

214 lines (175 loc) · 4.95 KB

CodeBlame

RawCopyDownload

```
1  #include <Windows.h>
2  #include <tchar.h>
3  #include <Psapi.h>
4
5  #define DLLEXPORT __declspec(dllexport)
6  #define AmsiProviderName TEXT("FakeAmsiProvider")
7  #define USERNAME_LENGTH 512
8  #define DOMAINNAME_LENGTH 512
9
10 GUID guid_GTAMsiProvider =
11 {
12     0x00000000, 0xDEAD, 0xDEAD, {0xDE, 0xAD, 0xb2, 0xb2, 0xe0, 0x85, 0x90, 0x59}
13 };
14
15 HMODULE g_currentModule;
16
17 BOOL GetProcessUsername(HANDLE hProcess, LPTSTR lpUserName)
18 {
19
20     HANDLE hToken = NULL;
21     PTOKEN_USER ptuTokenInformation = NULL;
22     DWORD dwTokenLength;
23     DWORD dwUserNameLen = USERNAME_LENGTH;
24     DWORD dwDomainNameLen = DOMAINNAME_LENGTH;
25     TCHAR szUserName[USERNAME_LENGTH];
26     TCHAR szDomainName[DOMAINNAME_LENGTH];
```

```
27     SID_NAME_USE snuSidUse;
28     TCHAR strNameBuf[USERNAME_LENGTH + 1 + DOMAINNAME_LENGTH] = { 0 };
29
30     if (!OpenProcessToken(hProcess, TOKEN_QUERY, &hToken))
31     {
32         lpUserName = TEXT("UNKNOWN");
33         return FALSE;
34     }
35
36     GetTokenInformation(hToken, TokenUser, NULL, 0, &dwTokenLength);
37     ptuTokenInformation = (PTOKEN_USER)LocalAlloc(LPTR, dwTokenLength);
38     if (NULL == ptuTokenInformation)
39     {
40         CloseHandle(hToken);
41         lpUserName = TEXT("UNKNOWN");
42         return FALSE;
43     }
44
45     if (!GetTokenInformation(hToken, TokenUser, ptuTokenInformation, dwTokenLength, &dwTokenLength))
46     {
47         CloseHandle(hToken);
48         LocalFree(ptuTokenInformation);
49         lpUserName = TEXT("UNKNOWN");
50         return FALSE;
51     }
52
53     if (!LookupAccountSid(NULL, ptuTokenInformation->User.Sid, szUserName, &dwUserNameLen, szDomainName, &dwDomainNameLen))
54     {
55         CloseHandle(hToken);
56         LocalFree(ptuTokenInformation);
57         lpUserName = TEXT("UNKNOWN");
58         return FALSE;
59     }
60
61     _stprintf_s(strNameBuf, _countof(strNameBuf), TEXT("%s\\%s"), szDomainName, szUserName);
62     _tcscpy_s(lpUserName, _countof(strNameBuf), strNameBuf);
63
64     LocalFree(ptuTokenInformation);
65     CloseHandle(hToken);
66     return TRUE;
67 }
68
69
70 DLLEXPORT
71 STDAPI
72DllRegisterServer()
```

```
73     {
74         TCHAR modulePath[MAX_PATH];
75         int dwRet;
76         LSTATUS lStatus;
77         TCHAR keyPath[200];
78
79         if (GetModuleFileName(g_currentModule, modulePath, _countof(modulePath)) >= _countof(modulePath))
80         {
81             return HRESULT_FROM_WIN32(GetLastError());
82         }
83
84         wchar_t clsidwString[40]; //always wchar
85         if (0 == StringFromGUID2(&guid_GTAMsiProvider, clsidwString, _countof(clsidwString)))
86         {
87             return E_UNEXPECTED;
88         }
89
90         dwRet = _stprintf_s(keyPath, _countof(keyPath), TEXT("Software\\Classes\\CLSID\\%ls"), clsidwString);
91         if (-1 == dwRet)
92         {
93             return E_INVALIDARG;
94         }
95
96         lStatus = RegSetKeyValue(HKEY_LOCAL_MACHINE, keyPath, NULL, REG_SZ, AmsiProviderName, (DWORD)strlen(AmsiProviderName));
97         if (ERROR_SUCCESS != lStatus)
98         {
99             return lStatus;
100         }
101
102         dwRet = _stprintf_s(keyPath, _countof(keyPath), L"Software\\Classes\\CLSID\\%ls\\InProcServer32", clsidwString);
103         if (-1 == dwRet)
104         {
105             return E_INVALIDARG;
106         }
107
108         lStatus = RegSetKeyValue(HKEY_LOCAL_MACHINE, keyPath, NULL, REG_SZ, modulePath, (DWORD)strlen(modulePath));
109         if (ERROR_SUCCESS != lStatus)
110         {
111             return lStatus;
112         }
113
114
115         lStatus = RegSetKeyValue(HKEY_LOCAL_MACHINE, keyPath, TEXT("ThreadingModel"), REG_SZ, TEXT("Apartment"));
116         if (ERROR_SUCCESS != lStatus)
117         {
118             return lStatus;
```

```
140         return lStatus;

141     {
142         wchar_t clsidwString[40]; //always wchar
143         TCHAR keyPath[200];
144         int dwRet;
145         LSTATUS lStatus;

146         if (0 == StringFromGUID2(&guid_GTAmsiProvider, clsidwString, _countof(clsidwString)))
147         {
148             return E_UNEXPECTED;
149         }

150         dwRet = _stprintf_s(keyPath, _countof(keyPath), L"Software\\Microsoft\\AMSI\\Providers\\%ls", clsidwString);
151         if (-1 == dwRet)
152         {
153             return E_INVALIDARG;
154         }

155         lStatus = RegDeleteTree(HKEY_LOCAL_MACHINE, keyPath);
156         if (lStatus != NO_ERROR && lStatus != ERROR_PATH_NOT_FOUND)
157         {
158             return lStatus;
159         }

160         return lStatus;
161     }
162 }
163
164 // ...
```

```
164         dwRet = _stprintf_s(keyPath, _countof(keyPath), L"Software\\Classes\\CLSID\\%ls", clsidwstr);
165         if (-1 == dwRet)
166         {
167             return E_INVALIDARG;
168         }
169
170         lStatus = RegDeleteTree(HKEY_LOCAL_MACHINE, keyPath);
171         if (lStatus != NO_ERROR && lStatus != ERROR_PATH_NOT_FOUND)
172         {
173             return lStatus;
174         }
175
176         return S_OK;
177     }
178
179
180     BOOL WINAPI DllMain(HMODULE hModule,
181                        DWORD dwReason,
182                        LPVOID lpReserved
183     )
184     {
185         TCHAR strMsg[1024] = {0};
186         TCHAR szFilePath[MAX_PATH] = {0};
187         TCHAR szUserName[USERNAME_LENGTH + 1 + DOMAINNAME_LENGTH];
188
189         g_currentModule = hModule;
190
191         switch (dwReason)
192         {
193             case DLL_PROCESS_ATTACH:
194                 GetProcessImageFileName(GetCurrentProcess(), szFilePath, MAX_PATH);
195                 GetProcessUsername(GetCurrentProcess(), szUserName);
196                 _stprintf_s(strMsg, _countof(strMsg), TEXT("[GTamsiProvider] %hs says: USERNAME = %s"), szFilePath, szUserName);
197                 break;
198             case DLL_THREAD_ATTACH:
199                 break;
200             case DLL_THREAD_DETACH:
201                 break;
202             case DLL_PROCESS_DETACH:
203                 break;
204             default:
205                 break;
206         }
207
208         if (_tcslen(strMsg) > 0)
209         {
```

```
210             OutputDebugString(strMsg);
211         }
212
213         return TRUE;
214     }
```