PSBits / FakeAMSI / FakeAMSI.c

gtworek Create FakeAMSI.c 967b457 · 3 years ago History

Code Blame 214 lines (175 loc) · 4.95 KB Raw

```c
#include <Windows.h>
#include <tchar.h>
#include <Psapi.h>

#define DLLEXPORT __declspec(dllexport)
#define AmsiProviderName TEXT("FakeAmsiProvider")
#define USERNAME_LENGTH 512
#define DOMAINNAME_LENGTH 512

GUID guid_GTAmsiProvider =
{
	0x00000000, 0xDEAD, 0xDEAD, {0xDE, 0xAD, 0xb2, 0xb2, 0xe0, 0x85, 0x90, 0x59}
};

HMODULE g_currentModule;

BOOL GetProcessUsername(HANDLE hProcess, LPTSTR lpUserName)
{

	HANDLE hToken = NULL;
	PTOKEN_USER ptuTokenInformation = NULL;
	DWORD dwTokenLength;
	DWORD dwUserNameLen = USERNAME_LENGTH;
	DWORD dwDomainNameLen = DOMAINNAME_LENGTH;
	TCHAR szUserName[USERNAME_LENGTH];
	TCHAR szDomainName[DOMAINNAME_LENGTH];
	SID_NAME_USE snuSidUse;
	TCHAR strNameBuf[USERNAME_LENGTH + 1 + DOMAINNAME_LENGTH] = { 0 };

	if (!OpenProcessToken(hProcess, TOKEN_QUERY, &hToken))
	{
		lpUserName = TEXT("UNKNOWN");
		return FALSE;
	}

	GetTokenInformation(hToken, TokenUser, NULL, 0, &dwTokenLength);
	ptuTokenInformation = (PTOKEN_USER)LocalAlloc(LPTR, dwTokenLength);
	if (NULL == ptuTokenInformation)
	{
		CloseHandle(hToken);
		lpUserName = TEXT("UNKNOWN");
		return FALSE;
	}

	if (!GetTokenInformation(hToken, TokenUser, ptuTokenInformation, dwTokenLength,
	{
		CloseHandle(hToken);
		LocalFree(ptuTokenInformation);
		lpUserName = TEXT("UNKNOWN");
		return FALSE;
	}

	if (!LookupAccountSid(NULL, ptuTokenInformation->User.Sid, szUserName, &dwUserN
	{
		CloseHandle(hToken);
		LocalFree(ptuTokenInformation);
```

- Services
- ShutdownPriority
- SuspendProcess
- Sysmon
- TarpitFS
- TarpitIIS

```
                        lpUserName = TEXT("UNKNOWN");
                        return FALSE;
                }

                _stprintf_s(strNameBuf, _countof(strNameBuf), TEXT("%s\\%s"), szDomainName, szU
                _tcscpy_s(lpUserName, _countof(strNameBuf), strNameBuf);

                LocalFree(ptuTokenInformation);
                CloseHandle(hToken);
                return TRUE;
        }


        DLLEXPORT
        STDAPI
        DllRegisterServer()
        {
                TCHAR modulePath[MAX_PATH];
                int dwRet;
                LSTATUS lStatus;
                TCHAR keyPath[200];

                if (GetModuleFileName(g_currentModule, modulePath, _countof(modulePath)) >= _co
                {
                        return HRESULT_FROM_WIN32(GetLastError());
                }

                wchar_t clsidwString[40];  //always wchar
                if (0 == StringFromGUID2(&guid_GTAmsiProvider, clsidwString, _countof(clsidwStr
                {
                        return E_UNEXPECTED;
                }

                dwRet = _stprintf_s(keyPath, _countof(keyPath), TEXT("Software\\Classes\\CLSID\
                if (-1 == dwRet)
                {
                        return E_INVALIDARG;
                }

                lStatus = RegSetKeyValue(HKEY_LOCAL_MACHINE, keyPath, NULL, REG_SZ, AmsiProvide
                if (ERROR_SUCCESS != lStatus)
                {
                        return lStatus;
                }

                dwRet = _stprintf_s(keyPath, _countof(keyPath), L"Software\\Classes\\CLSID\\%ls
                if (-1 == dwRet)
                {
                        return E_INVALIDARG;
                }

                lStatus = RegSetKeyValue(HKEY_LOCAL_MACHINE, keyPath, NULL, REG_SZ, modulePath,
                if (ERROR_SUCCESS != lStatus)
                {
                        return lStatus;
                }


                lStatus = RegSetKeyValue(HKEY_LOCAL_MACHINE, keyPath, TEXT("ThreadingModel"), R
                if (ERROR_SUCCESS != lStatus)
                {
                        return lStatus;
```

```
{
        wchar_t clsidwString[40]; //always wchar
        TCHAR keyPath[200];
        int dwRet;
        LSTATUS lStatus;

        if (0 == StringFromGUID2(&guid_GTAmsiProvider, clsidwString, _countof(clsidwStr
        {
                return E_UNEXPECTED;
        }

        dwRet = _stprintf_s(keyPath, _countof(keyPath), L"Software\\Microsoft\\AMSI\\Pr
        if (-1 == dwRet)
        {
                return E_INVALIDARG;
        }

        lStatus = RegDeleteTree(HKEY_LOCAL_MACHINE, keyPath);
        if (lStatus != NO_ERROR && lStatus != ERROR_PATH_NOT_FOUND)
        {
                return lStatus;
        }

        dwRet = _stprintf_s(keyPath, _countof(keyPath), L"Software\\Classes\\CLSID\\%ls
        if (-1 == dwRet)
        {
                return E_INVALIDARG;
        }

        lStatus = RegDeleteTree(HKEY_LOCAL_MACHINE, keyPath);
        if (lStatus != NO_ERROR && lStatus != ERROR_PATH_NOT_FOUND)
        {
                return lStatus;
        }

        return S_OK;
}


BOOL APIENTRY DllMain(HMODULE hModule,
                      DWORD dwReason,
                      LPVOID lpReserved
)
{
        TCHAR strMsg[1024] = {0};
        TCHAR szFilePath[MAX_PATH] = {0};
        TCHAR szUserName[USERNAME_LENGTH + 1 + DOMAINNAME_LENGTH];

        g_currentModule = hModule;

        switch (dwReason)
        {
        case DLL_PROCESS_ATTACH:
                GetProcessImageFileName(GetCurrentProcess(), szFilePath, MAX_PATH);
                GetProcessUsername(GetCurrentProcess(), szUserName);
                _stprintf_s(strMsg, _countof(strMsg), TEXT("[GTAmsiProvider] %hs says:
                break;
        case DLL_THREAD_ATTACH:
                break;
        case DLL_THREAD_DETACH:
                break;
        case DLL_PROCESS_DETACH:
                break;
        default:
                break;
        }
```

```
	if (_tcslen(strMsg) > 0)
	{
		OutputDebugString(strMsg);
	}

	return TRUE;
}
```