Windows API Exploitation in Real World

By

Akash Sarode

Cyber Security in today’s world has become more challenging than traditional days where we had Firewall and antivirus as our main defense systems. Modern day adversary have got all the necessary resources – (Time, Team & Money) to execute their tasks of getting a victim compromised. Most of the offensive techniques tools such as Metasploit, Empire could get easily detected by Next-generation Antivirus or IDS/IPS running within the infrastructure of organization. In order to bypass the Cyber security defense, an adversary may write custom code using programming languages.

This article post is based on “Windows API Exploitation Recipes” course from Pentester Academy.

Exploiting victim using custom code would be golden game for an adversary as most of the signature based defense solutions would not be able to detect the attack. We would be utilising Windows built in APIs to look at some specific techniques which could ensure stealth and undetectability for adversary for as long time as possible. We would be mostly focusing on post exploitation phase of attacker life cycle. Below are few of the pre-requisites which would be required for us to perform this API exploitation techniques -

- Programming language – C
- Windows 10
- Visual Studio 2022

As mentioned earlier, powerful adversaries typically develop custom code to ensure stealthier persistence. We have divided this topic of API exploitation in multiple different sections. In this section, we will specifically look at Process Listing APIs using Windows.

Process listing could be very interesting in real world scenario, which helps as an adversary –

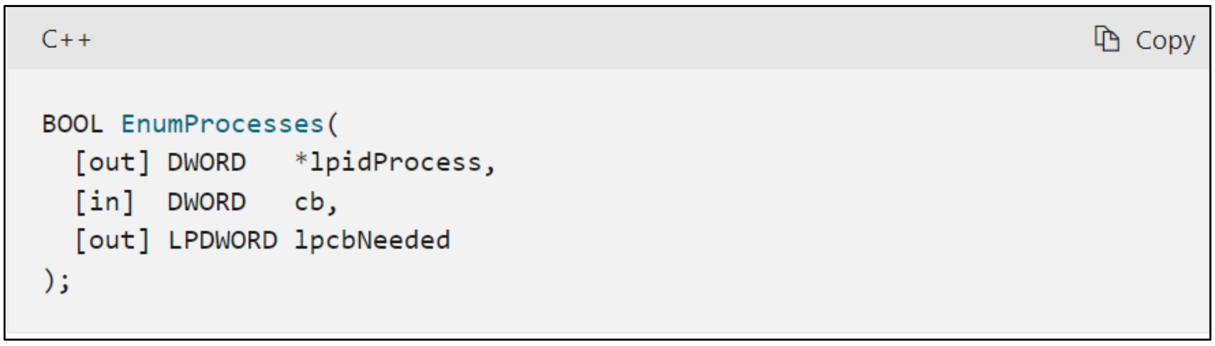
- To understand/enumerate which all processes are running on victim system
- Identify any antivirus, IDS, IPS running on system
- Starting point for process level attacks such as process injection, Memory dump, etc.

There are multiple ways which can be used to perform process listing using Windows APIs.

1. WTSEnumerateProcessesEx
2. CreateToolhelp32Snapshot
3. PSAPI Enumprocess

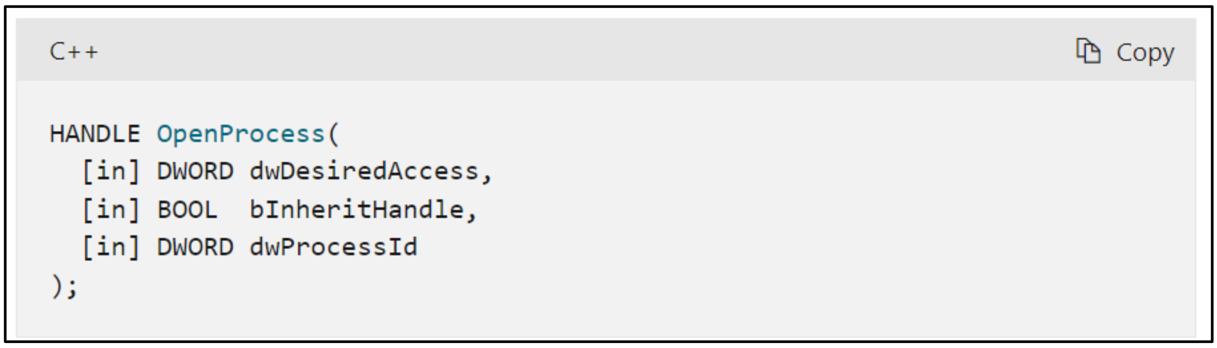
Let’s consider an example for process listing – PSAPI Enumprocess

Retrieves the process identifier for each process object in the system.


C++ Copy

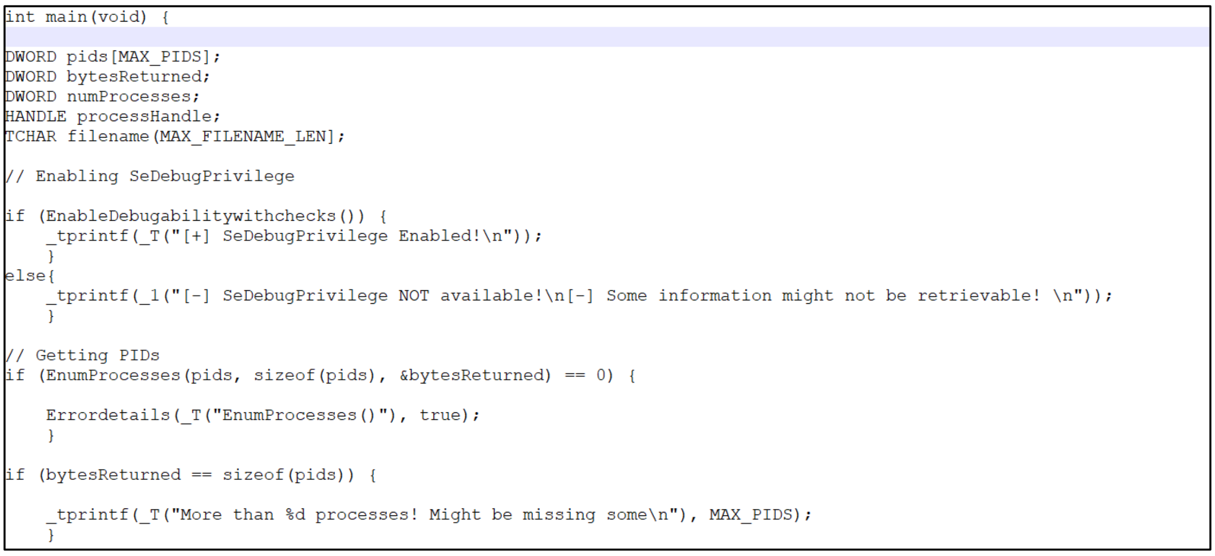
```
BOOL EnumProcesses(
  [out] DWORD   *lpidProcess,
  [in]  DWORD   cb,
  [out] LPDWORD lpcbNeeded
);
```

Enumprocesses will give us the list of process IDs. In order to get more information for particular process, we need to call OpenProcess() API after Enumprocesses.


C++ Copy

```
HANDLE OpenProcess(
  [in] DWORD dwDesiredAccess,
  [in] BOOL  bInheritHandle,
  [in] DWORD dwProcessId
);
```

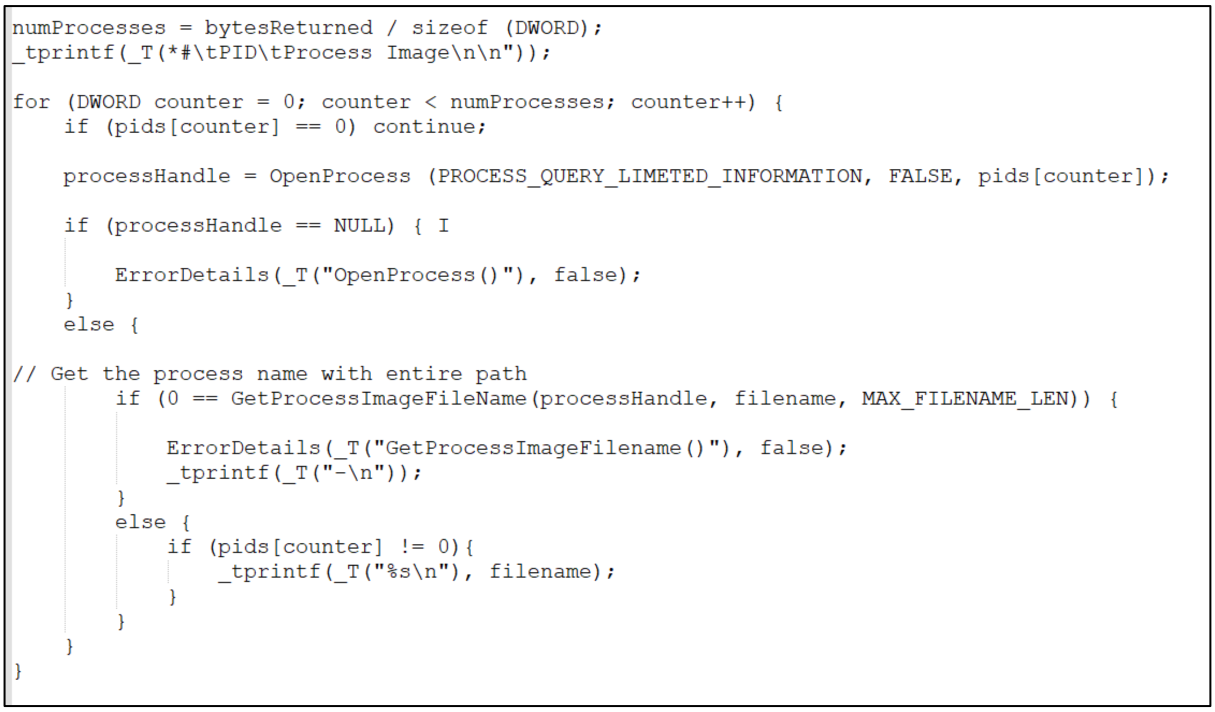
We need to consider that few of the API functions and methods will fail in the if we don’t have privilege session, in such cases we need to enable SeDebugPrivilege on current process token. This would be considered in our code walk through-



```
int main(void) {

DWORD pids[MAX_PIDS];
DWORD bytesReturned;
DWORD numProcesses;
HANDLE processHandle;
TCHAR filename(MAX_FILENAME_LEN];

// Enabling SeDebugPrivilege

if (EnableDebugabilitywithchecks()) {
    _tprintf(_T("[+] SeDebugPrivilege Enabled!\n"));
    }
else{
    _tprintf(_1("[-] SeDebugPrivilege NOT available!\n[-] Some information might not be retrievable! \n"));
    }

// Getting PIDs
if (EnumProcesses(pids, sizeof(pids), &bytesReturned) == 0) {

    Errordetails(_T("EnumProcesses()"), true);
    }

if (bytesReturned == sizeof(pids)) {

    _tprintf(_T("More than %d processes! Might be missing some\n"), MAX_PIDS);
    }
```

Enumprocesses() – this function as seen in code snippet is going to fill in all the pids currently in the system and will provide the output to us.

Next step involves actually applying OpenProcess() API in order to get processhandle and once we get process handle to any particular process -- > GetProcessImageFileName() is called in order to get entire path of the exe which is currently running on system.

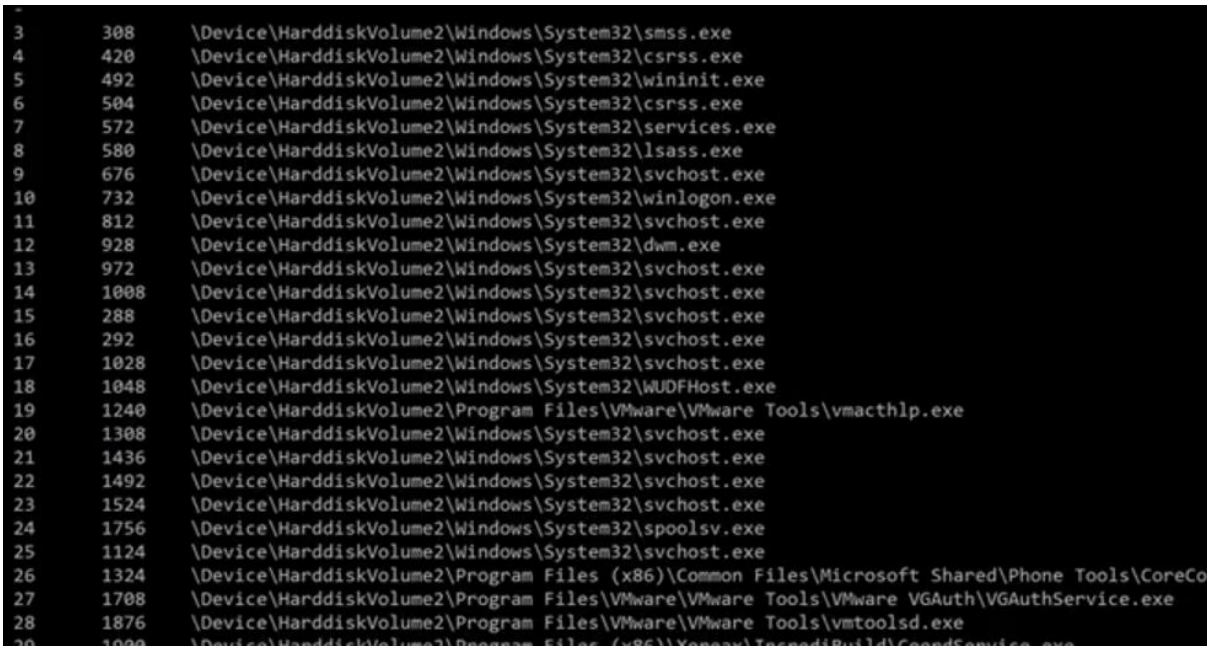


```
numProcesses = bytesReturned / sizeof (DWORD);
_tprintf(_T(*#\tPID\tProcess Image\n\n"));

for (DWORD counter = 0; counter < numProcesses; counter++) {
    if (pids[counter] == 0) continue;

    processHandle = OpenProcess (PROCESS_QUERY_LIMETED_INFORMATION, FALSE, pids[counter]);

    if (processHandle == NULL) { I

        ErrorDetails(_T("OpenProcess()"), false);
    }
    else {

// Get the process name with entire path
        if (0 == GetProcessImageFileName(processHandle, filename, MAX_FILENAME_LEN)) {

            ErrorDetails(_T("GetProcessImageFilename()"), false);
            _tprintf(_T("-\n"));
        }
        else {
            if (pids[counter] != 0){
                _tprintf(_T("%s\n"), filename);
            }
        }
    }
}
```

This code needs to be compiled using Microsoft visual studio and once we compile the code, we need to run the output exe file using command prompt.

There would be an interesting observation once we run the code using normal user cmd prompt. Our code would be able to provide us results in most of the cases but there would be few entries where "**<u>Error: Access is denied</u>**" is specified.

Reason being we don't have enough privileges to get more interesting information, running the code using Administrative command prompt will help us to get results similar to what we would observe using tool like process Explorer.



```
3       308     \Device\HarddiskVolume2\Windows\System32\smss.exe
4       420     \Device\HarddiskVolume2\Windows\System32\csrss.exe
5       492     \Device\HarddiskVolume2\Windows\System32\wininit.exe
6       504     \Device\HarddiskVolume2\Windows\System32\csrss.exe
7       572     \Device\HarddiskVolume2\Windows\System32\services.exe
8       580     \Device\HarddiskVolume2\Windows\System32\lsass.exe
9       676     \Device\HarddiskVolume2\Windows\System32\svchost.exe
10      732     \Device\HarddiskVolume2\Windows\System32\winlogon.exe
11      812     \Device\HarddiskVolume2\Windows\System32\svchost.exe
12      928     \Device\HarddiskVolume2\Windows\System32\dwm.exe
13      972     \Device\HarddiskVolume2\Windows\System32\svchost.exe
14      1008    \Device\HarddiskVolume2\Windows\System32\svchost.exe
15      288     \Device\HarddiskVolume2\Windows\System32\svchost.exe
16      292     \Device\HarddiskVolume2\Windows\System32\svchost.exe
17      1028    \Device\HarddiskVolume2\Windows\System32\svchost.exe
18      1048    \Device\HarddiskVolume2\Windows\System32\WUDFHost.exe
19      1240    \Device\HarddiskVolume2\Program Files\VMware\VMware Tools\vmacthlp.exe
20      1308    \Device\HarddiskVolume2\Windows\System32\svchost.exe
21      1436    \Device\HarddiskVolume2\Windows\System32\svchost.exe
22      1492    \Device\HarddiskVolume2\Windows\System32\svchost.exe
23      1524    \Device\HarddiskVolume2\Windows\System32\svchost.exe
24      1756    \Device\HarddiskVolume2\Windows\System32\spoolsv.exe
25      1124    \Device\HarddiskVolume2\Windows\System32\svchost.exe
26      1324    \Device\HarddiskVolume2\Program Files (x86)\Common Files\Microsoft Shared\Phone Tools\CoreCo
27      1708    \Device\HarddiskVolume2\Program Files\VMware\VMware Tools\VMware VGAuth\VGAuthService.exe
28      1876    \Device\HarddiskVolume2\Program Files\VMware\VMware Tools\vmtoolsd.exe
```

As simple as that!

This is how modern day threat actors develop custom code toolsets which can be used to perform multiple operations and which could get undetected by most of our prevention/detection security solutions.

As an example, we have discussed about process listing APIs but in reality we can have many of such use cases - Process token dumping, Read process memory, writing to process memory using set of Windows APIs and by developing custom code template to perform particular operations.

This would require some basic knowledge of programming but the application of exploiting the knowledgebase of Windows APIs would be huge. And by understanding the approach of how modern day threat adversary would definitely help blue teamers to improve their defense mechanism and stay a step ahead of an attacker.