Victima	Printif
Protección	UPX, Serial
Herramientas	Olly, Filemon, [PE Explorer]
Objetivo	Parchear limite de 30 días

Introducción

Hola a todos,

Bueno, antes de seguir con WL hacemos un alto en el camino y atendemos una petición de mi amigo Ralba.

En este caso es un programa creado con un compilador nuevo para mí y que nos puede traer un poco de cabeza en futuras víctimas.

El objetivo que me pidió Ralba era quitar el límite de 30 días, no sé como de complejo es analizar el serial ya que me centré solo en esto.

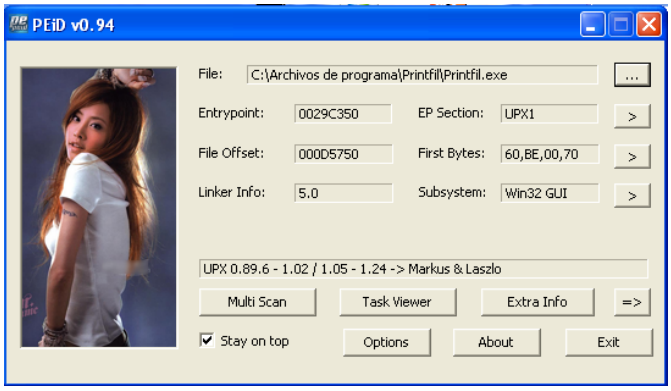
Recordar que todo lo que se muestra aquí es meramente educativo, el que vaya a utilizar el software que lo compre.

Dicho esto empecemos.

Analizando al enemigo

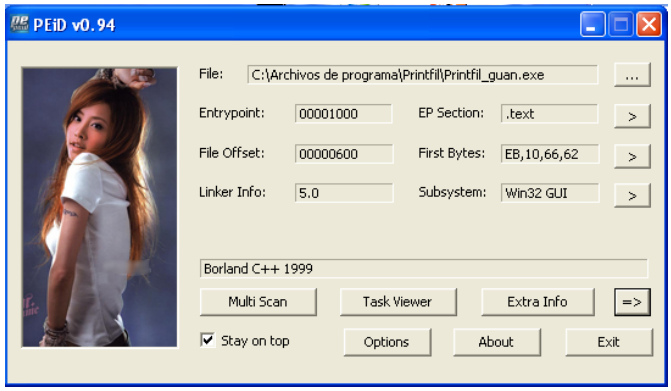
Aunque no creo que en este caso sea relevante decir que tengo el Olly protegido con el phanom.

Una vez dicho esto, si analizamos el ejecutable con el PEiD vemos que está comprimido con UPX



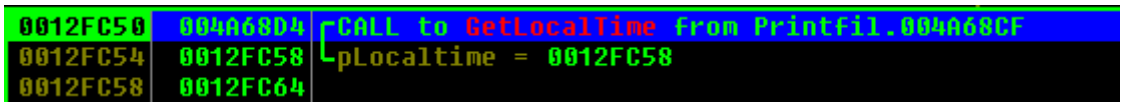
Como andamos un poco con fiaca y tenemos el PE Explorer a mano simplemente lo analizamos y guardamos por ejemplo con Printfil_guan.exe

Si lo analizamos ahora vemos

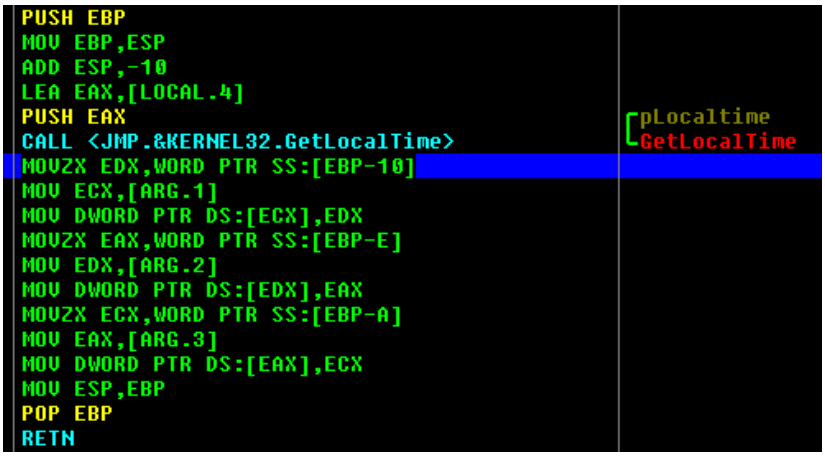


Si ejecutamos el programa nos aparece un splash screen indicando que tenemos 30 días

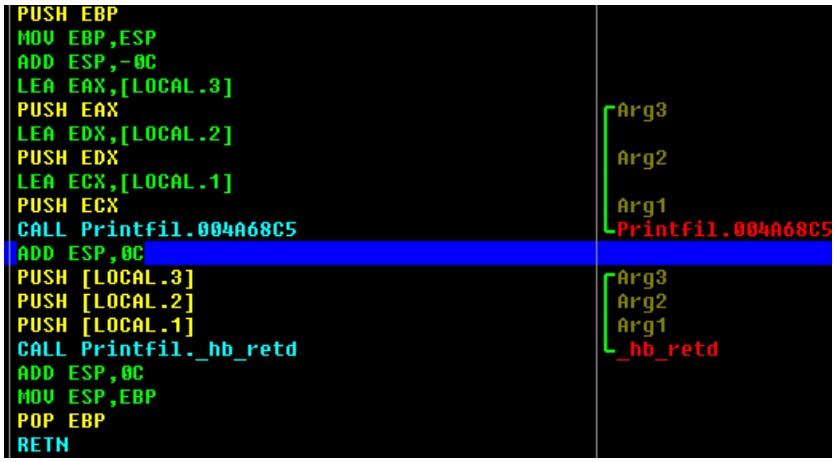
Como sabemos que es por tiempo pues pongamos los bps en las típicas APIs (GetLocalTime y GetSystemTime), veamos si para y en donde.



La primera vez nos para y es llamado desde el programa veamos donde estamos.



Bueno aquí toma los datos y los guarda en una estructura en memoria. Salimos de esta rutina y estamos:



Pasa los datos guardados y llama a la función _hb_retd. Como cosa que llama la atención es que parece que este programa tiene los símbolos de parte del programa y por eso vemos los nombres de ciertas funciones.

```
.   TEST BYTE PTR DS:[EBX+5],1
.v  JE SHORT Printfil.0046DD59
.   PUSH DWORD PTR DS:[EAX+4]
.   PUSH DWORD PTR DS:[EAX]
.   CALL Printfil._hb_vmExecute
.   ADD ESP,8
.v  JMP SHORT Printfil.0046DDC6
>   CALL EAX
.v  JMP SHORT Printfil.0046DDC6
>   PUSH -1                          Arg5 = FFFFFFFF
.   PUSH DWORD PTR DS:[EBX]          Arg4
.   PUSH 0                           Arg3 = 00000000
.   PUSH 3E9                         Arg2 = 000003E9
.   PUSH 0C                          Arg1 = 0000000C
.   CALL Printfil.00459961           Printfil.00459961
```

Llegados a este punto ya me hizo plantearme las cosas de otra forma. Ese nombre de función de _hb_vmExecute y por lo que vi traceando me indica que estamos dentro de una máquina virtual.

Intentar seguir los datos tomados de la API GetLocalTime también es complicado, más cuando esta API es llamada desde varias partes ante de arrancar el programa y se hace complicado saber que llamada es la que realmente realiza la comparación.

Para intentar reducir el tema adelante el año del reloj, con lo que el programa ya no arranca apareciendo el siguiente cartel

El resultado fue parecido, se llama como 4 o 5 veces a la API GetLocalTime antes de sacar la Nag.

Mirando por las referencias a cadenas podemos ver algo como esto:

```
005F29B8 ASCII "xHarbour build %d.%d.%d Intl. (%s) (Rev. %d)"
```

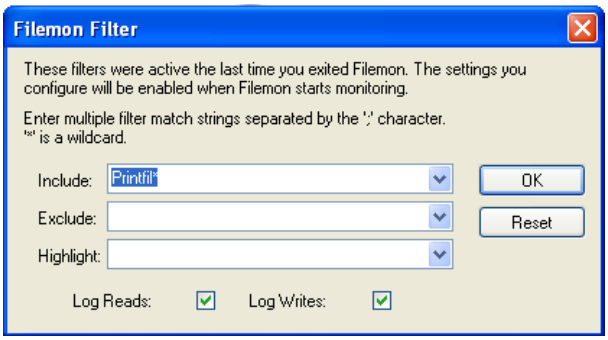
Esto me hizo buscar un poco en la red y encontré esto http://www.xharbour.org/

Al parecer el programa está hecho en xHarbour, y me imagino que esta Virtual Machine pues pertenece al núcleo del compilador xHarbour. Este tipo de compiladores nos complican un poco la vida a la hora de tracear el código.

Volviendo al programa, tras adelantar el reloj y reponerlo nos encontramos con que el programa no vuelve funcionar, solo nos muestra la nag, pero de alguna forma debe de saber que ya caduco y es justo aquí donde encontramos la debilidad del programa.

Los programas actuales para llevar la cuenta suelen usar el registro, pero un estudio con el ntregmon no vi nada, por lo que si no usa el registro debe de usar un fichero, más alternativas no quedan.

Abrimos el filemon y lo configuramos para que lo haga log de nuestro progama

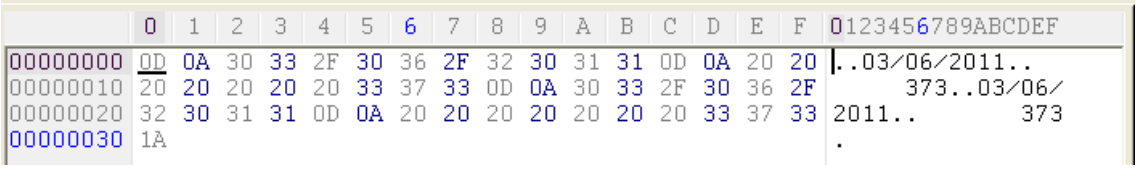


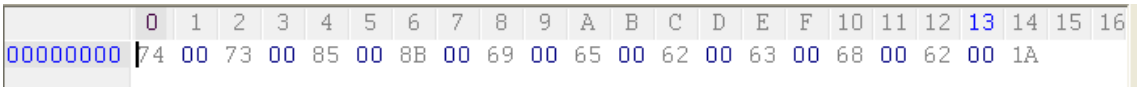
Arrancamos el programa y cuando sale la nag paramos el log y vemos que tenemos.

1779	23:47:23	Printfil.exe:304	FASTIO_QUERY_STAN...	C:\WINDOWS\161491591.dll
1780	23:47:23	Printfil.exe:304	IRP_MJ_READ	C:\WINDOWS\161491591.dll
1781	23:47:23	Printfil.exe:304	FASTIO_READ	C:\WINDOWS\161491591.dll
1782	23:47:23	Printfil.exe:304	IRP_MJ_CLEANUP	C:\WINDOWS\161491591.dll
1783	23:47:23	Printfil.exe:304	IRP_MJ_CREATE	C:\WINDOWS\161491592.dll
1784	23:47:23	Printfil.exe:304	FASTIO_QUERY_STAN...	C:\WINDOWS\161491592.dll

Tenemos un par de dll un poco sospechosas en c:\Windows, vamos a ver que son

Al parecer no son dlls, sino ficheros binarios a los que se le ha dado esa extensión para despistar.





¿Qué pasa si las borramos? Pues que el programa vuelve a funcionar como si estuviéramos en el día 1 jejeje.

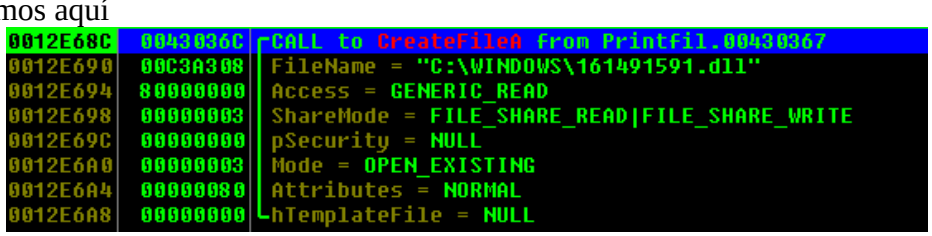
Creando el injerto

En este caso, lo más fácil puede ser crear un loader que borre las 2 dlls antes de ejecutar el programa, por lo que así saltamos la comprobación del tiempo y nunca caduca.

Hablando de esto con Ralba nos dimos cuenta de que las dlls tenían unos valores distinto en mi máquina que en la suya, así que para hacerlo un poco más genérico lo que vamos a hacer es crear un injerto que borre las dlls antes de que las vaya a abrir, el resultado es el mismo, pero como el programa es el que genera el nombre de las dlls pues así ya funcionaría para todas las máquinas.

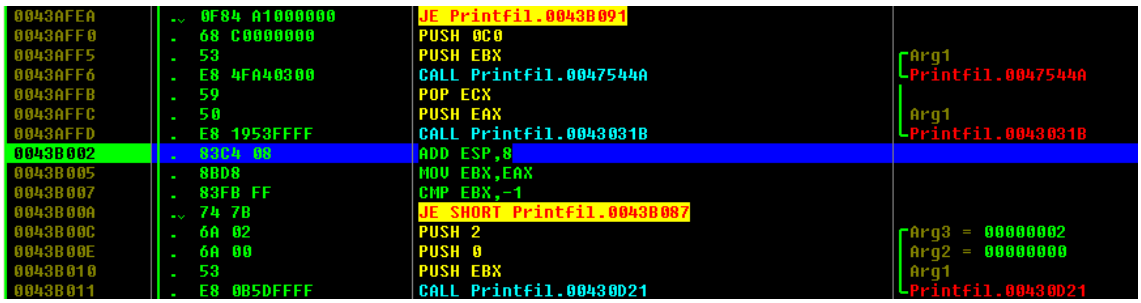
Así que volvemos a poner nuestro programa en Olly y ponemos un BP en CreateFileA, vamos dando a Run hasta que veamos que quiere abrir alguna de las 2 dlls.

Y llegamos aquí



Vemos que se llama desde el programa salimos de la API y estamos aquí

Salimos de esta función y llegamos aquí

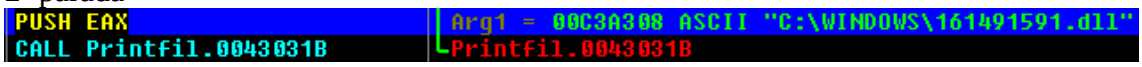


Aquí es donde se comprueba si el Handle es bueno o hubo error, vamos a considera buena esta zona para el injerto y como creo esto es de la virtual machine, vamos a poner un HE en la dirección 43AFFD (el PUSH EAX) y vamos a comprobar si en esta zona solo se abre nuestra dll falsa o pasan por aquí la apertura de todos los ficheros que use el programa.

Reiniciamos el Olly y tenemos esto:

1º parada

2º parada

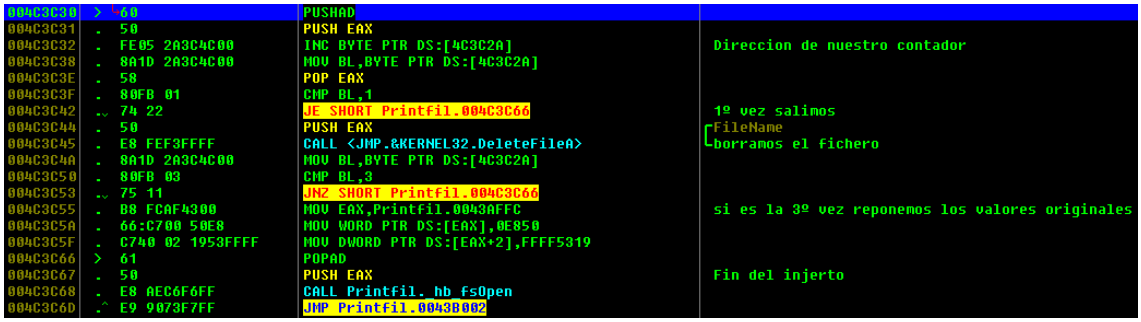


3º parada

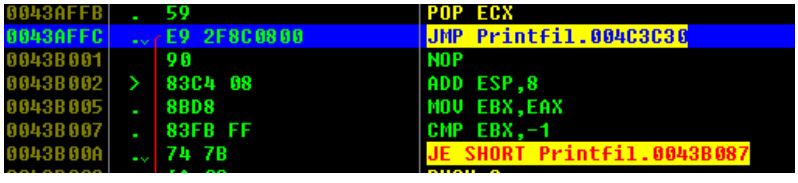
Y ya no hay más paradas salvo que mandemos a imprimir algo, por lo que sabemos que esta zona es la que se usa para abrir los ficheros, sea el que sea.

En este caso podemos hacer un injerto un poco inteligente usando un contador, de forma que la primera vez que pase por aquí no haga nada, la 2º y 3º vez borren el fichero apuntado por el registro EAX antes de mandarlo a abrir y después de borrar en el fichero la tercera vez pues simplemente se anula el injerto el solito.

Buscamos una zona de ceros y el injerto queda como sigue.

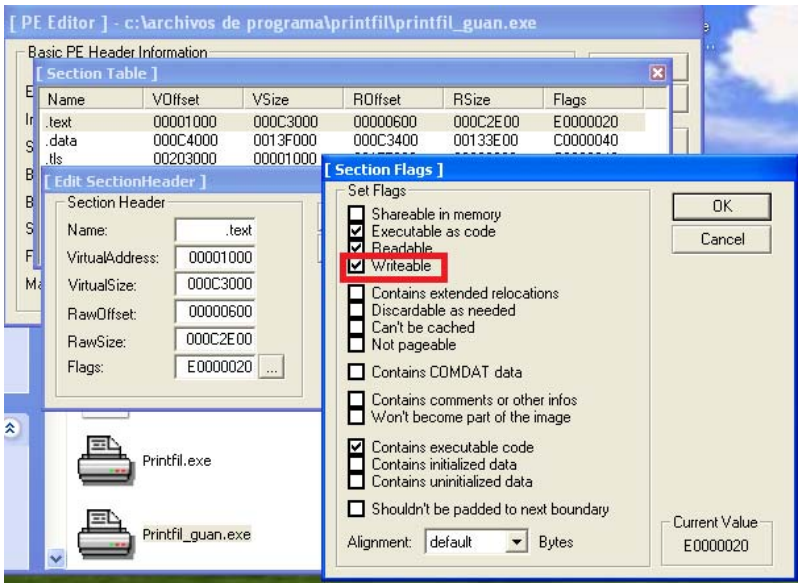


El cambio de flujo a nuestro injerto lo tenemos aquí



Para que todo esto funcione queda un detalle, y es que tenemos que poner la región de código como escritura, sino no podremos incrementar nuestro contador que lo puse en la misma sección de código.

Así que con LordPE nos vamos a secciones y cambiamos las características



Salvamos los cambios, arrancamos y todo va de maravilla.
Adelantamos el reloj lo atrasamos y todo sigue bien ☺

Agradecimientos

Especialmente a Ralba que con este mini reto me ha hecho pasar unas horas divertidas, a todo el grupo que forman CLS y especialmente a ti por haber leído este tutorial

Hasta la próxima.