Why Anti-Virus Software Cannot Stop the Spread of Email Worms

Matt Curtin Gary Ellison Doug Monroe
Interhack Corporation
{cmcurtin,gfe,monwel}@interhack.net
http://www.interhack.net/

May 11, 2000

Abstract

With the attention received by the "ILOVEYOU" worm that floated around the Internet in the early part of May 2000, many people are wondering why their anti-virus software didn't prevent them from becoming infected and how they can protect themselves in the future. Here we argue that this approach to the problem, though popular, is fatally flawed and simply *cannot* work.

1 Introduction

Apparently everyone is looking for a solution to the problem of rogue software. When asked how to defend against such attacks, some "experts" will immediately jump into a discussion of firewalls, intrusion detection systems, and anti-virus software. Commonly, you'll also hear the word "vigilance" thrown in there someplace.

The picture is especially grim among end-users and non-expert information technology managers. Experts at least will recognize the roles of policy and education, though some of them need to be prompted to say much on that topic.

We have known about problems like this "in the wild" (as opposed to "in the laboratory") at least since 1988, when graduate student Robert T. Morris released his worm on the Internet. That worm, intended to be harmless, contained a fatal flaw in logic that would cause it to crash the machine it infected.

Before we get too deep into this discussion, we're going to have to spell out some terminology because this article is aimed at non-experts and the media have done such a ridiculous job mangling terms. (Note ye well, would-be defenders of the media's actions in this regard: using the wrong words for things won't make them any more understandable to non-experts. This practice does nothing more than confuse the issue, diluting the precision of our terminology, making it difficult for anyone to determine what is being said.)

1.1 Viruses, Worms, and Trojan Horses (Oh my!)

We have not attempted to compile a comprehensive list of every term used to describe the kind of destructive software that people think about when they heard a word like "virus". We merely want to illustrate the primary types of this software and to explain the primary differences among them so the topic at hand can be clearly understood, irrespective of the reader's background.

Virus A *code fragment* that attaches itself to an executable program. Just as a biological virus does not exist without a "host", neither can a virus exist without some other program to which it can attach. [1]

Worm A *program* that will duplicate itself, usually through some sort of network connection.

Trojan Horse A *program* with a hidden feature. An example would be a program that claims to display something entertaining on the user's screen but secretly deletes the user's files as the expected behavior is taking place.

[1] The plural of "virus", by the way, is "viruses". Neither "viri" nor "virii" make the least bit of sense to anyone but a clueless script kiddie. Tom Christiansen has put this matter to rest, hopefully for good, in "What's the Plural of 'Virus'?", online at http://language.perl.com/misc/virus.html.

Rogue Software Any software whose job is to do something "bad", including viruses, worms, and trojan horses.

Malware This is a relatively new term that has generally been used to apply to software that combines the properties of a virus and a worm. That is, instead of being a standalone executable program that replicates, it's a piece of software that uses a popular "host" like JavaScript, VBScript, or some application macro language to do its work. If part of that work includes replication via a network connection, voilà, you have malware. ExploreZip, Happy99, Melissa, and the ILOVEYOU variants are all examples.

1.2 Detecting Rogue Software

If a computer infected with rogue software were to be continuously reinfected, one of two things would likely happen, exposing the fact that the computer has a problem:

1. The machine would run out of disk space holding instance after instance of the rogue software.
2. The machine would run out of memory or processor cycles needed to manage each instance of the rogue software.

Relatively early examples of rogue software including PC-based viruses and the 1988 Internet Worm made an attempt to stay hidden for as long as possible by taking precautions to see whether the intended target was already infected. Some of these attempts were more effective than others.

Typically, the way to identify whether the intended target was already infected was to examine the machine for a particular "signature", a small stream of data that would be (theoretically) unique to that software. If the signature was found, the software would not install itself. If the signature was not found, the software would continue, installing itself, and looking for new targets.

In today's malware, similar mechanisms *can* exist, but often do not. The primary reason for this is that email-based malware does not tend to burden the individual machine that the victim is running, but rather the servers that provide email delivery services. Further, compared to most PC-based viruses, today's malware is very primitive.

In practice, server-based solutions simply throw away messages whose `Subject` header appears to match one of the known patterns for malware messages. Client-side solutions are not much more sophisticated, typically looking for a combination of factors, or something closer to the root cause, like a client that is attempting to execute some VBScript code attached to email.

Either way, it's an arms race: a significant change by the malware itself, particularly in the case of malware that has the ability to mutate, and the detector—client or server-based—is rendered useless.

This sells lots of software, pays lots of consultants, and it can even put out the fire. But it's no solution to the problem.

1.3 Circumventing the Detector

Attempting to avoid detection, some rogue software will use more sophisticated means of hiding itself. Some can mutate over time so that as detectors are created for the original rogue software, successive generations will change their identity, rendering the detectors useless against the new generations. In practice, virus detectors are able to identify these mutant versions and to stop them, but this is probably only because the virus writers aren't especially clever in their means of mutation.

Solutions for email-based malware typically come in two forms:

1. Either server-based or client-based software that will identify and discard the messages through which the malware attempts to spread and
2. Client-based software that will identify and remove the malware itself, hopefully returning the infected machine to its pre-infection condition.

Generally, the first category is the most important, as this is where both infection and propagation take place. Infected machines aren't generally interesting because the malware has already run its course, doing whatever damage it will to the victim and spreading, often by means of email to everyone in the victim's email address book.

Malware that spreads via email is in a simple, easy-to-identify form. It's a message that claims to be sent from the victim to an individual in the victim's address book. The `Subject` header is typically unchanged, and the text of the message will say something that encourages the recipient to view the attached data. The attached data, when evaluated by the appropriate interpreter, infects the recipient's machine, starting the process over again.

In general, the user will have to run the attachment explicitly. However, a certain feature present in Microsoft's Outlook is that of automatically opening attachments. Microsoft claims this is an "ease of use" feature. We assert that this is nothing more than an "ease of abuse" feature, because it places the same level of trust in data that comes from an unknown source as it has with data from a known source. Blurring the boundaries of "operating system" vs. "application" and "program" vs. "data" is not only generally poor design, but is what makes malware possible.

2 The Issue of Trust

The real issue at stake here is one of trust. Who trusts whom? If the malware were to arrive via email, claiming to be from some random user, of whom the target has never heard, the message would be much more likely to go unread, and the target unaffected. Many people will open the mail, however, because it claims to be from someone they know. (This is an important distinction: without proof, perhaps in the form of a valid cryptographic-strength digital signature, there is absolutely no reason to believe that mail is actually from whom it claims to be.)

2.1 Software Trusts "Local" Data

Computers implicitly trust the data "local" to them. The very programs that they run are in fact data that are from a known source, either the internal hard drive, or perhaps a known and trusted local area network connection. In many cases, this is a generally reasonable thing to do, since a machine whose local data cannot be trusted is likely to have bigger and more serious problems than the sort of thing that the sort of malware we've seen so far covers.

2.2 Software Shouldn't Trust "Remote" Data

With the rise of the Internet, it is now possible for anyone to get data from anywhere. Bad guys and good guys alike are thrown together onto the same, ubiquitous network. Thus, data that comes from an unknown source should be *distrusted,* since it could well be from said bad guy.

The contents of email are such data. Email often originates from outside of the local area network. As such, it could be from literally anywhere, sent by literally anyone. This is why malware spreads: a bad guy has to inject it into the network in the first place. If no one ever trusted any such data, the malware would fail to infect anyone.

Thus, having been bitten by these sort of problems in the past, many users have demanded software that will take precautions to warn the user when something potentially dangerous has been requested. Computists around the world spend countless hours now answering dialogue boxes that say "Are you sure?"

2.3 Software Trusts its Users

Software trusts data it considers to be local. Software generally distrusts data it considers to be remote. But more important than the trust that software places in data is the trust that it places in users. Though certain "dangerous" requests might prompt an "Are you sure?" from the computer, if the user answers "Yes, I'm sure", the computer will typically do its best to fulfill the request. This is as it should be, for computers are the tools of humans.

However, to maintain the integrity of the computer and its data, a basic dependency upon the user is now placed. When the computer asks "Are you sure?" there is never any consideration to the question "is the user qualified to know?" And the answer is that many times, the user is not. The spread of ILOVEYOU and its predecessors is evidence of this.

As long as there are users who can be fooled, malware will continue to plague us. So far we've been very lucky that the malware has been largely benign and too primitive to avoid even the most trivial forms of detection.

3 The Only Solution

Part of the last paragraph is key and it bears repeating here. As long as there are users who can be fooled, malware will continue to plague us. The problem, therefore, can be solved one of two ways:

1. Get rid of the users or

2. Help them to avoid getting fooled.

Drivers of automobiles need not be master mechanics. However, they do need to understand that the ton of steel that they're driving around needs to be used responsibly. Failing to drive responsibly has consequences ranging from minor inconvenience to the loss of human life. (Most of the time the damage will be somewhere in the middle.)

Users of computers need not be master hackers. However, they do need to understand that the hunk of silicon and plastic on their desks needs to be used responsibly. Failing to use computers responsibly has consequences ranging from minor inconvenience to the loss of human life. (Most of the time the damage will be somewhere in the middle.)

3.1 Educate

People who use computers need to understand the risks associated with computing. Some will resist, saying they need not know what the difference is between a *Word* document and a VBScript file in order to accomplish their jobs. They must be corrected and helped to understand the need to compute responsibly.

3.2 Guide

People who use computers need to be guided. That means a *clear* articulation of policy. Buzzword-laden corporate newspeak does not count. Rather than trying to cover every single case, establish general principles that easily translate into practices, without regard to the technology that happens to be popular at the second that the policy was drafted.

3.3 Assist

Only after the users have been educated and guided will technology be able to help curb the flow of malware. Technology itself can always be circumvented by users, so do not attempt to skip directly to this step.

Some technological and architectural considerations that help:

- Properly designed and implemented software. There is no excuse for Microsoft Outlook's "feature" of automatically opening attachments, particularly executable ones. That bypasses the user's ability to exercise any measure of competence and to prevent his machine from being infected. That is a critically stupid misfeature and Microsoft's "defense" ("users requested it") is no defense at all, it's a weak excuse that exposes Microsoft's complete failure to understand computing in a networked world.

- Granting users minimum levels of access. Users who compute in environments where their access to the system is no greater than necessary to do their jobs pose a significantly smaller threat to an organization than users with privileges to do anything. Malware that deletes everything, for example, will only delete one user's files in the former environment. In the latter, it will wipe out everything.

- Diversity in computing platforms. The reason why ILOVEYOU, Happy99, and friends were effective is because they were binary executable programs that would run on the targets' machines. Unix users, for example, could not fall victim to these menaces by virtue of being unable to run the Windows/$x86$ program. The greater the diversity in our computing environments, the fewer number of machines that can be targeted.

- Safer software. What we mean by this is software that will not perform "dangerous" operations without specific authorization from a user with the competence to understand the issues at hand. Where potentially dangerous operations are not excluded by these other principles, they should be allowed only after confirmation that the associated risk is acceptable.

It's time we take a step back from the technology and assess our position. Instead of insisting on more and more features from vendors and giving them less and less time to implement them, we need to focus on the *correctness* of our systems' designs and implementations. People need to understand what they're using and how to avoid falling victim to these problems.

We need to get away from the syndrome of thinking that whatever the "computer says" is correct. When people are using tools that take reasonable precautions against doing the wrong thing and they understand how not to compromise the integrity of their systems, we'll all be in a much better position.