DNS

Al analizar el archivo DNS.log, se observa que, además de peticiones DNS comunes a dominios legítimos, existen múltiples solicitudes dirigidas a subdominios del dominio getdata.io. Estos subdominios contienen cadenas aparentemente aleatorias que no corresponden a nombres habituales, lo que levanta sospechas de una posible exfiltración de datos mediante DNS.

```
[28/02/2021] [REQUEST] www.telefonica.es - [RESPONSE] 194.224.110.41
[28/02/2021] [REQUEST] www.telefonica.es - [RESPONSE] 194.224.110.41
[28/02/2021] [REQUEST] www.telefonica.es - [RESPONSE] 194.224.110.41
[28/02/2021] [REQUEST] igoXkiaQqbr0LzNx/nt43mqtB.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] www.telefonica.com - [RESPONSE] 212.170.36.79
[28/02/2021] [REQUEST] www.ihacklabs.com - [RESPONSE] 172.67.97.11
[28/02/2021] [REQUEST] www.telefonica.es - [RESPONSE] 194.224.110.41
[28/02/2021] [REQUEST] www.telefonica.com - [RESPONSE] 212.170.36.79
[28/02/2021] [REQUEST] www.ihacklabs.com - [RESPONSE] 172.67.97.11
[28/02/2021] [REQUEST] www.google.com - [RESPONSE] 142.250.185.4
[28/02/2021] [REQUEST] www.telefonica.com - [RESPONSE] 212.170.36.79
[28/02/2021] [REQUEST] www.telefonica.es - [RESPONSE] 194.224.110.41
[28/02/2021] [REQUEST] cyberacademy.ihacklabs.com - [RESPONSE] 104.25.145.11
[28/02/2021] [REQUEST] cyberacademy.ihacklabs.com - [RESPONSE] 104.25.145.11
[28/02/2021] [REQUEST] www.ihacklabs.com - [RESPONSE] 172.67.97.11
[28/02/2021] [REQUEST] cyberacademy.ihacklabs.com - [RESPONSE] 104.25.145.11
[28/02/2021] [REQUEST] www.telefonica.com - [RESPONSE] 212.170.36.79
[28/02/2021] [REQUEST] www.ihacklabs.com - [RESPONSE] 172.67.97.11
[28/02/2021] [REQUEST] BAE0AAACjAQAAAAA=.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] www.telefonica.es - [RESPONSE] 194.224.110.41
```

```
import re

ciphers = []

with open("log.txt", "r", encoding="utf-8", errors="ignore") as f:
for line in f:
m = re.search(r"PASSWORD:\s*([A-Za-z0-9+/=]+)", line)
if m:
ciphers.append(m.group(1))

ciphers = sorted(set(ciphers), key=len, reverse=True)

print("[+] Ciphertexts encontrados:", len(ciphers))
print("[+] Usando el más largo:")
print(ciphers[0])
import re
```

```
with open("DNS.log", "r") as f:
for line in f:
if "getdata.io" in line:
print(line.strip())
```

```
└─$ python3 extraer.py
[28/02/2021] [REQUEST] igoXkiaQqbr0LzNx/nt43mqtB.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] BAE0AAACjAQAAAAA=.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] VffDRr9vnY1hze6vF2AitrRvz.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] vClLfpeyyrNIIbVTZXBOH/VnB.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] N4pAFP+cy3g77L057iJevQM/E.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] AAAAAAABAAAApIEAAAAAbG9nL.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] cqIKxSmattw86qVcklEhrjkpR.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] nR4dFVUBQAD8d04YHV4CwABBA.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] 07vBTf2twkLlZWza+/iVIJ4Q1.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] WrGYP/sN2ECPxxk5AxzvoPcdW.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] aKh/AggSfmZSXgEDZYZCG1z2O.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] OF8puXvwFQSwECHgMUAAAACAD.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] it4JMHRBIeEwtsEgJO7hADUe5.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] UEsDBBQAAAAIAO9kWlK3tRNAY.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] eZKWUJUWB1k26lq6X+tlnSS78.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] ihd92hr27mxTy/xVjhaPTXpob.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] tmGbeh3opqL7GeC5CEzSfa8Vg.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] vZFpSt7UTQGIBAAB3BwAABwAY.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] uNrinJscJEwSxKl5S6MRIKVYx.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] /LJXIN2zP0aqsF5GodCmP6Fw5.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] AAAAAEAAAAAFBLBQYAAAAAAQA.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] J5cHqb60oPcog5GkrxCYTej8A.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] AAAAABAAAAAC9k0tPwjAAx+98.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] /+X6OSIkyGLGMgjtxwBjilEgy.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] Ab0XR/il0ZmBtmFV2wsnl0rJt.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] gEAAHcHAAAHABwAbG9nLnR4dF.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] k5HA6NeoQPRCqvO4rO80hRPC7.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] VUCQAD8d04YPndOGB1eAsAAQQ.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] WzE1LDI3LDIxLDE3LDgsMjQsM.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] TIsMjUsMjAsMTgsOSwyMiw0LD.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] AsMTYsNSwxMSwyMywxMCwxNCw.getdata.io - [RESPONSE] 127.0.0.1
[28/02/2021] [REQUEST] yNiwxOSwzLDYsNywxLDEzLDJd.getdata.io - [RESPONSE] 127.0.0.1
```

Para centrarnos únicamente en la información relevante, se filtran todas las líneas del log que contienen peticiones hacia getdata.io y se extrae únicamente el subdominio. Tras este proceso se obtienen 32 fragmentos (chunks), todos con caracteres compatibles con Base64

```
import re

chunks = []

rx = re.compile(r"\[REQUEST\]\s+([A-Za-z0-9+/=]+)\.getdata\.io")
```

```
with open("DNS.log") as f:
for line in f:
m = rx.search(line)
if m:
chunks.append(m.group(1))

print("Total chunks:", len(chunks))
print("Primeros 5:", chunks[:5])
print("Últimos 5:", chunks[-5:])
```

Para centrarnos únicamente en la información relevante, se filtran todas las líneas del log que contienen peticiones hacia getdata.io y se extrae únicamente el subdominio. Tras este proceso se obtienen 32 fragmentos (chunks), todos con caracteres compatibles con Base64.

```
└─$ python3 subdominios.py
Total chunks: 32
Primeros 5: ['igoXkiaQqbr0LzNx/nt43mqtB', 'BAE0AAACjAQAAAAA=', 'VffDRr9vnY1hze6vF2AitrRvz', 'vClLfpeyyrNIIbVTZXBOH/
VnB', 'N4pAFP+cy3g77L057iJevQM/E']
Últimos 5: ['VUCQAD8d04YPndOGB1eAsAAQQ', 'WzE1LDI3LDIxLDE3LDgsMjQsM', 'TIsMjUsMjAsMTgsOSwyMiw0LD', 'AsMTYsNSwxMSwyM
ywxMCwxNCw', 'yNiwxOSwzLDYsNywxLDEzLDJd']
```

```
import re
import base64

# 1 Leer el log y extraer los subdominios
chunks = []

rx = re.compile(r"\[REQUEST\]\s+([A-Za-z0-9+/=]+)\.getdata\.io")

with open("DNS.log", "r", encoding="utf-8", errors="ignore") as f:
for line in f:
m = rx.search(line)
if m:
chunks.append(m.group(1))

print("[+] Total chunks encontrados:", len(chunks))

# 2 Tomar los últimos 4 (bloque de control)
control = "".join(chunks[-4:])

# Arreglar padding base64 si falta
```

```
control += "=" * ((-len(control)) % 4)

# 3 Decodificar
decoded = base64.b64decode(control)

print("[+] Control decodificado:")
print(decoded.decode(errors="replace"))
```

Al inspeccionar los fragmentos obtenidos, se observa que los últimos cuatro chunks tienen un comportamiento distinto al resto. Al concatenarlos y decodificarlos en Base64, el resultado es un arreglo de números enteros:

```
└─$ python3 decodificar.py
[+] Total chunks encontrados: 32
[+] Control decodificado:
[15,27,21,17,8,24,12,25,20,18,9,22,4,0,16,5,11,23,10,14,26,19,3,6,7,1,13,2]
```

```
import re
import base64

# 1 Extraer chunks
chunks = []
rx = re.compile(r"\[REQUEST\]\s+([A-Za-z0-9+/=]+)\.getdata\.io")

with open("DNS.log", "r", encoding="utf-8", errors="ignore") as f:
for line in f:
m = rx.search(line)
if m:
chunks.append(m.group(1))

print("[+] Total chunks:", len(chunks))

# 2 Separar
data_chunks = chunks[:-4]
control_chunks = chunks[-4:]

print("[+] Data chunks:", len(data_chunks))
print("[+] Control chunks:", len(control_chunks))

# 3 Decodificar vector de orden
control = "".join(control_chunks)
control += "=" * ((-len(control)) % 4)
```

```
decoded = base64.b64decode(control).decode(errors="replace")
print("[+] Vector crudo:", decoded)

# Extraer números
order = list(map(int, re.findall(r"\d+", decoded)))
print("[+] Vector de orden:", order)
print("[+] Longitud vector:", len(order))

# 4 Reordenar los datos según el vector
reordered = []

for i in range(len(order)):
idx = order.index(i)
reordered.append(data_chunks[idx])

# Unir todo
final_b64 = "".join(reordered)

print("[+] Base64 reconstruido (inicio):", final_b64[:60])
print("[+] Longitud base64 final:", len(final_b64))

# 5 Decodificar Base64 y guardar ZIP
final_b64 += "=" * ((-len(final_b64)) % 4)

zip_bytes = base64.b64decode(final_b64)

with open("exfiltrated.zip", "wb") as f:
f.write(zip_bytes)

print("[+] ZIP escrito como exfiltrated.zip")
print("[+] Magic bytes:", zip_bytes[:2])
```

Aplicando el vector de permutación sobre los chunks de datos, se reconstruye una única cadena Base64 de gran tamaño. Al decodificar esta cadena, se obtiene un archivo binario cuyos primeros bytes corresponden a la firma PK, lo que identifica claramente un archivo ZIP válido.

```
└─$ python3 reordenar.py
[+] Total chunks: 32
[+] Data chunks: 28
[+] Control chunks: 4
[+] Vector crudo: [15,27,21,17,8,24,12,25,20,18,9,22,4,0,16,5,11,23,10,14,26,19,3,6,7,1,13,2]
[+] Vector de orden: [15, 27, 21, 17, 8, 24, 12, 25, 20, 18, 9, 22, 4, 0, 16, 5, 11, 23, 10, 14, 26, 19, 3, 6, 7, 1
, 13, 2]
[+] Longitud vector: 28
[+] Base64 reconstruido (inicio): UEsDBBQAAAAIAO9kWlK3tRNAYgEAAHcHAAAHABwAbG9nLnR4dFVUCQAD8d04
[+] Longitud base64 final: 692
[+] ZIP escrito como exfiltrated.zip
[+] Magic bytes: b'PK'
```

```python
import re

ciphers = []

with open("log.txt", "r", encoding="utf-8", errors="ignore") as f:
for line in f:
m = re.search(r"PASSWORD:\s*([A-Za-z0-9+/=]+)", line)
if m:
ciphers.append(m.group(1))

ciphers = sorted(set(ciphers), key=len, reverse=True)

print("[+] Ciphertexts encontrados:", len(ciphers))
print("[+] Usando el más largo:")
print(ciphers[0])
import re

with open("DNS.log", "r") as f:
for line in f:
if "getdata.io" in line:
print(line.strip())
```

Al extraer el contenido del ZIP, se encuentra un único archivo llamado log.txt. Este archivo contiene registros de autenticación en los que aparecen usuarios y valores asociados a PASSWORD. Dichos valores están codificados en Base64, pero al decodificarlos no se obtiene texto legible, lo que indica que los datos se encuentran cifrados y no solo codificados.

```
└─$ python3 extraer.py
[+] Ciphertexts encontrados: 16
[+] Usando el más largo:
CU01h+7UmzsFXA+LAScdtQRrcxssJhs=
```

```
import base64

def rc4(key, data):
S = list(range(256))
j = 0

for i in range(256):
j = (j + S[i] + key[i % len(key)]) % 256
S[i], S[j] = S[j], S[i]

i = j = 0
out = bytearray()

for byte in data:
i = (i + 1) % 256
j = (j + S[i]) % 256
S[i], S[j] = S[j], S[i]
out.append(byte ^ S[(S[i] + S[j]) % 256])

return bytes(out)

cipher_b64 = "CU01h+7UmzsFXA+LAScdtQRrcxssJhs="
cipher = base64.b64decode(cipher_b64)

with open("/usr/share/wordlists/rockyou.txt", "r", errors="ignore") as f:
for n, word in enumerate(f, 1):
key = word.strip().encode()
if not key:
continue

pt = rc4(key, cipher)

if b"flag{" in pt:
print(" 🔥 FLAG ENCONTRADA 🔥 ")
print("Clave:", key.decode())
print("Texto:", pt.decode())
break

if n % 200000 == 0:
print("Probadas:", n)
```

El análisis del patrón de cifrado, junto con la información del reto, permite determinar que las contraseñas están cifradas mediante el algoritmo RC4, un cifrador de flujo común en este tipo de ejercicios. Además, se asume que la clave utilizada es una contraseña débil, por lo que se plantea un ataque de fuerza bruta utilizando la wordlist rockyou.txt