

EPISODIO 1

982

Célula, desde un universo paralelo, ha vuelto al futuro de nuestros héroes en búsqueda de su cuerpo perfecto, para ello necesita una información que sólo una persona es capaz de proporcionarle. En la búsqueda de la información, y tras un enfrentamientos con uno de nuestros héroes, Célula consigue escapar con vida jurando que volvería con el cuerpo perfecto y eliminaría la Tierra por completo.

Debes ayudar a los héroes de la Tierra con el fin de evitar que Célula consiga su objetivo. Llegan a la conclusión de que han de encontrar a la persona buscada por Célula antes que éste. Para ello, y con la ayuda del radar de Bulma, deciden ir en busca de las bolas de dragón para conocer quién es el objetivo de Célula a través de Shenron. ¿Serás capaz de conseguir el nombre?

**** Es necesario que deis acceso a vuestra ubicación para que funcione correctamente ****

Servicio contra el que comprobar el nombre: 34.253.120.147:9999

Radar de Bulma: <https://34.253.120.147/dragonball/episodio1/>

Info: La flag tiene el formato UAM{md5}

Si cargamos la Web del Radar con el navegador, nos pide la geolocalización y nos muestra un radar que indica que no hay ninguna bola en los alrededores:



Mirando las peticiones HTTP, vemos como hace dos:

Estado	Método	Archivo	Tipo	Transferido	Tamaño	Duración
200	GET	/dragonball/episodio1/	html	646 B	505 B	453 ms
200	POST	server.php	html	216 B	13 B	93 ms

La primera de ellas carga la página que se visualiza. La segunda se invoca a continuación, y es la petición que envía las coordenadas de ubicación, y con la que calcula si hay alguna bola cerca:

E...	Mé...	Archivo	Tipo	Transferido	Ta...	Duración	Cabeceras	Cookies	Parámetros	Respuesta
200	GET	/dragonball/episodio1/	html	646 B	505 B	453 ms	Filtrar los parámetros de la petición			
200	POST	server.php	html	216 B	13 B	93 ms	▼ Datos de formulario			
404	GET	favicon.ico	html	506 B	290 B	74 ms	lat: 3324037			
							lng: 5097480796			

Y la respuesta que devuelve lleva formato JSON:

E...	Mé...	Archivo	Tipo	Transferido	Ta...	Duración	Cabeceras	Cookies	Parámetros	Respuesta
200	GET	/dragonball/episodio1/	html	646 B	505 B	453 ms	Filtrar propiedades			
200	POST	server.php	html	216 B	13 B	93 ms	▼ JSON			
							success: 0			
							▼ Vista preliminar			
							{ "success": 0 }			

Por tanto, debemos montar un proceso automático que vaya probando combinaciones de latitud y longitud hasta que nos vaya encontrando bolas. Para ello usaremos Python. Lo primero preparamos una llamada de ejemplo, hasta que consigamos que nos devuelva ese mismo resultado:

```
dataGeo = "lat=57.23452625&lng=-24.76549375"
r = session.post("https://34.253.120.147/dragonball/episodio1/server.php", data=dataGeo, verify=False, proxies=dict_http_proxy)
contenido = r.content
print contenido
```

Las primeras llamadas nos devuelven un “forbidden”, por lo que faltarán algunos headers en la petición. Miramos desde el navegador qué cabeceras importantes envía, vamos probando y enviando estos headers ya nos devuelve correctamente la respuesta:

```
session.headers.update ({ "User-Agent": "Mozilla/5.0 (X11; Linux x86_64; rv:57.0) Gecko/20100101 Firefox/57.0" })
session.headers.update ({ "Referer": "https://34.253.120.147/dragonball/episodio1/" })
session.headers.update ({ "Content-Type": "application/x-www-form-urlencoded; charset=UTF-8" })
session.headers.update ({ "X-Requested-With": "XMLHttpRequest" })
```

Así que ahora ya es aplicar fuerza bruta. Buscamos cuales son los rangos exactos de longitud y latitud:

Coordenadas decimales

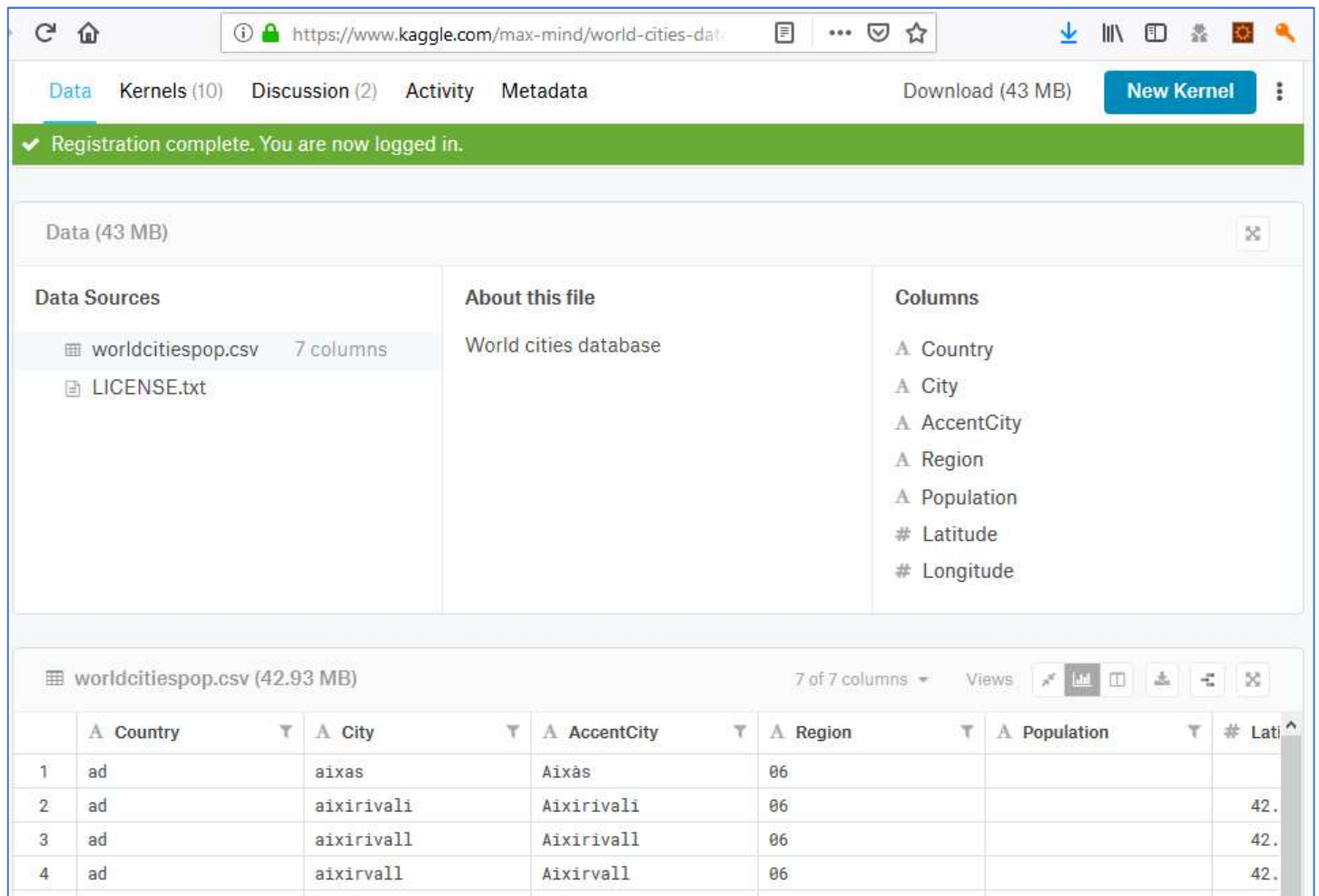
La latitud y longitud son números decimales con las siguientes características:

- Latitud entre 0° y 90 °: Hemisferio Norte,
- Latitud entre 0° y -90°: Hemisferio Sur,
- Longitud entre 0° y 180°: Al este del meridiano de Greenwich,
- Longitud entre 0° y -180°: Al oeste del meridiano de Greenwich.

Vamos a probar con una precisión de incremento de 0.1 grados cada vez. Por tanto, a nivel de total de petición saldrían 180*10 para la latitud, y 360 * 10 para la longitud, lo que hará un total de 6.480.000 peticiones al servidor.

Para acelerar el proceso, vamos a aplicar multithreading en el script, lanzaremos 20 threads en paralelo. Sin embargo, la velocidad no es muy alta, conseguimos aprox. unos 1,5 peticiones / segundo en cada threads, así que el proceso podrá tardar unas 60 horas aproximadamente. Sin embargo, cada cierto tiempo el servidor sufre cortes, o parones, por lo que es imposible conseguir resultados con el script.

Vamos a cambiar de estrategia, probamos también a descargar una base de datos de ciudades del mundo, con sus coordenadas, y probarlas una a una desde el script.



Data Sources

- worldcitiespop.csv 7 columns
- LICENSE.txt

About this file

World cities database

Columns

- Country
- City
- AccentCity
- Region
- Population
- Latitude
- Longitude

worldcitiespop.csv (42.93 MB)

7 of 7 columns Views

	Country	City	AccentCity	Region	Population	Latitude
1	ad	aixas	Aixàs	06		
2	ad	aixirivali	Aixirivali	06		42.
3	ad	aixirivall	Aixirivall	06		42.
4	ad	aixirvall	Aixirvall	06		42.

Descargamos el fichero, y lo formateamos para quedarnos con los campos que nos interesan. Y vemos que algunas ciudades están repetidas, por tener distintos nombres parece, le pasamos por uniq para descartarlas:

```
nacho@kali:~/UAM/201907-BolaDrac1$ cut -d "," -f 6-8 worldcitiespop.csv | uniq | wc -l
2980162
```

Generamos el fichero:

```
nacho@kali:~/UAM/201907-BolaDrac1$ cut -d "," -f 6-8 worldcitiespop.csv | uniq > ciudades.csv
```

Vemos que, aun así, sigue habiendo coordenadas repetidas, pero no van en orden en el fichero, así que le pasamos un sort | uniq de nuevo:

```
nacho@kali:~/UAM/201907-BolaDrac1$ sort ciudades.csv | uniq > ciudadesSinRepetir.csv
```

Y en total nos quedarán algo más de 2.000.000 de ciudades:

```
nacho@kali:~/UAM/201907-BolaDrac1$ wc -l ciudadesSinRepetir.csv
2243467 ciudadesSinRepetir.csv
```

Con esto el tiempo se reduciría a alrededor de 20 horas, preparamos el script y lo lanzamos. Vemos que nos salen ya algunas bolas (3) después de 2 / 3 horas de ejecución:

```
Geo: lat=33.5166667&lng=36.2666667

Resul: {"stars":1,"city":"Damasco","lat":33.513645,"lng":36.276762,"locInRadar": "<circle cx=\"150\" cy=\"150\" r=\"10\"></circle>"}

Geo: lat=36.746115&lng=-5.161443

Resul: {"stars":2,"city":"Ronda","lat":36.745473,"lng":-5.161438,"locInRadar": "<circle cx=\"250\" cy=\"125\" r=\"10\"></circle>"}

```


Geo: lat=64.15&lng=-21.95

```
Resul: {"stars":6,"city":"Reikiavik","lat":64.145144,"lng":-21.942496,"locInRadar":"<circle cx=\"80\" cy=\"80\" r=\"10\"></circle>"}
```

Sin embargo, seguimos sufriendo cortes en el servidor, que provocan que haya que reiniciarlo continuamente, y sea imposible completar el script (ni siquiera una parte significativa), por lo que no conseguimos más que esas tres bolas.

Finalmente, los admins suben el rango de coordenadas al PHP que busca las bolas, así que a su vez nosotros podemos rebajar la precisión del script. En el de buscar ciudades no puedo hacer mucho, pero retomo la estrategia del script inicial que probaba todas las combinaciones, y le subo la precisión (incremento de peticiones desde 0.1 a 0.6). También elimino algunos rangos (0 a 20 y de 70 a 90), ya que viendo que realmente lo que busca son ciudades y no puntos perdidos por el mapa, en dichos rangos extremos hay muy pocas ciudades.

Con estos nuevos rangos, ya conseguimos hasta 6 bolas:

Geo: lat=33.5166667&lng=36.2666667

```
Resul: {"stars":1,"city":"Damasco","lat":33.513645,"lng":36.276762,"locInRadar":"<circle cx=\"150\" cy=\"150\" r=\"10\"></circle>"}
```

Geo: lat=36.746115&lng=-5.161443

```
Resul: {"stars":2,"city":"Ronda","lat":36.745473,"lng":-5.161438,"locInRadar":"<circle cx=\"250\" cy=\"125\" r=\"10\"></circle>"}
```

Geo: lat=47.1833333&lng=106.3

```
Resul: {"stars":4,"city":"Ulan Bator","lat":47.906641,"lng":106.895085,"locInRadar":"<circle cx=\"50\" cy=\"240\" r=\"10\"></circle>"}
```

Geo: lat=60.0&lng=19.0

```
Resul: {"stars":5,"city":"Estocolmo","lat":59.328694,"lng":18.068505,"locInRadar":"<circle cx=\"320\" cy=\"270\" r=\"10\"></circle>"}
```

Geo: lat=64.15&lng=-21.95

```
Resul: {"stars":6,"city":"Reikiavik","lat":64.145144,"lng":-21.942496,"locInRadar":"<circle cx=\"80\" cy=\"80\" r=\"10\"></circle>"}
```

Geo: lat=47.183333000000005&lng=31.033333000000002

```
Resul: {"stars":7,"city":"Odessa","lat":46.482921,"lng":30.722892,"locInRadar":"<circle cx=\"30\" cy=\"280\" r=\"10\"></circle>"}
```

No tengo muy claro lo que buscamos con el resultado de las bolas, parece que buscamos un nombre. Podemos usar las letras mayúsculas iniciales de cada bola para formar un nombre, en este caso, con dichas letras y dicho orden quedaría: DR_UERO siendo el “_” la letra que falta para la bola 3, que no apareció.

Buscamos en esta Web todos los nombres de personajes de la serie Bola de Dragón y encontramos este que se parece:

86



Dr. Gero

El **Dr.** Gero o Androide #20 (人造人間20号, Jinzō'ningen Nijūgō?), llamado **Dr.** Maki Gero en el doblaje latinoamericano, es un personaje de ficción de la serie manga y anime, Dragon Ball. Se trata del creador de todos los androides. Este semi-androide era originariamente el prestigioso y cruel... Ver mas

Vótalo:

1

2

3

4

5

PTO

PTO

PTO

PTO

PTO

Ha recibido 11379 puntos

El nombre no es exactamente igual, aunque se comentó en el grupo de Telegram que, a propósito, no estaba correctamente escrito, así que vamos a comprobar contra el servidor de nombres si puede ser DRGUERO, que se parece y cuadra con las iniciales de las bolas. Hacemos un “nc” contra el servidor de nombres, y nos sale la flag:

```
nacho@kali:~/UAM/201907-BolaDrac1$ nc 34.253.120.147 9999
DRGUERO
UAM{2f3c45a7fdd272de9f43836e5ca2f39c}nacho@kali:~/UAM/20190
```

Por tanto, la flag es *UAM{2f3c45a7fdd272de9f43836e5ca2f39c}*

José Ignacio de Miguel González:

User UAM: nachinho3

Telegram: @jignaciodemiguel