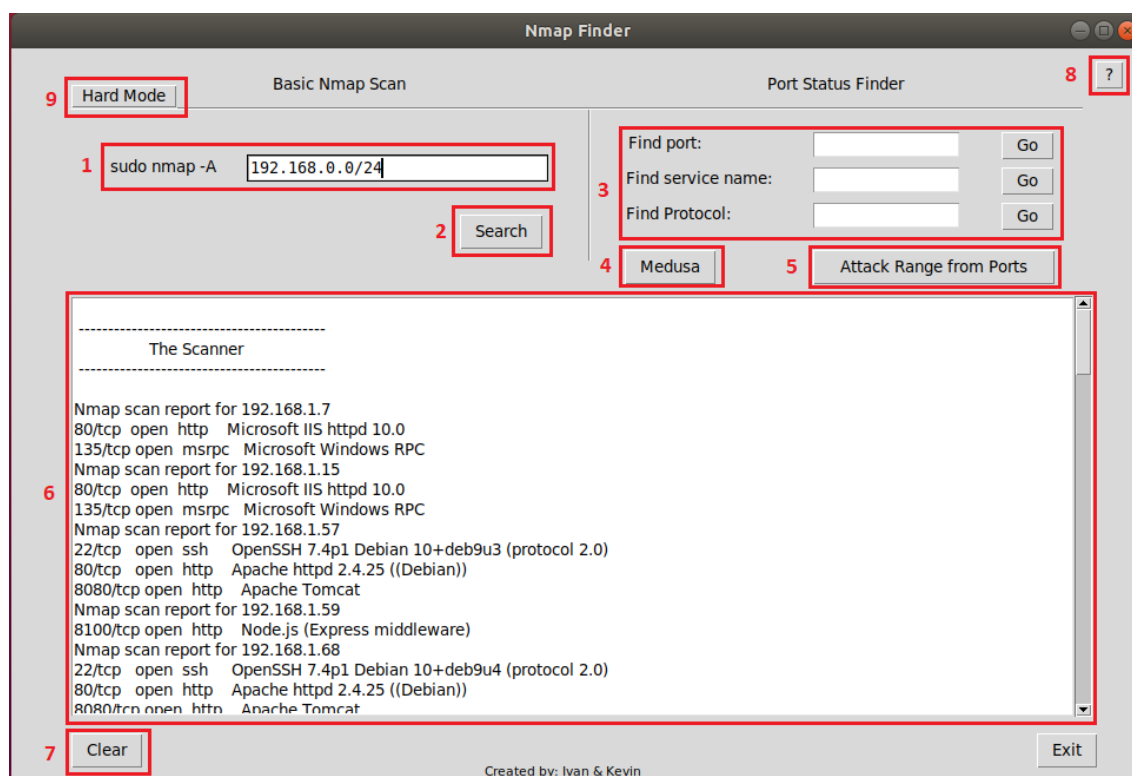
El cual llama a la función attackRange que hace lo siguiente:

```
def attackRange():
    f2 = open(os.getcwd() + '/files/nmap/medusaRange.l2p', 'r')
    for line in f2:
        ll = line.split(',')
        os.system('medusa -h '+ll[0]+' -U ' + os.getcwd() +
            '/files/medusa/medusa_users.l2p -P ' + os.getcwd() +
            '/files/crunch/listAttack.l2p '+' -M '+ll[1].rstrip('\n')+' -f -b -v 6 -e ns')
```

Leemos el fichero donde previamente hemos almacenado las **ip's** y los puertos de los **host** registrados por el **Nmap**, luego procedemos a pasarlos por un 'for' el cual va hacer un **ataque** de **Medusa** por cada línea dentro del fichero.

En conclusión tenemos un **programa** que recoge **ip's** de un rango y aplica un bucle atacando a todas las **ip's** y puertos seleccionados.

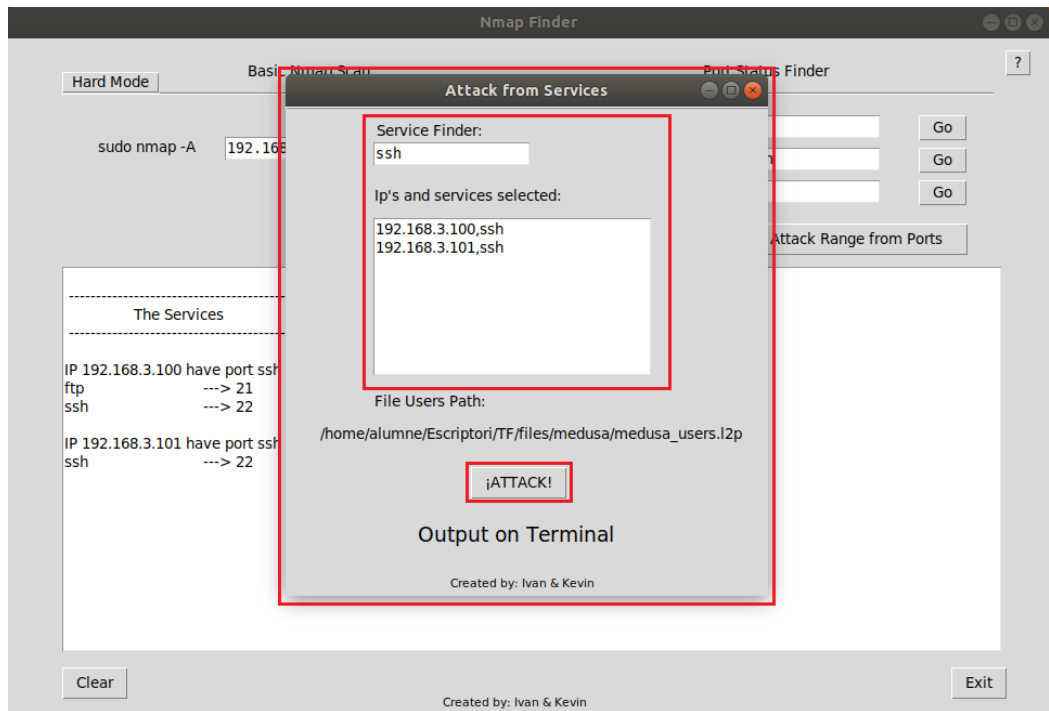
5 FUNCIONALIDADES DE NCJM



5.1 Nmap

1. Cuadro de texto para indicar la IP, rango de **ip's** o una **red** entera como en nuestro caso.
2. Botón 'Search' como su nombre indica para buscar. Este botón ejecuta un comando '**nmap**' para escanear la **red**.
3. Sección para filtrar el escaneo realizado previamente.
 - a. *Find port*: filtro por puerto.
 - b. *Find service name*: filtro por nombre.
 - c. *Find Protocol*: filtro por protocolo.
4. Botón para abrir otra de las **funcionalidades** creadas, el **Medusa**.

5. Esta opción es una **funcionalidad** “nueva” que hemos creado, ya que **funciona** igual que el **Medusa** ataca una **ip** a fuerza bruta, pero ejecutamos el comando **Medusa** en un bucle para el **ataque** a una **lista** de IPs.



6. Cuadro de texto, con logs de lo que está ocurriendo.
7. Botón para limpiar el cuadro de texto.
8. *Helper* (?), para una breve explicación de sus **funcionalidades**.
9. *Hard mode* nos **importa** un escaneo de una **red** xxx.xxx.xxx.xxx/24 ya realizada previamente.

5.2 Crunch

The screenshot shows the 'Crunch' application window titled 'Crunch'. The main heading is 'Make Wordlist with Crunch'. The interface includes several input fields and checkboxes, each highlighted with a red box and a number:

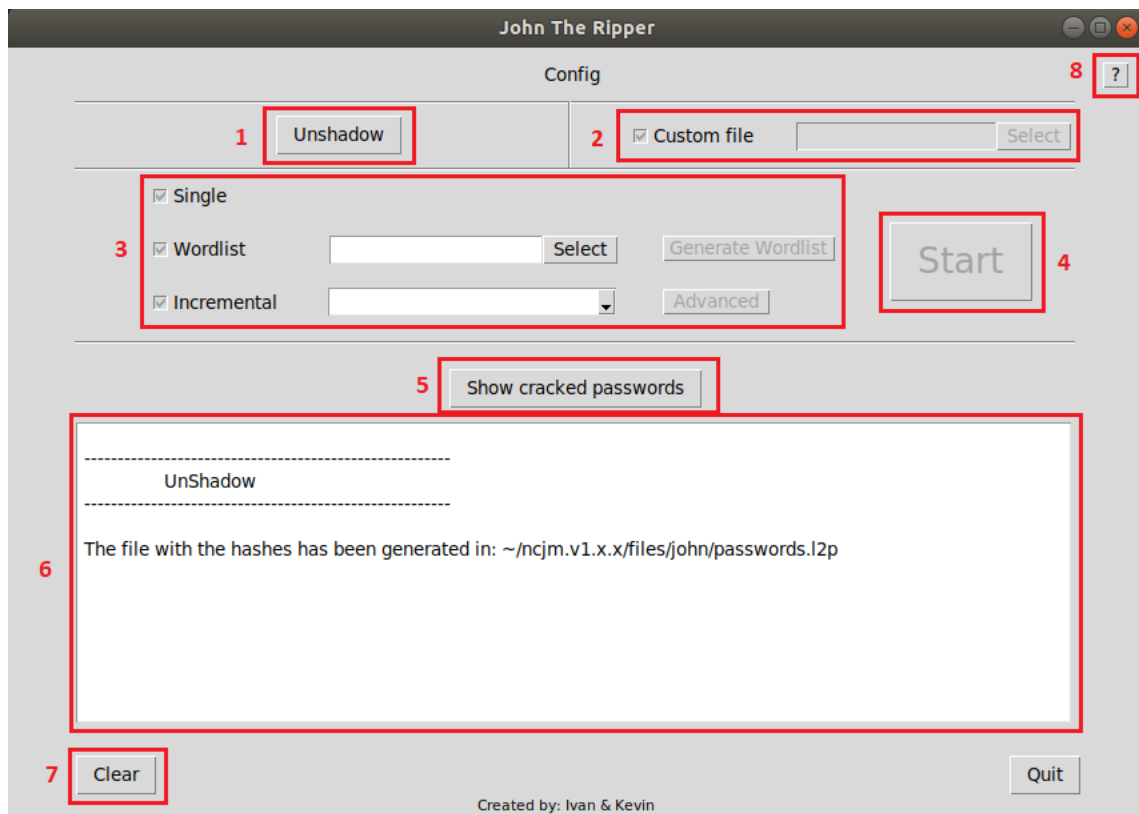
- 1**: A box containing 'Min' and 'Max' fields, both set to '4'.
- 2**: A checkbox labeled 'Destination File' followed by an empty text input field.
- 3**: A group of options including a checked 'Restricted Characters' checkbox with a text field containing '.,@&', and two unchecked checkboxes for 'Mixed Words' and 'Mixed Characters'.
- 4**: A 'Custom Crunch' checkbox followed by an empty text input field and a 'GO!' button.
- 5**: A large text area labeled 'Result' showing a list of generated passwords: AAaa, AAab, AAac, AAad, AAae, AAaf, AAag, AAah, AAai, and AAai.
- 6**: A help icon (?) in the top right corner.

At the bottom of the window, there is an 'Exit' button and a footer that reads 'Created by: Ivan & Kevin'.

1. Parámetro mínimo y máximo de longitud de la contraseña a generar.
2. *CheckBox* para indicar el nombre del fichero al que queremos guardar nuestra **Wordlist**. La ruta por defecto con el *checkbox* marcado es `\"./files/crunch/nombredelfichero\"`.
3. Diferentes opciones que nos permite generar una contraseña.
 - a. Restricted Characters: Especifica un patrón, por ejemplo: `\"@@@con@@@@\"` donde solo las `\"@\"` cambiará.
 - i. `\"@\"` insertará caracteres en minúsculas
 - ii. `\",\"` insertará mayúsculas

- iii. “%” insertará números
 - iv. “^” insertará símbolos
 - b. *Mixed Words*: podemos insertar distintas palabras para hacer la combinación entre ellas.
 - c. *Mixed Characters*: insertamos una cadena de caracteres para hacer todas las combinaciones posibles.
4. Si tenemos conocimientos de **Crunch**, y NCJM no nos da la posibilidad de ejecutar un comando más complejo, lo podemos hacer en esta sección.
 5. Cuadro de texto, con logs de lo que está ocurriendo.
 6. Helper (?), para una breve explicación de sus **funcionalidades**.

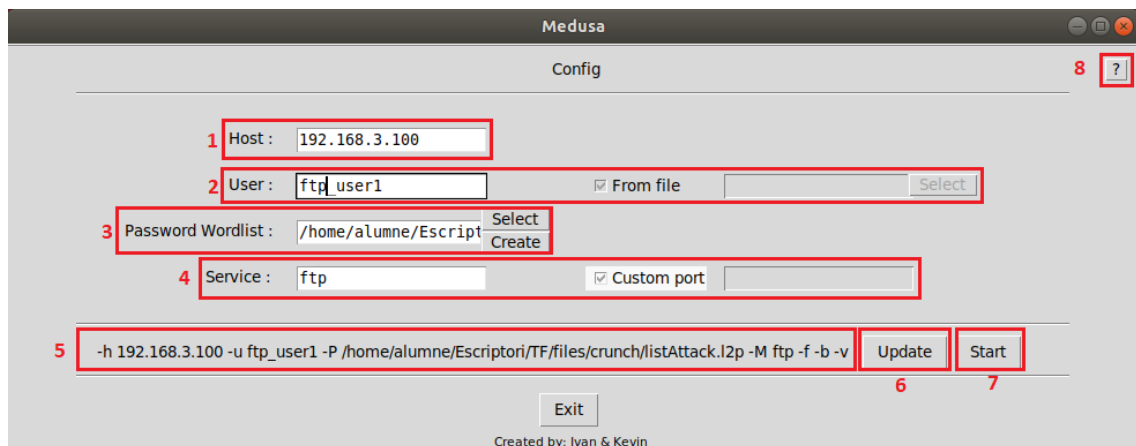
5.3 John The Ripper



1. Botón "Unshadow", el cual ejecuta un comando nativo de **JTR** que aprovecha los archivos `/etc/passwd` y `/etc/shadow` para encontrar el Hash de la contraseña de un usuario de sistema.
2. En este CheckBox podemos seleccionar un archivo propio con un **listado** de Hashes propio.
3. **Funcionalidades** principales de **JTR**:
 - a. Single: Utilizará los nombres de inicio de sesión, el nombre completo del usuario y los nombres de los directorios personales como contraseñas candidatas, también con un gran conjunto de reglas de gestión de errores aplicadas.

- b. Este es el modo de craqueo más simple admitido por **John**. Todo lo que necesita hacer es especificar una **lista** de palabras (un archivo de texto que contenga una palabra por línea).
 - i. Tenemos, además, el botón “Generate **Wordlist**” el cual abre ‘**Crunch**’ y cuando generemos un **listado** de palabras, la ruta donde se guarda esta **Wordlist** se **importa** a **JTR**.
 - c. Este es el modo de craqueo más poderoso, puede probar todas las combinaciones de caracteres posibles como contraseñas. Sin embargo, se supone que el craqueo con este modo nunca terminará debido a que el número de combinaciones es demasiado grande (en realidad, terminará si establece un límite de longitud de contraseña baja o hace que use un juego de caracteres pequeño), y parará antes.
 - i. Para establecer estas limitaciones tenemos el botón “Advanced”, que como su nombre indica es de uso avanzado, pero no tiene complicación alguna. Una vez abierto la **configuración** con este botón, busquemos el formato de contraseña dentro del archivo y editamos la longitud mínima-máxima y los caracteres a combinar.
4. Botón “Start”, que ejecuta el modo marcado en el CheckBox.
 5. Show cracked passwords: enseña todas las contraseñas encontradas, relacionadas con el archivo **importado** o unshadow.
 6. Cuadro de texto, con logs de lo que está ocurriendo.
 7. Botón para limpiar el cuadro de texto.
 8. Helper (?), para una breve explicación de sus **funcionalidades**.

5.4 Medusa



1. Indicamos el **host** a atacar.
2. Tenemos dos opciones: o introducir un nombre o una **lista** de nombres para probar el acceso a un **servicio**.
3. Opción para **importar** una **Wordlist**, la podemos seleccionar o también crear con el botón "Create".
4. Indicamos el nombre del **servicio**, además podemos introducirle un puerto diferente del defecto. No todos los **servicios** existentes pueden ser atacados con **Medusa**, tiene unos en concreto que son: ftp, http, imap, mssql, mysql, ncp, nntp, pcanywhere, pop3, postgres, rexec, rlogin, rsh, rlogin, smbnt, smtp-vrfy, smt, snmp, **ssh**, svn, telnet, vmauthd, vnc, web-form y wrapper.
5. Nos muestra la línea de comando a ejecutar con las configuraciones definidas en los campos anteriores. Se actualiza esta línea con el botón 'Update'.
6. Botón para actualizar la línea de comando.
7. Botón para ejecutar la línea de comando.
8. Helper (?), para una breve explicación de sus **funcionalidades**.

6 CONCLUSIONES

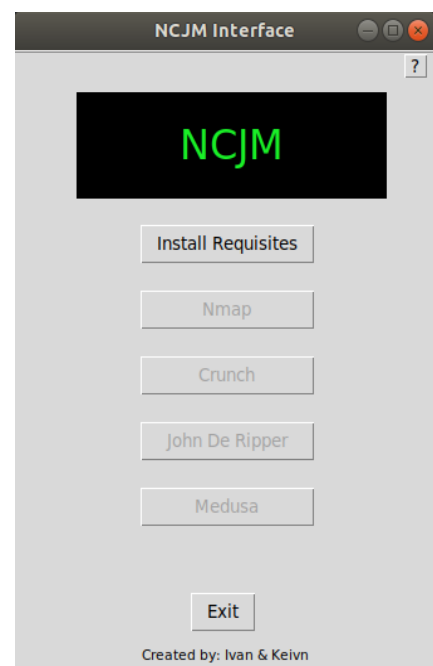
6.1 Objetivos cumplidos

Podríamos empezar nombrando que nuestra **plataforma** además de ser ligera en cuanto a tamaño, también es liviana al uso.

Hemos conseguido integrar en una misma **plataforma** cuatro **programas**, de tipologías muy parecidas, incluso se podría decir que de uso conjunto ya que por ejemplo **Medusa** necesita una **Wordlist** para poder hacer un **ataque** o **John The Ripper** tiene una **funcionalidad** de la cual también depende de una **Wordlist**. ¿Y quién las genera? **Crunch**, pero no tan solo hemos conseguido implementar un unión entre esas necesidades, sino que también hemos conseguido que **Medusa**, con la ayuda de **Nmap**, donde podemos encontrar puertos vulnerables en la **red**, pueda atacar no solo por fuerza bruta si no en masa a un conjuntos de **ip's** rescatadas de **Nmap**, filtrando por **servicios** vulnerables.

Así que podemos concluir que cumplimos el 90% de nuestros objetivos, entre ellos tener una **interfaz** gráfica fácil de usar en la cual incluimos botones de ayuda (para usuarios con cero experiencia en esos aplicativos).

Como último detalle, matizar que hemos intentado generar las máximas restricciones posibles. Un ejemplo claro sería nuestro primer botón del menú llamado “Install Requisites” el cual si todos los **programas** necesarios están instalados se muestra desactivado, pero de lo contrario se activará y desactivará el resto del menú.



6.2 Objetivos no cumplidos

Este tema giraría más entorno a la **programación** debido a los siguientes puntos:

- El código no es limpio
- Se repite código
- Validar entradas por **pantalla**
- Control de errores (**Try/Catch**)
- Comentarios en el código
- Creación de **métodos globales**
- Creación de variables **globales**
- **Redirección** de salidas a terminal (en algunos casos no fue posible)
- Enlaces a documentación oficial dentro de las ayudas

En cuanto a nivel de **plataforma** nos gustaría haber implementado todas las posibilidades de todos los **programas** y sobre todo conseguir hacer un auditor de **red** que fuese capaz de atacar y registrar cómo de segura es una **red**. Ese era el objetivo inicial, un objetivo que nos quedó grande.

6.3 Viabilidad y aplicabilidad de NCJM

Después de todo, podemos afirmar que nuestra **plataforma** es viable según en el entorno en el cual vaya a ser usada.

Por ejemplo, en un grado medio, inclusive un superior para explicar y usar **herramientas** como **Crunch**, **Nmap**, **Medusa** y **Jhon the Ripper**. Debido a que estas **herramientas** a veces pueden ser difíciles de entender, sobre todo si no se sabe interpretar la documentación. Es una forma fácil y sencilla de usarlos.

Ya que el coste del **proyecto** son 2100 € contando que el **hardware** lo ponga quien contrate el **servicio**, podríamos afirmar que tener a dos **programadores** que diseñen, creen, programen, debuguen y testeen la **plataforma** no es un precio demasiado alto, sobre todo teniendo en cuenta que las **utilidades** requeridas están implementadas y fusionan de forma fluida.

En la parte menos buena de todo el **proyecto** deberíamos ser **realistas** ya que realmente es poco tiempo y poco dinero invertido. Esto será proporcional a la calidad de nuestra **aplicación**, con esto nos referimos a que no todas la **funcionalidades** de todos los **programas** han sido implementadas y de hecho quedarían muchas cosas que mejorar a nuestro ver.

Pero afirmamos que en un entorno **low level**, nuestra **plataforma** puede tener una buena **utilidad**.

7 CONCLUSIONS

7.1 Objectives not met

This topic would turn more around to the programming due to the following points:

- The code is not clean
- Code is repeated
- Validate entries by screen
- Error control (**Try / Catch**)
- Comments in the code
- Creation of **global** methods
- Creation of **global** variables
- **Redirection** of outputs to terminal (in some cases it was not possible)
- Links to official documentation within the aid

Regarding the platform **level**, we would like to have implemented all the possibilities of all the programs and, above all, to get a network auditor capable of attacking and registering how secure a network is. That was the initial objective, an objective that was great for us.

7.2 Viability and applicability of NCJM

After all, we can affirm that our platform is viable according to the environment in which it will be used.

For example, in a medium degree, including a superior to explain and use tools such as **Crunch**, **Nmap**, **Medusa** and **Jhon the Ripper**. Because these tools can sometimes be difficult to understand, especially if you do not know how to interpret the documentation. It is an easy and simple way to use them.

Since the cost of the project is € 2100, counting on the **hardware** provided by the person who hires the service, we could say that having two programmers design, create, program, debug and test the platform is not a very high price, especially considering account that the required utilities are implemented and merged smoothly.

In the less good part of the whole project we should be realistic since it is really little time and little money invested. This will be proportional to the quality of our application, with this we mean that not all the functionalities of all the programs have been implemented and in fact there would be many things to improve our view.

But we affirm that in a **low-level** environment, our platform can have a good use.

8 PRESUPUESTO

En este apartado se describe una estimación detallada de los costes de realización de este trabajo final de grado. Estos costes se presentan agrupados en diversas categorías.

- Costes del personal involucrado en el **proyecto**.
- Costes de elementos **software** y **hardware**.

Para el cálculo del presupuesto se debe tomar en cuenta los siguientes aspectos:

- Los costes que se mostrarán estarán expresados en euros.
- Se tomarán dos decimales para las cantidades económicas, **redondeando** en el caso de que sea necesario.

Nombre	Categoría	Euros/hora	Horas Trabajo	Coste Total
Kevin Morillo Gallego	Junior Developer	15	70¹	1050 €
Ivan Chirmici Coeva	Junior Developer	15	70¹	1050 €
Software				Coste Total
Ubuntu Linux 18.04				0 €
Page GUI Generator				0 €
Atom Editor				0 €
Python2.7 , Python3, Python-tk				0 €
Hardware²				Coste Total
2x Ordenadores				1500 €
Periféricos				200 €

¹ [Distribución de horas.](#)

² Costes incluidos solo en el caso de empezar un **proyecto** des de cero, ya que normalmente un desarrollador tiene su propia máquina de trabajo.

Descripción	Coste
Personal	2100 €
Equipos sin Hardware / Equipo con Hardware	0 € / 1700 €
COSTE TOTAL DEL PROYECTO ³ : €	2100 € / 3800

³ Coste dividido en: coste básico sin **hardware** y coste avanzado con **hardware**.

REFERENCIAS Y BIBLIOGRAFIA

[1] Manuales de Python.

<https://www.python.org/doc/>

[2] Manual Pages GUI Generator.

<http://page.sourceforge.net/html/index.html>

[3] Manual Atom Editor.

<https://atom.io/docs>

[4] Python-tk manual.

<https://docs.python.org/2/library/tkinter.html>

[5] Oracle VirtualBox.

<https://www.virtualbox.org/wiki/Documentation>

[6] Manuales de los distintos comandos de bash.

<http://manpages.ubuntu.com/>

[7] GitHub NCJM.

<https://github.com/kmg19951/NCJM>