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.

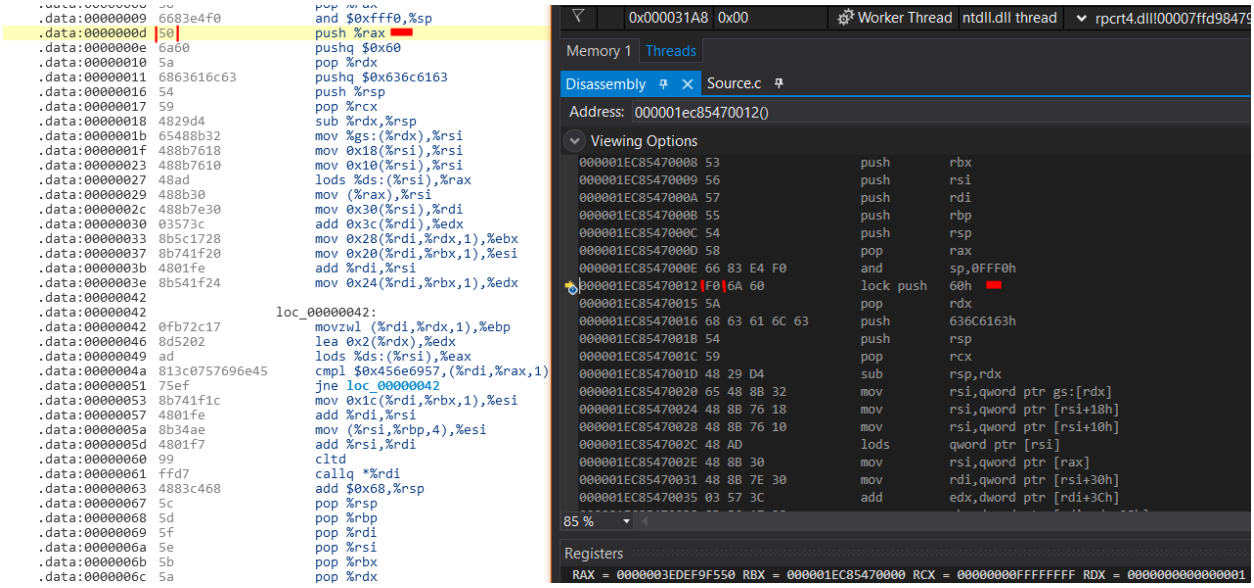
Build Optimization

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.

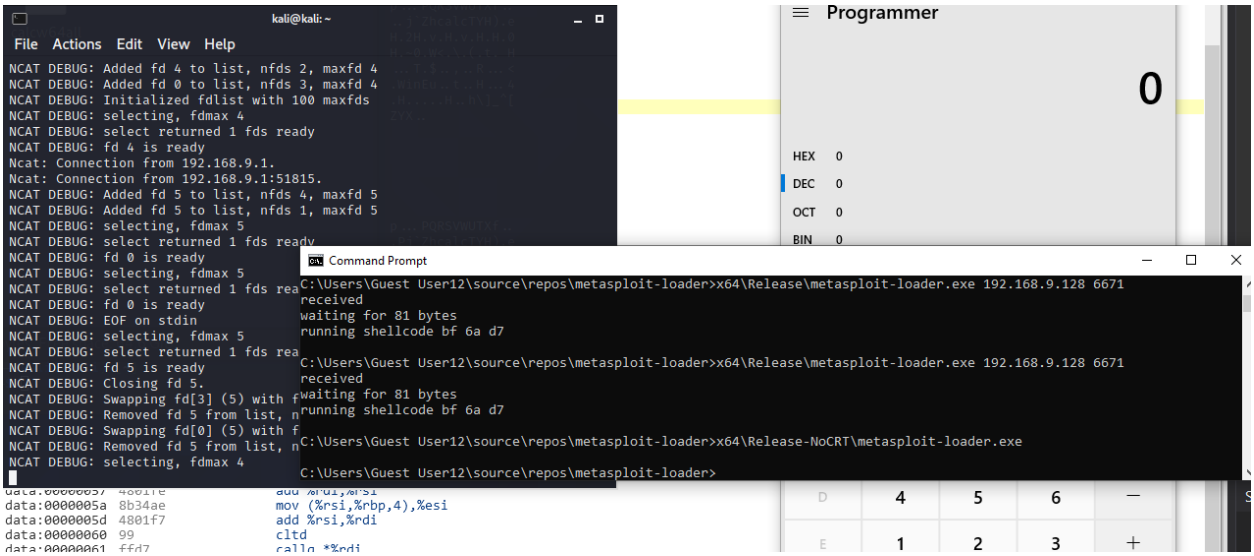
```
#ifdef NO_CRT
#define printf(...)
void memcopy(void* dest, void* src, size_t n) {
    for (int i = 0; i < n; i++)
        ((char*)dest)[i] = ((char*)src)[i];
}
#else
#include <stdio.h>
#define memcopy memcpy
#endif // NO_CRT

#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.

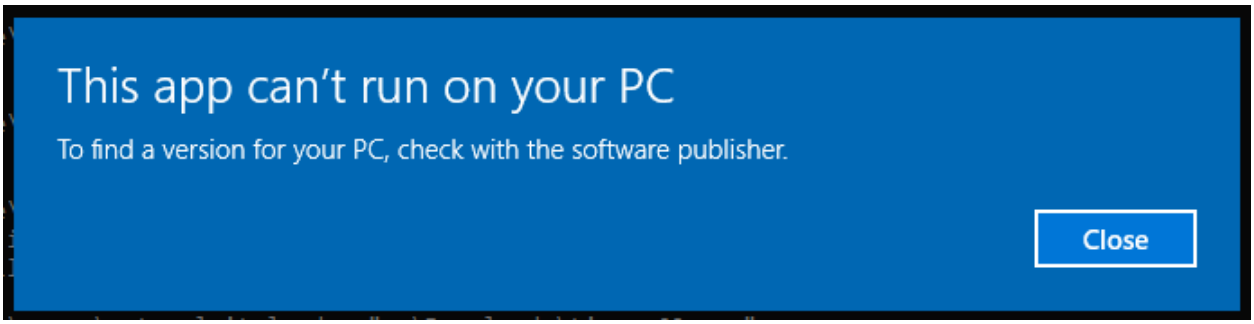


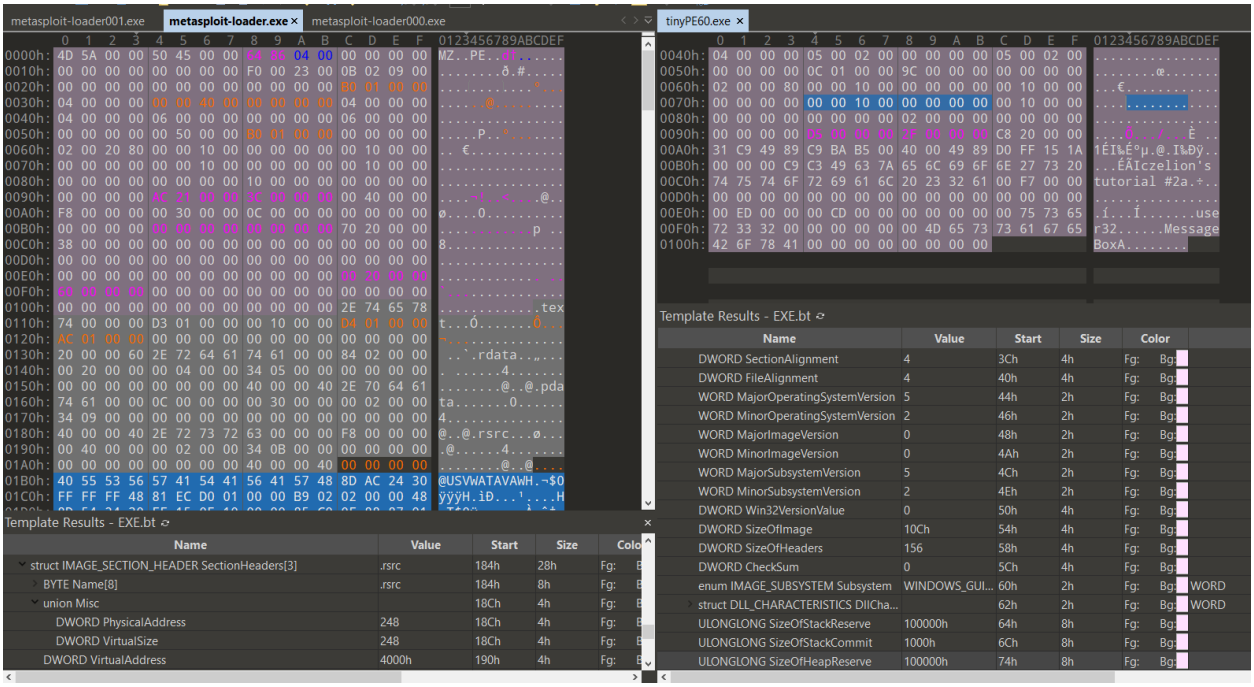
And we popped calc, so that's cool.



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 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.