

PRÁCTICA 8

CTF 3

Realizada por:

Javier Rojas Horrillo

Se nos proporciona un archivo PDF así que lo primero que probamos a hacer es **extraer posibles ficheros embebidos**:

```
remnux@remnux:~/Desktop/Malware/practica8$ 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/registered/macron/degree/plusminus/twosuperior/threesuperior/acute/mu 183/periodcen"
1879562	0x1CAE0A	gzip compressed data, from Unix, last modified: 1970-01-01 00:00:00 (null date)

```
remnux@remnux:~/Desktop/Malware/practica8$ ls
pegasus.pdf  _pegasus.pdf.extracted
```

Dentro de ese directorio tenemos:

```
remnux@remnux:~/Desktop/Malware/practica8$ cd _pegasus.pdf.extracted/
remnux@remnux:~/Desktop/Malware/practica8/_pegasus.pdf.extracted$ ls
1CAE0A  1CAE0A.gz
```

Podemos ver que se trata de **txt**:

```
remnux@remnux:~/Desktop/Malware/practica8/_pegasus.pdf.extracted$ exiftool 1CAE0A
```

ExifTool Version Number	: 12.42
File Name	: 1CAE0A
Directory	: .
File Size	: 58 kB
File Modification Date/Time	: 2023:05:03 09:40:26-04:00
File Access Date/Time	: 2023:05:03 09:40:26-04:00
File Inode Change Date/Time	: 2023:05:03 09:40:26-04:00
File Permissions	: -rw-rw-r--
File Type	: TXT
File Type Extension	: txt
MIME Type	: text/plain
MIME Encoding	: us-ascii
Newlines	: Unix LF
Line Count	: 758
Word Count	: 758

El **contenido** de este es:

```
remnux@remnux:~/Desktop/Malware/practica8/_pegasus.pdf.extracted$ cat 1CAE0A
f0VMRgIbAQAAAAAAAAAMAPgABAAAAICyAAAAAAAAABAAAAAAAAAGChAAAAAAAAAAAAEAA0AAN
AEAAHQAcAAyAAAAEAAAAQAAAAAAAAABAAAAAAAAAEAAAAAAAAA2AIAAAAAADYAgAAAAAAAgA
AAAAAAAAAAwAAAAQAAAAyAwAAAAAAABgDAAAAAAAAAGAMAAAAAAAcAAAAAAAAABwAAAAAAQA
AAAAAAAABAAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAAAACgRAAAAAAAKBEAAAAAAAAEAAA
AAAAAEAAAAFAAAAAACAAAAAAAAAIAAAAAAAAAAGAAAAAAAUv0AAAAAAABRXQAAAAAAQAQAAA
AAAAAQAAAAQAAAAAgAAAAAAACAAAAAAAAAIAAAAAAAC4EgAAAAAALgSAAAAAAABAAAAAA
AAABAAAABgAAACiAAAAAAAAAKKwAAAAAAArAAAAAAAAAEAAAAAAAMAQAAAAAAAEAAAAAA
AAIAAAAGAAA0JwAAAAAA4rAAAAAAADisAAAAAAAEIAAAAAAQAgAAAAAAAgAAAAAAAB
BAAAAAQAAAA4AwAAAAAADgDAAAAAAAAA0MAAAAAAAAgAAAAAAACAAAAAAACAAAAAAAE
AAAAABAAAFgDAAAAAAAwMAAAAAABYAwAAAAAAEQAAAAAAARAAAAAAAEAAAAAAAFPl
dGQEA AAA0MAAAAAAA4AwAAAAAADgDAAAAAAAIAAAAAAAgAAAAAAAgAAAAAAAU0V0
ZAQAAADYigAAAAAANiKAAAAAA2IoAAAAAACUAQAAAAAAJQBAAAAABAAAAABR5XRk
BgAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFLldGQE
AAAAKJwAAAAAAArAAAAAAACisAAAAAA2MAAAAAADYAwAAAAAAEAAAAAAAL2xpYjY0
L2xkLWxpbnV4LXg4Ni02NC5zby4yAAAAAAEAAAAEAAAAUAAABHTLUAAgAAwAQAAAAADAAAAAA
AAQAAAAUAAAAAwAAAE0VQAnZadrNcaug2gCQRVU9X+NwFLVTwQAAAAQAAAAQAAAE0VQAAAAA
AwAAAAIAAAAAAAAAAAAAIAAA0AAAAQAAAAyAAAAAYEAAAAAJQAAAAIAAA0WX0bTnyixwA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAQAAAAEgAAAAAAAAAAAAAAAAAAAA4AQAAEgAAAAA
```

Podemos ver que **acaba** con **"=="** lo que podría ser un **base64**.

```
remnux@remnux:~/Desktop/Malware/practica8/_pegasus.pdf.extracted$ cat 1CAE0A | base64 -d > decoded
remnux@remnux:~/Desktop/Malware/practica8/_pegasus.pdf.extracted$ ls
1CAE0A  1CAE0A.gz  decoded
```

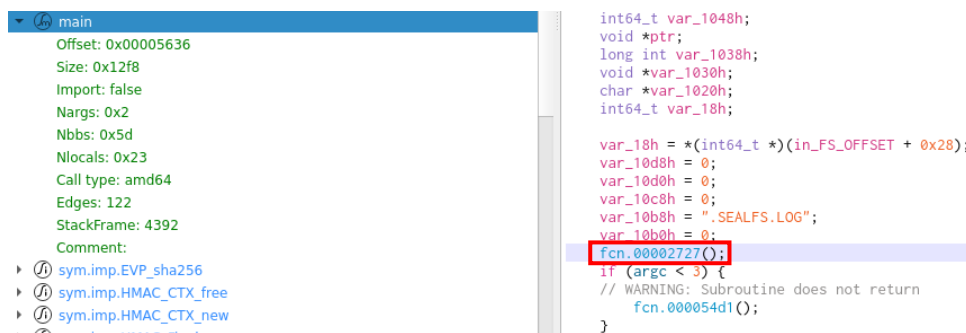
Este archivo vemos que es un **ELF**:

```
remnux@remnux:~/Desktop/Malware/practica8/_pegasus.pdf.extracted$ exiftool decoded
ExifTool Version Number      : 12.42
File Name                    : decoded
Directory                   : .
File Size                    : 43 kB
File Modification Date/Time  : 2023:05:03 10:00:28-04:00
File Access Date/Time       : 2023:05:03 10:01:06-04:00
File Inode Change Date/Time  : 2023:05:03 10:00:53-04:00
File Permissions             : -rwxrwxr-x
File Type                    : ELF shared library
File Type Extension         : so
MIME Type                    : application/octet-stream
CPU Architecture            : 64 bit
CPU Byte Order              : Little endian
Object File Type            : Shared object file
CPU Type                    : AMD x86-64
```

Le damos **permisos de ejecución** y lo abrimos con **cutter** para **analizarlo**.

```
remnux@remnux:~/Desktop/Malware/practica8/_pegasus.pdf.extracted$ chmod +x decoded
remnux@remnux:~/Desktop/Malware/practica8/_pegasus.pdf.extracted$ cutter decoded
"0.4.0" "0.4.0"
Setting PYTHONHOME = "/tmp/.mount_cutter6tyt7x/usr" for AppImage.
PYTHONHOME = "/tmp/.mount_cutter6tyt7x/usr"
Plugins are loaded from "/home/remnux/.local/share/rizin/cutter/plugins"
Native plugins are loaded from "/home/remnux/.local/share/rizin/cutter/plugins/native"
```

Decompilamos el main y analizamos la primera función que encontramos:



```
main
Offset: 0x00005636
Size: 0x12f8
Import: false
Nargs: 0x2
Nbbs: 0x5d
Nlocals: 0x23
Call type: amd64
Edges: 122
StackFrame: 4392
Comment:
  sym.imp.EVP_sha256
  sym.imp.HMAC_CTX_free
  sym.imp.HMAC_CTX_new
  sym.imp.EVP_sha256

int64_t var_1048h;
void *ptr;
long int var_1038h;
void *var_1030h;
char *var_1020h;
int64_t var_18h;

var_18h = *(int64_t *) (in_FS_OFFSET + 0x28);
var_10d8h = 0;
var_10d0h = 0;
var_10c8h = 0;
var_10b8h = ".SEALFS.LOG";
var_10b0h = 0;
var_10a0h = 0;
fcn.00002727();
if (argc < 3) {
  // WARNING: Subroutine does not return
  fcn.000054d1();
}
```

Esta función a su vez **llama** a la función **fcn.00002709**:

```
void fcn.00002727(void)
{
  int32_t iVar1;
  int64_t in_FS_OFFSET;
  int64_t var_28h;
  void *var_20h;
  int64_t var_18h;
  int64_t var_10h;
  int64_t canary;

  canary = *(int64_t *) (in_FS_OFFSET + 0x28);
  iVar1 = fcn.00002709();
  if (iVar1 == 0) {
    var_20h = (void *) malloc(0x400);
    uncompress(var_20h, &var_28h, 0x8020, 300);
    var_18h = (int64_t) var_20h;
    var_10h = (uint64_t) var_20h & 0xffffffffffff000;
    mprotect(var_10h, 0x1000, 7);
    (*(code *) var_18h)();
  }
  if (canary != *(int64_t *) (in_FS_OFFSET + 0x28)) {
    // WARNING: Subroutine does not return
    __stack_chk_fail();
  }
  return;
}
```

La función `fcn.00002709` **comprueba** si el binario se está **ejecutando** en una **máquina virtual o no** (no actuará de la misma forma):

```
bool fcn.00002709(void)
{
    int32_t iVar1;

    iVar1 = system("grep -qs vboxvideo /proc/modules");
    return iVar1 == 0;
}
```

Como lo **ejecutamos** en una **máquina virtual**, devolvería 1, por lo tanto, **no entraría en if** (`iVar1 == 0`).

Dentro de este **if** se **descomprimen 300 bytes** comprimidos en `var_20h`, se guarda la **dirección de memoria** en `var_18h` y se llama a la **función** almacenada en esa dirección de memoria con **"code"**.

Abrimos el ELF con el **modo depuración** de **radare2** (-d) y analizamos la función que nos interesa:

```
remnux@remnux:~/Desktop/Malware/practica8/_pegasus.pdf.extracted$ r2 -d decoded
-- Learn pancake as if you were radare!
[0x7f75087dc100]> db main
[0x7f75087dc100]> dc
hit breakpoint at: 0x560bbf1a3636
[0x560bbf1a3636]> pd 30
;-- main:
;-- rax:
;-- rip:
0x560bbf1a3636 b f30f1efa endbr64
0x560bbf1a363a 55 pushq %rbp
0x560bbf1a363b 4889e5 movq %rsp, %rbp
0x560bbf1a363e 53 pushq %rbx
0x560bbf1a363f 4881ec001000 subq $0x1000, %rsp
0x560bbf1a3646 48830c2400 orq $0, (%rsp)
0x560bbf1a364b 4881ec080100 subq $0x108, %rsp
0x560bbf1a3652 89bdfceeffff movl %edi, -0x1104(%rbp)
0x560bbf1a3658 4889b5f0eeff movq %rsi, -0x1110(%rbp)
0x560bbf1a365f 64488b042528 movq %fs:0x28, %rax
0x560bbf1a3668 488945e8 movq %rax, -0x18(%rbp)
0x560bbf1a366c 31c0 xorl %eax, %eax
0x560bbf1a366e c7851cefffff movl $1, -0x10e4(%rbp)
0x560bbf1a3678 48c78528efff movq $0, -0x10d8(%rbp)
0x560bbf1a3683 48c78530efff movq $0, -0x10d0(%rbp)
0x560bbf1a368e 48c78538efff movq $0, -0x10c8(%rbp)
0x560bbf1a3699 488d05c72a00 leaq str..SEALFS.LOG, %rax ; 0x560bbf1a6167 ; ".SEALFS.LOG"
0x560bbf1a36a0 48898548efff movq %rax, -0x10b8(%rbp)
0x560bbf1a36a7 48c78550efff movq $0, -0x10b0(%rbp)
0x560bbf1a36b2 e870d0ffff callq 0x560bbf1a0727
0x560bbf1a36b7 83bdfceeffff cmpl $2, -0x1104(%rbp)
0x560bbf1a36ba 7f05 jg 0x560bbf1a36c5
```

Saltamos a ella y localizamos la **comprobación del valor de eax**:

```
[0x560bbf1a3636]> db 0x560bbf1a0727
[0x560bbf1a3636]> dc
hit breakpoint at: 0x560bbf1a0727
[0x560bbf1a0727]> pd 30
;-- rip:
0x560bbf1a0727 b f30f1efa endbr64
0x560bbf1a072b 55 pushq %rbp
0x560bbf1a072c 4889e5 movq %rsp, %rbp
0x560bbf1a072f 4883ec30 subq $0x30, %rsp
0x560bbf1a0733 64488b042528 movq %fs:0x28, %rax
0x560bbf1a073c 488945f8 movq %rax, -8(%rbp)
0x560bbf1a0740 31c0 xorl %eax, %eax
0x560bbf1a0742 e8c2ffff callq 0x560bbf1a0709
0x560bbf1a0747 85c0 testl %eax, %eax
0x560bbf1a0749 7561 jne 0x560bbf1a07ac
0x560bbf1a074b bf00040000 movl $0x400, %edi ; 1024
0x560bbf1a0750 c9bfc0ffff callq sym.imp.callq
```

Ahora debemos **modificar** (dr) el **valor de eax** para que el **resultado** de test sea **desfavorable** y parezca que **no estamos en una máquina virtual**:

```
[0x560bbf1a0727]> db 0x560bbf1a0747
[0x560bbf1a0727]> dc
(40707) Created process 40713
[0x7f75083fa125]> dc
[+] SIGNAL 17 errno=0 addr=0x3e800009f09 code=1 si_pid=40713 ret=0
[+] signal 17 aka SIGCHLD received 0 (Child)
[0x7f750831e30b]> dc
hit breakpoint at: 0x560bbf1a0747
[0x560bbf1a0747]> dr eax
0x00000001
[0x560bbf1a0747]> dr eax=0
0x00000001 ->0x00000000
```

Podemos ver que entre medias **se crea un proceso de una shell**, esta es la que se utiliza para hacer el **grep** y **comprobar si es una máquina virtual o no**.

Vemos el **valor de var18_h** almacenado en rax:

```
0x560bbf1a0779 488945e8 movq %rax, -0x18(%rbp)
0x560bbf1a077d 488b45e8 movq -0x18(%rbp), %rax
0x560bbf1a0781 482500f0ffff andq $0xffffffffffff000, %rax
0x560bbf1a0787 488945f0 movq %rax, -0x10(%rbp)
0x560bbf1a078b 488b45f0 movq -0x10(%rbp), %rax
0x560bbf1a078f ba07000000 movl $7, %edx

[0x560bbf1a0747]> db 0x560bbf1a0781
[0x560bbf1a0747]> dc
hit breakpoint at: 0x560bbf1a0781
[0x560bbf1a0781]> dr rax
0x560bbffc72a0
```

Ponemos un **breakpoint** en la llamada a esta función:

```
0x560bbf1a07a1 488b55e8 movq -0x18(%rbp), %rdx
0x560bbf1a07a5 b800000000 movl $0, %eax
0x560bbf1a07aa ffd2 callq *%rdx
0x560bbf1a07ac 90 nop
0x560bbf1a07ad 488b45f8 movq -8(%rbp), %rax
0x560bbf1a07b1 644833042528 xorq %fs:0x28, %rax
0x560bbf1a07ba 7405 je 0x560bbf1a07c1

[0x560bbf1a0727]> db 0x560bbf1a07aa
[0x560bbf1a0727]> dc
hit breakpoint at: 0x560bbf1a07aa
[0x560bbf1a07aa]> pd 5
;-- rip:
0x560bbf1a07aa b ffd2 callq *%rdx
0x560bbf1a07ac 90 nop
0x560bbf1a07ad 488b45f8 movq -8(%rbp), %rax
0x560bbf1a07b1 644833042528 xorq %fs:0x28, %rax
0x560bbf1a07ba 7405 je 0x560bbf1a07c1
[0x560bbf1a07aa]> ds
[0x560bbffc72a0]> pd 5
;-- rdx:
;-- rip:
0x560bbffc72a0 e89d000000 callq 0x560bbffc7342
0x560bbffc72a5 0f8497000000 je 0x560bbffc7342
0x560bbffc72ab 0f8591000000 jne 0x560bbffc7342
```

La función en la dirección de memoria **var_18h** **llama a otra función**.

Si accedemos al contenido de esta nueva función:

```
[0x560bbffc72a0]> ds
[0x560bbffc7342]> pd 30
;-- rip:
0x560bbffc7342 415f      popq %r15
0x560bbffc7344 4983c70c  addq $0xc, %r15 ; 12
0x560bbffc7348 4d89fd    movq %r15, %r13
0x560bbffc734b 41be23130701 movl $0x1071323, %r14d
0x560bbffc7351 49c7c48c0000. movq $0x8c, %r12 ; 140
0x560bbffc7358 453137    xorl %r14d, (%r15)
0x560bbffc735b 4983c704  addq $4, %r15
0x560bbffc735f 4983ec04  subq $4, %r12
0x560bbffc7363 75f3      jne 0x560bbffc7358
0x560bbffc7365 48c7c0010000. movq $1, %rax
0x560bbffc736c 48c7c7010000. movq $1, %rdi
0x560bbffc7373 498d7500  leaq (%r13), %rsi
0x560bbffc7377 48c7c2520000. movq $0x52, %rdx ; 'R' ; 82
0x560bbffc737e 0f05      syscall
0x560bbffc7380 49c7c48c0000. movq $0x8c, %r12 ; 140
0x560bbffc7387 4d89ef    movq %r13, %r15
0x560bbffc738a 453137    xorl %r14d, (%r15)
0x560bbffc738d 4983c704  addq $4, %r15
0x560bbffc7391 4983ec04  subq $4, %r12
0x560bbffc7395 75f3      jne 0x560bbffc738a
0x560bbffc7397 48c7c03c0000. movq $0x3c, %rax ; '<' ; 60
0x560bbffc739e 4831ff    xorq %rdi, %rdi
0x560bbffc73a1 0f05      syscall
```

Esta función utiliza **140 bytes (0x8c)** para el **XOR** entre %r14d y %r15, y luego sólo **imprime 82 bytes (0x52)**.

Así que probamos a modificar ese 0x52 por 0x8c saltando a la primera llamada al sistema (syscall):

```
[0x560bbffc7342]> db 0x560bbffc737e
[0x560bbffc7342]> dc
hit breakpoint at: 0x560bbffc737e
[0x560bbffc737e]> dr rdx
0x00000052
[0x560bbffc737e]> dr rdx=0x8c
0x00000052 ->0x0000008c
[0x560bbffc737e]> ds
B000M! HAS EJECUTADO EL MALWARE, ACABAS DE SOLTAR UN GUSANO PELIGROSO EN TU RED.
ESTA ES LA FLAG QUE BUSCAS: [Who watches the watchmen?]

[0x560bbffc7380]> █
```

De esta forma saldría la **flag que buscábamos** y el **mensaje que se imprime normalmente al ejecutar “la acción maliciosa”** (que en este caso no lo es) sin que se llegue a ejecutar (cortaríamos la ejecución del programa aquí).