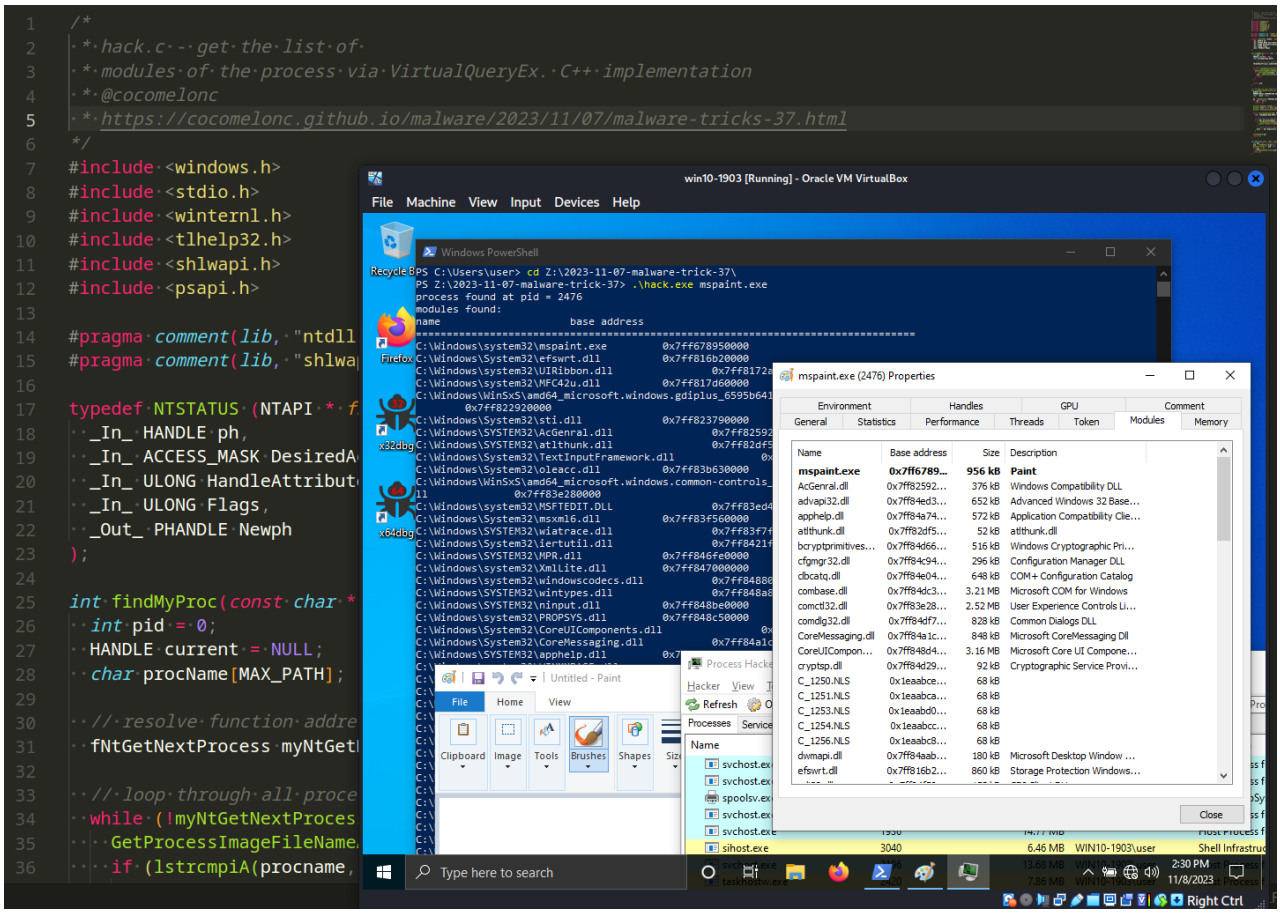
Malware development trick - part 37: Enumerate process modules via VirtualQueryEx. Simple C++ example.

cocomelonc.github.io/malware/2023/11/07/malware-trick-37.html

November 7, 2023

4 minute read

Hello, cybersecurity enthusiasts and white hackers!



Today, this post is the result of my own research on another popular malware development trick: get list of modules of target process.

It's similar to my previous post about enum list of modules, but in this case I used VirtualQueryEx

practical example

First of all, we just use one of the methods to find target process PID. For example I used this one:

```
typedef NTSTATUS (NTAPI * fNtGetNextProcess)(
  _In_ HANDLE ph,
  _In_ ACCESS_MASK DesiredAccess,
  _In_ ULONG HandleAttributes,
  _In_ ULONG Flags,
  _Out_ PHANDLE Newph
);

int findMyProc(const char * procname) {
  int pid = 0;
  HANDLE current = NULL;
  char procName[MAX_PATH];

  // resolve function address
  fNtGetNextProcess myNtGetNextProcess = (fNtGetNextProcess)
GetProcAddress(GetModuleHandle("ntdll.dll"), "NtGetNextProcess");

  // loop through all processes
  while (!myNtGetNextProcess(current, MAXIMUM_ALLOWED, 0, 0, &current)) {
    GetProcessImageFileNameA(current, procName, MAX_PATH);
    if (lstrcmpiA(procname, PathFindFileName((LPCSTR) procName)) == 0) {
      pid = GetProcessId(current);
      break;
    }
  }

  return pid;
}
```

Then, create function which opens a specified process, iterates through its memory regions using `VirtualQueryEx`, and retrieves information about the loaded modules, including their names and base addresses:

```
// function to list modules loaded by a specified process
int listModulesOfProcess(int pid) {
  HANDLE ph;
  MEMORY_BASIC_INFORMATION mbi;
  char * base = NULL;

  ph = OpenProcess(PROCESS_QUERY_INFORMATION | PROCESS_VM_READ, FALSE, pid);
  if (ph == NULL)
    return -1;

  printf("modules found:\n");
  printf("name\t\t\t base address\n");

printf("================================================================================
====\n");

  while (VirtualQueryEx(ph, base, &mbi, sizeof(mbi)) ==
sizeof(MEMORY_BASIC_INFORMATION)) {
    char szModName[MAX_PATH];

    // only focus on the base address regions
    if ((mbi.AllocationBase == mbi.BaseAddress) && (mbi.AllocationBase != NULL)) {
      if (GetModuleFileNameEx(ph, (HMODULE) mbi.AllocationBase, (LPSTR) szModName,
sizeof(szModName) / sizeof(TCHAR)))
        printf("%#25s\t\t%#10llx\n", szModName, (unsigned long
long)mbi.AllocationBase);
      }
      // check the next region
      base += mbi.RegionSize;
    }

  CloseHandle(ph);
  return 0;
}
```

As you can see, the code enters a while loop that continues as long as the `VirtualQueryEx` function successfully retrieves memory information. This loop iterates through memory regions within the target process.

Then checks whether the `AllocationBase` of the current memory region matches the `BaseAddress`. This condition ensures that it only focuses on the base address regions. If the conditions are met, it proceeds to retrieve the module name.

`if (GetModuleFileNameEx(ph, (HMODULE) mbi.AllocationBase, (LPSTR) szModName, sizeof(szModName) / sizeof(TCHAR)))` - The `GetModuleFileNameEx` function is called to retrieve the module filename associated with the current memory region's base address. If successful, it stores the filename in `szModName`.

If the module name retrieval was successful, the code prints the module name and base address in a formatted manner.

After processing the current region, the `base` pointer is incremented by the size of the region to check the next region in the subsequent iteration of the loop.

That's all.

So, the full source code is looks like this (`hack.c`):

```
/*
 * hack.c - get the list of
 * modules of the process via VirtualQueryEx. C++ implementation
 * @cocomelonc
 * https://cocomelonc.github.io/malware/2023/11/07/malware-tricks-37.html
*/
#include <windows.h>
#include <stdio.h>
#include <winternl.h>
#include <tlhelp32.h>
#include <shlwapi.h>
#include <psapi.h>

#pragma comment(lib, "ntdll.lib")
#pragma comment(lib, "shlwapi.lib")

typedef NTSTATUS (NTAPI * fNtGetNextProcess)(
  _In_ HANDLE ph,
  _In_ ACCESS_MASK DesiredAccess,
  _In_ ULONG HandleAttributes,
  _In_ ULONG Flags,
  _Out_ PHANDLE Newph
);

int findMyProc(const char * procname) {
  int pid = 0;
  HANDLE current = NULL;
  char procName[MAX_PATH];

  // resolve function address
  fNtGetNextProcess myNtGetNextProcess = (fNtGetNextProcess)
GetProcAddress(GetModuleHandle("ntdll.dll"), "NtGetNextProcess");

  // loop through all processes
  while (!myNtGetNextProcess(current, MAXIMUM_ALLOWED, 0, 0, &current)) {
    GetProcessImageFileNameA(current, procName, MAX_PATH);
    if (lstrcmpiA(procname, PathFindFileName((LPCSTR) procName)) == 0) {
      pid = GetProcessId(current);
      break;
    }
  }

  return pid;
}

// function to list modules loaded by a specified process
int listModulesOfProcess(int pid) {
  HANDLE ph;
  MEMORY_BASIC_INFORMATION mbi;
  char * base = NULL;

  ph = OpenProcess(PROCESS_QUERY_INFORMATION | PROCESS_VM_READ, FALSE, pid);
```

```
  if (ph == NULL)
    return -1;

  printf("modules found:\n");
  printf("name\t\t\t base address\n");

printf("================================================================================
====\n");

  while (VirtualQueryEx(ph, base, &mbi, sizeof(mbi)) ==
sizeof(MEMORY_BASIC_INFORMATION)) {
    char szModName[MAX_PATH];

    // only focus on the base address regions
    if ((mbi.AllocationBase == mbi.BaseAddress) && (mbi.AllocationBase != NULL)) {
      if (GetModuleFileNameEx(ph, (HMODULE) mbi.AllocationBase, (LPSTR) szModName,
sizeof(szModName) / sizeof(TCHAR)))
        printf("%#25s\t\t%#10llx\n", szModName, (unsigned long
long)mbi.AllocationBase);
      }
      // check the next region
      base += mbi.RegionSize;
    }

  CloseHandle(ph);
  return 0;
}

int main(int argc, char* argv[]) {
  int pid = 0; // process ID
  pid = findMyProc(argv[1]);
  printf("%s%d\n", pid > 0 ? "process found at pid = " : "process not found. pid = ",
pid);
  if (pid != 0)
    listModulesOfProcess(pid);
  return 0;
}
```

demo

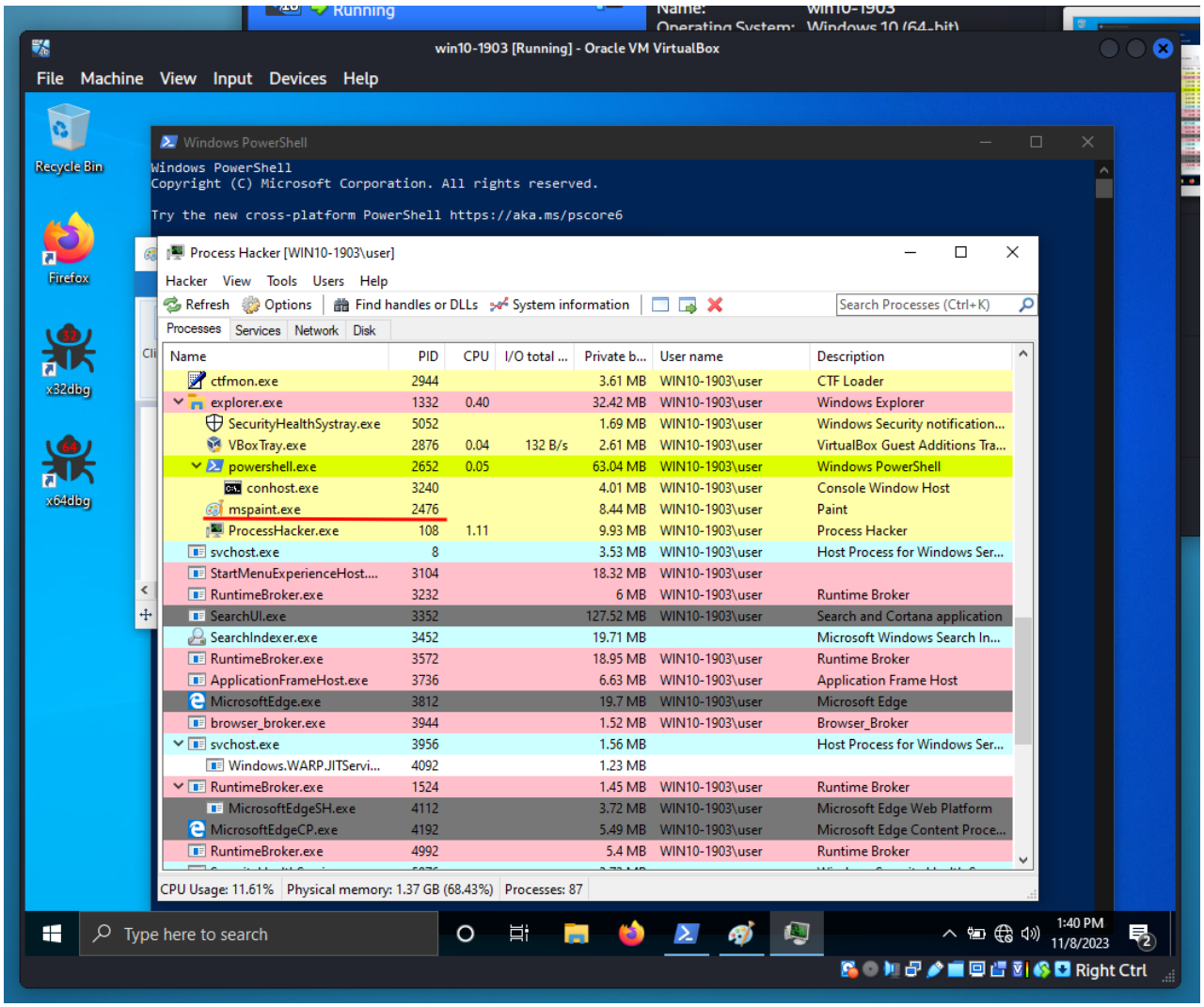
Let’s go to see this logic in action.

Compile it:

```
x86_64-w64-mingw32-g++ -O2 hack.c -o hack.exe -I/usr/share/mingw-w64/include/ -s -
ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-
constants -static-libstdc++ -static-libgcc -fpermissive -lpsapi -lshlwapi
```

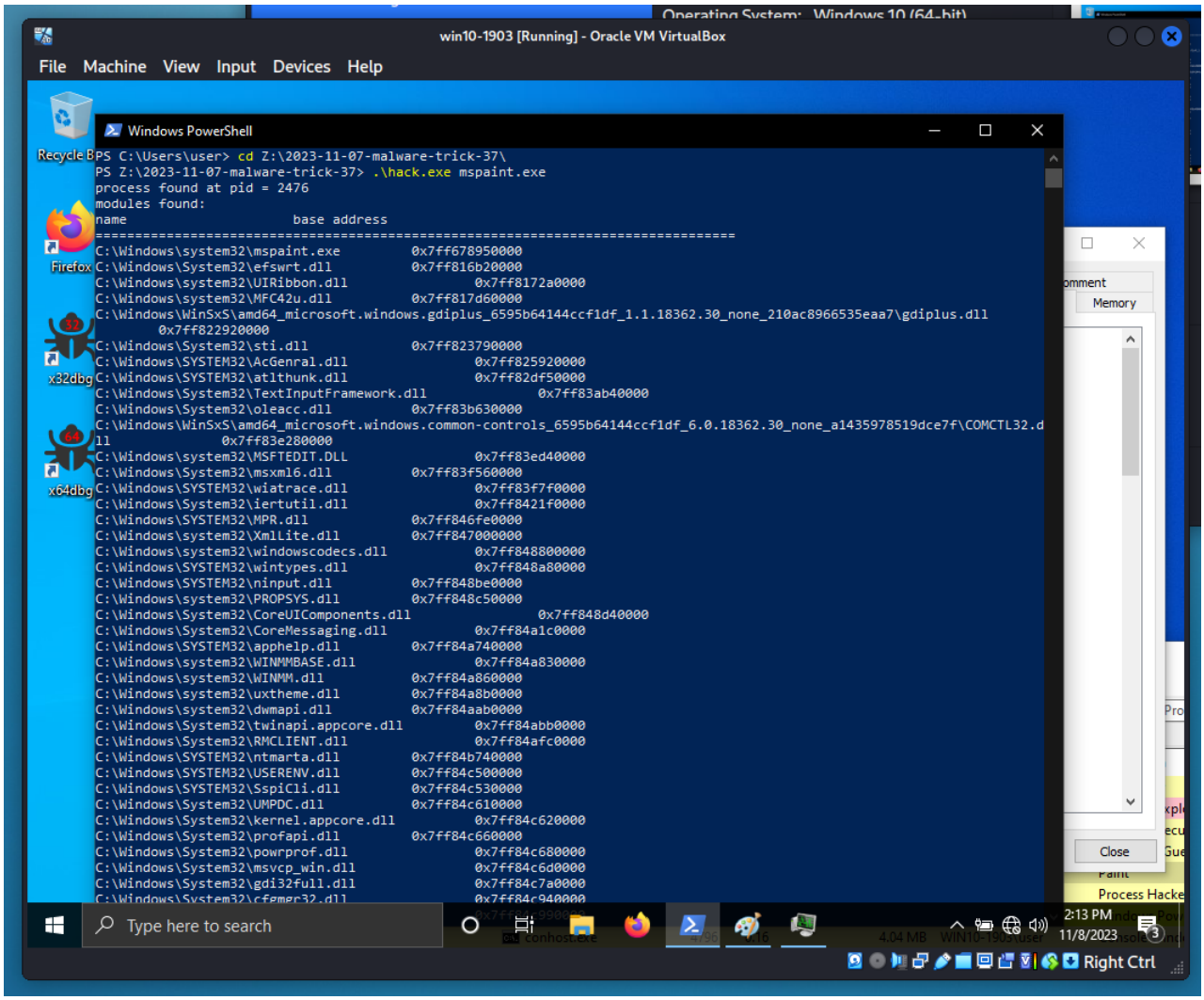
```
┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/meow/2023-11-07-malware-trick-37]
└─$ x86_64-w64-mingw32-g++ -O2 hack.c -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive -lpsapi -lshlwapi
In file included from hack.c:9:
/usr/share/mingw-w64/include/winternl.h:1130:14: warning: 'void RtlUnwind(PVOID, PVOID, PEXCEPTION_RECORD, PVOID)' redeclared without dllimport attribute: previous dllimport ignored [-Wattributes]
 1130 |    VOID NTAPI RtlUnwind (PVOID TargetFrame,PVOID TargetIp,PEXCEPTION_RECORD ExceptionRecord,PVOID ReturnValue);
      |              ^~~~~~~~

┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/meow/2023-11-07-malware-trick-37]
└─$ ls -lt
total 48
-rwxr-xr-x 1 cocomelonc cocomelonc 41472 Nov  8 17:12 hack.exe
-rw-r--r-- 1 cocomelonc cocomelonc  2398 Nov  8 17:12 hack.c

┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/meow/2023-11-07-malware-trick-37]
└─$
```

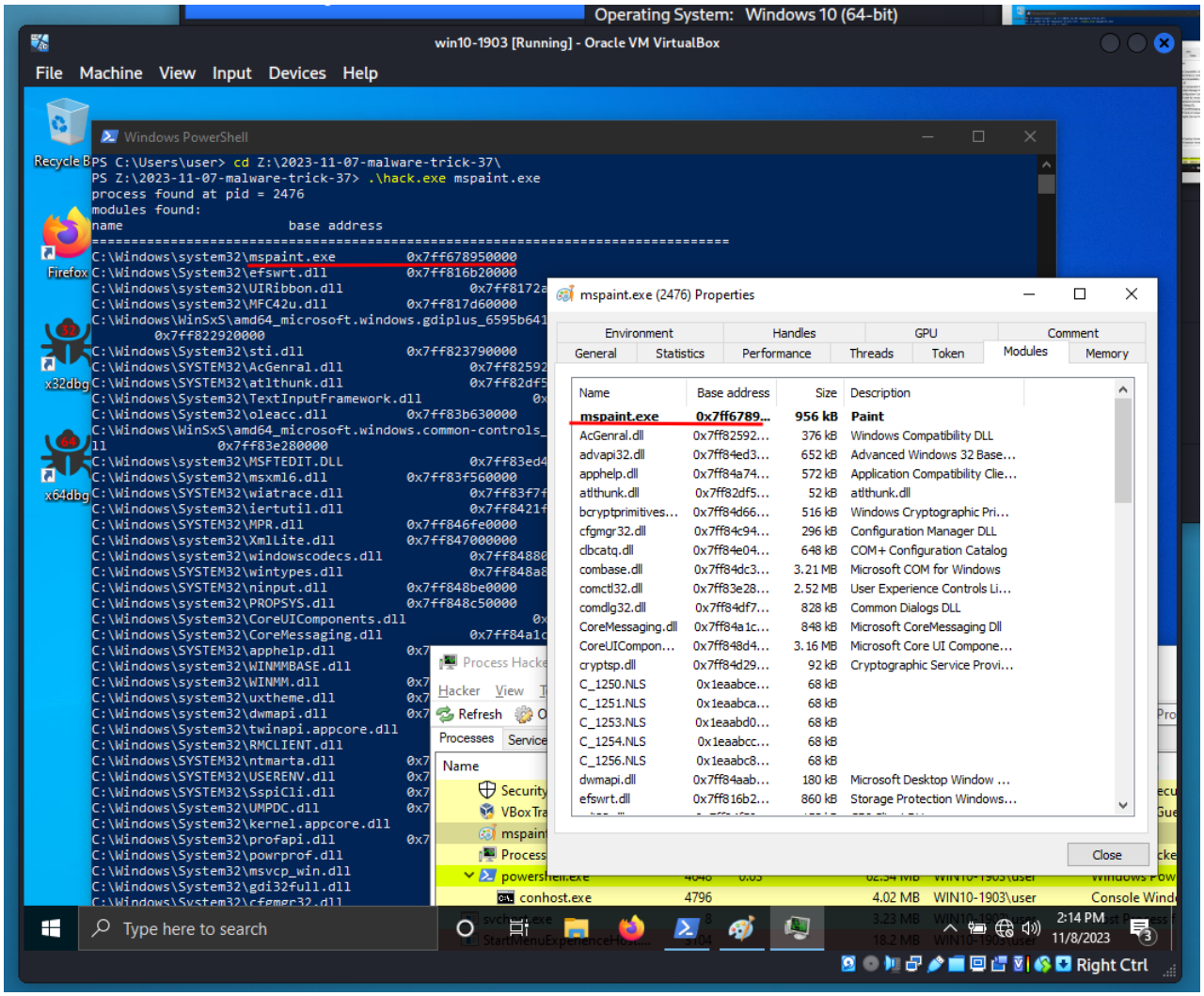
Then, open target process in the victim's machine:



And just run our `hack.exe`:

```
.\hack.exe mspaint.exe
```


win10-1903 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Windows PowerShell
PS C:\Users\user> cd Z:\2023-11-07-malware-trick-37\
PS Z:\2023-11-07-malware-trick-37> .\hack.exe mspaint.exe
process found at pid = 2476
modules found:
name base address
C:\Windows\system32\mspaint.exe 0x7ff678950000
C:\Windows\System32\efswrt.dll 0x7ff816b20000
C:\Windows\system32\UIRibbon.dll 0x7ff8172a0000
C:\Windows\system32\MFC42u.dll 0x7ff817d60000
C:\Windows\WinSxS\amd64_microsoft.windows.gdiplus_6595b64144ccf1df_1.1.18362.30_none_210ac8966535eaa7\gdiplus.dll
0x7ff822920000
C:\Windows\System32\sti.dll 0x7ff823790000
C:\Windows\SYSTEM32\AcGenral.dll 0x7ff825920000
C:\Windows\SYSTEM32\atlthunk.dll 0x7ff82df50000
C:\Windows\System32\TextInputFramework.dll 0x7ff83ab40000
C:\Windows\System32\oleacc.dll 0x7ff83b630000
C:\Windows\WinSxS\amd64_microsoft.windows.common-controls_6595b64144ccf1df_6.0.18362.30_none_a1435978519dce7f\COMCTL32.dll 0x7ff83e280000
C:\Windows\system32\MSFTEDIT.DLL 0x7ff83ed40000
C:\Windows\System32\msxml6.dll 0x7ff83f560000
C:\Windows\SYSTEM32\wiatrace.dll 0x7ff83f7f0000
C:\Windows\System32\iertutil.dll 0x7ff8421f0000
C:\Windows\SYSTEM32\MPR.dll 0x7ff846fe0000
C:\Windows\system32\XmlLite.dll 0x7ff847000000
C:\Windows\system32\windowscodecs.dll 0x7ff848800000
C:\Windows\SYSTEM32\wintypes.dll 0x7ff848a80000
C:\Windows\SYSTEM32\ninput.dll 0x7ff848be0000
C:\Windows\system32\PROPSYS.dll 0x7ff848c50000
C:\Windows\System32\CoreUIComponents.dll 0x7ff848d40000
C:\Windows\System32\CoreMessaging.dll 0x7ff84a1c0000
C:\Windows\SYSTEM32\apphelp.dll 0x7ff84a740000
C:\Windows\system32\WINMMBASE.dll 0x7ff84a830000
C:\Windows\system32\WINMM.dll 0x7ff84a860000
C:\Windows\system32\uxtheme.dll 0x7ff84a8b0000
C:\Windows\system32\dwmapi.dll 0x7ff84aab0000
C:\Windows\System32\twinapi.appcore.dll 0x7ff84abb0000
C:\Windows\System32\RMCLIENT.dll 0x7ff84afc0000
C:\Windows\SYSTEM32\ntmarta.dll 0x7ff84b740000
C:\Windows\SYSTEM32\USERENV.dll 0x7ff84c500000
C:\Windows\SYSTEM32\SspiCli.dll 0x7ff84c530000
C:\Windows\System32\UMPDC.dll 0x7ff84c610000
C:\Windows\System32\kernel.appcore.dll 0x7ff84c620000
C:\Windows\System32\profapi.dll 0x7ff84c660000
C:\Windows\System32\powrprof.dll 0x7ff84c680000
C:\Windows\System32\msvcp_win.dll 0x7ff84c6d0000
C:\Windows\System32\gdi32full.dll 0x7ff84c7a0000
Recycle Bin
Firefox
x32dbg
x64dbg
Memory
Close
Process Hacke
Type here to search
2:13 PM
11/8/2023
Right Ctrl



As you can see, everything is worked perfectly! =^..^=

Keep in mind that this code may have limitations and dependencies on specific Windows APIs. Additionally, it relies on the process name for identification, which may not be unique.

This code can also help you develop your own script to work with process memory, for example for forensics or other tasks on blue team practical cases.

I hope this post spreads awareness to the blue teamers of this interesting malware dev technique, and adds a weapon to the red teamers arsenal.

VirtualQueryEx
GetModuleFileNameEx
Find process ID by name and inject to it
Find PID via NtGetNextProcess
source code in github

> This is a practical case for educational purposes only.

Thanks for your time happy hacking and good bye!
PS. All drawings and screenshots are mine