```
.text:00406514                          loc_406514:
.text:00406514 E8 89 41 04 85           call    near ptr 8544A6A2h
```

Figure 5: Anti-Disassembly null deference followed by junk jumps

```
.text:0040650B 90                       nop
.text:0040650C 90                       nop
.text:0040650D 90                       nop
.text:0040650E 90                       nop
.text:0040650F 90                       nop
.text:00406510 90                       nop
.text:00406511 90                       nop
.text:00406512 8B 4D E8                 mov     ecx, [ebp+Block]
.text:00406515 89 41 04                 mov     [ecx+4], eax
.text:00406518 85 C0                    test    eax, eax
.text:0040651A 75 22                    jnz     short loc_40653E.text:00406553 E8 5A
```

Figure 6: Anti-Disassembly removed

Anti-Debugging Techniques

The anti-debugging techniques used throughout this challenge are fairly common and most are handled by a plugin for x64dbg, ScyllaHide as seen in Figure 7 shows many of the options for configuring ScyllaHide(ScyllaHide, 2021) . Some efforts were added to further obfuscate anti-debugging techniques, below is a list of anti-debugging techniques used:

- IsDebuggerPresent
  - Determined by checking `PEB.BeingDebugged (offset 0x02)` to be set.
- IsRemoteDebuggerPresent
  - Calls `CheckRemoteDebuggerPresent` on current process and checks if the return has a debug port.
- IsNtGlobalFlag
  - Checks the `PEB.NtGlobalFlag (offset 0x68)` for `the following flags described in` Table 1: NtGlobalFlags`.`

Flag	Value
FLG_HEAP_ENABLE_TAIL_CHECK	0x10
FLG_HEAP_ENABLE_FREE_CHECK	0x20
FLG_HEAP_VALIDATE_PARAMETERS	0x40
Total	0x70

Table 1: NtGlobalFlags

- IsSeDebugPrivsEnabled

    - This sample doesn't enable `DebugPrivs` so if they are it is a red flag. Another test that is better for this kind of test, but not implemented is to check the parent process and see if that process has DebugPrivs enabled.
- IsHardwareBreakpointPresent
    - This checks if any of the hardware breakpoints are set by checking the CONTEXT record for the running thread.
- IsTooSlow
    - This check is a simple timing check that does a simple back to back call to `GetTickCount` and compares the delta and looks for a threshold of anything over two seconds constitutes debugging

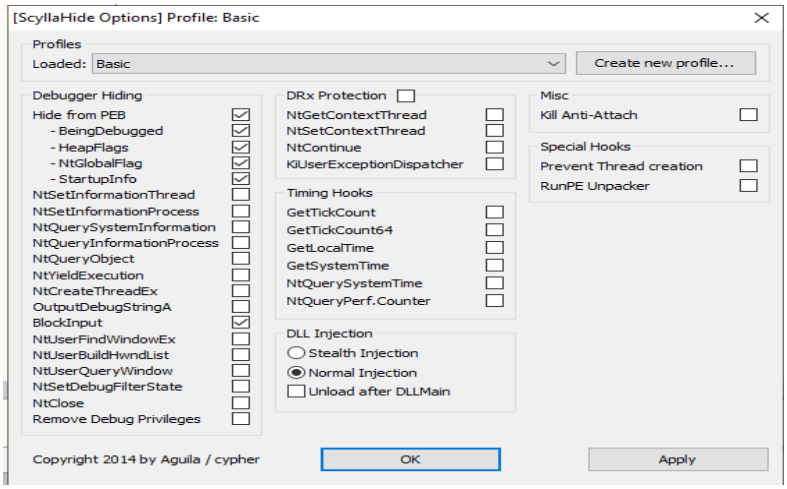

Figure 7: Scylla Hide Configuration

These debug checks are run periodically and randomly throughout the challenge. A random test will run randomly between zero and ten seconds. This is to bypass a common practice of letting all anti-debug logic to run and then attach to the process for debugging. There are places during the challenge that debug checks are called randomly. Not only is this a hassle, it creates the problem that if one wants to NOP out the anti-debugging and just handles the creation of debugging object, the program will crash when the object is trying to be referenced later. If the anti-debug thread is killed it will restart itself. Killing both threads is an option. All anti-debug checks will call `ExitProcess` if debugging is detected, because of this, modifying `ExitProcess` to return immediately is an option too.

The constructor for the `AntiDebug` class has added functionality that is only called once. Two of these fall under anti-debug and two under anti-virtualization. The two anti-debug features are as follows:

- `PatchDbgBreakPoint`
    - `DbgBreakPoint` is resolved and the first byte of the function is replaces with a `ret` instruction (0xC3).
- `PatchDbgUiRemoteBreakIn`
    - Patching `DbgUiRemoteBreakin` goes a step further and inserts shellcode that will call `TerminateProcess` when this function is called.

Anti-Virtualization Techniques

The anti-virtualization techniques used in this challenge are meant to detect running in either VmWare or VirtualBox. The techniques are standard and well covered over the years.

VmWare Detection starts with a common VmWare detection method of getting the memory size by making using the `in` operation from assembly. There are some magic values that are associated with this method and therefore are obfuscated by adding two values together to create the desired value. If running outside of a VM the in call is