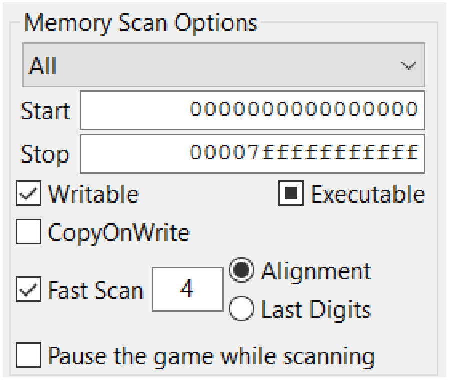
## When the Memory Scan Fails

Sometimes, despite your best efforts, you can't seem to find the value you're looking for in a memory scanner like Cheat Engine. There are several possible reasons for this.

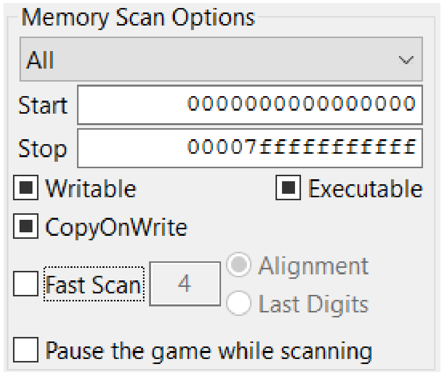
- **The memory scanner isn't looking everywhere it could be.**



Memory Scan Options
All
Start 0000000000000000
Stop 00007fffffffffff
[✓] Writable [■] Executable
[ ] CopyOnWrite
[✓] Fast Scan 4 (●) Alignment ( ) Last Digits
[ ] Pause the game while scanning

On x86 processors, memory is organized into *pages*, which are contiguous blocks of memory. This is due to the way memory management is implemented in the processor's hardware. On a typical machine, pages are 4096 (0x1000) bytes large although this can vary. Pages are important for us, because each memory page is assigned *protection bits*. The exact details aren't important to us now (you'll learn more in CS2200 😊) but all that we need to know for now is the following: the 3 protection bits (**R**ead, **W**rite, e**X**ecute) basically tell the processor whether we're allowed to access the memory. For example, if you try to write to a read-only page (no W bit set), the processor will generate an exception. The operating system will receive that exception and, believing that the program has malfunctioned, kill it. In short, your program crashes with an Access Violation (aka Segfault).

Now what does this have to do with our memory scan? Look at the three checkboxes: "Writeable," "Executable," and "CopyOnWrite". These checkboxes basically tell the scanner which pages of memory to look through and which to skip. For example, currently it would only look in writeable pages, skip COW pages, and doesn't care whether pages are marked executable or not. If you're stuck and can't find a memory value, relaxing these settings may help:



Memory Scan Options
All
Start 0000000000000000
Stop 00007fffffffffff
[■] Writable [■] Executable
[■] CopyOnWrite
[ ] Fast Scan 4 (●) Alignment ( ) Last Digits
[ ] Pause the game while scanning

Now the scanner will basically look through all the memory it can. Also note the Fast Scan and Start/Stop settings: these adjust at which addresses the scanner will look at memory. Fast Scan makes it only scan at addresses which are multiples of 4 and might cause the scanner to miss results in unaligned addresses. The Start/Stop setting is pretty self-explanatory: they're the start and stop addresses of the scan. In this course we'll only be working on 32-bit processes, so you should start the scan at 0x00000000 and stop at 0xffffffff. If you were on 64-bit, userland memory ranges from 0x0000000000000000 to 0x00007fffffffffff.

As to why we use those start and stop values, remember what a memory address actually is: it just identifies a specific byte in the program's address space memory. When we say "32-bit" and "64-bit", we're really just referring to how big our address space is: on 32-bit, we have 32-bit addresses, meaning addresses range from 0 to ($2^{32}$-1). Likewise, the registers are 32-bits wide meaning they hold 32-bit values. In other words, there are $2^{32}$ bytes we can address. That's why you hear people say that "32-bit computers can't use more than 4GiB of RAM": there's simply no way to refer to, or *address*, any more memory past that limit of $2^{32}$ bytes. Meanwhile, on 64-bit, registers and addresses are twice as large, and the address space's size is squared ($2^{64}$ bytes). Hence, 64-bit computers can support practically unlimited amounts of RAM.

Of course, the programs we're dealing with can't use the entire address space; we actually work with "virtual addresses", an abstraction the OS provides user-mode programs (more on this in CS2200). While that last detail isn't really important to us right now, it's good to keep in mind.

- **The value is hidden, encrypted, or obfuscated.**

Some games try to combat memory hacking by changing or hiding their values in memory so they can't be found easily. CSGO's branch of Source Engine, for example, now encrypts the values of all of the console variables (cvars) using XOR. This means that the value in memory doesn't actually equal the value of the variable in-game. Game Maker used to hide values by storing them as one plus twice the actual in-game value. Hence, there might be some formula you have to know to actually scan for the value.

Other times, the value might be getting loaded into memory then immediately getting destroyed. This is hard to deal with, and the only true way to approach values like this is using a debugger, which you will learn how to do later in the course.

- **User error.**

Maybe it really is your fault. :^) Perhaps you're using the wrong scan type or scan value, or maybe you're scanning the wrong process. If the process is 32-bit, you have to use 32-bit Cheat Engine; likewise, with 64-bit processes you must use 64-bit Cheat Engine. If you're stuck, phone a friend.