Hacking Tools



Hacking Tools es una herramienta de hacking desarrollada en **Python 3.7.2**. Ésta librería o herramienta es modular, con la posibilidad de ampliarse fácilmente.

Iremos viendo detenidamente durante el documento como funciona y qué podemos llegar a hacer con ésta librería.

Contenido

[Estructura 3](#_Toc31229176)

[¿Cómo se usa en Python ésta librería? 4](#_Toc31229177)

[Instalación 4](#_Toc31229178)

[Instalación Web GUI Django 4](#_Toc31229179)

[Comandos básicos 5](#_Toc31229180)

[Nombre de los módulos 5](#_Toc31229181)

[Nombres de las categorías 5](#_Toc31229182)

[¿De qué categoría es un módulo en concreto? 5](#_Toc31229183)

[¿Qué módulos son de una categoría en concreto? 5](#_Toc31229184)

[Cargar un módulo 5](#_Toc31229185)

[¿Qué funciones tiene un módulo? 6](#_Toc31229186)

[¿Qué es un worker o thread? 6](#_Toc31229187)

[Librerías del core 7](#_Toc31229188)

[Config (python/json) 7](#_Toc31229189)

[Connections 8](#_Toc31229190)

[Logger 9](#_Toc31229191)

[printMessage 9](#_Toc31229192)

[print\_and\_return 9](#_Toc31229193)

[Pool 10](#_Toc31229194)

[Desarrollando Módulos 11](#_Toc31229195)

[¿Cómo se desarrolla un módulo? 11](#_Toc31229196)

[¿Cómo se desarrolla una simple función en un módulo? 12](#_Toc31229197)

[¿Cómo implemento una API en una función de un módulo? 18](#_Toc31229198)

[¿Cómo hago un fichero cifrado de mis claves APIs portable? 19](#_Toc31229199)

[¿Cómo uso el Logger? 22](#_Toc31229200)

## Estructura

**hackingtools**

* **modules**
  + **\* Categories \***
    - **\* Module Name \***
      * **\*ht\_moduleName.py \***
* **core**
  + **Config**
  + **Connections**
  + **Logger**
  + **Objects (En desarrollo)**
  + **Pool**
  + **Repositories (Para desarrollo en Django)**
  + **Utils**

Así sería la estructura básica de la librería como tal. **En** **core** encontraremos varios scripts y ficheros en json para la configuración; y **en modules** encontraremos una estructura de carpetas que serán las categorías de los módulos y luego dentro de cada categoría hay X módulos como tal.

#### Si vas a desarrollar un módulo para la librería

Con la ayuda de Django Hacking Tools ( hackingtoolsgui ) podemos crear módulos y probarlos como si fuesen microprogramas.

Gracias a la Web en Django y todo el desarrollo en backend propio, se ha conseguido hacer una web dinámica que se crea sola a partir de los módulos que detecta creados en ésta librería.

Automáticamente, una vez desarrollemos un módulo, la web se creará automáticamente con los cambios, de forma que no tendremos que tocar AJAX, HTML, CSS, ni hacer formularios para cada función, etc.

Simplemente se recarga la web y tendrás tus funciones con formularios modales, mediante AJAX y CSRF Token como método de seguridad.

La librería de Hacking Tools se sitúa en el directorio:

***hackingtoolsgui/core/library/hackingtools***

# ¿Cómo se usa en Python ésta librería?

## Instalación

Primero que todo debemos tener la versión más actualizada de Hacking Tools en el entorno de python. Para ello lo instalaremos de ésta forma:

**pip install hackingtools -U**

El parámetro -U forzará a nuestro sistema a revisar si hay versiones nuevas, ya que muchas veces, nuestro sistema intenta buscar en caché los ficheros que tenemos de instalación y es posible que tengáis cacheada una versión quizás desactualizada de la librería.

Si tenemos algún problema con alguna dependencia, la intentaremos resolver con pip mediante la instalación de los paquetes faltantes. En caso de no ser efectivo hay 2 formas:

1 - Revisar en GitHub el archivo requirements.txt que está en la raíz. Podemos instalar de golpe todos los requirements haciendo:

**pip install -r requirements.txt**

2 - Contactar con *@ConnorHack* para resolver dudas o problemas con la herramienta.

Una vez tenemos la librería instalada, podremos: cargar una consola en python y empezar a importar hackingtools y llamar a sus funciones, o podemos abrirnos un fichero en python e importar la librería y empezar a usarla.

Así se importa la librería en un script:

**import hackingtools as ht**

Así importamos librerías que queramos del **core** (por ejemplo):

**from hackingtools.core import Connections, Logger, Utils, Pool**

## Instalación Web GUI Django

Para la instalación de la web con la librería, tengo un repositorio en GitHub donde tenéis una documentación básica para instalar y cargar la web:

<https://github.com/ConnorXploit/hackingtoolsgui>

## Comandos básicos

Para empezar con Hacking Tools, deberemos conocer al menos alguna que otra función principal que tiene la librería.

### Nombre de los módulos

Con esta función podremos obtener los nombres de todos los módulos que han cargado con HackingTools en forma de lista. *Si no aparece en dicha lista un módulo que sabemos que existe, será que no se ha podido importar dicho módulo por 2 posibles problemas:*

*1 - Fallo de sintaxis en algún punto de ese script ( ht\_crypter.py - por ejemplo )*

*2 - Alguna importación de dicho módulo ( ht\_bruteforce.py - por ejemplo) no ha podido realizarse porque no está instalado ese paquete del import en el entorno python.*

Comando: **ht.getModulesNames()**

### Nombres de las categorías

Las categorías son una forma de representar a un conjunto de módulos del mismo tipo. Igual que podemos tener una herramienta que hace búsqueda de información pública en shodan, otro módulo para instagram, otro con nmap, etc. podríamos juntarlos en un mismo grupo o categoría. En ese caso, serían módulos de la categoría OSINT.

Para obtener la lista de nombres de categorías existentes, podemos hacer:

Comando: **ht.getCategories()**

### ¿De qué categoría es un módulo en concreto?

Para resolver de forma sencilla esta pregunta, simplemente deberemos ejecutar la siguiente función:

Comando: **ht.getModuleCategory(‘ht\_shodan’)**

### ¿Qué módulos son de una categoría en concreto?

Con este siguiente ejemplo, obtendremos una lista de todos los módulos de una categoría concreta:

Comando: **ht.getModulesFromCategory(‘osint’)**

### Cargar un módulo

Para inicializar un módulo deberemos asignar en una variable la respuesta de este comando:

Comando: **modulo\_nmap = ht.getModule(‘ht\_nmap’)**

### ¿Qué funciones tiene un módulo?

Para obtener una lista de las funciones que tiene un módulo hay 2 formas de hacerlo:

1 - Podemos ejecutar un simple comando desde Hacking Tools para obtener una lista con las funciones de un módulo:

Comando: **funciones\_modulo = ht.getFunctionsNamesFromModule(‘ht\_nmap’)**

2 - Iniciamos el módulo y llamamos a su función help() de ésta forma:

**modulo\_nmap = ht.getModule(‘ht\_nmap’)**

**modulo\_nmap.help()**

**# ht.getModule(‘ht\_nmap’).help()**

### ¿Qué es un worker o thread?

Un worker es un thread para manejar de forma sencilla los hilos del programa. Podremos ejecutar periódicamente de forma totalmente asíncrona cualquier función que queramos.

Para ello, imaginad que tenemos la siguiente función:

from **datetime** import **datetime**

def **dimeLaHora**():

print(**datetime.now()**)

Ésto imprimirá la hora actual cada vez que ejecutemos la función *dimeLaHora*. Bien, pues imaginemos que necesitamos que cada 1 minuto se imprima en consola, sin importar lo que esté haciendo el programa por delante. Vamos a crear un worker para ver cómo funciona:

**ht.worker(‘nombre-identificativo’, dimeLaHora, timesleep=60)**

La ejecución del programa sigue pero el worker se queda en ejecución en segundo plano. Cada minuto veremos en consola la hora sin tener que llamar a la función.

Atentos! La función que se le pasa como parámetro no lleva paréntesis, ya que no la tenemos que ejecutar nosotros. Le hemos puesto un nombre identificativo para poder controlar dicho worker más tarde. Los workers se guardan dentro de Hacking Tools y debemos controlar los nombres para cuando necesitemos coger el worker por algún motivo:

**el\_worker = ht.getWorker(‘nombre-identificativo’)**

Actualmente con el worker no podríamos hacer prácticamente nada más que apagarlo para que deje de funcionar. Para ello podemos hacer join() al worker para pararlo:

**el\_worker.join()**

## Librerías del core

### Config (python/json)

Este fichero es **esencial** para cargar la configuración de Hacking Tools, parámetros de los módulos, APIs, etc. Principalmente, este fichero tiene que ir junto con **config.json.**

*Recomiendo usar siempre el Config que lleva cargado Hacking Tools ya que podríamos tener conflicto si hay cambios en local del fichero config.json.* ***No importar con la línea****:*

***~~from hackingtools.core import Config~~****.* *Puede provocar conflictos.*

Para usar el Config de Hacking Tools es tan sencillo como hacer:

**ht.Config**

Y las funciones más esenciales son las siguientes:

#### Cargar la configuración de un módulo en una variable

**config\_shodan = ht.Config.getConfig(‘modules’, ‘ht\_shodan’)**

*( Devuelve un diccionario { ‘clave’ : ‘valor’, … } )*

#### Ver las posibles APIs

**ht.Config.getAPIsNames()**

*( Devuelve una lista [ ‘ipinfodb\_api’, ‘shodan\_api’, … ] )*

#### Coger una API de la lista de APIs:

**clave = ht.Config.getAPIKey(‘shodan\_api’)**

*( Devuelve un string con la clave si existe, si no texto vacío )*

#### Establecer una clave a una API, aunque no exista:

**ht.Config.setAPIKey(‘shodan\_api’, ‘abcdef0123456789’)**

#### Exportar las APIs a un fichero cifrado htpass:

**ht.Config.saveRestAPIsFile(‘fichero\_claves.htpass’, ‘password’)**

#### Cargar APIs de un fichero cifrado htpass:

**ht.Config.loadRestAPIsFile(‘fichero\_claves.htpass’, ‘password’)**

#### Añadir un servidor Hacking Tools como nodo de la pool:

**ht.Config.add\_pool\_node(‘https://hackingtools-hosted.com’)**

#### Quitar un nodo de la pool de Hacking Tools:

**ht.Config.remove\_pool\_node(‘https://hackingtools-hosted.com’)**

### Connections

Desde este fichero se controlan las conexiones, las IPs, los servicios e incluso alguna opción dedicada para los servidores de Heroku (su comportamiento en Heroku cambia).

Para acceder a las conexiones será tan sencillo como:

**ht.Connections**

Si no queremos tener problemas en **Heroku** con los servicios, deberemos establecer una variable de entorno desde el servidor. Debe llamarse: **DYNO**. Le pondremos valor **1**.

#### ¿Cuáles son mis servicios iniciados?

**ht.Connections.getMyServices()**

*( Devuelve una lista [ ‘http://127.0.0.1:5000’, ‘http://192.168.1.123:5000’, ... ] )*

#### Añadir un servicio

**ht.Connections.addMineService(‘http://192.168.1.140:1234’)**

#### ¿Cuál es mi IP pública?

**ht.Connections.getMyPublicIP()**

*( Devuelve un string con la dirección ip pública )*

#### ¿Cuál es mi IP en Lan?

**ht.Connections.getMyLanIP()**

*( Devuelve un string con la dirección ip de la red wifi o lan )*

#### Iniciar Ngrok con Puerto

**ngrok\_url = ht.Connections.startNgrok(1234)**

*( Devuelve un string con la dirección de ngrok )*

#### Detener Ngrok por URL

**ht.Connections.stopNgrok( ngrok\_url )**

## Logger

Este fichero es una forma sencilla de imprimir mensajes con colores y categorizando el tipo de mensaje ( user / module / core ) en la terminal.

Las funciones que tiene esenciales son:

### printMessage

- Argumentos obligatorios:

*message* {String} -- El mensaje principal para el print

- Argumentos opcionales:

*description* {String} -- Una descripción (default: {None})

*debug\_module* {bool} -- ¿Es llamado por un módulo? (default: {False})

*debug\_core* {bool} -- ¿Es llamado por el core? (default: {False})

*is\_error* {bool} -- Si es error, saldrá de otro color (default: {False})

*is\_warn* {bool} -- Si es warn, saldrá de otro color (default: {False})

*is\_info* {bool} -- Si es info, saldrá de otro color (default: {False})

*is\_success* {bool} -- Si success, saldrá de otro color (default: {False})

*color* {[type]} -- Para mostrar con un color concreto (default: {None})

**ht.Logger.printMessage(‘un mensaje’, description=‘con su descripción’)**

### print\_and\_return

Hace lo mismo que el printMessage, pero, se puede asignar a una variable la respuesta de ésta función para no tener que almacenar un valor, imprimir y devolverlo. Un par de ejemplos:

**modules\_names = ht.Logger.print\_and\_return( ht.getModulesNames() )**

Ésta línea de arriba recoge los nombres de los módulos, imprime en consola la lista de módulos y a su vez lo devuelve a la variable modules\_names sin tener que hacer lo siguiente:

*(ESTO NO)*

**modules\_names = ht.getModulesNames()**

**ht.Logger.printMessage(modules\_names)**

## Pool

La pool es un concepto que se añadió pensando en la distribución de muchos nodos de Hacking Tools donde las herramientas puedan ayudarse entre sí con las tareas complejas, las que queramos anonimizar o distribuir por varias partes del mundo.

La idea es la siguiente: tenemos que hacer un escaneo de nmap pero quizás desde nuestro punto donde tenemos montado Hacking Tools tenemos prohibida la visión de ese objetivo. Podemos conectarnos con otros nodos y hacer que hagan un escaneo de nmap sobre el objetivo que nosotros queremos.

La pool se maneja mediante los servicios de las Connections y un ID de nodo único que se le da a cada aplicación de Hacking Tools cargada para poder comunicarse por la pool sin tener problemas con bucles infinitos en caso de un rebote de petición de nuevo contra nosotros mismos. Esto puede suceder si hubiese por ejemplo 3 nodos y se conocen entre ellos, ya que pueden intentar rebotar la conexión a otro nodo de forma aleatorio y volver a nosotros mismos creando un conflicto con las peticiones y pudiendo entrar en bucle infinito.

Para acceder a la Pool, podemos usar la pool cargada con Hacking Tools:

**ht.Pool**

#### Ejecutar en la Pool una función (que lo haga otro)

***params = {}***

***params[‘ip’] = ‘8.8.8.8’***

***params[‘tcp’] = True***

***params[‘udp’] = True***

**respuesta = ht.Pool.sendNow(‘ht\_nmap’, ‘getDevicePorts’, params=params )**

#### Activar o desactivar la Pool

**ht.Pool.switchPool()**

#### Añadir Nodo a la Pool

**ht.Pool.addNodeToPool(‘http://hackingtools-node.com’)**

#### Informar otros Nodos sobre mis Servicios

**ht.Pool.callNodesForInformAboutMyServices()**

#### Ver la Lista de Nodos en la Pool

**ht.Pool.getPoolNodes()**

*( Devuelve una lista [ ‘http://server1.com’, ‘http://server2.com’, ... ] )*

#### Borrar Nodo de la Pool

**ht.Pool.removeNodeFromPool(‘http://hackingtools-node.com’)**

# Desarrollando Módulos

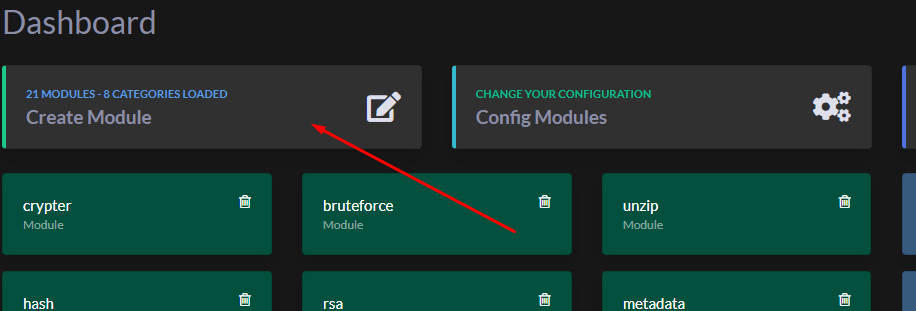
## ¿Cómo se desarrolla un módulo?

Principalmente tenemos que tener **Python >= 3.7.X**, la **última versión de HackingTools** instalada en vuestro entorno de Python **y todos los requirements** que trae el repositorio de GitLab (mirar en [Instalación](#_Instalación_Web_GUI), más arriba).

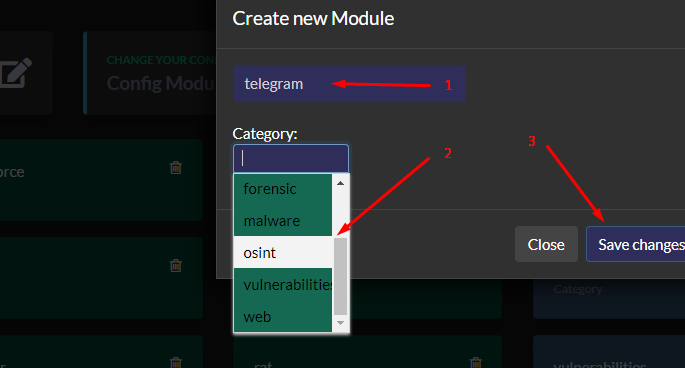
Una vez iniciemos HackingTools (**python server.py**), veremos que carga un servidor que podremos visitar en <http://127.0.0.1:2222>

Dicho esto, vamos a la URL que indico anteriormente, cargaré la web y le daremos a éste botón

:



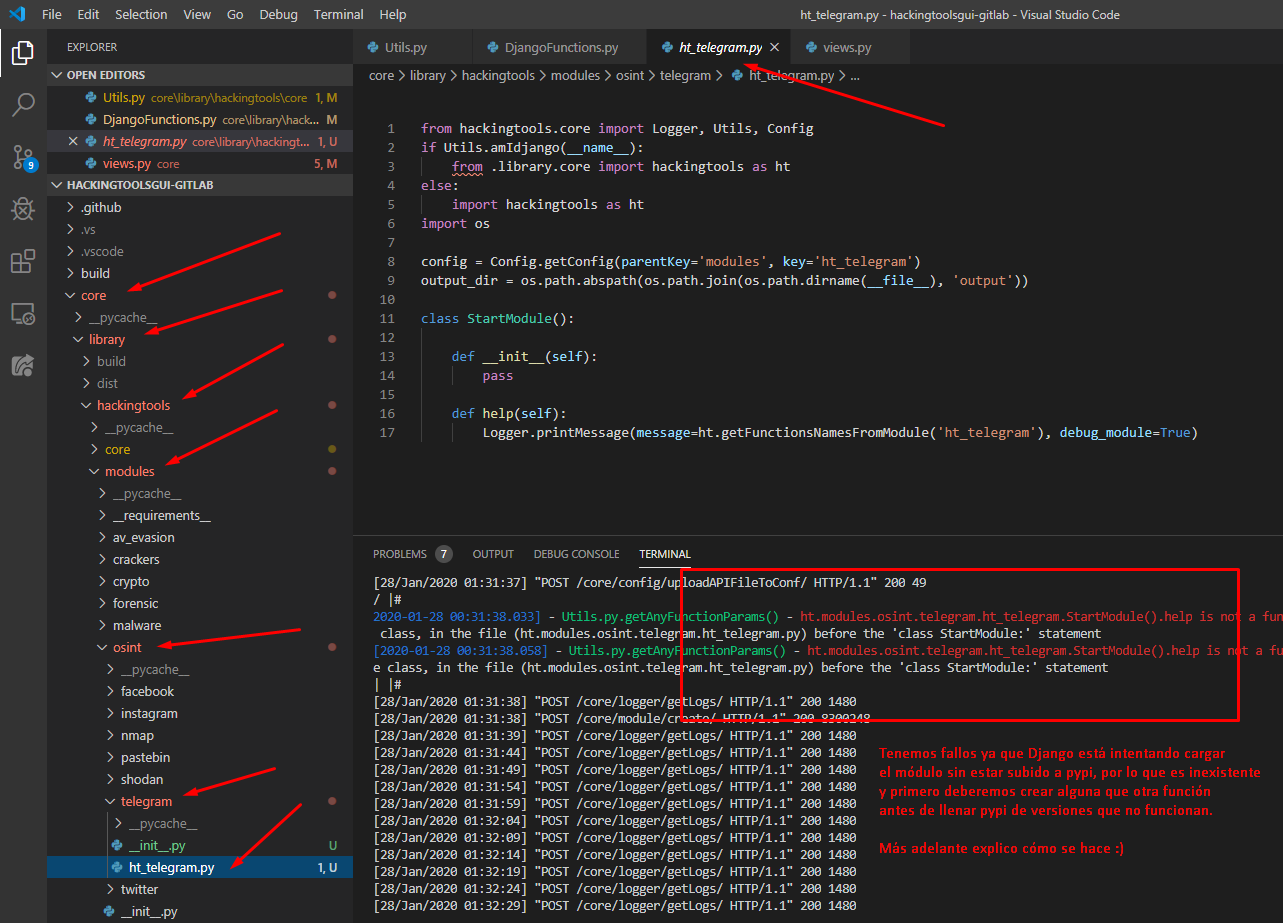
Nos aparecerá la siguiente ventana donde tendremos que poner un nombre de módulo, por ejemplo: *Telegram*, y meterlo en una categoría existente e incluso podemos escribir una nueva en ése mismo campo de selección:



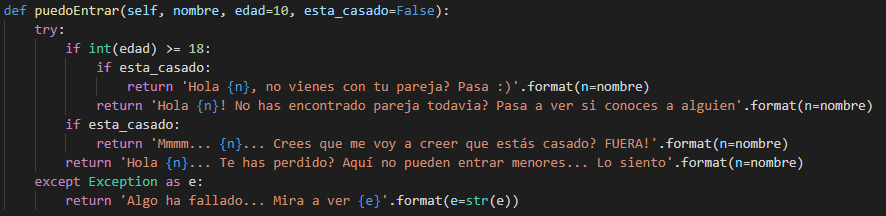
Una vez se refresque la web, todo se habrá creado y ya podremos volver a nuestro IDE de desarrollo favorito, en mi caso Visual Studio Code, y comenzar a editar nuestro módulo.

## ¿Cómo se desarrolla una simple función en un módulo?

Ya tenemos un módulo donde lo ubicaremos en **core/library/hackingtools/modules**, dentro de **la categoría** que hemos elegido o creado, dentro de una carpeta llamada como **el módulo** que has creado. Aquí veremos **2** ficheros**: \_\_init\_\_.py** (necesario para poder importar el módulo) y el módulo como tal, **ht\_telegram.py** en este caso.

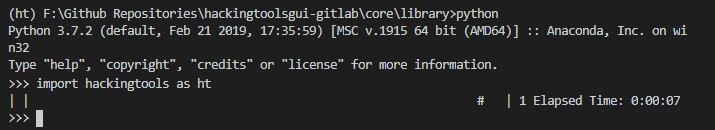


Ahora crearemos una función. Para no complicarnos con el manual, haremos una función simple que reciba como parámetro el nombre de una persona, una edad y si está casado o no. Como digo, esto solo será un simple ejemplo:

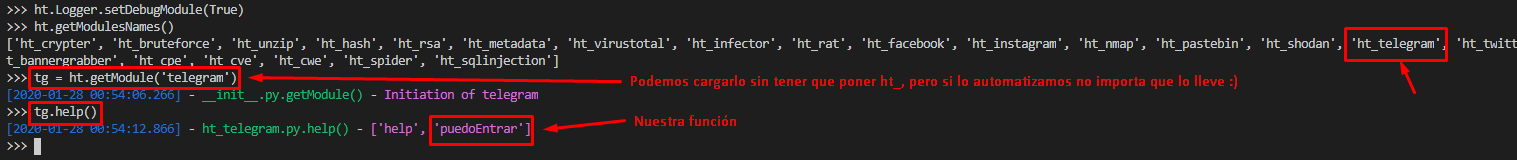


Al guardar veremos que **Django reinicia la interfaz gráfica**. De momento no hagamos caso a la web.

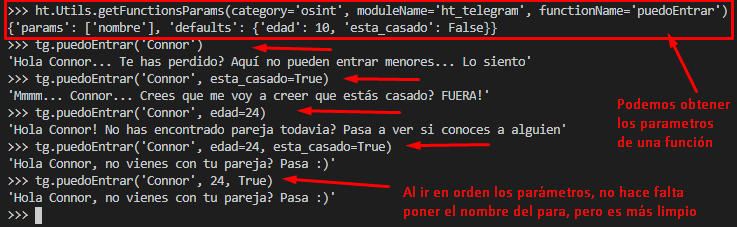
**Vamos a una terminal** y nos situamos en la carpeta del proyecto **core/library**. Una vez ahí, **ejecutamos Python** e **importamos hackingtools** (lo hacemos desde éste directorio para que el comando que os marco en la imagen de abajo pueda importar la versión local y no la versión instalada en el entorno de Python).



Ahora ya podemos ver que módulos están cargados e incluso cargar uno. Primero activaré el log de los módulos para ver que nos dice la función **help()** de nuestro módulo:

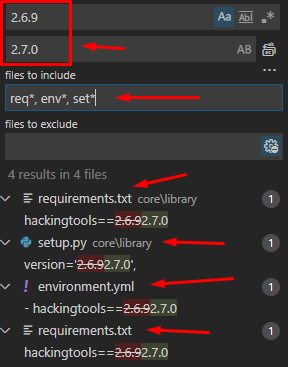


Ahora podemos **obtener los parámetros necesarios** de la función a la que vamos a llamar si lo necesitamos y después simplemente instanciamos nuestro módulo para finalmente ejecutar la función con los parámetros que queramos.



El siguiente paso será **subir ésta versión** que funciona a pypi para que Django deje de gritar y nos cree automáticamente todas las URLs, vistas, HTML y AJAX necesario para que funcione nuestra web con la función que hemos creado.

**Volvemos a la raíz** **del proyecto** y veremos 2 ficheros que se llaman **upload\_to\_pypi.bat** y **upload\_to\_pypi.sh**. Podéis ver lo que hacen, pero ya están pensados para que entren al directorio necesario, compile el proyecto, borra automáticamente los directorios y carpetas temporales que genera y sube a pypi la versión nueva del proyecto. Para éste proceso será importante editar la versión en los siguientes **4 ficheros** (podéis hacer la búsqueda con los parámetros que tengo puestos yo):



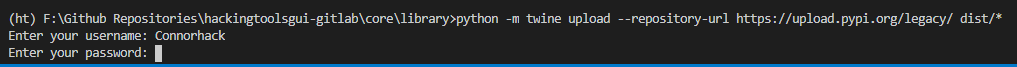
***Antes de darle a “remplazar todo”,*** *asegúrate que están nada más los 4 ficheros donde debe aparecer referenciada la versión de “****hackingtools****” con ello y en versión del setup.py, ya que puede haber librerías de JS que tendrán listas de versiones a las que apuntarán fuera del servicio y eso no necesitamos que se modifique.*

Una vez hecho el remplazo, **ejecutamos el archivo para subir el fichero a pypi**.

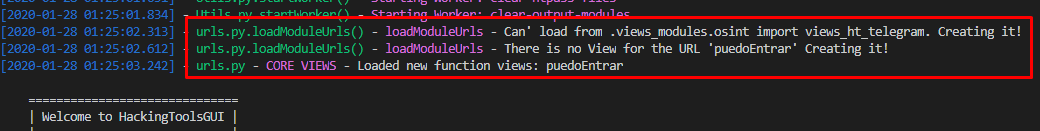


Una vez empiece su ejecución, **no parar bajo ningún concepto** ya que está tocando todos los ficheros del proyecto y al generar la versión, puede dejar ficheros residuales (no se sincronizan con GitLab) pero te dejará archivos en local que son innecesarios. Además, si paras su ejecución, asegúrate estar de nuevo en el directorio raíz del proyecto, ya que éste script trabaja todo sobre el directorio **core/library**.

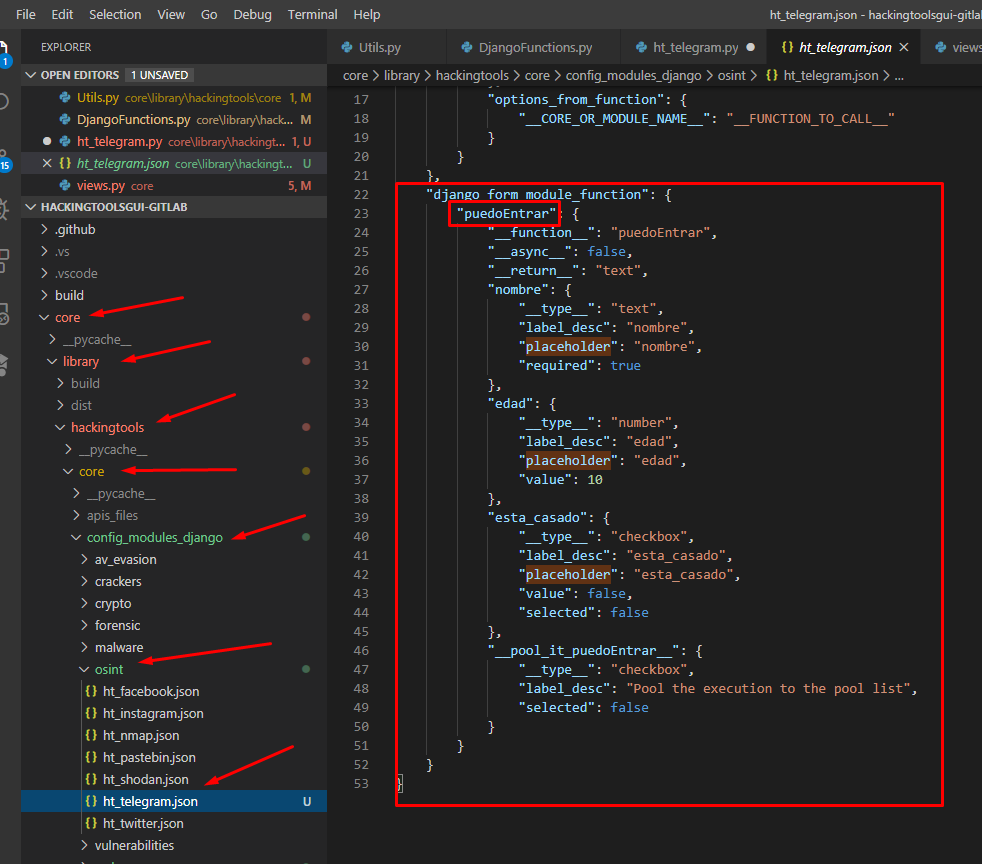
Durante su ejecución, nos pedirá las credenciales para subir cambios a pypi. **Deberéis tener permisos** que antes tendréis que pedirme expresamente a mi @connor.



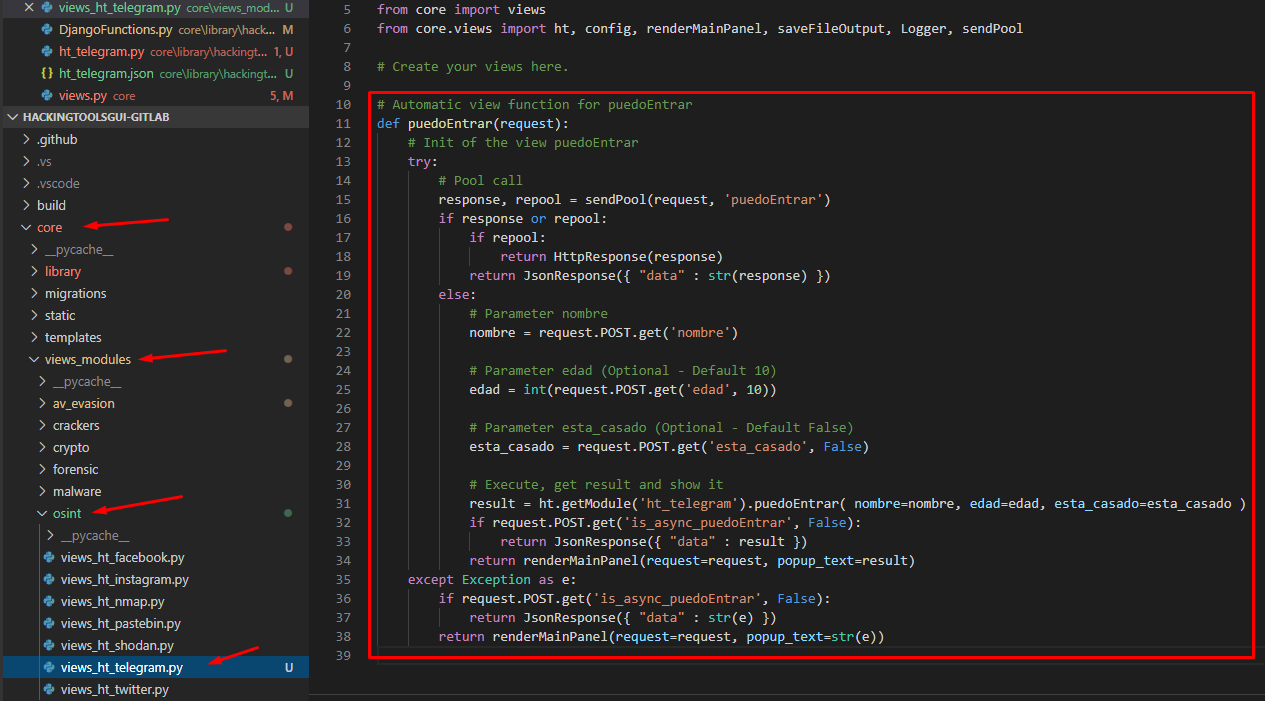
Cuando finalice, veremos que nuestro script habrá actualizado de manera automática y repetida Hacking Tools para verificar su satisfactoria instalación.

Ya podemos iniciar la interfaz gráfica ejecutando **Python server.py**. Ahora veremos que al iniciar nos pone bastantes mensajes en el log que antes no salían y se reinicia de manera automática:

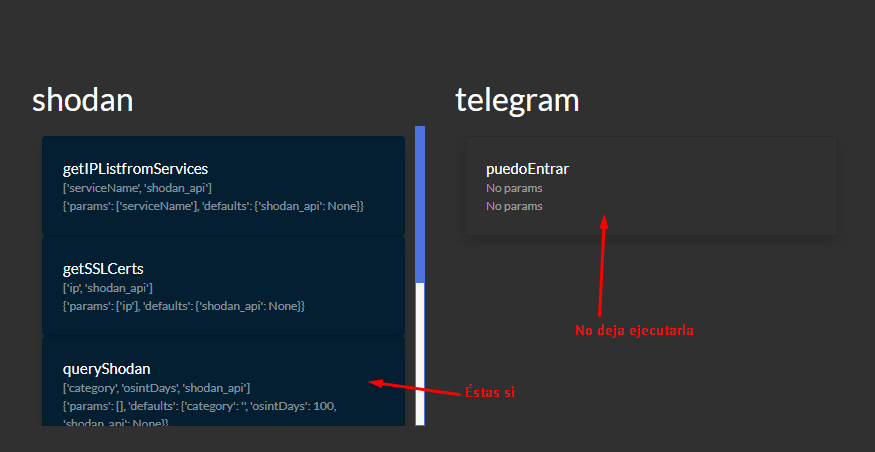
Podemos ver cómo nos ha generado un archivo de configuración para la interfaz de manera automática:



Y también tenemos otro fichero automáticamente generado que sería la vista de nuestra función del módulo:



Ahora si vamos a la interfaz web, veremos que sí encuentra nuestra función, pero no podemos todavía ejecutarla de manera gráfica.

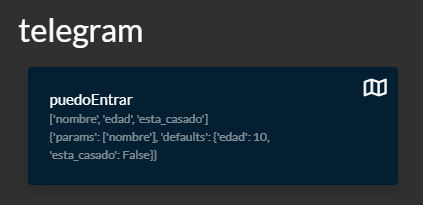


Esto pasa por lo mismo que antes. Django está usando la librería instalada hace un par de pasos, pero si nos fijamos, al iniciar el servidor hace un momento, cuando nos ha reiniciado, nos ha creado las vistas y la configuración, y claro, Django no lo encuentra en la libraría que está instalada desde Pypi, por lo que es necesario volver a hacer una nueva versión con la configuración actual y ya tendremos el formulario.

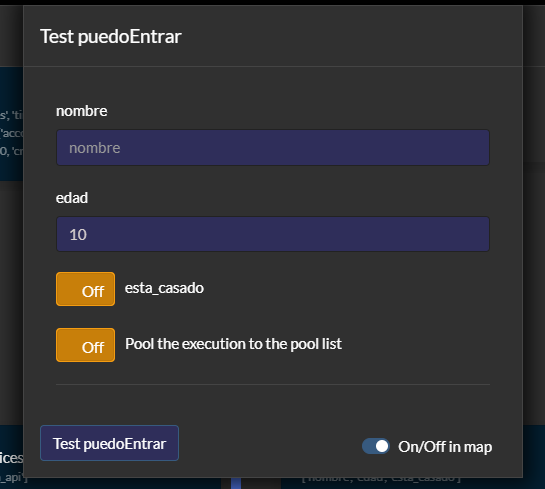
**Haremos los mismos pasos que antes:**

1. **Cambiar la versión**
2. **Subir a pypi**
3. **Ejecutamos el servidor**

Ahora no deberían salir errores en la consola. *En caso de algún error, borrar el archivo dentro de* ***core/views\_modules/*** *que corresponda con tu módulo, o borra la función de dentro de ése fichero que corresponda a la función que no está funcionando en la web. También hacer lo mismo con* ***core/library/hackingtools/core/config\_modules\_django/*** *y el fichero que corresponde con la misma lógica del anterior fichero de las vistas.*

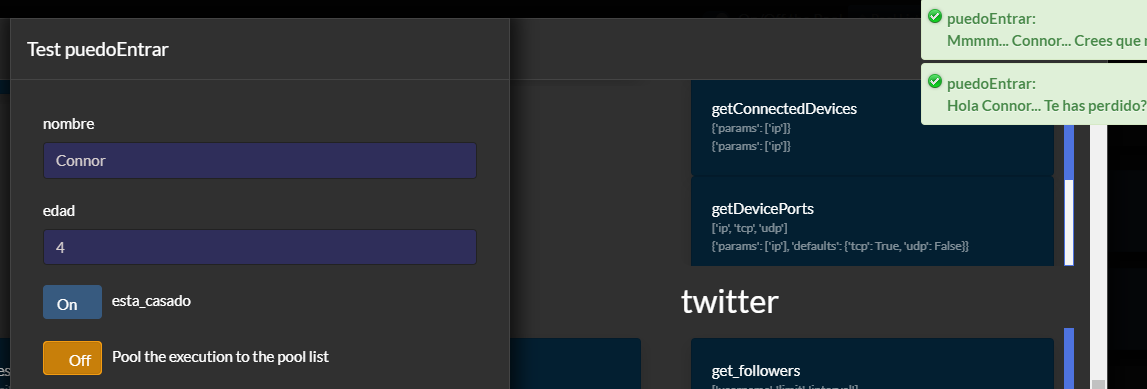


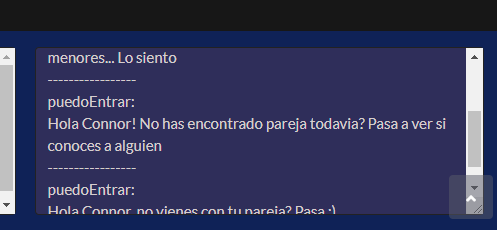
Si hacemos clic, podremos ver el formulario gráfico. Si aparece vacío únicamente con el botón de probar módulo, cerrar y el switch del mapa, haz el paso anterior que está en *cursiva.*



Como podemos ver, tiene los parámetros por defecto asignados. Podemos darle a “Test” para probarlo y recibir la respuesta tanto por un pop-up como en una caja de respuestas que está situada en la parte inferior del panel principal de la web.

Aquí un ejemplo de cómo funciona y donde aparece la notificación y el resultado, pudiendo copiar sin problema el texto si es necesario para el usuario.

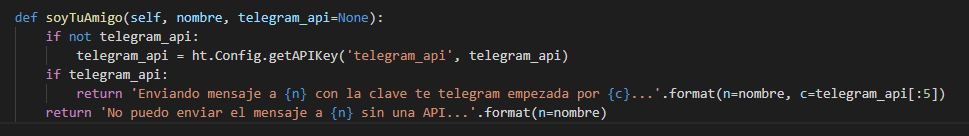




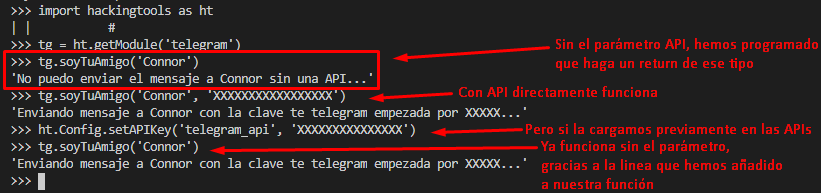
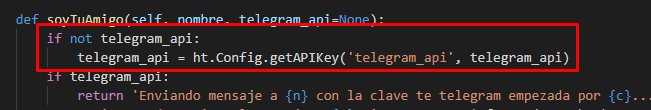
Y si queremos que otros usuarios no tengan que generar todos éstos ficheros al cargar Django y hacer que se reinicie el servidor, o si queremos que ciertos cambios que hagamos a mano, como por ejemplo los ficheros que hemos detallado antes de configuración **JSON** y el que corresponde a la **vista del módulo**, como por ejemplo lo tiene el *crypter*, podemos subirlo a GitLab para que tenga efecto en el resto con la nueva versión de Hacking Tools que hayas subido.

Ya tenemos la forma de hacer un módulo con funciones y como usarlas de manera gráfica.

## ¿Cómo implemento una API en una función de un módulo?

Para implementar una API, únicamente tenemos que hacer una función donde reciba un **parámetro para la API**, pero **no es un parámetro obligatorio**, es decir, le daremos él valor **por defecto** de **None**. Deberemos configurar algo muy simple para que nuestro módulo coja la API de la configuración en caso de no ser pasada como parámetro. También tendremos que tener en cuenta que el nombre del parámetro para la API, debe ser **nombremodulo\_api**, es decir, **sin ht\_**. Por ejemplo:

Nos ponemos en el directorio para desarrollo en **una terminal** (**core/library**) como antes, iniciamos Python, importamos Hacking Tools y probamos la función:



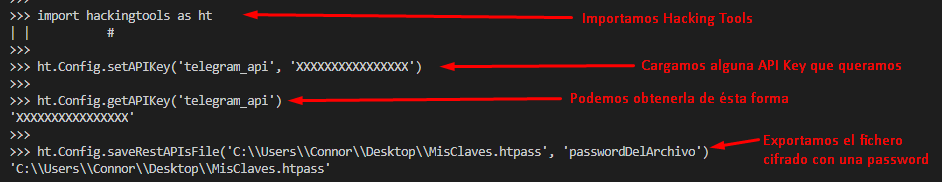
Ya tenemos una función con un parámetro para usar una API. Ahora ya puedes usarla en la ejecución de lo que necesites usando ése parámetros.

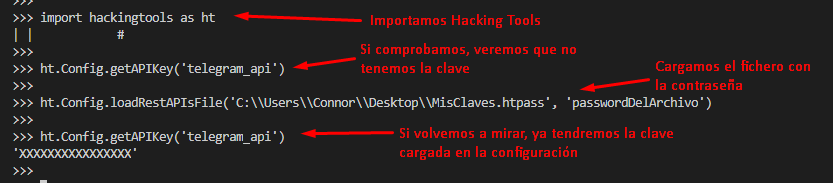
## ¿Cómo hago un fichero cifrado de mis claves APIs portable?

Puede parecer engorroso tener que meter las claves API en cada script que hacemos usando Hacking Tools. Por eso, he creado esta solución muy simple para poder guardar de manera cifrada las claves APIs, para poder llevarnos con nosotros un simple archivo **.htpass**, la cual podremos importar con Hacking Tools y la clave que le hemos puesto.

Veamos cómo podemos darle uso desde la consola y desde la web.

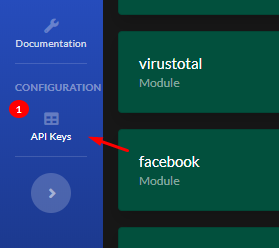
Desde una consola, podemos importar primero las claves manualmente y después exportar un fichero cifrado de ésta forma:



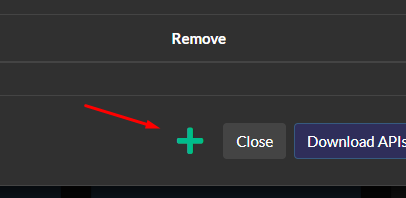
Ahora podremos en otro script hacer lo siguiente para cargar las claves APIs:

Desde el entorno web, podemos acceder al panel de API Keys, donde agregaremos a mano las APIs por primera vez y después podremos usar el mismo formulario para cargar dichas APIs:

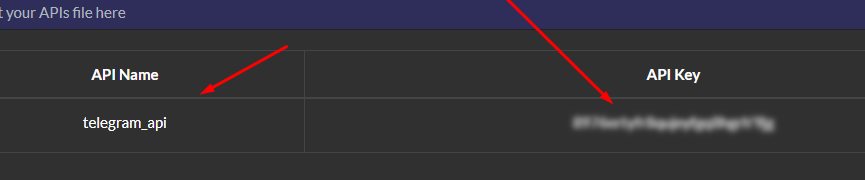
1. Primero entraremos en el panel:

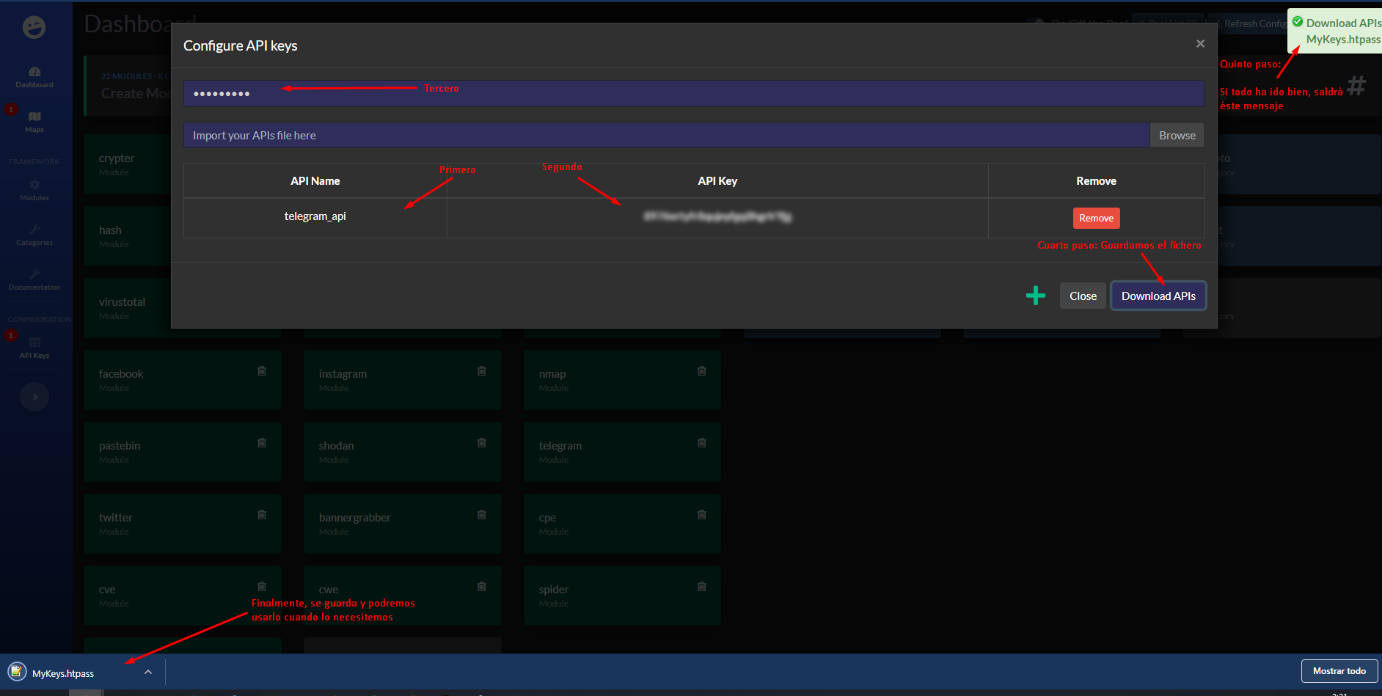


1. Ahora le daremos al botón del más cada vez que queramos crear una nueva entrada en una lista de APIs:



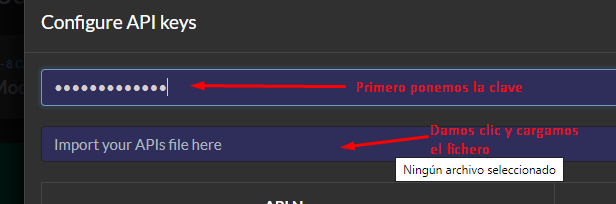
1. Rellenamos el nombre y la API:



1. Ahora deberemos poner una contraseña y guardar las claves agregadas para poder cargarlas, ya que el entorno web solo carga desde los ficheros las claves. Si solo añadimos en la tabla la clave, pero no ha sido cargada desde un fichero, no será detectada por Django. Lo dicho: ponemos clave y guardamos:

Ahora procedemos a cargar el fichero en la web o en otra instancia cualquiera de Hacking Tools. *Éste fichero serviría tanto para Web como para los scripts que nos hagamos y viceversa.*

1. Cargamos la web, vamos al menú que ya conocemos de API Keys.
2. Ponemos primero la contraseña del fichero que vamos a cargar.
3. Elegimos el fichero en el campo de debajo.

*Una vez seleccionado el fichero, el formulario es enviado automáticamente y se refrescará la web si todo ha ido bien.*

1. Podemos ver las claves cargadas abriendo de nuevo el menú de API Keys.

## ¿Cómo uso el Logger?

Para usar el Logger, hay 2 formas de importarlo. Los módulos, ya lo traen importado desde **hackingtools.core**, ya que está pensado para ser ejecutado como módulo.

La otra forma es **usada por Django** y es usada para coger los ficheros locales y se haría importando **ht.Logger** para usar la instancia que mismamente usa la herramienta, que también deberá estar importada en el fichero que estés trabajando.

**Si no es así**, por estar en el directorio **core**, será tan simple como hacer: **from . import Logger**.

Puedes basarte en los ejemplos que verás en otros ficheros en la misma altura de directorios que el tuyo, ya que Django no trabaja igual que la librería que estamos creando debajo de él.

Ten en cuenta que son 2 proyectos en 1.