Embrace Your Inner Virus

Michael F. Nowlan and Bryan Ford
Yale University

Introduction

A huge amount of systems research effort is currently spent on fighting viruses and other forms of infective malware. But in our so far vain struggle to eradicate viruses, are we failing to learn important lessons they teach? Might viruses point the way toward the next fundamental and beneficial computing paradigm shift? Consider the many technical advantages viruses offer over other software, both conventional (e.g., MS-Word) and "cloud-based" (e.g., Google Docs):

- Viruses are *easy to install*. You don't have to insert a DVD and run an installer; you don't even have to visit a particular web site. The virus comes to you.
- Viruses are *easy to manage*. You don't have to copy or re-install it if you move to another machine. The virus moves with you automatically via any means available: USB stick, Webmail or Facebook account, etc.
- Viruses are *highly available*. Unlike conventional and cloud-based software, viruses keep operating, communicating, and propagating even among machines that can't reach the Internet or the original provider's site.
- Viruses are *highly resilient*. If a node gets wiped or otherwise loses its state, a virus will just re-infect the node the next time the node communicates.

As networked computing infrastructure becomes ever more crucial to the operation human society at all levels, we must ask ourselves whether we can afford *not* to be using a computing model that offers the above advantages. What we need is not to eradicate viruses but harness them. We need a viral computing model that offers the above properties and two additional ones: *transparency*—visibility into what a virus is doing and what information and computing resources it is using; and *choice*—control over how and when a virus spreads and what information and resources it may access.

Virix: An Operating System to Host Viruses

We propose *Virix*, a new operating system that expects all applications to be viruses. These viral applications, or *vapps*, propagate automatically via any communication or storage channel the OS allows them to use, online or offline. A Virix user creates one or more *profiles* to enforce policies on where a vapp may propagate and what computing resources it may access. Virix assigns vapps to profiles, and determines their resource allocations, according to their origin and perhaps other criteria.

When Alice visits Bob's web site or contacts Bob via E-mail, for example, Alice's Virix nodes allow themselves to become infected with the vapps contained in Bob's public profile. These vapps effectively create an "embassy of Bob" on each of Alice's machines, through which Alice can interact with Bob's public persona via software and communication protocols defined and controlled by Bob. In exchange for devoting some local resources to hosting Bob's public vapps, Alice can use Bob's vapps to find and retrieve (public) information about Bob, or to contact Bob via channels approved by Bob: e.g., only via E-mail and not voice, only during business hours in Bob's time zone, or only via Bob's secretary. If Alice and Bob establish a "friends" relationship, then Alice's nodes might allocate more resources to Bob's vapps, and become infected with the additional vapps comprising Bob's more private "friends profile." Finally, if Alice logs in to her employer's private network from her laptop, her laptop automatically becomes infected with all the vapps comprising her employer's computing ecosystem—or at least the vapps inhabiting the profile representing Alice's job role.

Virix treats new versions of vapps, and changes to their associated data, exactly like brand-new vapp infections. When Bob edits a document associated with a vapp in his profile, Alice's devices pull these changes whenever connectivity permits, making them highly available even under intermittent connectivity.

Vapps and their data propagate relentlessly wherever Virix offers them resources. Once Alice has shown an interest in Bob, granting him a share of her resources, Alice's devices constantly attempt to locate and download Bob's new or changed vapps from *any* machine hosting them—e.g., from Charlie, if Bob is offline but Charlie communicated with him recently. Similarly, Virix embraces the "Windows autorun vulnerability" as a feature, treating the insertion of a CD-ROM or USB drive as a signal to grant a (perhaps small) resource quota to any vapps found on the drive. Bob's devices automatically keep any spare capacity on his USB drives stuffed with his vapps and recent updates, so Bob can easily infect and use his vapps on a new machine—or quickly bring a friend's machine up-to-date—anywhere he goes.

The only way to delete a vapp is by denying it resources. If Alice contacts Bob once and henceforth ignores him, for example, then her resource quota for Bob's vapps will gradually diminish, squeezing them until they can no longer execute, and finally disappear. If Alice does contact Bob again or establishes a long-lived relationship, her quota for Bob increases again, and her system is reinfected or updated with Bob's latest vapps.

For efficient resource sharing, Virix enables vapps to build on other vapps "by reference": e.g., two location-aware applications might (perhaps unknowingly) share a single viral mapping component on each machine they infect. Many software layers such as GUIs and network protocols might be composed this way: in the Virix architecture, "it's viruses all the way down."

The Well-Tempered Virus

Vapps will of course follow the mantra, "Don't Be Evil." But just to keep them honest, Virix should offers users intuitive, easy-to-use policy controls both over the resources they make available to foreign vapps they host, and over which foreign *devices* may host private vapps containing the user's sensitive data.

To control resources, we distinguish *computing resources* from *information resources.* Computing resources are those such as CPU, storage, and network bandwidth, which enable vapps to execute but—barring side-channels—do not grant them information they could not otherwise acquire. Virix by default shares computing resources liberally, automatically granting a controlled share to vapps from any source the user has shown any interest in—and perhaps granting more limited shares to "friends of friends" or even to random strangers.

Information resources, in contrast—such as a keyboard, GPS, or webcam—can potentially produce "sensitive" data, for which Virix adapts recent Information Flow Control (IFC) techniques. A vapp that has been "tainted" with sensitive data becomes a *sensitive vapp*, which Virix allows to propagate in unencrypted form only to devices the user has specifically authorized for that sensitivity level. Encrypted copies of sensitive vapps and their data still propagate freely, however, enabling Alice to use Bob's spare storage capacity—or that of any cloud storage provider—as a storage repository or a "data mule," without having to trust the storage.

Different types of devices will require different and perhaps evolving types of information flow policies and other device-specific behavior. Like everything else in a Virix system, device drivers are viruses—created by the hardware vendor or driver writer—and by default keep themselves up-to-date with no user action.

The Vapp Store: An Economy of Viruses

What kind of business models might be compatible with and foster the development of a (preferably legal) ecosystem of vapps for an operating system like Virix?

An advertising-based model is an obvious possibility: a software vendor embeds in its vapps a reference to an "ad virus," which pulls ads from both ad servers and peers, displaying them when the embedding vapp has access to the display. An ad virus can select targeted ads using any information the user has granted the embedding vapp—but cannot send information back to ad servers except as the user's information flow policies permit.

Today's malware economy might also hint at a next-generation economic model for software. Just as malware writers create and sell malware "toolkits" to spammers, vapp vendors might make their profits by selling access to custom "vapp generators" to non-programmers. For example, Bob purchases a "Skype virus" customized to himself, which provides a "Contact Me" service in his public profile. Bob can then spread his customized vapp to anyone—no DRM or license counting—but it's useful *only* for contacting Bob. If Charlie sees and likes Bob's Skype vapp, he must purchase his own customized version to get one suited for use in his own public profile.

Finally, a vapp economy might even be based on "quid pro quo" or bartering. In exchange for on-demand weather reports, a user might allow WeatherVapp.com access to the external thermometer/barometer connected to the user's device, an otherwise restricted information resource. WeatherVapp's users thus collectively form a distributed sensor network, providing WeatherVapp a goldmine of free information. A parking vapp might offer predictions of available parking spaces, in exchange for GPS information from the user's car indicating where the user has parked (and left) recently, which the vapp uses to make predictions for other users. Finally, vapp vendors can always offer services in exchange for users' computing resources such as storage and CPU time, giving the vendor a massive supply of processing power and/or storage—essentially forming a "legal botnet."

Stop Worrying and Love the Virus

This proposal certainly leaves a few interesting questions unanswered. How should Virix control interaction between vapps? How should Virix and its vapps interact with the emasculated world of non-viral legacy software? What if a bad virus infects a good virus? Once everything is a virus, will anti-virus software vendors die out or become all-powerful? These and other details we leave as exercises for the reader. For now, we merely offer a parting thought: given how pragmatically impossible we have found it to eradicate viruses we *don't* want, who really wants to place a bet against viruses we *do*?