Starting on the other side of this airtight hatchway: Overwhelming the system

devblogs.microsoft.com/oldnewthing/20221004-00

October 4, 2022

Raymond Chen

A security vulnerability report arrived that went roughly like this:

> The attached script uses the `SendKeys` function to generate random input. Let this script run for a few minutes, and then terminate it. Observe that the system remains sluggish for several minutes after the script stopped running.

The finder never actually stated what the security vulnerability was. They just left it at that with a "So there."

But the reported behavior is not surprising.

The `SendKeys` function is almost certainly a wrapper around the `SendInput` function, and we saw some time ago that `SendInput` adds the injected into to a queue, and when something gets added to a queue, it takes time for it to come out the front of the queue.

Programs are optimized to the point where they can achieve their goals within the allotted interaction class responsiveness targets, but really there's no point optimizing your program so it can respond to a keypress faster than the threshold of human perception.

On the other hand, the script injected input at a rate far faster than humanly possible. If you simulate somebody typing 10,000 characters per second, you're probably going to be injecting the keys faster than the receiving program can process them. If you simulate a million keypresses over a few minutes, it's going to take a long time for the program to get through all of those keypresses.

When the program that is generating the synthetic input exits, it stops producing new injected input, but the old injected input is not recalled. It already got sent into the system, and those keypresses will work their way through the system to their final destination.

It's like writing a program that submits print jobs as fast as it can. You stop it after letting it run for a minute, but in that time, it managed to submit 10,000 jobs. The printer can print only 60 pages per minute, so it'll be a few hours before the printer can print anything else.

The system doesn't cancel a program's print jobs when it exits. Once they are accepted by the spooler, the program isn't needed any more.

No security boundary was crossed here. You ran a program and gave it input injection permission (by virtue of running it at medium integrity), and it injected a ton of input. Code execution leads to code execution. And maybe that's what you wanted: The purpose of the script might have been to automate some lengthy process that requires 10,000 keystrokes.

To prevent this from happening, stop running untrusted scripts in a trusted context.

Bonus bogus security vulnerability: A customer reported that the following batch file renders a system unusable:

```
:loop
start C:\Windows\System32\eventvwr.exe
goto :loop
```

They reported this as a security flaw in Event Viewer.

But really, Event Viewer is doing nothing wrong. You can get the same effect by running a billion copies of any other program. Eventually, they will consume so much memory and CPU that the system becomes unusable. In fact, you didn't even need Event Viewer to get the same effect. Just write a program that spawns a million copies of itself, each one allocating tons of memory in an infinite loop.

Again, no security boundary was crossed here. You ran an untrusted program in a trusted context. Code execution leads to code execution. If you don't want that to happen, don't run random batch files you find on USB sticks on the sidewalk.

By default, Windows lets programs use as much memory as they like. After all, programs are the reason why you bought the computer! What would be the point of adding more memory to your system if the Contoso app couldn't use it?

If you want to run a program in a more restrictive environment, you can put it in a job object with resource constraints.

Raymond Chen

Follow