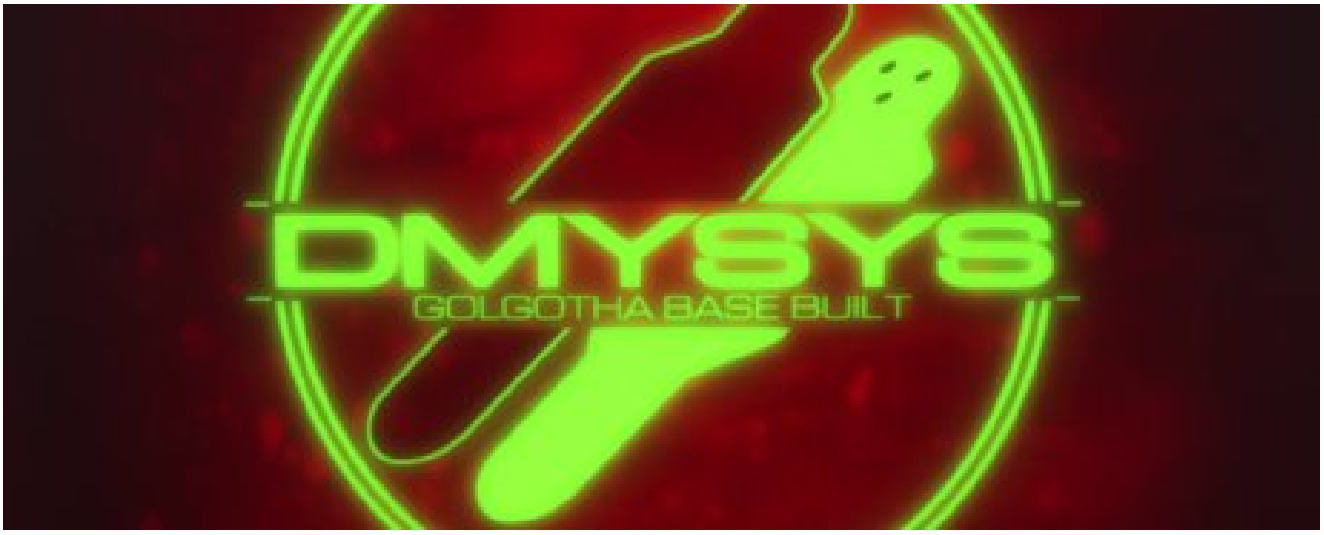
(13) X

x.com/tomiesghost_/article/1871653802245259725



Constructing a Win32 Control Handler in MASM

For my first Github project, I wanted to make something simple that could also simultaneously be something I could use to show off my low-level skills.

Feeling inspired by a blog post from fellow Cr0w Academy colleague
x0reaxeax
, I had the idea to implement a control handler in x64 MASM.

For those who don’t know what signals are, I’ll provide a brief explanation from the
UNIX Code Migration Guide
on MSDN:

> "UNIX signals are software interrupts that catch or indicate different types of events. Windows on the other hand supports only a small set of signals that is restricted to exception events only."

So of course, we were gonna have to puzzle out how signals are handled on Windows in order to build our program.

Unlike Linux, Windows doesn’t have the
sigaction
struct to handle control signals from a process. What we do have however, is the
HandlerRoutine

callback function, which does exactly the same thing.
Next, we're going to need to use the
SetConsoleCtrlHandler
function in order to add our Handler to the calling process.

Now that we have all that down, we can write a short C program to test our newfound knowledge:

c

```
#include <stdio.h>
#include <Windows.h>

BOOL WINAPI HandlerRoutine(DWORD fdwCtrlType) {
    if (fdwCtrlType == CTRL_C_EVENT)
    {
        printf("Signal caught\n");
        return EXIT_SUCCESS;
    }
}

int main(void) {
    if (!SetConsoleCtrlHandler(HandlerRoutine, TRUE))
    {
        printf("Could not install Control handler\n");
        return EXIT_FAILURE;
    }

    printf("Control Handler is installed\n");

    while (1);

    return EXIT_SUCCESS;
}
```

Upon running and hitting CTRL+C, we'll see:

> Signal caught.

Sexy. Let's convert this baby to MASM.

A little stack shadow space here, some function calls there, and voila:

x86asm

```
includelib legacy_stdio_definitions.lib
includelib ucrt.lib
includelib kernel32.lib

SetConsoleCtrlHandler proto
printf                                  proto
HandlerRoutine          proto

.data
caughtSignal                    byte    "Signal caught", 10, 0
handlerInstalled                byte    "Control Handler is installed", 10, 0

CTRL_C_EVENT                    equ             0
EXIT_SUCCESS                    equ             0


.code
main PROC
         call setConsole
         cmp  rax, EXIT_SUCCESS
         jmp  exit

         call installedHandler
         cmp  rax, EXIT_SUCCESS
         jmp  exit

exit:
     xor rax, rax
     ret

main ENDP


print PROC
          sub  rsp, 40
          call printf
          add  rsp, 40
          ret
print ENDP


setConsole PROC
              sub  rsp, 40
              lea  rcx, [setHandler]
              mov  rdx, 1
              call SetConsoleCtrlHandler
              add  rsp, 40

setConsole ENDP


installedHandler PROC
```

```
                    sub  rsp, 40
                    lea  rcx, handlerInstalled
                    call print
                    add  rsp, 40

                    jmp whileLp
whileLp:
        jmp whileLp

installedHandler ENDP


setHandler PROC

             sub  rsp, 40
             mov  rcx, CTRL_C_EVENT
             call HandlerRoutine
             add  rsp, 40

             cmp  rcx, 0
             jz   signalCaught

signalCaught:
             lea  ecx, caughtSignal
             call printf

setHandler ENDP


end
```

Run the program, and-

> LNK2019 unresolved external symbol HandlerRoutine referenced in function setHandler

Beg your pardon?

Something is clearly going on with HandlerRoutine. Hm.

After a few minutes of internal screaming and a quick inspection of the library file, I was led to the root cause of the problem, that being:

Callback functions aren't defined in the kernel32 library.

…Which meant I had to learn how they were implemented in MASM.

About 5 or 6 hours into scouring the web for info, I eventually ended up on a Youtube tutorial for creating a Window using MASM, which uses the Winproc callback function.

Using the information from this, I quickly adapted my function from:

x86asm

```
setHandler PROC

             sub  rsp, 40
             mov  rcx, CTRL_C_EVENT
             call HandlerRoutine
             add  rsp, 40

             cmp  rcx, 0
             jz   signalCaught

signalCaught:
             lea  ecx, caughtSignal
             call printf

setHandler ENDP
```

To:

x86asm

```
HandlerRoutine PROC dwCtrlType:DWORD

             mov  rcx, CTRL_C_EVENT
             cmp  rcx, 0
             jz   signalCaught

signalCaught:
             lea ecx, caughtSignal
             call printf

HandlerRoutine ENDP
```

And after running the changed code, I saw a glorious:

> Control Handler is installed

Marveling at my genius, I clicked CTRL+C, expecting to see the holy words of “Signal caught” before me; However, I was met with disappointment.

Although my program was indeed hitting the infinite loop, what it wasn’t doing was intercepting the signal like it was supposed to.

This is odd. After all, I implemented the proper condition to make sure that a CTRL_C_EVENT triggers the message response, right?

Is it time to pack up and finally consider the abomination that is webdev?

Before we decide to sell our soul to Javascript, let's retrace our steps back to the list of parameter values for the HandlerRoutine function, specifically CTRL_C_EVENT:

> "A CTRL+C signal was received, either from keyboard input or from a signal generated by the
> GenerateConsoleCtrlEvent
> function"

So the function intercepts a CTRL_C_EVENT as long as we provide the keyboard input for it..meaning we don't need to write a conditional in order to catch the signal.

Bet.

With that in mind, we can rewrite the previous code in our callback function:

x86asm

```
HandlerRoutine PROC dwCtrlType:DWORD

             mov  rcx, CTRL_C_EVENT
             cmp  rcx, 0
             jz   signalCaught

signalCaught:
             lea ecx, caughtSignal
             call printf

HandlerRoutine ENDP
```

To this:

x86asm

```
HandlerRoutine PROC dwCtrlType:DWORD

            lea rcx, caughtSignal
            call printf

HandlerRoutine ENDP
```

And after giving our newly run program an anxiety-ridden click of CTRL+C, we see our final result:

> Signal caught.