There Is No Ideal Virus Scanner

Dr. Mark A. Ludwig
September 15, 1992

Abstract

A simple corrolary to the classic halting problem for a Turing machine proves that one cannot write a computer virus scanner which can identify computer viruses with 100% accuracy.

One can mathematically prove that a computer program cannot accurately determine whether or not another given program has a virus in it by passive examination. (The problem is very similar to the halting problem for a Turing machine,[1] and the proof goes along the same lines.) In layman's terms, an ideal scanner is a mathematical impossibility. To demonstrate such an assertion, we first define a virus and an operating environment in general terms:

An operating environment consists of an operating system on a computer and any relevant application programs which are resident on a computer system and are normally executed under that operating system.

A ***virus*** is any program which, when run, modifies the operating environment (apart from itself).

We say that a program P ***spreads a virus*** on input x if running P in the operating environment with input x (designated P(x)) alters the operating environment. A program is ***safe for input x*** if it does not spread a virus for input x. A program is ***safe*** if it does not spread a virus for all possible inputs.

Generally speaking, the input of program P, designated by x, represents all inputs to the program, including any system information provided to the program by the operating system, any keystrokes passed to the program, data files and audio or video input fed to it, data passed to it through the internet, etc. All of this data is digitizable into what is essentially one huge binary number, x. The program P itself is written using the notation of functions since the program, given the input x, will likewise generate some output which can be digitized (provided, of course, that the program halts, providing finite output) and represented as a number.

Obviously ours are very general definitions—more general than we are used to when defining viruses—but they are all that is necessary to prove our point. What we generally call a virus is a program that modifies a part of its environment specifically by infecting another application program, so that the (now infected) application program will in turn modify its environment by infecting the next application program, etc. In other words, our definition of a virus above does not say anything about self-replication. It includes, for example, a Trojan Horse program that simply wipes out all of the application programs. What our definition does is draw a line between an ordinary application program, which makes modifications restricted to its own code and to associated data files, and programs that modify other things. Every self-replicating program that attaches itself to other programs falls within this definition because it either modifies other application programs or the operating system itself. If we can prove a statement about a virus as defined here, it will apply to all self-replicating code such as is normally considered to be a virus, it will apply to all Trojan Horse programs, etc.

One further assumption is necessary: *the assumption that a virus is possible.* This is essentially a function of the operating environment, and it need not always be the case with a computerized device.

For example, if the operating environment could not be modified (as might be the case in a ROM-based microcontroller which never allowed any program but that stored in its ROM to run) then a virus is not possible. Given that a virus is possible, we can state the following theorem:

Theorem: There is no program SCAN(P,x) which will correctly determine whether any given program P is safe for input x.

In order to prove this theorem, let us first invent a numbering system for programs and inputs. Since programs essentially consist of binary information, they can be sequentially ordered: 1, 2, 3, 4 . . . etc. For example, since a program on a PC is just a file of bytes, all those bytes strung together could be considered to be a large positive integer. Most useful programs will be represented by ridiculously large numbers, but that is no matter. Likewise, inputs, which may consist of data files, keystroke, I/O from the COM port, etc., being nothing but binary data, can be sequentially ordered in the same fashion. Within this framework, let us assume SCAN(P,x) exists. SCAN(P,x) is simply a function of two positive integers:

$$\text{SCAN}(P,x) = \begin{cases} 0 \text{ if } P(x) \text{ is safe} \\ 1 \text{ if } P(x) \text{ spreads a virus} \end{cases}$$

We can write SCAN in tabular for like this:

P \ X	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0
2	0	1	1	0	0	0	0
3	1	1	1	1	1	1	1
4	0	0	0	0	0	0	0
5	1	0	0	1	0	0	0
6	0	0	1	0	0	0	0

This table shows the output of our hypothetical SCAN for every conceivable program and every conceivable input. The problem is that we can construct a program V with input x as follows:

$$V(x) = \begin{cases} \text{Terminate if SCAN}(x,x) = 1 \\ \text{Spread a virus if SCAN}(x,x) = 0 \end{cases}$$

(remember, the parameters in SCAN are just positive integers). This construction is known as the Cantor diagonal slash. We have defined a program which, for input x, has

$$\text{SCAN}(V,x) = \overline{\text{SCAN}(x,x)}$$

Thus its values in the table for SCAN should always be *exactly opposite* to the diagonal values in the table for SCAN:

P \ X	0	1	2	3	4	5	6	
0	0	0	0	0	0	0	0	. . .
1	0	0	1	0	1	0	0	
2	0	1	1	0	0	0	0	
3	1	1	1	1	1	1	1	
4	0	0	0	0	0	0	0	
5	1	0	0	1	0	0	0	
6	0	0	1	0	0	0	0	
.					.			
.						.		
.							.	
V	1	1	0	0	1	1	1	. . . ?

The problem here is that—since V is just another program, represented by a number—we must have SCAN(V,V) = $\overline{\text{SCAN}(V,V)}$ as represented by the question mark in the table above. This is an obvious contradiction. Since the construction of V(x) is straightforward, the only possible conclusion is that our function SCAN does not exist. This proves the theorem.

We must conclude that an ideal virus scanner is a mathematical impossibility. **Any real scanner must either fail to catch some viruses or flag some programs as unsafe even though they are, in fact, safe.** Such are the inherent limitations of scanners.

However, all is not lost. Although the program V above beats the scanner SCAN, one can construct a new scanner SCAN2, which can improve on SCAN and in corporate V into its scheme. The trouble is, our theorem just says that there will be some other program V2 that will fool SCAN2. So, although there may be no virus which can fool all conceivable scanners, a new scanner can be written that will catch any given virus. The scanner / virus game is mathematically guaranteed to be endless. That is good news for the antivirus developers, because they'll always be able to sell updates, but bad news for the computer user. One should remember to keep their scanner up to date, and understand that it is *always* vulnerable to new attacks, no matter how bullet-proof anyone claims it is.

Notes:

1. A lucid and easy to find introduction to the halting problem and Turing machines in general is presented in Roger Penrose, *The Emperor's New Mind*, (Oxford University Press, New York: 1989).

2. The theorem and proof presented here are adapted from William F. Dowling, "There Are No Safe Virus Tests," *The Teaching of Mathematics*, (November, 1989), p. 835.