Manually Enumerating Process Modules

codereversing.com/archives/265

This post will show how to enumerate a processes loaded modules without the use of any direct Windows API call. It will rely on partially undocumented functionality both from the native API and the undocumented structures provided by them. The implementation discussed is actually reasonably close to how EnumProcessModules works.

Undocumented Functions and Structures

The main undocumented function that will be used here is NtQueryInformationProcess, which is a very general function that can return a large variety of information about a process depending on input parameters. It takes in a *PROCINFOCLASS* as its second parameter, which determines which type of information it is to return. The value of this parameter are largely undocumented, but a complete definition of it can be found here. The parameter of interest here will be *ProcessBasicInformation*, which fills out a *PROCESS_BASIC_INFORMATION* structure prior to returning. In code this looks like the following:

```
PROCESS_BASIC_INFORMATION procBasicInfo = { 0 };
ULONG ulRetLength = 0;
NTSTATUS ntStatus = NtQueryInformationProcess(hProcess,
    PROCESS_INFORMATION_CLASS_FULL::ProcessBasicInformation,
&procBasicInfo,
    sizeof(PROCESS_BASIC_INFORMATION), &ulRetLength);
if (ntStatus != STATUS_SUCCESS)
{
    fprintf(stderr, "Could not get process information. Status = %X\n",
        ntStatus);
    exit(-1);
}
```

This structure, too, is largely undocumented. Its full definition can be found here. The field of interest is the second one, the pointer to the processes PEB. This is a very large structure that is mapped into every process and contains an enormous amount of information about the process. Among the vast amount of information contained within the *PEB* are the loaded modules lists. The *Ldr* member in the *PEB* is a pointer to a PEB_LDR_DATA structure which contains these three lists. These three lists contain the same modules, but ordered differently; either in load order, memory initialization order, or initialization order as their names describe. The list consists of LDR_DATA_TABLE_ENTRY entries that contain extended information about the loaded module.

Retrieving Module Information

The above definitions are all that is needed in order to implement manual module traversal. The general idea is the following:

1. Open a handle to the target process and obtain the address of its *PEB* (via *NtQuerySystemInformation*).
2. Read the PEB structure from the process (via ReadProcessMemory).
3. Read the PEB_LDR_DATA from the PEB (via *ReadProcessMemory*).
4. Store off the top node and begin traversing the doubly-linked list, reading each node (via *ReadProcessMemory*).

Writing it in C++ translates to the following:

```
void EnumerateProcessDlls(const HANDLE hProcess)
{
    PROCESS_BASIC_INFORMATION procBasicInfo = { 0 };
    ULONG ulRetLength = 0;
    NTSTATUS ntStatus = NtQueryInformationProcess(hProcess,
        PROCESS_INFORMATION_CLASS_FULL::ProcessBasicInformation, &procBasicInfo,
        sizeof(PROCESS_BASIC_INFORMATION), &ulRetLength);
    if (ntStatus != STATUS_SUCCESS)
    {
        fprintf(stderr, "Could not get process information. Status = %X\n",
            ntStatus);
        exit(-1);
    }

    PEB procPeb = { 0 };
    SIZE_T ulBytesRead = 0;
    bool bRet = BOOLIFY(ReadProcessMemory(hProcess,
(LPCVOID)procBasicInfo.PebBaseAddress, &procPeb,
        sizeof(PEB), &ulBytesRead));
    if (!bRet)
    {
        fprintf(stderr, "Failed to read PEB from process. Error = %X\n",
            GetLastError());
        exit(-1);
    }

    PEB_LDR_DATA pebLdrData = { 0 };
    bRet = BOOLIFY(ReadProcessMemory(hProcess, (LPCVOID)procPeb.Ldr, &pebLdrData,
sizeof(PEB_LDR_DATA),
        &ulBytesRead));
    if (!bRet)
    {
        fprintf(stderr, "Failed to read module list from process. Error = %X\n",
            GetLastError());
        exit(-1);
    }

    LIST_ENTRY *pLdrListHead = (LIST_ENTRY
*)pebLdrData.InLoadOrderModuleList.Flink;
    LIST_ENTRY *pLdrCurrentNode = pebLdrData.InLoadOrderModuleList.Flink;
    do
    {
        LDR_DATA_TABLE_ENTRY lstEntry = { 0 };
        bRet = BOOLIFY(ReadProcessMemory(hProcess, (LPCVOID)pLdrCurrentNode,
&lstEntry,
            sizeof(LDR_DATA_TABLE_ENTRY), &ulBytesRead));
        if (!bRet)
        {
            fprintf(stderr, "Could not read list entry from LDR list. Error =
%X\n",
                GetLastError());
            exit(-1);
        }

        pLdrCurrentNode = lstEntry.InLoadOrderLinks.Flink;

        WCHAR strFullDllName[MAX_PATH] = { 0 };
        WCHAR strBaseDllName[MAX_PATH] = { 0 };
        if (lstEntry.FullDllName.Length > 0)
        {
            bRet = BOOLIFY(ReadProcessMemory(hProcess,
(LPCVOID)lstEntry.FullDllName.Buffer, &strFullDllName,
                lstEntry.FullDllName.Length, &ulBytesRead));
            if (bRet)
            {
                wprintf(L"Full Dll Name: %s\n", strFullDllName);
            }
        }

        if (lstEntry.BaseDllName.Length > 0)
        {
            bRet = BOOLIFY(ReadProcessMemory(hProcess,
(LPCVOID)lstEntry.BaseDllName.Buffer, &strBaseDllName,
                lstEntry.BaseDllName.Length, &ulBytesRead));
            if (bRet)
            {
                wprintf(L"Base Dll Name: %s\n", strBaseDllName);
            }
        }

        if (lstEntry.DllBase != nullptr && lstEntry.SizeOfImage != 0)
        {
            wprintf(
                L"  Dll Base: %p\n"
                L"  Entry point: %p\n"
                L"  Size of Image: %X\n",
                lstEntry.DllBase, lstEntry.EntryPoint, lstEntry.SizeOfImage);
        }

    } while (pLdrListHead != pLdrCurrentNode);
}
```

Code

The Visual Studio 2015 project for this example can be found here. The source code is viewable on Github here.
This code was tested on x64 Windows 7, 8.1, and 10.

Follow on Twitter for more updates