Sign in

S12cybersecurity / RDPCredentialStealer
Public

Notifications   Fork 35   Star 234

Code   Issues 1   Pull requests   Actions   Projects   Security   Insights

main   Go to file   Code

- APIHookInjectorBin
- RDPCredsStealerDLL
- APIHookInjectorBin.exe
- RDPCredentialStealer....
- RDPCredsStealerDLL....
- README.md

About

RDPCredentialStealer it's a malware that steal credentials provided by users in RDP using API Hooking with Detours in C++

Readme
Activity
234 stars
8 watching
35 forks
Report repository

Releases 1

v1 Latest
on Jun 13, 2023

Packages

No packages published

Languages

● C++ 86.7%   ● C 13.3%

README

RDPCredentialStealer

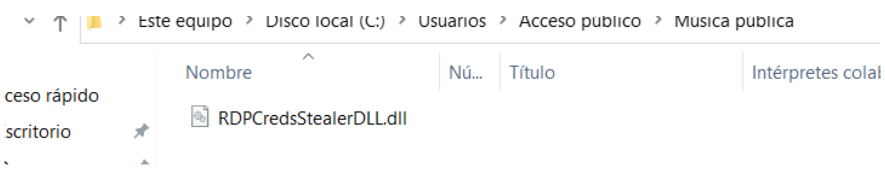
RDPCredentialStealer it's a malware that steal credentials provided by users in RDP using API Hooking with Detours in C++

Proof-of-Concept Tool for

Credential Theft

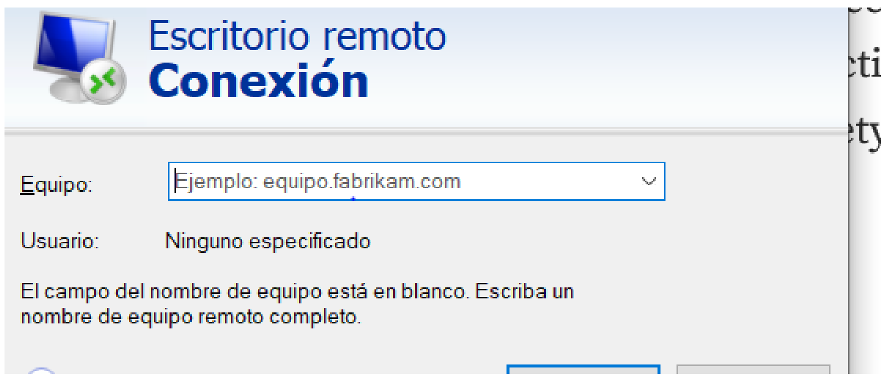
Let's try this:

First of all i transfer the DLL of the GitHub repository to the C:\Users\Public\Music path:


Este equipo › Disco local (C:) › Usuarios › Acceso público › Música pública
Nombre
Nú...
Título
Intérpretes colal
ceso rápido
scritorio

RDPCredsStealerDLL.dll

Now i open a RDP Connector Windows Application:

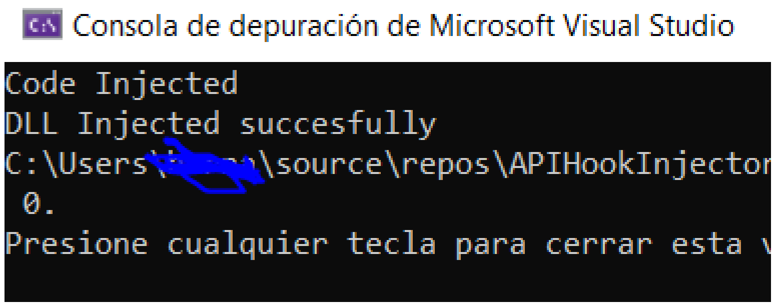


Equipo: Ejemplo: equipo.fabrikam.com

Usuario: Ninguno especificado

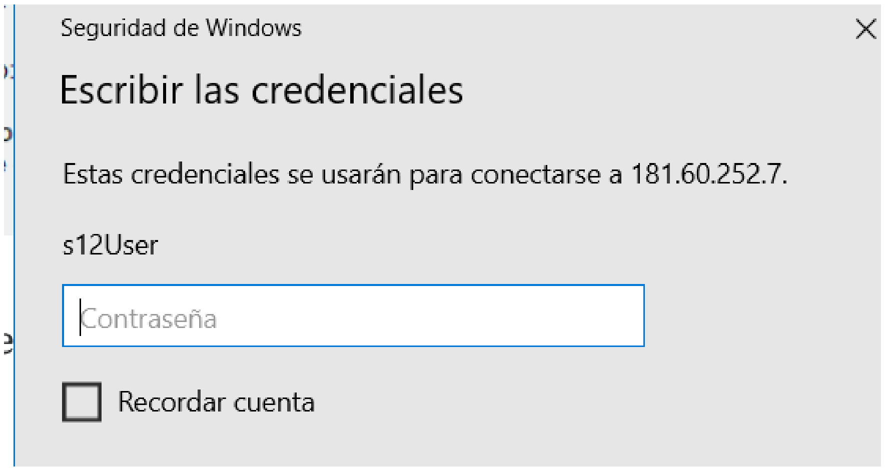
El campo del nombre de equipo está en blanco. Escriba un nombre de equipo remoto completo.

And now i execute the EXE in the GitHub repository:



```
Code Injected
DLL Injected succesfully
C:\Users\[redacted]\source\repos\APIHookInjector
 0.
Presione cualquier tecla para cerrar esta v
```

And when i try to access into a rdp server:


Seguridad de Windows

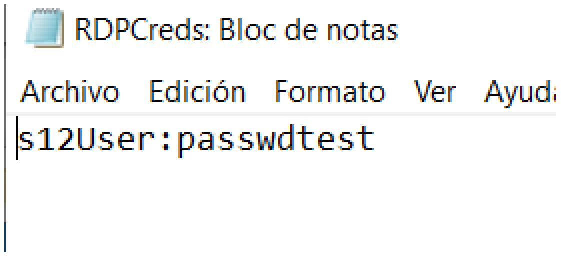
Escribir las credenciales

Estas credenciales se usarán para conectarse a 181.60.252.7.

s12User

I put the user: s12User and password: passwdtest

And now i check the file: C:\Users\Public\Music\RDPCreds.txt



```
s12User:passwdtest
```

Code

RDPCredStealerDLL:

This code is an implementation of a hooking technique in C++ using the Detours library. It specifically targets the CredUnPackAuthenticationBufferW function from the credui.dll library, which is responsible for unpacking authentication buffers used in credential operations. Let's break down the code step by step: The necessary header files are included, such as windows.h, wincred.h, detours.h, and others. A function pointer type CredUnPackAuthenticationBufferW_t is defined, representing the original CredUnPackAuthenticationBufferW function's signature. The function pointer pCredUnPackAuthenticationBufferW is declared, which will be used to store the address of the original function.

The MyCredUnPackAuthenticationBufferW function is implemented, which serves as the hook for the original function. It is called when the hooked function is invoked. This function first calls the original function using the stored function pointer pCredUnPackAuthenticationBufferW. Then, it converts the retrieved username and password from wide strings (LPWSTR) to UTF-8 encoded strings (std::string).

Finally, it opens a file in append mode and writes the username and password to it. The DllMain function serves as the entry point for the hooking DLL. It is called when the DLL is loaded or unloaded. When ul_reason_for_call is DLL_PROCESS_ATTACH, indicating that the DLL is being loaded, it loads the credui.dll library using LoadLibraryA.

Then, it obtains the address of the original function CredUnPackAuthenticationBufferW using GetProcAddress. If successful, it starts the hooking process by calling DetourTransactionBegin, DetourUpdateThread, and DetourAttach. When ul_reason_for_call is DLL_PROCESS_DETACH, indicating that the DLL is being unloaded, it reverses the hooking process by calling DetourTransactionBegin, DetourUpdateThread, and DetourDetach.

The purpose of this code is to hook the CredUnPackAuthenticationBufferW function and intercept the credentials passed to it, allowing the hooking code to extract and log the username and password to a file. This technique can be used maliciously to capture sensitive information such as RDP credentials, which is a significant security concern.

Inject.h

This code provides functions for injecting a DLL (Dynamic Link Library) into a target process on the Windows operating system. Let's break down the code and understand its functionality:

The necessary header files are included, such as windows.h, stdio.h, and tlhelp32.h. These headers provide the required functions and data types for interacting with the Windows API.

The getPIDbyProcName function takes a process name as input and returns the corresponding process ID (PID). It uses the CreateToolhelp32Snapshot function to create a snapshot of the current processes, and then iterates through the snapshot using Process32FirstW and Process32NextW functions to find the process with a matching name. If found, it returns the process ID; otherwise, it returns 0.

The DLLinjector function takes a process ID (pid) and a DLL path as input. It injects the specified DLL into the target process. Here's how it works: a. It opens the target process using OpenProcess with the PROCESS_ALL_ACCESS flag. b. It retrieves the handle of the Kernel32 module using GetModuleHandleW. c. It obtains the address of the LoadLibraryW function within Kernel32 using GetProcAddress. d. It allocates memory in the target process using VirtualAllocEx. e. It writes the DLL path to the allocated memory in the target process using WriteProcessMemory. f. It creates a remote thread in the target process using CreateRemoteThread and passes the address of LoadLibraryW and the allocated memory as parameters. g. If the thread creation is successful, it returns true indicating that the DLL injection was successful.
The main function is not provided in this code snippet. You can use these functions within your own application to inject a DLL into a target process by providing the process name and DLL