



Universidad  
Rey Juan Carlos

# Análisis extra

MALWARE Y AMENAZAS DIRIGIDAS

Javier Burgos Chamorro

# 1-INTRODUCION

Se nos da un pdf por lo que primero que hago es un file para ver si me da algo diferente

```
remnux@remnux:~/Desktop/pegasus analisis$ file pegasus.pdf
pegasus.pdf: PDF document, version 1.4
```

Como podemos ver dice que es un pdf algo que ya sabíamos, por lo que pruebo a usar bindwalk para ver si hay archivos embebidos.

```
remnux@remnux:~/Desktop/pegasus analisis$ binwalk -e pegasus.pdf

DECIMAL          HEXADECIMAL      DESCRIPTION
-----
0                0x0              PDF document, version: "1.4"
1320759          0x142737         MySQL ISAM compressed data file Version 7
1860507          0x1C639B         Copyright string: "copyright/ordfeminine 172/logicalnot/.notdef/regist
ered/macron/degree/plusminus/twosuperior/threesuperior/acute/mu 183/periodcen"
1879562          0x1CAE0A         gzip compressed data, from Unix, last modified: 1970-01-01 00:00:00 (
ull date)
```

Como suponía tiene archivos embebidos dentro, el que más me llama de primeras es el archivo '1CAE0A' por lo que uso file, para ver de que tipo es.

```
remnux@remnux:~/Desktop/pegasus analisis/_pegasus.pdf.extracted$ ls
1CAE0A  1CAE0A.gz
remnux@remnux:~/Desktop/pegasus analisis/_pegasus.pdf.extracted$ file 1CAE0A
1CAE0A: ASCII text
remnux@remnux:~/Desktop/pegasus analisis/_pegasus.pdf.extracted$
```

Al hacer file me muestra que es un txt. Le hago un cat para ver que tiene dentro el txt.

```
remnux@remnux:~/Desktop/pegasus analisis/_pegasus.pdf.extracted$ cat 1CAE0A
f0VMRgIBAQAAAAAAAAAAAAAAAAMAPgABAAAAAICYAAAAAAAAABAAAAAAAAAAGChAAAAAAAAAAAAAAAAEAA0AAN
AEAAHQACAAAYAAAAEAAAAQAAAAAAAAABAAAAAAAAAAEAAAAAAAAA2AIAAAAAADYAgAAAAAAAAAgA
AAAAAAAAAAwAAAAQAAAAAYwAAAAAAABgDAAAAAAAAAGAMAAAAAAAAACAAAAAAAAABwAAAAAAAAAAQAA
AAAAAAAABAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAACgRAAAAAAAAKBEAAAAAAAAAAEAAA
AAAAAAEAAAAFAAAAAACAAAAAAAAAAAAIAAAAAAAAAAAgAAAAAAAAAUv0AAAAAAAAABRXQAAAAAAAAAQAAA
AAAAAQAAAAQAAAAAGAAAAAAAAACAAAAAAAAAAIAAAAAAAAC4EgAAAAAAALgSAAAAAAAAABAAAAAA
AAABAAAAABgAAACiCAAAAAAAAAAKKwAAAAAAAAAoRAAAAAAAAAAAEAAAAAAAAAMQAAAAAAAAAAEAAAAAA
AAIAAAAGAAAAOJwAAAAAAAAA4rAAAAAAAAADisAAAAAAAAAEIAAAAAAAQAgAAAAAAAgAAAAAAAgA
AAAAAQAAAAA4AwAAAAAAADgDAAAAAAAAAOAMAAAAAAAAAgAAAAAAAAACAAAAAAAAACAAAAAAAAAAE
```

Creo que está cifrado así que pruebo a descodificar en base 64

```
remnux@remnux:~/Desktop/pegasus analisis/_pegasus.pdf.extracted$ base64 -d 1CAE0A >descodificado
remnux@remnux:~/Desktop/pegasus analisis/_pegasus.pdf.extracted$ ls
1CAE0A  1CAE0A.gz  descodificado
```

El archivo que me saca al que he llamado descodificado es este.

```
remnux@remnux:~/Desktop/pegasus analisis/_pegasus.pdf.extracted$ file descodificado
descodificado: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter
lib64/ld-linux-x86-64.so.2, BuildID[sha1]=2765a76b35c6ae836802411554f57f8dc059554f, for GNU/Linux 3.2.0,
tripped
```

Le doy permisos y lo ejecuto a ver que ocurre.

```
remnux@remnux:~/Desktop/pegasus analisis/_pegasus.pdf.extracted$ chmod +x descodificado
remnux@remnux:~/Desktop/pegasus analisis/_pegasus.pdf.extracted$ ls
1CAE0A  1CAE0A.gz  descodificado
remnux@remnux:~/Desktop/pegasus analisis/_pegasus.pdf.extracted$ ls
1CAE0A  1CAE0A.gz  descodificado
remnux@remnux:~/Desktop/pegasus analisis/_pegasus.pdf.extracted$ ./descodificado
USAGE: verify dir kalpha kbeta [-Dh] [-n lfilename] [-i inode begin end] [-nfs0 nlog0 -nfs1 nlog1...]
```

Al ver que ejecuta, empiezo a analizarlo.

## 2-ANALISIS DEL BINARIO

Lo analizo en radare2 y busco que strings me tira el comando iz pues podría estar ahí la flag.

```
remnux@remnux:~/Desktop/pegasus analysis/_pegasus.pdf.extracted$ radare2 -d descodificado
Process with PID 13062 started...
= attach 13062 13062
bin.baddr 0x55fc33a8d000
Using 0x55fc33a8d000
asm.bits 64
Warning: r_bin_file_hash: file exceeds bin.hashlimit
-- Mind that the 'g' in radare is silent
[0x7fccdbae100]> aaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Check for objc references
[x] Check for vtables
[TOFIX: aaft can't run in debugger mode.ions (aaft)]
[x] Type matching analysis for all functions (aaft)
[x] Propagate noreturn information
[x] Use -AA or aaaa to perform additional experimental analysis.
[0x7fccdbae100]> iz
[Strings]
nth paddr          vaddr          len size section type  string
0 0x0000804b 0x55fc33a9504b 4 5 .rodata ascii \frgg
1 0x00008056 0x55fc33a95056 9 11 .rodata utf8 WDD[EB]!XXK blocks=Basic Latin,Arabic
2 0x00008081 0x55fc33a95081 4 5 .rodata ascii H^c+
3 0x0000809f 0x55fc33a9509f 5 8 .rodata utf8 9'W*^ blocks=Basic Latin,Georgian
4 0x000080aa 0x55fc33a950aa 4 5 .rodata ascii 8D2Q
5 0x000080b8 0x55fc33a950b8 8 9 .rodata ascii \dq\f\#8A
6 0x000080de 0x55fc33a950de 5 7 .rodata utf8 fbjV.5 blocks=Basic Latin,Arabic
```

Al analizar la lista de strings ,me llama la atención uno en concreto.

```
7 0x000080f9 0x55fc33a950f9 4 5 .rodata ascii 0n +
8 0x00008120 0x55fc33a95120 32 33 .rodata ascii grep -qs vboxvideo /proc/modules
9 0x00008141 0x55fc33a95141 13 14 .rodata ascii out of memory
10 0x0000814f 0x55fc33a9514f 17 18 .rodata ascii can't open dir %s
11 0x00008161 0x55fc33a95161 5 6 .rodata ascii %s/%s
12 0x00008167 0x55fc33a95167 11 12 .rodata ascii .SEAFES.LOG
```

Investigando, básicamente lo que hace esa líneas es comprobar si en nuestro sistema , hay un módulo de virtual box , el cual se instala al cargar una imagen de un sistema .Por lo que me da a entender ;es que le programa desde dentro consulta si esta ese modulo en nuestro sistema , de tal manera que si esta, quiere decir que estamos en una máquina virtual , por lo que supongo que el binario modificara su comportamiento .

Al analizar el código descubro que la función que hace grep fcn 00002709.

```
30: fcn.00002709 ();
0x00002709      endbr64
0x0000270d      push rbp
0x0000270e      mov rbp, rsp
0x00002711      lea rdi, str.grep__qs_vboxvideo__proc_modules ; 0x8120 ; const char *string
0x00002718      call system ; sym.imp.system ; int system(const char *string)
0x0000271d      test eax, eax
0x0000271f      sete al
0x00002722      movzx eax, al
0x00002725      pop rbp
```

Podemos ver que básicamente lo que hace es comprobar si es una máquina virtual ,si es así el grep devuelve un cero el cual comprueba al hacer

Al fijarnos detenidamente podemos apreciar que la función de abajo es la que la llama a esta .test eax eax de tal manera que si es 0 dará cero pues , hace un and de eax , tras esto esta la intrusión sete al que comprueba si es igual a 0 o no lo anterior , de tal manera que pone a 1 al. Lo siguiente que hace es mover el valor de al a eax con movzx para retornarlo.

```

156: fcn.00002727 ();
; var int64_t var_28h @ rbp-0x28
; var void *var_20h @ rbp-0x20
; var void *var_18h @ rbp-0x18
; var int64_t var_10h @ rbp-0x10
; var int64_t canary @ rbp-0x8
0x00002727     endbr64
0x0000272b     push rbp
0x0000272c     mov rbp, rsp
0x0000272f     sub rsp, 0x30
0x00002733     mov rax, qword fs:[0x28]
0x0000273c     mov qword [canary], rax
0x00002740     xor eax, eax
0x00002742     call fcn.00002709
0x00002747     test eax, eax
0x00002749     jne 0x27ac
0x0000274b     mov edi, 0x400 ; size_t size
0x00002750     call malloc ; sym.imp.malloc
0x00002755     new qword [var_20h] ; var

```

De aquí pasamos a la fcn 00002727 quien es quien llama primeramente a la función anterior. A esta le llega el valor de la función anterior y realiza otro **test eax eax**.

En base al valor que le de , hará el salto condicional o no, de tal manera que si en el paso anterior le da 1 quiere decir que es una máquina virtual y que las intruciones referentes al malware .

Para solucionar esto basta con cambiar el salto ,invertir **jne** por un **je** de tal manera que al ejecutar el binario consigo que ejecute esas líneas y me de esta salida

```

remnux@remnux:~/Desktop/pegasus analisis/_pegasus.pdf.extracted$ ./descodificado
BOOOM! HAS EJECUTADO EL MALWARE, ACABAS DE SOLTAR UN GUSANO PELIGROSO EN TU RED.

```

Tras esto me fijo en el código que he vitado saltar al invertir el salto

```

0x0000274b     mov     edi, 0x400 ; size_t size
0x00002750     call   malloc ; sym.imp.malloc ; void *malloc(size_t size)
0x00002755     mov     qword [var_20h], rax
0x00002759     lea     rsi, [var_28h]
0x0000275d     mov     rax, qword [var_20h]
0x00002761     mov     ecx, 0x12c
0x00002766     lea     rdx, [0x00008020]
0x0000276d     mov     rdi, rax
0x00002770     call   uncompress ; sym.imp.uncompress
0x00002775     mov     rax, qword [var_20h]
0x00002779     mov     qword [var_18h], rax
0x0000277d     mov     rax, qword [var_18h]
0x00002781     and     rax, 0xffffffffffff0000
0x00002787     mov     qword [var_10h], rax
0x0000278b     mov     rax, qword [var_10h]
0x0000278f     mov     edx, 7
0x00002794     mov     esi, 0x1000
0x00002799     mov     rdi, rax
0x0000279c     call   mprotect ; sym.imp.mprotect
0x000027a1     mov     rdx, qword [var_18h]
0x000027a5     mov     eax, 0
0x000027aa     call   rdx
0x000027ac     nop
0x000027ad     mov     rax, qword [canary]
0x000027b1     xor     rax, qword fs:[0x28]
0x000027ba     je      0x27c1
0x000027bc     call   __stack_chk_fail ; sym.imp.__stack_chk_fail ; void __stack_chk_fail
0x000027c1     leave
0x000027c2     ret

```

Si me fijo puedo intuir que en esta parte es para conceder permisos de ejecución , para algo que descomprime.

Si nos fijamos ,descomprime un total de 300 bytes (**mov ecx, 0x12**) y esto lo hace en la dirección 0x00008020.

Estoy lo voy a hacer yo manualmente para ver que contiene eso que descomprime .Utilizo xxd para que me haga un volcado de estos datos .





### 3-BUSCO LA FLAG

Lo primero que hago es analizar el código en radare2 .

Lo primero es acceder al main.

```
; arg char**argv @ f30f1efa          endbr64
0x555555559636 b          55          push rbp
0x55555555963a          4889e5        mov rbp, rsp
0x55555555963b          53          push rbx
0x55555555963f          4881ec001000. sub rsp, 0x1000
0x555555559640          48830c2400    or qword [rsp], 0
0x555555559646          4881ec080100. sub rsp, 0x108
0x555555559652          89bdfceeffff. mov dword [var_1104h], edi ; argc
0x555555559658          4889b5f0eef. mov qword [var_1110h], rsi ; argv
0x55555555965f          64488b042528. mov rax, qword fs:[0x28]
0x555555559668          488945e8      mov qword [var_18h], rax
0x55555555966c          31c0         xor eax, eax
0x55555555966e          c7851ceffff. mov dword [var_10e4h], 1
0x555555559678          48c78528eff. mov qword [var_10d8h], 0
0x555555559683          48c78530eff. mov qword [var_10d0h], 0
0x55555555968e          48c78538eff. mov qword [var_10c8h], 0
0x555555559699          488d05c72a00. lea rax, str..SEALFS.LOG ; 0x55555555c167 ; ".
0x5555555596a0          48898548eff. mov qword [var_10b8h], rax
0x5555555596a7          48c78550eff. mov qword [var_10b0h], 0
0x5555555596b2          e870d0ffff. call 0x55555556727
0x5555555596b5          93bdfceffff. mov dword [var_1104h], 0
```

El primer call que realiza es a la función 00002727 la que contenía el shellcode.

Nos metemos en esta función :

```
; CALL XREF from main @ 0x5555555596b2
;-- rip:
0x5555555556727 b          f30f1efa          endbr64
0x555555555672b          55          push rbp
0x555555555672c          4889e5        mov rbp, rsp
0x555555555672f          4883ec30      sub rsp, 0x30
0x5555555556733          64488b042528. mov rax, qword fs:[0x28]
0x555555555673c          488945f8      mov qword [rbp - 8], rax
0x5555555556740          31c0         xor eax, eax
0x5555555556742          e8c2fffffff. call 0x5555555556709
0x5555555556747          85c0         test eax, eax
0x5555555556749          7461         je 0x55555555567ac
0x555555555674b          bf00040000    mov edi, 0x400 ; 1024
0x5555555556750          e89bfcffff. call sym.imp.malloc ; void
0x5555555556755          488945e0      mov qword [rbp - 0x20], rax
0x5555555556759          488d75d8      lea rsi, [rbp - 0x28]
0x555555555675d          488b45e0      mov rax, qword [rbp - 0x20]
0x5555555556761          b92c010000    mov ecx, 0x12c ; 300
0x5555555556766          488d15b35800. lea rdx, [0x55555555c020]
0x555555555676d          4889c7        mov rdi, rax
0x5555555556770          e88bfeffff. call sym.imp.uncompress
0x5555555556775          488b45e0      mov rax, qword [rbp - 0x20]
0x5555555556779          488945e8      mov qword [rbp - 0x18], rax
0x555555555677d          488b45e8      mov rax, qword [rbp - 0x18]
0x5555555556781          482500f0ffff. and rax, 0xffffffffffff000
0x5555555556787          488945f0      mov qword [rbp - 0x10], rax
0x555555555678b          488b45f0      mov rax, qword [rbp - 0x10]
0x555555555678f          ba07000000    mov edx, 7
0x5555555556794          be00100000    mov esi, 0x1000
0x5555555556799          4889c7        mov rdi, rax
0x555555555679c          e89ffbffff. call sym.imp.mprotect
0x55555555567a1          488b55e8      mov rdx, qword [rbp - 0x18]
0x55555555567a5          b800000000    mov eax, 0
0x55555555567aa          ffd2         call rdx
```

Si nos fijamos después de conceder permisos , hace un call a rdx por lo que accedo a esta llamada.

```
[0x7ffff7b1248b]> dc
hit breakpoint at: 5555555567aa
[0x5555555567aa]> ds
[0x55555555602a0]> pd
;-- rdx:
;-- rip:
0x55555555602a0          e89d000000    call 0x5555555560342
0x55555555602a5          0f8497000000. je 0x5555555560342
0x55555555602ab          0f8591000000. jne 0x5555555560342
0x55555555602b1          61          invalid
0x55555555602b2          5c          pop rsp
0x55555555602b3          484e6e      outsb dx, byte [rsi]
0x55555555602b6          3227        xor ah, byte [rdi]
0x55555555602b8          49          invalid
0x55555555602b9          62          invalid
0x55555555602ba          40          invalid
0x55555555602bb          27          invalid
0x55555555602bc          446956445477. imul r10d, dword [rsi +
0x55555555602c4          4e03564b    add r10, qword [rsi + 0
0x55555555602c8          216e52      and dword [rsi + 0x52],
```

Esta llamada tiene como primera instrucción otra llamada , de tal manera que entro a esta nueva llamada .



```

[0x555555560348]> db 0x555555560365
[0x555555560348]> dc
hit breakpoint at: 555555560365
[0x555555560365]> px 300 @0x5555555602b1
offset - 0 1 2 3 4 5 6 7 8 9 A B C D E F 0123456789ABCDEF
0x5555555602b1 424f 4f4f 4d21 2048 4153 2045 4a45 4355 B000M! HAS EJECU
0x5555555602c1 5441 444f 2045 4c20 4d41 4c57 4152 452c TADO EL MALWARE,
0x5555555602d1 2041 4341 4241 5320 4445 2053 4f4c 5441 ACABAS DE SOLTA
0x5555555602e1 5220 554e 2047 5553 414e 4f20 5045 4c49 R UN GUSANO PELI
0x5555555602f1 4752 4f53 4f20 454e 2054 5520 5245 442e GROSO EN TU RED.
0x555555560301 0a45 5354 4120 4553 204c 4120 464c 4147 .ESTA ES LA FLAG
0x555555560311 2051 5545 2042 5553 4341 533a 205b 5768 QUE BUSCAS: [Wh
0x555555560321 6f20 7768 6174 6368 6573 2074 6865 2077 o whatches the w
0x555555560331 6174 6368 6d65 6e3f 5d0a 0a0a 2919 0d0b atchmen?].)...).
0x555555560341 0041 5f49 83c7 0c4d 89fd 41be 2313 0701 _A_I...M..A.#...
0x555555560351 49c7 c48c 0000 0045 3137 4983 c704 4983 I.....E17I...I.
0x555555560361 ec04 75f3 48c7 c001 0000 0048 c7c7 0100 ..u.H.....H...
0x555555560371 0000 498d 7500 48c7 c252 0000 000f 0549 ..I.u.H..R....I
0x555555560381 c7c4 8c00 0000 4d89 ef45 3137 4983 c704 .....M..E17I...
0x555555560391 4983 ec04 75f3 48c7 c03c 0000 0048 31ff I...u.H..<...H1.
0x5555555603a1 0f05 ffff ffff ffff ffff ffff ffff ffff .....
0x5555555603b1 ffff ffff ffff ffff ffff ffff ffff .....
0x5555555603c1 ffff ffff ffff ffff ffff ffff ffff .....
0x5555555603d1 0000 0000 0000 0000 0000 0000 0000 .....

```

Como suponía mi teoría es correcta y esta es la flag .