

Rans0m - 100 pts - Reversing

La descripción del reto dice lo siguiente:

“Un virus de tipo ransomware encripto los archivos de nuestra Empresa, por suerte obtuvimos el trafico de red que creemos capturo dicho malware con el nombre svchost.exe.tar.gz.”

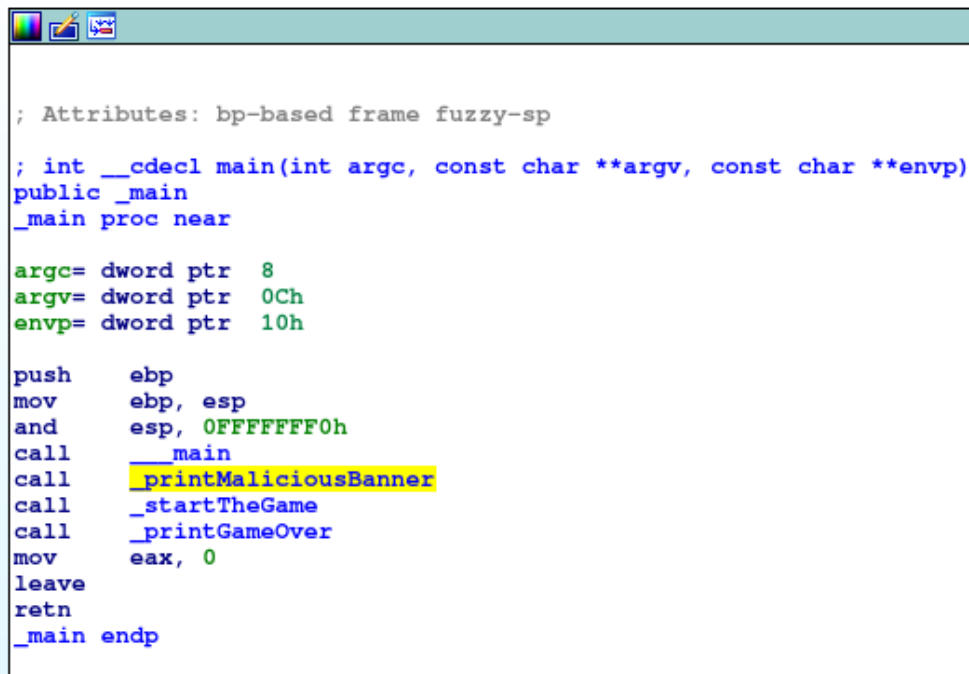
Y nos dan los siguientes archivos:

- Important_Message.txt.ransm
- Ransmutation.pcapng

Revisando el .pcapng podemos extraer el archivo svchost.exe.tar.gz que fue enviado mediante FTP. Despues de extraer el ejecutable svchost.exe podemos notar mediante el comando file que se trata de un binario para Windows PERO comprimido con UPX.

```
./svchost.exe: PE32 executable (console) Intel 80386, for MS Windows, UPX compressed
```

Una vez que conseguimos descomprimir el binario ya podiamos ver el ensamblado.

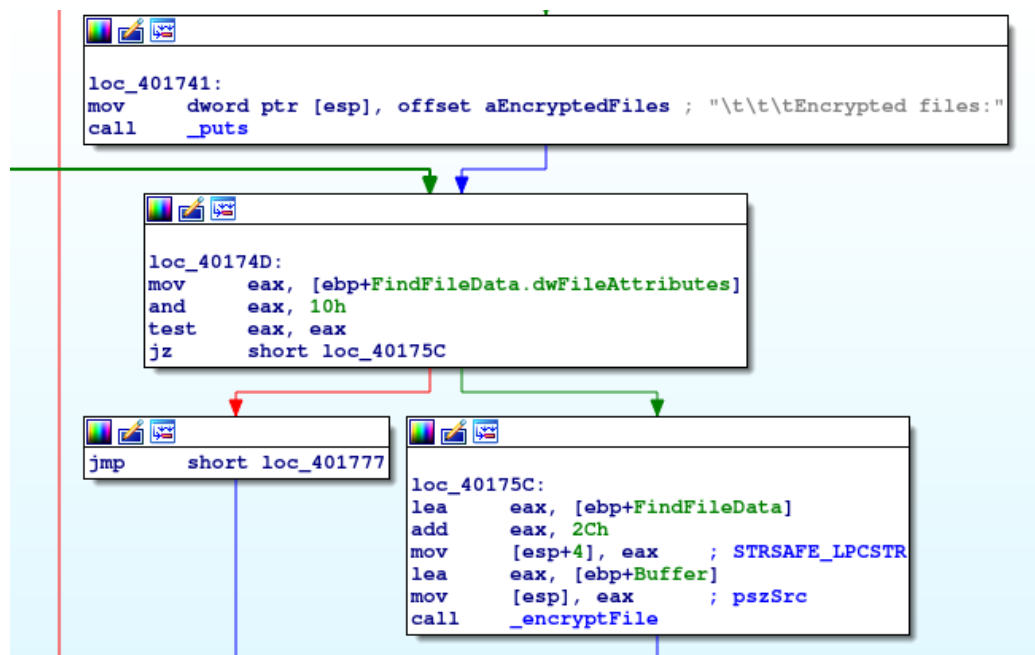


```
; Attributes: bp-based frame fuzzy-sp
; int __cdecl main(int argc, const char **argv, const char **envp)
public _main
_main proc near

    argc= dword ptr 8
    argv= dword ptr 0Ch
    envp= dword ptr 10h

    push    ebp
    mov     ebp, esp
    and     esp, 0FFFFFFF0h
    call    __main
    call    printMaliciousBanner
    call    _startTheGame
    call    _printGameOver
    mov     eax, 0
    leave
    retn
_main endp
```

Incluso podemos ver la sección de código encargada de cifrar el archivo.



Aquí es donde está agregando la extensión ".ransm". Por lo que sabemos que el archivo originalmente se llamaba Important_Message.txt

```

mov     dword ptr [esp+4], 104h ; cchDest
lea     eax, [ebp+FileName]
mov     [esp], eax             ; pszDest
call    _StringCchCatA@12
sub     esp, 0Ch
mov     eax, [ebp+arg_4]
mov     [esp+8], eax           ; pszSrc
mov     dword ptr [esp+4], 104h ; cchDest
lea     eax, [ebp+FileName]
mov     [esp], eax             ; pszDest
call    _StringCchCatA@12
sub     esp, 0Ch
mov     dword ptr [esp+8], offset aRansm ; ".ransm"
mov     dword ptr [esp+4], 104h ; cchDest
lea     eax, [ebp+FileName]
mov     [esp], eax             ; pszDest
call    _StringCchCatA@12
sub     esp, 0Ch
mov     dword ptr [esp+18h], 0 ; hTemplateFile
mov     dword ptr [esp+14h], 80h ; dwFlagsAndAttributes
mov     dword ptr [esp+10h], 2 ; dwCreationDisposition
mov     dword ptr [esp+0Ch], 0 ; lpSecurityAttributes
mov     dword ptr [esp+8], 0 ; dwShareMode
mov     dword ptr [esp+4], 40000000h ; dwDesiredAccess
lea     eax, [ebp+FileName]
mov     [esp], eax             ; lpFileName

```

Pero lo que nos importa es el siguiente pedazo de código ya que es el encargado de tomar los bytes del archivo y los convierte hasta obtener los bytes del archivo cifrado.

```

LOC_401A4/:
lea     edx, [ebp+Buffer]
mov     eax, [ebp+i]
add     eax, edx
movzx   ebx, byte ptr [eax]
mov     ecx, [ebp+i]
mov     edx, 92492493h
mov     eax, ecx
imul    edx
lea     eax, [edx+ecx]
sar     eax, 4
mov     edx, eax
mov     eax, ecx
sar     eax, 1Fh
sub     edx, eax
mov     eax, edx
mov     edx, eax
lea     eax, ds:0[edx*4]
mov     edx, eax
lea     eax, ds:0[edx*8]
sub     eax, edx
sub     ecx, eax
mov     eax, ecx
movzx   eax, byte ptr [ebp+eax+var_F44AC]
xor     ebx, eax
mov     ecx, ebx
mov     edx, [ebp+lpBuffer]
mov     eax, [ebp+i]
add     eax, edx
mov     [eax], cl
add     [ebp+i], 1

```

Lo principal a notar es que esta tomando cada byte del archivo original y le hace xor con otro byte generado y el resultado es lo que va a reemplazar en el archivo cifrado. Una posible solución es hacer el reversing de todo el proceso que genera el byte con el que se hace el xor PERO notamos que esos

bytes son una secuencia que no depende de los bytes originales. Esto implica que siempre se usa una misma secuencia para cifrar todos los archivos. Así que nuestro siguiente paso fue cifrar el mismo programa para posteriormente realizar el xor de ambos y con eso obtener la secuencia de bytes usada para el cifrado.



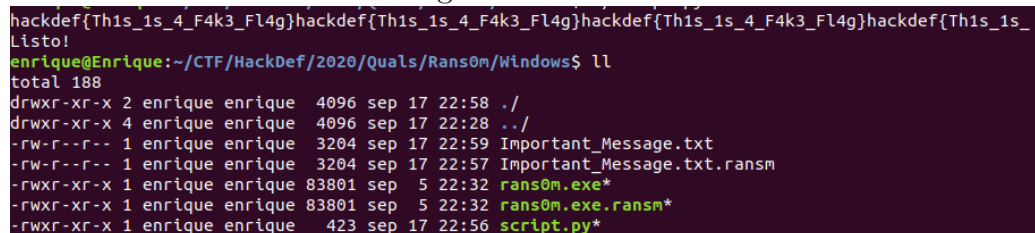
Aquí el script utilizado para obtener la secuencia de bytes y al mismo tiempo, generar Important_Message.txt

```
#!/usr/bin/python
a = open("./rans0m.exe", "rb").read()
b = open("./rans0m.exe.ransm", "rb").read()
key = ''.join([chr(ord(a[i]) ^ ord(b[i])) for i in range(len(a))])

# Primeros 100 caracteres de la secuencia
print key[:100]

c = open("Important_Message.txt.ransm", "rb").read()
N = len(c)
with open("Important_Message.txt", "wb") as f:
    for i in range(N):
        f.write(chr(ord(c[i]) ^ ord(key[i])))
print "Listo!"
```

Al ejecutar el script podemos observar la secuencia utilizada por el programa, así como también el archivo original.



Pero resulto ser una cadena en base64:

Que al intentar decodificar, notamos que se trataba de un PNG.

Así que lo último fue redirigir la salida a un archivo y abrirlo. Con lo cual obtuvimos la flag.

```
hackdef{R4nsom_L1f3_F0r3v3r_D13_B4ckup5!}
```