Baking really good x86/x64 shellcode for Windows

My idea is to create small, position-independent, cross-platform x86/x64 code with some nice tricks. Here are some snippets commonly used in shellcode for example.

Get PEB and kernel32.dll base address

Opcode	x86	x64
6A 60	push 60h	push 60h
5A	pop edx	pop rdx
31C0	xor eax, eax	xor eax, eax
50	push eax	push rax
48 64:0F481D 30000000	dec eax cmovs ebx, fs:[30h]	cmovs ebx, fs:[rip+30h]
0F491A	cmovs edx, esp	cmovs edx, esp
65:48 0F491A	gs:dec eax cmovns ebx, [edx]	cmovns rbx, gs:[rdx]

The trick is to use the *DEC EAX*/*REX prefix* and *CMOVcc* to conditionally get the data we need: in x86 we get the PEB address in *EBX*; in x64 has no effect. In x86 *CMOVS* moves *ESP* to *EDX*, but not in x64, *RDX* remains 60h. In x86 *GS:DEC EAX* and *CMOVNS* have no effect. In x64, we get the PEB address in *RBX*.

Opcode	x86	x64
59	pop ecx	pop rcx
0F94D1	setz cl	setz cl
6BF9 08	imul edi, ecx, 8	imul edi, ecx, 8
FEC1	inc cl	inc cl
6BD1 0C	imul edx, ecx, 0Ch	imul edx, ecx, 0ch
48 8B1C13	dec eax mov ebx, [ebx+edx]	mov rbx, [rbx+rdx]
01FA	add edx, edi	add edx, edi
48 8B1C13	dec eax mov ebx, [ebx+edx]	mov rbx, [rbx+rdx]
48 8B33	dec eax mov esi, [ebx]	mov rsi, [rbx]
48 AD	dec eax lodsd	lodsq
FF7438 18	push [eax+edi+18h]	push [rax+rdi+18h]
5D	pop ebp	pop rbp

In x64 *SETZ* sets *CL*=1. We use *IMUL* to dynamically adjust the offsets to read Ldr and InLoadOrderModuleList. *PUSH/POP* don't need *REX*, it's compatible for both modes, it's a nice optimization trick. The same play with *SETZ/IMUL* can be used to parse the PE when looking for API addresses in a DLL.

How to call APIs

W64 uses FASTCALL, so some APIs will require 4 QWORD slots to spill registers, it's called the "shadow space". We will make the slots using *PUSH* that in W32's STDCALL will have the effect of pushing a parameter, it would look like this in W64:

```
mov  rdx, lpFindFileData
mov  rcx, lpFileName
push rax                    ;align before call
push rax                    ;shadow space slot
push rax                    ;shadow space slot
push rdx                    ;x86: push lpFindFileData
push rcx                    ;x86: push lpFileName
push myapis.FindFirstFileW ;for example, 0ch
pop  eax
call jump2api
```

When I find the addresses of the APIs I need, I push them onto the stack, but to pick an API address from it, we again need to calculate the correct offset. The idea is to use the offset for x86 and multiply it by 2 in a trampoline code I call *jump2api*. *EAX* = API offset, *ESI* is a pointer to the API addresses in stack:

Opcode	x86	x64
51	push ecx	push rcx
E8 xxxxxxxx	call is64bit	call is64bit
D3E0	shl eax, cl	shl eax, cl
59	pop ecx	pop rcx
FF2406	jmp [esi+eax]	jmp [rsi+rax]

What is *is64bit*? It's a detection gem by qkumba for my BEAUTIFULSKY codebase:

Opcode	x86	x64
31C9	xor ecx, ecx	xor ecx, ecx
63C9	arpl cx, cx	movsxd ecx, ecx
0F94D1	setz cl	setz cl
C3	ret	ret

XOR sets *ZF*=1 in both modes. *ARPL* sets *ZF*=0 in x86 but here is the trick: in x64, *ARPL* opcode was reassigned to be *MOVSXD* that doesn't alter any flag!

Bonus: Exception handling

Using Vectored Exception Handling it's possible to create a compatible handler for both modes. Here begins our handler:

Opcode	x86	x64
5A	pop edx	pop rdx
58	pop eax	pop rax
53	push ebx	push rbx
50	push eax	push rax
5B	pop ebx	pop rbx
31C0	xor eax, eax	xor eax, eax
50	push eax	push rax
48 0F49D9	dec eax cmovns ebx, ecx	cmovns rbx, rcx
59	pop ecx	pop rcx
0F94D1	setz cl	setz cl
E8 xxxxxxxx	call set_newIP	call set_newIP

We use the *REX prefix/CMOVNS* trick to get the pointer to *EXCEPTION_POINTERS* in *EBX/RBX*, which in x64 is passed to the handler via *RCX*, and in x86 via the stack. We use *CALL* to "push" to the stack the address that we use to continue execution replacing EIP/RIP in CONTEXT, otherwise it would continue where the exception occurred. So *set_newIP* is this code:

Opcode	x86	x64
48 8B5C8B 04	dec eax mov ebx, [ebx+ecx*4+4]	mov rbx, [rbx+rcx*4+4]
6BC140	imul eax, ecx, 40h	imul eax, ecx, 40h
8F8403 B8000000	pop [ebx+eax+0b8h]	pop [rbx+rax+0b8h]
5B	pop ebx	pop rbx
C1E1 03	shl ecx, 3	shl ecx, 3
48 29CC	dec eax sub esp, ecx	sub rsp, rcx
83C8 FF	or eax, -1	or eax, -1
FFE2	jmp edx	jmp rdx

We get the pointer to CONTEXT and calculate the offset to *EIP/RIP* and with *POP* we replace it with the "pushed" address, then return EXCEPTION_CONTINUE_EXECUTION and the execution continues after "call set_newIP".