Quick Distributed Brute Force

Una herramienta simple para realizar ataques de fuerza bruta distribuidos.

Flags

`-b <cantidad>` (500) Cantidad de peticiones en paralelo (reducir si el programa crashea)

`-o <nombre archivo>` (out.log) Archivo de salida de peticiones que cumplen con el criterio

`-p <puerto>` (7575) Puerto para la comunicación entre helpers e instancia principal

Uso

`qdbf [<flags>] [<config.yml>]`

Si no se especifica archivo de configuración se inicia en modo "helper" donde se espera a que una instancia principal se comunique y le envié lo que tiene que hacer.

Helpers

La herramienta se puede usar de forma 'standalone' o distribuida. En el archivo de configuración se pueden especificar de 0 a muchas direcciones IP de máquinas que ayudarán en el ataque. Estas máquinas serán llamadas "helpers".

Estructura de archivo de configuración (config.yml)

```
request:
 method: <GET | POST | PUT | ...>
 url: http://test/
 params: # parámetros query en formato clave: valor
   username: Test
   password: bee
   id: "$id$" # las variables se definen entre '$'
 headers:
   cookie: "sessionid=ab3235bac5"
 body: '{"user":"test", "tst":55}'


params: # valores que asumirán las variables
 id: # nombre de variable
   type: RANGE # tipos: RANGE, DICT, FILE
   from: 1
   to: 10000


criteria:
 type: <STOP | LOG> # frenar o guardar y seguir
 response:
```

```
    status: 200
    headers:
      cookie: "sesion=sa4d65541a"
    body: 'sjdhaksjdh' # soporta REGEX


helpers: # otras instancia que ayudarán
 - 192.168.1.1
 - 123.45.67.8
```

Request

Se especifica todo lo necesario para realizar la petición: URL, método, headers, body y parámetros de query (en formato `<nombre param>: valor`). Los parámetros query, el body y los headers son opcionales.

```
request:
 method: <GET | POST | PUT | ...>
 url: <url>
 params:
   nombre_param: <valor>
 headers:
   nombre_header: <valor>
 body: <contenido del body>
```

Variables/Params

Se definen entre '$' (ej. `$nombre$`) y pueden estar en el body, algún header, el path de la URL, o algún parámetro de query. En la sección de `params` se define qué valores tendrá la variable en cada petición. No es necesario que se utilice la variable, si quieres realizar 100 peticiones pero no quieres que cambie nada entre ellas simplemente crea una variable RANGE de 1 al 100 y no la uses en ninguna parte de la petición.

Las variables RANGE se incrementan de uno en uno en cada petición desde `from:` hasta `to:`.

```
nombre_var:
   type: RANGE
   from: <inicio>
   to: <fin>
```

Las variables DICT toman sus valores de una lista de valores especificada en el mismo archivo de configuración con `dict:`

```
nombre_var:
   type: DICT
   dict: ['a', 'b', 'c']
```

Las variables FILE toman sus valores de una lista de valores que se encuentra en un archivo, cada línea de archivo se toma como un valor de la lista. Se especifica el nombre del archivo con `file:`

```
nombre_var:
   type: FILE
   file: <nombre archivo>
```

Criterio de corte [opcional]

Indica que debe contener una respuesta para ser aceptada. Se puede especificar contenido de encabezado con `headers:` (igual formato que la petición), código de status con `status:` y contenido del body con `body:` especificando una expresión regex.

```
criteria:
   type: <STOP | LOG>
   response:
       status: <codigo status>
       headers:
           nombre_header: <valor>
       body: <regex del contenido esperado>
```

No es necesario especificar todos estos campos, por ejemplo si quiero frenar el ataque cuando el código de status es 200 seria:

```
criteria:
   type: STOP
   response:
       status: 200
```

Se puede especificar si cuando una respuesta cumpla con el criterio se frene el ataque `type: STOP` o solamente se la guarde y el ataque continúa `type: LOG`. En ambos casos los resultados serán guardados en el archivo de salida especificado con el flag `-o` (out.log por defecto), si se están usando helpers este archivo estará solo en la instancia principal pero loguea los resultados de todas las demás instancias.