

# Strike Back - CobalStrike C2 reversing

WriteUp realizado por Jorge Espinosa

## Análisis forense completo: Descifrando el canal C2 de Cobalt Strike De evidencias en bruto a la reconstrucción total del ataque

---

### 1. Contexto inicial y punto de partida

**Objetivo:** Partir desde evidencias en bruto (PCAP + memoria) y reconstruir completamente el ataque sin conocimiento previo del tipo de malware.

**Evidencias disponibles:**

- `capture.pcap` - Tráfico de red capturado durante el incidente
- `freesteam.dmp` - Volcado de memoria de un proceso sospechoso

**Estado mental inicial:** No sabemos qué tipo de malware enfrentamos. Nuestra misión es desarrollar hipótesis basadas en evidencia técnica y confirmarlas mediante análisis forense riguroso.

---

### 2. Fase de reconocimiento: Análisis superficial del PCAP

#### 2.1. Primera inspección del tráfico de red

**Herramienta utilizada:** Wireshark

**Objetivo:** Identificar patrones anómalos en el tráfico

**Proceso mental:**

- Busco protocolos dominantes (HTTP, HTTPS, DNS)
- Analizo conversaciones TCP para detectar beaconing
- Examino URIs y payloads para identificar tráfico no legítimo

**Observaciones clave:**

1. **Múltiples sesiones HTTP** con URIs sospechosas (`/view`, `/submit.php`)
2. **Payloads binarios** en lugar de contenido web típico
3. **Periodicidad** en las comunicaciones ( posible beaconing )
4. **Tamaños específicos** de datos: 16, 32, 48, 64 bytes...

**Primera hipótesis:** El patrón de tamaños múltiplos de 16 sugiere **cifrado por bloques** (probablemente AES-128 con bloques de 16 bytes).

#### 2.2. Análisis detallado de las peticiones HTTP

**Comando ejecutado:**

```
# Filtrar tráfico HTTP en Wireshark
# Aplicar filtro: http
```

**Proceso mental:**

- Los URIs (`/freesteam.exe`, `/submit.php?id=1909272864`) no corresponden a navegación legítima
- El contenido de respuesta es binario, no HTML/CSS/JS típico
- El User-Agent y headers parecen legítimos ( posible evasión )

**Segunda hipótesis:** Estamos ante un **canal C2 encapsulado en HTTP** que utiliza perfiles maleables para evadir detección.

No.	Time	Source	Destination	Protocol	Length	Info
7	0.022...	192.168.1.7	192.168.1.9	HTTP	507	GET /freesteam.exe HTTP/1.1
23	29.57...	192.168.1.7	192.168.1.9	HTTP	249	GET /iVd9 HTTP/1.1
56	29.58...	192.168.1.7	192.168.1.9	HTTP	439	GET /match HTTP/1.1
66	89.63...	192.168.1.7	192.168.1.9	HTTP	439	GET /match HTTP/1.1
86	149.6...	192.168.1.7	192.168.1.9	HTTP	439	GET /match HTTP/1.1
119	211.3...	192.168.1.7	192.168.1.9	HTTP	439	GET /match HTTP/1.1
153	272.5...	192.168.1.7	192.168.1.9	HTTP	439	GET /match HTTP/1.1
214	333.7...	192.168.1.7	192.168.1.9	HTTP	439	GET /match HTTP/1.1
234	393.7...	192.168.1.7	192.168.1.9	HTTP	439	GET /match HTTP/1.1
12	0.023...	192.168.1.9	192.168.1.7	HTTP	1250	HTTP/1.0 200 OK (application/x-msdos-program)
47	29.57...	192.168.1.9	192.168.1.7	HTTP	29795	HTTP/1.1 200 OK
58	29.59...	192.168.1.9	192.168.1.7	HTTP	169	HTTP/1.1 200 OK
69	89.63...	192.168.1.9	192.168.1.7	HTTP	102	HTTP/1.1 200 OK
78	89.64...	192.168.1.9	192.168.1.7	HTTP	154	HTTP/1.1 200 OK
101	149.6...	192.168.1.9	192.168.1.7	HTTP	10322	HTTP/1.1 200 OK
111	151.3...	192.168.1.9	192.168.1.7	HTTP	154	HTTP/1.1 200 OK
135	211.3...	192.168.1.9	192.168.1.7	HTTP	5202	HTTP/1.1 200 OK
145	212.5...	192.168.1.9	192.168.1.7	HTTP	154	HTTP/1.1 200 OK
190	272.5...	192.168.1.9	192.168.1.7	HTTP	28690	HTTP/1.1 200 OK
206	273.7...	192.168.1.9	192.168.1.7	HTTP	154	HTTP/1.1 200 OK
217	333.7...	192.168.1.9	192.168.1.7	HTTP	134	HTTP/1.1 200 OK
226	333.7...	192.168.1.9	192.168.1.7	HTTP	154	HTTP/1.1 200 OK
237	393.7...	192.168.1.9	192.168.1.7	HTTP	134	HTTP/1.1 200 OK
256	393.8...	192.168.1.9	192.168.1.7	HTTP	154	HTTP/1.1 200 OK
76	89.64...	192.168.1.7	192.168.1.9	HTTP	405	POST /submit.php?id=1909272864 HTTP/1.1
109	151.3...	192.168.1.7	192.168.1.9	HTTP	1062	POST /submit.php?id=1909272864 HTTP/1.1
143	212.5...	192.168.1.7	192.168.1.9	HTTP	886	POST /submit.php?id=1909272864 HTTP/1.1
204	273.7...	192.168.1.7	192.168.1.9	HTTP	4570	POST /submit.php?id=1909272864 HTTP/1.1
224	333.7...	192.168.1.7	192.168.1.9	HTTP	662	POST /submit.php?id=1909272864 HTTP/1.1
254	393.8...	192.168.1.7	192.168.1.9	HTTP	2766	POST /submit.php?id=1909272864 HTTP/1.1

### 3. Descarga y análisis inicial del ejecutable

### 3.1. Extracción del malware desde el PCAP

**Herramienta:** Wireshark (File > Export Objects > HTTP)

**Archivo extraído:** freesteam.exe

## Comando de verificación:

```
file freesteam.exe
# Salida: PE32 executable (GUI) Intel 80386, for MS Windows
```

tcp.stream eq 0

No.	Time	Source	Destination	Protocol	Length	Info
5	0.000...	192.168.1.7	192.168.1.9	TCP	60	49753 → 8080 [ACK] Seq=1 Ack=1 Win=2102272 Len=0
14	0.023...	192.168.1.7	192.168.1.9	TCP	60	49753 → 8080 [ACK] Seq=454 Ack=14543 Win=2102272 Len=0
13	0.023...	192.168.1.7	192.168.1.9	TCP	60	49753 → 8080 [ACK] Seq=454 Ack=6046 Win=2102272 Len=0
15	0.028...	192.168.1.7	192.168.1.9	TCP	60	49753 → 8080 [FIN, ACK] Seq=454 Ack=14543 Win=2102272 Len=0
1	0.000...	192.168.1.7	192.168.1.9	TCP	66	49753 → 8080 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
8	0.022...	192.168.1.9	192.168.1.7	TCP	54	8080 → 49753 [ACK] Seq=1 Ack=454 Win=64128 Len=0
16	0.028...	192.168.1.9	192.168.1.7	TCP	54	8080 → 49753 [ACK] Seq=14543 Ack=455 Win=64128 Len=0
9	0.023...	192.168.1.9	192.168.1.7	TCP	259	8080 → 49753 [PSH, ACK] Seq=1 Ack=454 Win=64128 Len=205 [TCP PDU reassembled in 12]
10	0.023...	192.168.1.9	192.168.1.7	TCP	7354	8080 → 49753 [PSH, ACK] Seq=206 Ack=454 Win=64128 Len=7300 [TCP PDU reassembled in 12]
11	0.023...	192.168.1.9	192.168.1.7	TCP	5894	8080 → 49753 [PSH, ACK] Seq=7506 Ack=454 Win=64128 Len=5840 [TCP PDU reassembled in 12]
2	0.000...	192.168.1.9	192.168.1.7	TCP	66	8080 → 49753 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460 SACK_PERM WS=128
7	0.022...	192.168.1.7	192.168.1.9	HTTP	507	GET /freesteam.exe HTTP/1.1
12	0.023...	192.168.1.9	192.168.1.7	HTTP	1250	HTTP/1.0 200 OK (application/x-msdos-program)

Wireshark · Seguir secuencia HTTP (tcp.stream eq 0) · capture.pcap

1 client pkt(s), 1 server pkt(s), 1 turn(s).

Conversación completa (14 kB) Mostrar como Raw No delta times Secuencia 0

Buscar:  Mayúsculas y minúsculas Buscar siguiente

Puedes descargar o la conversación completa o solo la descarga desde el server. Resulta que ambos tiene reportes. You can download either the entire conversation or just the download from the server. It turns out that both have reports.

Filtrar secuencia Imprimir Guardar como... Atrás Cerrar Ayuda

(4056 bits), Dst: PCSSystemtec\_59:38:62 (08:00:27:59:38:62)

0000	08	00	27	59	38	62	08
0010	01	ed	d6	48	40	00	80
0020	01	09	c2	59	1f	90	h1

## ⌚ Amenazas actuales

Se han detectado amenazas. Inicia las acciones recomendadas.

Behavior:Win32/CobaltStrike.gen!A  
07/10/2025 6:14 (Activo) Grave

Backdoor:Win64/CobaltStrike!rfn  
07/10/2025 6:14 (Activo) Grave

[Iniciar acciones](#)

[Opciones de examen](#)

[Amenazas permitidas](#)

[Historial de protección](#)

## 3.2. Análisis de reputación en VirusTotal

Hash SHA256: [d3e38d55a3161cc27813d4c1d046479eb1fb61fea55d9a23a302f0b9875b7340](#)

Resultado: 21/62 motores lo detectan como malicioso

Etiquetas principales:

- trojan.cobaltstrike/cobalt
- Win32:MalwareX-gen [Hack]
- Trojan.Win32.CobaltStrike

Proceso mental: La detección específica de "CobaltStrike" confirma nuestra hipótesis. Ahora sabemos que enfrentamos un Beacon de Cobalt Strike, pero necesitamos descifrar su canal C2.

23 / 62

Community Score

23/62 security vendors flagged this file as malicious

d3e38d55a3161cc27813d4c1d046479eb1fb61fea55d9a23a302f0b9875b7340  
freesteam.exe

Size 14.64 KB | Last Analysis Date 14 hours ago

DETECTION DETAILS COMMUNITY 1

Crowdsourced YARA rules ⓘ

- ⚠️ Matches rule Windows\_API\_Function from ruleset Windows\_API\_Function at <https://github.com/inQuest/yara-rules-vt> by [inQuest Labs](#)
  - ↳ This signature detects the presence of a number of Windows API functionality often seen within embedded executables. When this signature alerts on an executable, it is not an indication of malicious behavior. However, if seen firing in other file types, deeper investigation may be warranted. - 14 hours ago
- ⚠️ Matches rule Windows\_Trojan\_CobaltStrike\_1787eef5 from ruleset Windows\_Trojan\_CobaltStrike at <https://github.com/elastic/protections-artifacts> by [Elastic Security](#)
  - ↳ CS shellcode variants - 14 hours ago
- ⚠️ Matches rule Windows\_Trojan\_CobaltStrike\_7f8da98a from ruleset Windows\_Trojan\_CobaltStrike at <https://github.com/elastic/protections-artifacts> by [Elastic Security](#)
- ⚠️ Matches rule CobaltStrike\_Resources\_Artifact32\_v3\_14\_to\_v4\_x from ruleset CobaltStrike\_\_Resources\_Artifact32\_and\_Resources\_Dropper\_v1\_45\_to\_v4\_x at <https://github.com/chronicle/GCTI> by [gssincla@google.com](mailto:gssincla@google.com)
  - ↳ Cobalt Strike's resources/artifact32.dll, .exe, big.exe, big.dll, bigsvc.exe} signature for versions 3.14 to 4.x and resources/artifact32svc.exe for 3.14 to 4.x and resources/artifact32uac.dll for v3.14 and v4.0 - 14 hours ago

Popular threat label ⓘ trojan.cobaltstrike/cobalt

Threat categories trojan

Family labels cobaltstrike cobalt hack

Security vendors' analysis ⓘ

Vendor	Signature	Category	Label	Action
AliCloud	Backdoor:Win/CobaltStrike.gyf	ALYac	Trojan.GenericKDZ.80482	Do you want to automate checks?
Arcabit	Trojan.Generic.D13A62	Avast	Win32:MalwareX-gen [Hack]	
AVG	Win32:MalwareX-gen [Hack]	BitDefender	Trojan.GenericKDZ.80482	
ClamAV	Win.Trojan.CobaltStrike-7899872-1	CTX	Unknown.trojan.generickdz	
Emsisoft	Trojan.GenericKDZ.80482 (B)	eScan	Trojan.GenericKDZ.80482	

## 4. Investigación: Metodologías para descifrar Cobalt Strike

### 4.1. Búsqueda de información técnica

**Fuentes consultadas:** NVISO Labs, SANS Internet Storm Center

**Hallazgo clave:** Existen herramientas específicas para descifrar tráfico de Beacon:

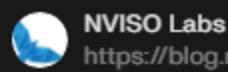
- `cs-extract-key.py` - Extrae claves desde memoria
- `cs-parse-http-traffic.py` - Descifra y parsea el tráfico

Como sabemos que CobalStrike 4.X utiliza cifrados AES (puede ser 128-256bits), NVISO sugiere extraer los blobs(Objeto Binario Grande) raw data y filtrar por aquellos cuyo tamaño sea múltiplo de 16 (indicio de ser AES/CBC - Cipher Block Chaining). Para ello, utilizaremos la herramienta `cs-parse-http-traffic.py`.

**Proceso mental:** Necesito las claves criptográficas (AES + HMAC) que usa el Beacon para cifrar su canal C2. Estas pueden extraerse del volcado de memoria o derivarse del tráfico.

# cobalt strike procces memory decrypt

Asistente Imágenes Videos Fuentes · 20 Pasos



**NVISO Labs**  
<https://blog.nviso.eu/2021/11/03/cobalt-strike-using-process-memory-to-decrypt-traffic-part-3>

## Cobalt Strike: Using Process Memory To Decrypt Traffic

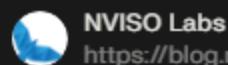
3 nov 2021 — In this blog post, we will explain how to decrypt Cobalt Strike traffic if you don't know the private RSA key but do have a process memory dump.



**SANS Internet Storm Center**  
<https://isc.sans.edu/diary/28006>

## Decrypting Cobalt Strike Traffic With Keys Extracted From ...

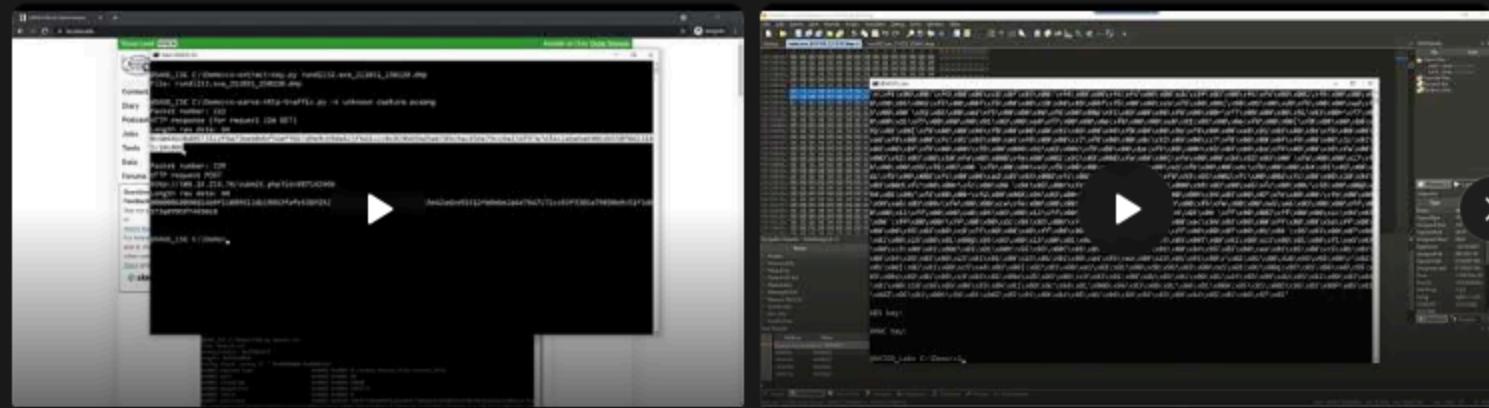
6 nov 2021 — In this diary entry, I will show how to decrypt Cobalt Strike network traffic with AES keys extracted from the beacon's process memory.



**NVISO Labs**  
<https://blog.nviso.eu/series/cobalt-strike-decrypting-traffic>

## Series: Cobalt Strike: Decrypting Traffic

Cobalt Strike: Using Process Memory To Decrypt Traffic – Part 3. We decrypt Cobalt Strike traffic with cryptographic keys extracted from process memory.



To decrypt Cobalt Strike process memory, the keys (AES and HMAC) used to encrypt its traffic can be extracted from the writable process memory of a running beacon. This method is useful

## 4.2. Descarga de herramientas NVISO

Repositorio: <https://github.com/NVISO-BE/cobalt-strike-tools>

Archivos descargados:

- `cs-extract-key.py`
- `cs-parse-http-traffic.py`

master	1 Branch	0 Tags	Go to file	t	Add file	Code
DidierStevens 20250612a			439d216 · 4 months ago		76 Commits	
OLETemplate.bt		20220327a			3 years ago	
README.md		Initial commit			9 years ago	
cs-analyze-processdump.py		20211103a			4 years ago	
cs-dns-stager.py		20210530a			4 years ago	
cs-extract-key.py		20211102a			4 years ago	
cs-parse-http-traffic.py		20211106a			4 years ago	
csv-cut.py		20170409a			8 years ago	
csv-lookup.py		20161219a			9 years ago	
csv-transform.py		20210811a			4 years ago	
decode-search.py		20170313a			8 years ago	
decode-search.txt		20170313a			8 years ago	
diffdump.py		20171101a			8 years ago	
dump-registry-hive.py		20220718a			3 years ago	
file-magic.def		20170129a			9 years ago	
file-magic.py		20170129a			9 years ago	
format-bytes.py		20171104a			8 years ago	

## 5. Identificación de Cobalt Strike 4.x y método alternativo requerido

### 5.1. Intento inicial con extracción desde memoria (falla esperada)

Comando ejecutado:

```
python3 cs-extract-key.py freesteam.dmp
```

```
> python3 cs-extract-key.py freesteam.dmp
File: freesteam.dmp
```

Resultado: X Sin output de claves

- El script no encuentra el offset tradicional 0xBEEF típico de Beacon 3.x
- Esto confirma que estamos ante **Cobalt Strike 4.x**, que almacena las claves de manera diferente en memoria

Proceso mental crítico:

Al no obtener claves directamente del volcado, debo utilizar el **método alternativo**: derivar las claves criptográficas desde los datos cifrados del propio PCAP, usando un bloque como "oráculo" de descifrado.

## 6. Derivación de claves desde el PCAP

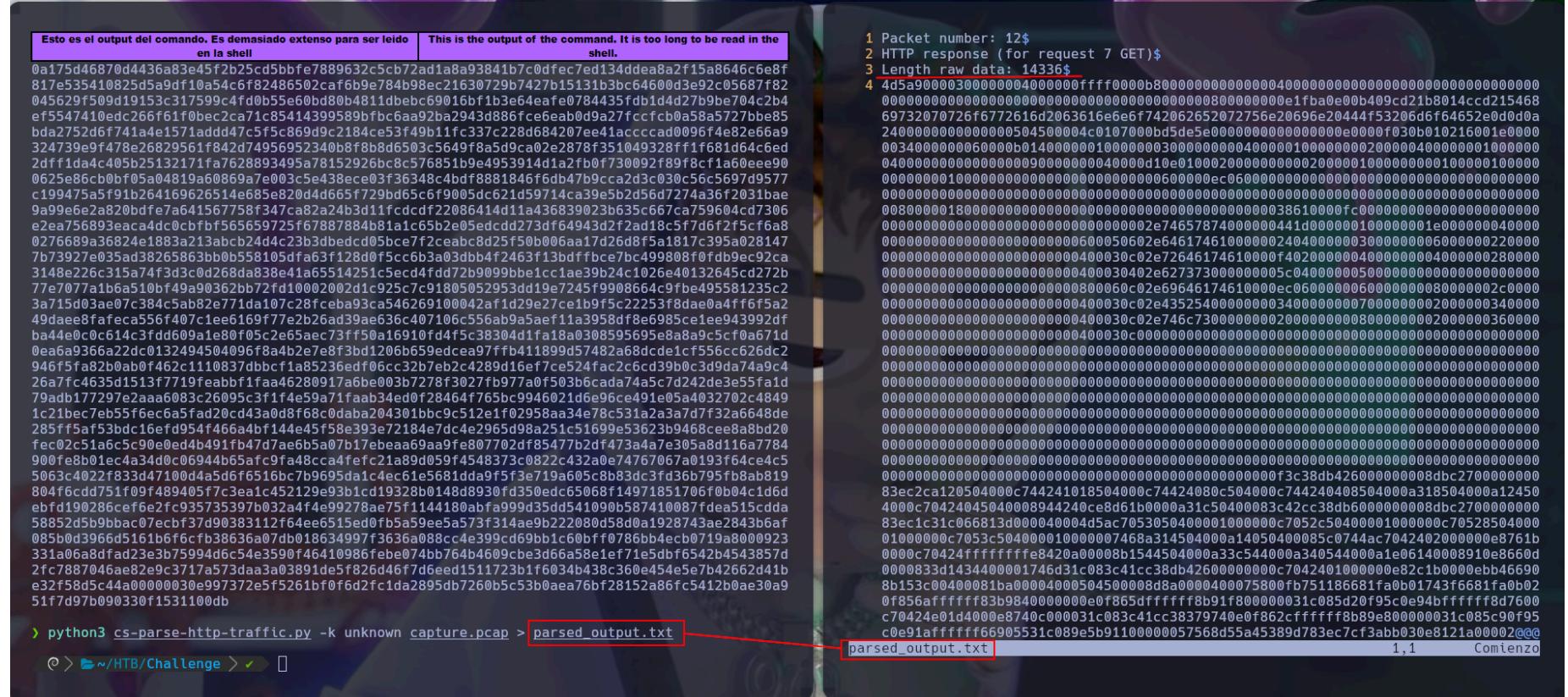
### 6.1. Exploración inicial del tráfico cifrado

## Comando base para inspección:

```
python3 cs-parse-http-traffic.py -k unknown capture.pcap
```

## Parámetros explicados:

- `cs-parse-http-traffic.py` : Parser especializado de NVISO para tráfico de Beacon
- `-k unknown` : Indica que no conocemos las claves (modo de exploración)
- `capture.pcap` : Nuestro archivo de tráfico capturado



```
Este es el output del comando. Es demasiado extenso para ser leido en la shell. This is the output of the command. It is too long to be read in the shell.  
0a175d46870d4436a83e45f2b25cd5bbfe7889632c5cb72ad1a8a93841b7c0defc7ed134ddeaa8a2f15a8646c6e8f  
817e535410825d5a9df10a5a4c6f82486502cafb9e784b98ec21630729b7427b15131b3bc64600d3e92c05687fb2  
045629f509d19153c317599c4fd0b55e60bd80b4811d0bec69016bf1b3e64eafe0784435fdb1d4d7b9be704c2b4  
ef5547410edc256f6f0bec2ca71c85414399589fbcb6aa92ba2943d886fce6ab09a27f7fcfb0a58a5727bbe85  
bda2752d6f741a4e1571add47c5f5c869d9c2184ce53f49b11fc337c228d684207ee4iaccccad0096f4e82e66a9  
324739e9f478e26829561f847d49569523408f88d6503c5649f8a5d9ca02e2878f351049328ff1f681d64c6ed  
2dff1da4c405b25132171fa7628893495a78152926bc8c5768519e94953914da2fb0f730092789f8cf1a60eee90  
0625e86cb0bf05a04819a60869a7e003c5e438ece03f36348c4bdf8881846f6db47b9cca2d3c030c56c5697d9577  
c199475a5f91b264169626514e685e820d4665f729b65c6f9005dc621d59714ca39e5b2d56d7274a36f2031bae  
9a99e6e2a820bdfc7a641567758f347ca82a24b3d11fcdd2208641d11a436839023b635c667ca759604cd7306  
e2ea756893eaca4dc0cbfbf56569725f67887884b81a1c65b2e05ecd273df64943d2f2ad18c5f7d6f25cf6a8  
ba276689a36824e1883a213abcb24d4c23b3dbcd05bce7f2ceabc8d25f50b006aa17d26d8f5a1817c395a028147  
7b73927e035ad38265863bb0b558105dfa63f128d0f5c6b3a03dbbf4263f13bdfbce7bc499808f0fb9ec92ca  
3148e226c315a71bf3d3c0d268da838e41a5514251c5ec4f4dd72b9099bce1ae39b24c1026e40132645cd272b  
77e7077a1b6a510849a0362bb72fd10002002d1c925c7c91805052935d1d19e7245f9908664c9fbe495581235c2  
3a715d03ae07c384c5ab82e771da07c28fcea93ca546269100042af1d29e27ce1b9f5c2253f8dae0a4ff6f5a2  
49da4687afecfa55674071ee6169f77e2b26d39ae636c407106c5659ab9a5eef1a3958df8e6985ce1ee943992df  
ba44e0c0c614c36d09a1e80f05c2e65aec73f50a16910f4df5c38304d1fa18a0308595698e8a9c5cf0a671d  
0ea6a9366a22dc0132494504096f84b2e7e8f3bd1206b659edcea97482a68dcd1c7f556cc626dc2  
946f51a82b0ab0f462c110837dbbcf1a85236edf06cc32b7eb24289d16ef7c524fac2c6cd9b8c3d9d47a94  
26a7f4635d15137719feabbf1aa46820917a6be003b7278f3027bf977a0f503b6cada7a5c7d242de355fa1d  
79da0177297e2aaa083c26095c3f14e59a71faab04ed0f28464f75f68f14971851706f0b04c1d6d  
1c21bec7eb55f6ec6a5fd20cd43a0d8f68c0daba204301bbc9c512ef02958aa34e78c531a23a7d7f732a6648de  
285ff5af53bdc16efd954f466a4bf144e45f58e393e72184e7dc4e2965d98a251c51699e53623b9468ce8a8bd20  
fec02c51a6c5c90e0d4b491f747d7ae6b5a07b17beaa9aa9fe807702df85477b2df473a47e305a8d116a7784  
900fe8b01ec4a34d0c069465a65af9fa48cc4f4efc21a89d059f4548373c082342a074767067a0193f64ce4c5  
5063c4022f833d471004a5d6f6516bc7b9695da1c4ec61e5681da0f5f3e719a605c8b83dc3fd36b795fb8ab19  
804f6cd751f09f489405f7c3a1c452129e93b0148d8930fd350edc65068f14971851706f0b04c1d6d  
ebfd190286cef6f2c935735397b032a4f4e99278a7f51144180abf999d35dd54109b68f14971851706f0b04c1d6d  
58852d59bbdc07ecbf37d0383112f64ee6515ed0fb5a59ee5a573f314ae9b222080d58d0a1928743ae2843b6af  
085b0d3966d161bcb6fc38636a07db018634997f3636a088cc4e399cd69bb1c60bfff0786bbf4ecb0719a00923  
331a06a8dfad23e3b75994d6c54e3590f46410986febe074bb764b4609cbe3d66a58e1ef71e5dbf6542b453857d  
21c78858d5c44a0000030e997372e5f5261bf0f6d2fc1da2895bd726b05c53b0aea76bf28152a86f5c5412b0ae30a9  
51f7d97b090330f1531100db  
python3 cs-parse-http-traffic.py -k unknown capture.pcap > parsed_output.txt
```

Ejecutamos este comando como fase inicial de reconocimiento para mapear la estructura del tráfico cifrado de Cobalt Strike sin conocer aún las claves de descifrado. El parámetro `-k unknown` le indica al parser que opere en modo "exploración", donde no intenta descifrar el contenido de los blobs, sino que simplemente los identifica, enumera y nos muestra sus características básicas como tamaños, posiciones y estructura. **Esto nos permite obtener un inventario completo de todos los bloques de datos presentes en el PCAP**, catalogando las longitudes de los payloads cifrados (16, 32, 48, 64, 80 bytes, etc.) junto con sus correspondientes datos hexadecimales. **Con esta información, podemos aplicar filtros estratégicos para identificar aquellos bloques que sean múltiplos de 16 bytes y de tamaño reducido, que son los candidatos ideales para usarse como oráculo en la derivación de las claves criptográficas**. Sin esta fase de mapeo inicial, sería imposible localizar eficientemente los bloques específicos que contienen comandos del protocolo Beacon cifrados con AES, ya que estarían "perdidos" entre cientos de otros elementos del tráfico HTTP.

## 6.2. Automatización de la búsqueda de bloques válidos

### Script personalizado creado (check\_parsed.sh):

```
#!/bin/bash

# Almacenamos todas las líneas del archivo en un array para poder acceder a la siguiente línea fácilmente

mapfile -t lines < "$archivo"

num_lines=${#lines[@]}

for (( i=0; i<num_lines; i++ )); do

    line="${lines[i]}"

    if [[ $line =~ data:\ \([0-9]+\)\ ]]; then

        value=${BASH_REMATCH[1]}

        if (( value % 16 == 0 && value <= 200 )); then

            # Mostrar la siguiente línea si existe

            if (( i+1 < num_lines )); then
```

```

echo "${lines[i+1]}"

fi

fi

done

```

#### Explicación del script:

- `mapfile -t lines` : Carga todo el archivo en un array
- `[[ $line =~ data:\ \ ([0-9]+) ]]` : Busca líneas que contengan "data:" seguido de números
- `value % 16 == 0` : Filtra solo múltiplos de 16 bytes
- `value <= 200` : Evita bloques demasiado grandes que compliquen la derivación
- `echo "${lines[i+1]}"` : Muestra la línea siguiente que contiene el hex

#### Observaciones iniciales:

- El script lista múltiples bloques con "Length raw data: N"

### 6.3. Ejecución y filtrado de candidatos

#### Comando de filtrado automático:

```

python3 cs-parse-http-traffic.py -k unknown capture.pcap > parsed_output.txt

./check_parsed.sh

```

#### Resultados obtenidos (múltiplos de 16 ≤ 200 bytes):

```

Length raw data: 48
a4940d6ff0a59421822467d80d1b620bc7ecfa661c452a85c0486b56aa752e908c4aeb3f2f0a64d9c02d7025713867ee

Length raw data: 80
70766735dedd9ba1fff6c36ed8d9eecc2bfed095de2439e347aea08157e01978eb7b7cab86bea3d97e7062c9990bf7407ee88eab9ee0
7bc9761a6371323f285bba8c2d1d4b4a04a09655d675e3ac7e8e

Length raw data: 80
46e10ae1e8c22141cc1242189a988f329cd1e1e4b6176a12b2de7dfb406b511907b78f39b04a139bfa61fed4fdc19dc3800c4e48b9a3
a8e97c30dd532ebdd90b1c0bccb346e85a455af0abdb29da44527

```

#### Proceso mental:

- El bloque de **48 bytes** es el más pequeño y por tanto el candidato ideal
- Los bloques de 80 bytes son más grandes y pueden contener más "ruido"
- Priorizo el de 48 bytes para minimizar complejidad en la derivación (aunque posteriormente también pruebo y es funcional los bloques de 80)

```

> python3 cs-parse-http-traffic.py -k unknown capture.pcap > parsed_output.txt
> ./check_parsred.sh
Líneas donde 'data' es múltiplo de 16 y <= 200, junto a la siguiente línea:
Length raw data: 48
a4940d6ff0a59421822467d80d1b620bc7ecfa661c452a85c0486b56aa752e908c4aeb3f2f0a64d9c02d7025713867ee
Length raw data: 80
7076675dbed9baf4ffcc36ed8d9eccc2bfed095de2439e347aea08157e01978eb7b7cab86bea3d97e7062c9990bf7407ee88eab9ee07bc9
761a6371323f285bba8c2d1d4b4a04a09655d675e3ac7e8e
Length raw data: 80
46e10ae1e8c22141cc1242189a988f329cd1e1e4b6176a12b2de7dfb406b511907b78f39b04a139bfa61fed4fdc19dc3800c4e48b9a3a8e9
7c30dd532ebdd90b1c0bccb346e85a455af0abdb29da4527
Hay al menos un valor de 'data' que NO es múltiplo de 16.
> ./check_parsred.sh > posibles_keyys.txt
> cat posibles_keyys.txt -l java

File: posibles_keyys.txt

Líneas donde 'data' es múltiplo de 16 y <= 200, junto a la siguiente línea:
Length raw data: 48
a4940d6ff0a59421822467d80d1b620bc7ecfa661c452a85c0486b56aa752e908c4aeb3f2f0a64d9c02d7025713867ee
7076675dbed9baf4ffcc36ed8d9eccc2bfed095de2439e347aea08157e01978eb7b7cab86bea3d97e7062c9990bf7407ee88eab9ee07bc9
062c9990bf7407ee88eab9ee07bc9761a6371323f285bba8c2d1d4b4a04a09655d675e3ac7e8e
Length raw data: 80
46e10ae1e8c22141cc1242189a988f329cd1e1e4b6176a12b2de7dfb406b511907b78f39b04a139bfa61fed4fdc19dc3800c4e48b9a3a8e97c30dd532ebdd90b1c0bccb346e85a455af0abdb29da4527
1fed4fdc19dc3800c4e48b9a3a8e97c30dd532ebdd90b1c0bccb346e85a455af0abdb29da4527
Hay al menos un valor de 'data' que NO es múltiplo de 16.

@ > ~/HTB/Challenge > ✓

1 #!/bin/bash
2 $_
3 archivo="parsed_output.txt"
4 all_ok=true
5 $_
6 echo "Líneas donde 'data' es múltiplo de 16 y <= 200, junto a la siguiente línea:$_
7 $_
8 # Almacenamos todas las líneas del archivo en un array para
9 # poder acceder a la siguiente línea fácilmente
10 mapfile -t lines < "$archivo"
11 num_lines=${#lines[@]}
12 $_
13 for (( i=0; i<num_lines; i++ )); do
14     line="${lines[$i]}"
15     if [[ $line =~ data:\ ([0-9]+) ]]; then
16         value=${BASH_REMATCH[1]}
17         if (( value % 16 == 0 && value <= 200 )); then
18             echo "$line"
19             if (( i+1 < num_lines )); then
20                 echo "${lines[$((i+1))]}"
21             fi
22         fi
23         if (( value % 16 != 0 )); then
24             all_ok=false
25         fi
26     fi
27 done
28 $_
29 if $all_ok; then
30     echo "Todos los valores de 'data' son múltiplos de 16."
31 else
32     echo "Hay al menos un valor de 'data' que NO es múltiplo d
e 16."
33 fi
34 $_
~
~ check_parsred.sh
1,1
Todo

```

```

52 UNKNOWN: 1
> ./check_parsred.sh > posibles_keyys.txt
> cat posibles_keyys.txt -l java

File: posibles_keyys.txt

Líneas donde 'data' es múltiplo de 16 y <= 200, junto a la siguiente línea:
Length raw data: 48
a4940d6ff0a59421822467d80d1b620bc7ecfa661c452a85c0486b56aa752e908c4aeb3f2f0a64d9c02d7025713867ee
Length raw data: 80
7076675dbed9baf4ffcc36ed8d9eccc2bfed095de2439e347aea08157e01978eb7b7cab86bea3d97e7062c9990bf7407ee88eab9ee07bc9761a6371323f285bba8c2d1d4b4a04a09655d675e3ac7e8e
Length raw data: 80
46e10ae1e8c22141cc1242189a988f329cd1e1e4b6176a12b2de7dfb406b511907b78f39b04a139bfa61fed4fdc19dc3800c4e48b9a3a8e97c30dd532ebdd90b1c0bccb346e85a455af0abdb29da4527
1fed4fdc19dc3800c4e48b9a3a8e97c30dd532ebdd90b1c0bccb346e85a455af0abdb29da4527
Hay al menos un valor de 'data' que NO es múltiplo de 16.

@ > ~/HTB/Challenge > ✓

```

## 6.4. Extracción del hex del bloque óptimo (48 bytes)

Hex seleccionado:

a4940d6ff0a59421822467d80d1b620bc7ecfa661c452a85c0486b56aa752e908c4aeb3f2f0a64d9c02d7025713867ee

## 7. Derivación de claves usando el bloque como oráculo

### 7.1. Comando de derivación crítica

Comando ejecutado:

```

python3 cs-extract-key.py -t
a4940d6ff0a59421822467d80d1b620bc7ecfa661c452a85c0486b56aa752e908c4aeb3f2f0a64d9c02d7025713867ee
freesteam.dmp

```

Parámetros explicados en detalle:

- `cs-extract-key.py` : Script de NVISO para extracción/derivación de claves
- `-t` : Parámetro para usar un bloque de datos como "template" u oráculo
- `a4940d6f...` : El hex de 48 bytes extraído del PCAP (bloque cifrado)
- `freesteam.dmp` : Volcado de memoria (usado como contexto adicional)

¿Cómo funciona internamente?

1. El script toma el bloque hex como "texto cifrado conocido"
2. Prueba diferentes combinaciones de claves AES/HMAC
3. Para cada combinación, intenta descifrar el bloque
4. Verifica si el resultado descifrado tiene estructura válida de protocolo Beacon
5. Cuando encuentra una combinación que produce datos coherentes, esas son las claves correctas

Este comando le dice al script "aquí tienes un fragmento cifrado que sabemos corresponde a un mensaje válido del Beacon; úsalo como pista para encontrar las claves en el volcado de memoria".

```
python3 cs-extract-key.py -t <HEX_DE_48_BYT> freesteam.dmp
```

El script toma ese bloque de datos cifrados de 48 bytes y prueba sistemáticamente diferentes pares de claves AES y HMAC extraídas del volcado de memoria. Para cada par, intenta descifrar el bloque y comprueba si el resultado tiene la estructura interna esperada de un mensaje Beacon (un ID de comando válido, longitud coherente, campos reconocibles). Tan pronto encuentra unas claves que descifran correctamente el mensaje, las reporta como las **claves de sesión** necesarias para descifrar todo el resto del tráfico C2.

```
> python3 cs-extract-key.py -t a4940d6ff0a59421822467d80d1b620bc7ecfa661c452a85c0486b56aa752e908c4aeb3f2f0a64d9c02d7025713867ee freesteam.dmp
File: freesteam.dmp
Searching for AES and HMAC keys
Searching after sha256\x00 string (0x4048a)
AES key position: 0x00447f81
AES Key: 3ae7f995a2392c86e3fa8b6fbc3d953a
HMAC key position: 0x0044b2a1
HMAC Key: bf2d35c0e9b64bc46e6d513c1d0f6ffe
SHA256 raw key: bf2d35c0e9b64bc46e6d513c1d0f6ffe:3ae7f995a2392c86e3fa8b6fbc3d953a
Searching for raw key
Searching after sha256\x00 string (0x441a49)
AES key position: 0x00447f81
AES Key: 3ae7f995a2392c86e3fa8b6fbc3d953a
HMAC key position: 0x0044b2a1
HMAC Key: bf2d35c0e9b64bc46e6d513c1d0f6ffe
Searching for raw key
```

## 7.2. Resultado exitoso de la derivación

Output completo:

```
File: freesteam.dmp

Searching for AES and HMAC keys

Searching after sha256\x00 string (0x4048a)

AES key position: 0x00447f81

AES Key: 3ae7f995a2392c86e3fa8b6fbc3d953a

HMAC key position: 0x0044b2a1

HMAC Key: bf2d35c0e9b64bc46e6d513c1d0f6ffe

SHA256 raw key: bf2d35c0e9b64bc46e6d513c1d0f6ffe:3ae7f995a2392c86e3fa8b6fbc3d953a
```

Análisis del resultado:

- ¡Éxito! El método alternativo funcionó perfectamente
- Se encontraron ambas claves necesarias:
  - **HMAC Key:** bf2d35c0e9b64bc46e6d513c1d0f6ffe (integridad)
  - **AES Key:** 3ae7f995a2392c86e3fa8b6fbc3d953a (cifrado)
- **Formato final:** bf2d35c0e9b64bc46e6d513c1d0f6ffe:3ae7f995a2392c86e3fa8b6fbc3d953a
- Se muestran las posiciones exactas en memoria donde se hallaron las claves

Proceso mental clave:

El script logró "romper" el cifrado usando el bloque de 48 bytes como oráculo. Esto confirma que:

1. El bloque seleccionado contenía un mensaje válido de Beacon
2. Las claves derivadas son correctas para toda la sesión
3. Ahora puedo descifrar todo el tráfico del PCAP

## 8. Descifrado completo del canal C2

### 8.1. Comando principal de descifrado

Comando ejecutado:

```
python3 cs-parse-http-traffic.py -k bf2d35c0e9b64bc46e6d513c1d0f6ffe:3ae7f995a2392c86e3fa8b6fbc3d953a
```

capture.pcap

## Parámetros críticos:

- **-k** : Especifica las claves en formato **HMAC:AES** (orden crucial)
- **bf2d35c0...** : HMAC key (primera)
- **3ae7f995...** : AES key (segunda)
- **capture.pcap** : Archivo de tráfico a descifrar

```
32 UNKNOWN: 1
> ./check_parsred.sh > posibles_keyys.txt [1]
> cat posibles_keyys.txt -l java
File: posibles_keyys.txt
1 Líneas donde 'data' es múltiplo de 16 y <= 200, junto a la siguiente línea:
2 Length raw data: 48
3 a4940d6ff0a59421822467d80d1b620bc7ecfa661c452a85c0486b56aa752e908c4aeb3f2f0a64d9c02d7025713867ee
4 Length raw data: 80
5 7076675d8ed9baf94ffcc36ed8d9eecc2bfed095de2439e347aea08157e01978eb7b7cab86bea3d97e7062c9990bf7407ee88eab9ee07bc9761a6371323f285bba8c2d1d4b4a04a09655d675e3ac7e8e
6 Length raw data: 80
7 46e10ae1e8c2214cc1242189a988f329cd1e1e4b6176a12b2de7dfb406b511907b78f39b04a139bfa61fed4fdc19dc3800c4e48b9a3a8e97c30dd532ebdd90b1c0bccb346e85a455af0abdb29da4527
8 Hay al menos un valor de 'data' que NO es múltiplo de 16.

> python3 cs-extract-key.py -t a4940d6ff0a59421822467d80d1b620bc7ecfa661c452a85c0486b56aa752e908c4aeb3f2f0a64d9c02d7025713867ee freesteam.dmp [2]
File: freesteam.dmp
Searching for AES and HMAC keys
Searching after sha256\x00 string (0x4048a)
AES key position: 0x00447f81
AES Key: 3ae7f995a2392c86e3fa8b6fbc3d953a
HMAC Key position: 0x0044b2a1
HMAC Key: bf2d35c0e9b64bc46e6d513c1d0f6ffe
SHA256 raw key: bf2d35c0e9b64bc46e6d513c1d0f6ffe:3ae7f995a2392c86e3fa8b6fbc3d953a
Searching for raw key
Searching after sha256\x00 string (0x441a49)
AES key position: 0x00447f81
AES Key: 3ae7f995a2392c86e3fa8b6fbc3d953a
HMAC key position: 0x0044b2a1
HMAC Key: bf2d35c0e9b64bc46e6d513c1d0f6ffe
Searching for raw key
> python3 cs-parse-http-traffic.py -k bf2d35c0e9b64bc46e6d513c1d0f6ffe:3ae7f995a2392c86e3fa8b6fbc3d953a -e capture.pcap [3]
Packet number: 12
HTTP response (for request 7 GET)
Length raw data: 14336
HMAC signature invalid
Packet number: 47
HTTP response (for request 23 GET)
Length raw data: 206401
HMAC signature invalid
Packet number: 69
```

```
Searching for raw key
> python3 cs-parse-http-traffic.py -k bf2d35c0e9b64bc46e6d513c1d0f6ffe:3ae7f995a2392c86e3fa8b6fbc3d953a -e capture.pcap
Packet number: 12
HTTP response (for request 7 GET)
Length raw data: 14336
HMAC signature invalid
Packet number: 47
HTTP response (for request 23 GET)
Length raw data: 206401
HMAC signature invalid
Packet number: 69
HTTP response (for request 66 GET)
Length raw data: 48
Timestamp: 1637354721 20211119-204521
Data size: 8
Command: 27 GETUID
Arguments length: 0

Packet number: 76
HTTP request POST
http://192.168.1.9/submit.php?id=1909272864
Length raw data: 68
Counter: 2
Callback: 16 BEACON_GETUID
b'WS02\\npatrick (admin)'

Packet number: 101
HTTP response (for request 86 GET)
Length raw data: 87648
Timestamp: 1637354781 20211119-204621
Data size: 87608
Command: 89 UNKNOWN
Arguments length: 87552
b'MZ\xe8\x00\x00\x00\x00[REU\x89\xe5\x81\xc3)\x1f\x00\x00\xff\xd3\x89\xc3Wh\x04\x00\x00\x00P\xff\xd0
MD5: 1e4b88220d370c6bc55e213761f7b5ac
Command: 40 UNKNOWN
Arguments length: 40
Unknown1: 0
Unknown2: 1602864
Pipename: b'\\\\.\\pipe\\8e09448'
Command: b'net user'

Packet number: 224
HTTP request POST
http://192.168.1.9/submit.php?id=1909272864
Length raw data: 324
Counter: 6
Callback: 22 TODO
b'\xff\xff\xff\xfe'

-----
C:\Users\patrick\Desktop\*
D      0      11/19/2021 12:24:08 .
D      0      11/19/2021 12:24:08 ..
F    5175    11/11/2021 03:24:13 cheap_spare_parts_for_old_blimps.docx
F    282    11/10/2021 07:02:24 desktop.ini
F   24704    11/11/2021 03:22:16 gogglestown_citizens_osint.xlsx
F   62393    11/19/2021 12:24:10 orders.pdf

-----
```

```
MD5: 025952a41uba97bac3ccc812c01b9000  
  
Packet number: 254  
HTTP request POST  
http://192.168.1.9/submit.php?id=1909272864  
Length raw data: 62572  
Counter: 7  
Callback: 2 DOWNLOAD_START  
parameter1: 0  
length: 62393  
filenameDownload: C:\Users\ncpytrick\Desktop\orders.pdf  
  
Counter: 8  
Callback: 8 DOWNLOAD_WRITE  
Length: 62393  
MD5: 00f542efefcccd7a89a55c133180d8581  
  
Counter: 9  
Callback: 9 DOWNLOAD_COMPLETE  
b'\x00\x00\x00\x00'  
  
Commands summary:  
11 DOWNLOAD: 1  
27 GETUID: 1  
40 UNKNOWN: 3  
44 UNKNOWN: 2  
53 LIST_FILES: 1  
89 UNKNOWN: 1  
  
Callbacks summary:  
2 DOWNLOAD_START: 1  
8 DOWNLOAD_WRITE: 1  
9 DOWNLOAD_COMPLETE: 1  
16 BEACON_GETUID: 1  
21 BEACON_OUTPUT_HASHES: 1  
22 TODO: 1  
24 BEACON_OUTPUT_NET: 1  
32 UNKNOWN: 1
```

```
python3 cs-parse-http-traffic.py -k bf2d35c0e9b64bc46e6d513c1d0f6ffe:3ae7f995a2392c86e3fa8b6fbc3d953a  
capture.pcap
```

El script toma las **claves de sesión** (HMAC y AES) ya derivadas y las utiliza para **descifrar completamente** el canal C2 de Cobalt Strike en el PCAP. Al especificar `-k HMAC:AES`, el script recorre cada bloque cifrado encontrado en las peticiones y respuestas HTTP, aplica el descifrado AES y valida la integridad con HMAC. El resultado es una **salida legible** que muestra los comandos internos del Beacon (por ejemplo, GETUID, LIST\_FILES, DOWNLOAD), los callbacks de respuesta y los metadatos asociados (hashes, nombres de archivos, longitudes). Con este paso, transformamos el tráfico “críptico” en un registro forense detallado de la actividad del atacante.

## 8.2. Análisis de comandos descifrados

Resultados parciales obtenidos:

Reconocimiento inicial:

```
Packet number: 69  
  
Command: 27 GETUID  
  
Arguments length: 0  
  
Callback: 16 BEACON_GETUID  
  
b'WS02\\ncpytrick (admin)'
```

Enumeración del sistema:

```
Command: 53 LIST_FILES  
  
[Resultados de listado de archivos]  
  
Command: OUTPUT_HASHES
```

[Hashes de archivos identificados]

#### Exfiltración detectada:

```
Command: 11 DOWNLOAD
Callback: 2 DOWNLOAD_START
Callback: 8 DOWNLOAD_WRITE
Callback: 9 DOWNLOAD_COMPLETE
Filename: orders.pdf
MD5: [hash verifiable]
Size: [tamaño en bytes]
```

#### Proceso mental de análisis:

- El atacante siguió una metodología sistemática
- **Reconocimiento:** Identificó usuario y privilegios (admin)
- **Enumeración:** Listó archivos para identificar objetivos
- **Exfiltración:** Descargó documentos específicos de interés

### 8.3. Extracción de artefactos transferidos

#### Comando con extracción habilitada:

```
python3 cs-parse-http-traffic.py -k bf2d35c0e9b64bc46e6d513c1d0f6ffe:3ae7f995a2392c86e3fa8b6fb3d953a -e
capture.pcap
```

Parámetro `-e` : Habilita extracción física de todos los payloads transferidos

#### Archivo extraído:

```
payload-00f542efefcccd7a89a55c133180d8581.vir
```

### 8.4. Validación del contenido exfiltrado

#### Verificación de tipo:

```
file payload-00f542efefcccd7a89a55c133180d8581.vir
# Resultado: PDF document, version 1.4
```

#### Renombrado y apertura:

```
mv payload-00f542efefcccd7a89a55c133180d8581.vir orders.pdf
libreoffice orders.pdf
```

#### Contenido descubierto:

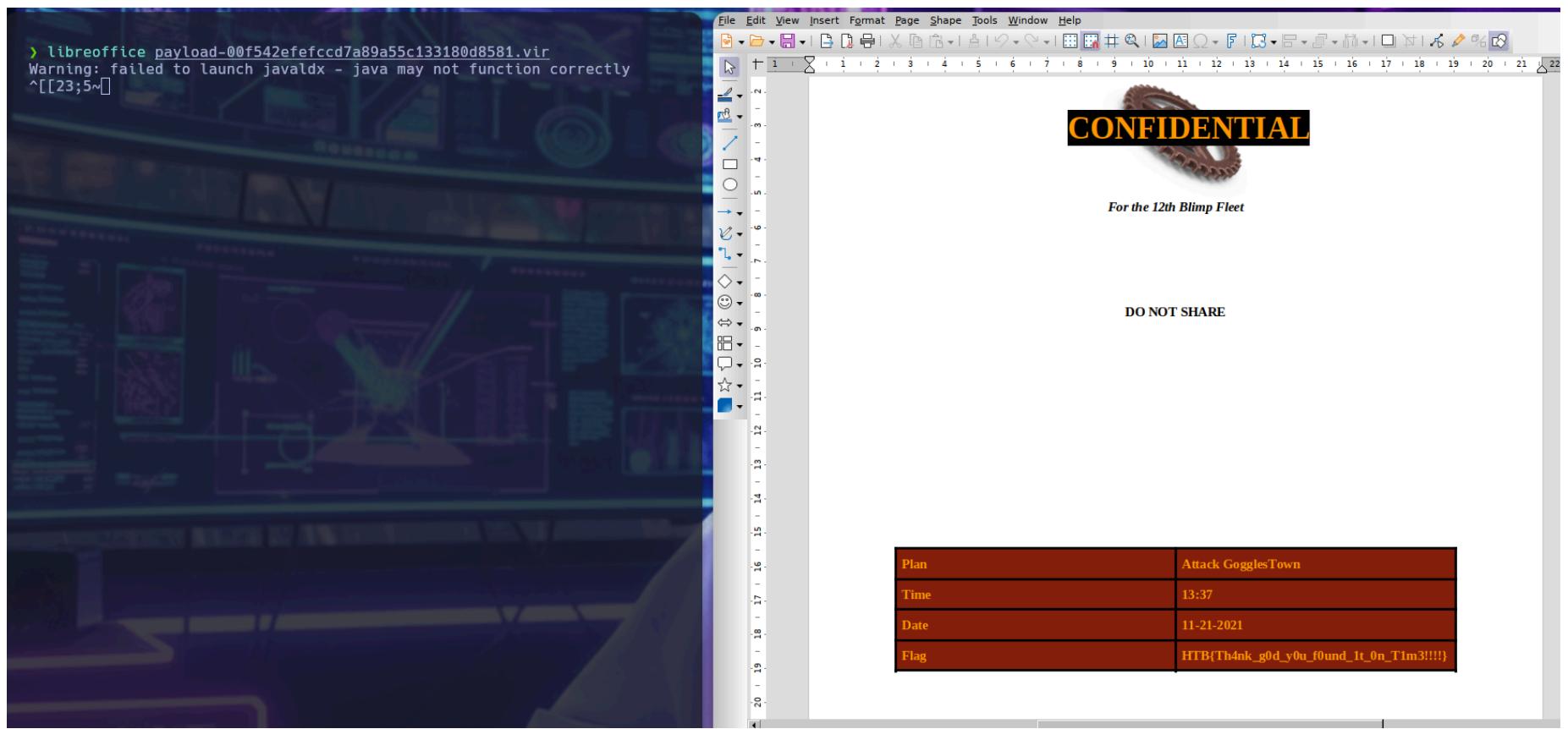
- **Documento confidencial:** "CONFIDENTIAL - For the 12th Blimp Fleet"
- **Información estratégica:** Plan de ataque contra "GogglesTown"

- **Detalles operacionales:**

- Fecha: 11-21-2021

- Hora: 13:37

- **Flag del desafío:** HTB{Th4nk\_g0d\_y0u\_f0und\_1t\_0n\_T1m3!!!!}



## Conclusión

Este análisis demuestra la importancia de correlacionar múltiples fuentes de evidencia forense. La combinación de tráfico de red cifrado y volcado de memoria nos permitió:

1. **Identificar** el tipo exacto de malware (Cobalt Strike Beacon)
2. **Extraer** las claves criptográficas de la sesión
3. **Descifrar** completamente el canal C2
4. **Reconstruir** la cadena de ataque completa
5. **Recuperar** los documentos exfiltrados

La metodología presentada es replicable para futuros incidentes de Cobalt Strike y otros frameworks C2 similares que utilicen cifrado simétrico y almacenen claves en memoria del proceso.