Open Problems in Computer Virus Research

Steve R. White
IBM Thomas J. Watson Research Center
Yorktown Heights, NY
USA

Abstract

Over a decade of work on the computer virus problem has resulted in a number of useful scientific and technological achievements. The study of biological epidemiology has been extended to help us understand when and why computer viruses spread. Techniques have been developed to help us estimate the safety and effectiveness of anti-virus technology before it is deployed. Technology for dealing with known viruses has been very successful, and is being extended to deal with previously unknown viruses automatically. Yet there are still important research problems, the solution to any of which significantly improve our ability to deal with the virus problems of the near future. The goal of this paper is to encourage clever people to work on these problems. To this end, we examine several open research problems in the area of protection from computer viruses. For each problem, we review the work that has been done to date, and suggest possible approaches. There is clearly enough work, even in the near term, to keep researchers busy for quite a while. There is every reason to believe that, as software technology evolves over the next century or so, there will plenty of important and interesting new problems that must be solved in this field.

Introduction

Some people believe that there is no longer any interesting research to do in the field of protection from computer viruses - that all of the important technology has already been developed - that it is now a simple matter of programming to keep up with the problem. Others believe that "virus research" simply means "analyzing viruses." To dispel these misimpressions, we discuss several important research problems in the area, reviewing what is known on each problem and what remains open.

The purpose of this paper is not to give solutions to these problems. Rather it is to outline the problems, to suggest approaches, and to encourage those interested in research in this field to pursue them.

The problems we have selected have two characteristics. The first is that, if the problem were solved, it would significantly improve our ability to deal with the virus problem as it is likely to evolve in the near future. The second is that the problem constitutes an actual research problem, so that a definitive solution would be publishable in peer-reviewed computer science journals, and could form the basis for an M.S. thesis or, in some cases, a Ph.D. thesis.

We discuss five problems:

1. As more viruses are written for new platforms, new heuristic detection techniques must be developed and deployed. But we often have no way of knowing, in advance, the extent to which these techniques will have problems with false positives and false negatives. That is, we don't know how well they will work or how many problems they will cause. We show that analytic techniques can be developed which estimate these characteristics and suggest how these might be developed for several classes of

heuristics.

2. We have a reasonable, qualitative understanding of the epidemiology of computer viruses, characterizing their spread in terms of birth rate, death rate, and the patterns of program transfer between computers. But a mystery remains. Evidence suggests that viruses are still relatively uncommon - that their prevalence has always been very low. But, according to our current theories, this can only happen if the birth rate of viruses is ever so slightly higher than their death rate, a coincidence too remarkable to believe. We discuss effects that might be responsible for this puzzling observation.

3. We are in the process of deploying digital immune system technology that finds new viruses, transmits them to an analysis center, analyzes them, and distributes cures worldwide, automatically, and very quickly. The current architecture for this system uses a centralized analysis center for a variety of good reasons. But a more distributed approach, perhaps even a massively distributed approach, has advantages as well. We outline the system issues that must be considered, and what simulation results would be useful, in understanding the tradeoffs.

4. There have been thankfully few instances of worms - freestanding virus-like programs that spread themselves and may never be present in the computer's file system at all. Yet virtually all of our anti-virus technology relies on detecting and removing viruses from a file system. We discuss the new problems that worms engender, and suggest some of the new technology that may be needed to deal with them.

5. Current anti-virus technology is largely reactive, relying on finding a particular virus before being able to deal with it well. Modern programming environments can give rise to viruses that spread increasingly rapidly, and for which a reactive approach becomes ever more difficult. We review the history of pro-active approaches, showing why traditional access controls are basically useless here, and describe newer approaches that show promise.

Analyzing Heuristic Detection Methods

Over the past ten years, a single method of detecting computer viruses has nearly eclipsed all others: scanning for known viruses. Originally, a string of bytes was selected from some known virus, and the virus scanner looked for that string in files as a way of determining if that file was infected with that virus. Later, more complex techniques were developed which involved looking for various substrings in various parts of the file. But all of these techniques have one thing in common: they look for static characteristics of viruses that are already known.

In that same ten years, around twenty thousand different viruses were created. How could a method that only deals with already-know viruses be effective in an environment with so many new viruses? The reason is simple: over the past ten years, only a few hundred of these viruses have actually been seen in real customer incidents (these are the viruses that are “in the wild”). Even those spread rather slowly on a global scale, typically requiring months or years to become prevalent around the world. This provided the anti-virus industry plenty of time to discover a new virus, derive a cure, and make it available – all before very many PCs had been infected.

The anti-virus industry also developed methods for detecting previously unknown viruses. These methods are usually called “heuristic” methods because they are, by their nature, inexact.[1] Heuristics are on the horns of the same dilemma as any other virus detection method: detecting as many viruses as possible while having as few false positives as possible. Authors of heuristics have dealt with these problems in two different ways. Some have conducted extensive beta tests of new heuristics, and tried to tune their heuristics to have acceptable false negative and false positive rates. Others have given up on selecting a single good trade-off and have let users try to make this trade-off themselves by adjusting parameters in the anti-virus program.

But the virus landscape is changing. No longer are we dealing with simple DOS file and boot viruses. Excel and Word macro viruses are currently the most prevalent kinds of viruses. Windows NT viruses are starting to be written. We have seen the first attempt at a Java application virus. And on the horizon are entirely new kinds of viruses that will take advantage of the Internet to spread themselves. Future kinds of viruses will arise and become widespread much more quickly than in the past. It is important that we have ways to find new instances of these viruses before they spread globally. We may not have the luxury of lengthy beta periods to help tune our heuristics to eliminate false positives. And we certainly can't expect users to be sophisticated enough to tune dozens of different, complex heuristics if the authors of the heuristics are unable to do so.

The difficulty is that very little work has been done in this area. Apart from experience with individual heuristics as they are used in individual products, we don't know how well they will work or how many problems they will cause. In fact, since few heuristics have been described in the open literature, it is hard to know how good even current heuristics are. To further complicate matters, virtually all heuristics in use today have been developed without regard to the ability to estimate their false positive and false negative rates before they are in wide-scale use.

So the challenge is to develop classes of broadly useful heuristics that can be understood analytically before they are deployed and, preferably, updated as the threat evolves without requiring entirely new methods.

One possible starting point is a heuristic based on traditional signatures, but signatures that are common to large classes of already-known viruses. Combinations of these signatures can detect variants of viruses in these classes. Probabilities that individual string signatures will cause false positives in non-infected files can be estimated with techniques that have already been developed. [2] Estimating false negative probabilities relies on characterizing new viruses in these classes as they appear over time.

A second possible starting point is to use neural networks to attempt to distinguish infected from uninfected files. This approach has already proved very successful for DOS boot viruses.[3] Neural networks have been studied as general classifiers for many years. Techniques are available for estimating the false positive probabilities of neural networks trained on a given distribution of positive and negative examples. Most of these estimates, however, focus on giving worst-case bounds on the false positive probabilities. As such, these bounds are rather loose. [4] An important part of this proposed analysis would be to establish expected-case estimates of false positive probabilities for the cases of interest.

Mysteries of Epidemiology

Models of computer virus epidemiology – how viruses spread – have been constructed, based on models for biological epidemiology. These models provide a reasonable qualitative understanding of the conditions under which viruses spread and why some viruses spread better than others. But our understanding remains incomplete in important ways.

The model we have today characterizes virus spread in terms of three things:[5]

1. The birth rate of the virus – the rate at which one infected system can spread the infection to other systems.
2. The death rate of the virus – the rate at which an infection is found on a system and eliminated.

3. The pattern (topology) of connections between systems along which viruses spread, whether by program sharing via diskettes, sending infected attachments via email, or other connections yet to come.

If we look at a very simplified model, in which any system is equally likely to infect any other system in the world, then there is a simple way to express how viruses spread.

If the death rate is larger than the birth rate, then the virus fails to spread globally. It may spread to a few other systems at first, but it will eventually be found on all infected systems and eradicated. It will become extinct until such time as a new copy of the virus is introduced into the world's systems.

If the birth rate is larger than the death rate, then the virus might spread, though this doesn't always happen. Sometimes, the virus is unlucky. It is found and eliminated from the first few systems it infects, and it does not end up spreading widely. Sometimes, it survives this tenuous childhood and spreads pervasively throughout the world. The probability that the virus will get lucky and spread pervasively after a single initial system is infected is given by:

$$1 - (\textit{death rate})/(\textit{birth rate})$$

If it does spread pervasively, it rises in prevalence until the chance of finding and disinfecting a system is the same as the chance of spreading the infection to an uninfected system. Then it stays at this level of prevalence. While some systems may be found and disinfected, other systems will be newly infected to take their place and the fraction of infected systems remains about the same. In this simple model, the fraction of systems infected when this equilibrium is reached is also expressed by:

$$1 - (\textit{death rate})/(\textit{birth rate})$$

In more complex models, which take into account more complex patterns of connections between systems, the details of the above results change. But, in the models examined to date, there is always a sharp transition between extinction and pervasive viral spread, and there is always an equilibrium prevalence that depends sensitively on the birth and death rates.[6]

Now here's the mystery. Our evidence on virus incidents indicates that, at any given time, few of the world's systems are infected. Similarly, we observe that very few of the known viruses have gotten lucky enough to spread pervasively in the world. If these viruses have reached equilibrium in their life cycle then, according to our current models, these two things are only possible if the birth rate is almost identical to the death rate, though just a tiny bit larger, for virtually every virus.

No one has come up with an explanation of why the birth rate should be just infinitesimally larger than the death rate and, if fact, such an explanation seems entirely unlikely. Rather, it points to something wrong with the model. This lapse is disconcerting. If the model fails to predict that virtually all current viruses will be very low in global prevalence, it probably won't help us understand the conditions under which viruses could rise to dangerous levels of prevalence.

One possibility is that viruses never actually reach equilibrium. Rather, they start to rise in prevalence as predicted, then something intervenes. Perhaps the dominant operating system in the world changes from DOS to Windows 3.1, all but eliminating DOS file viruses. Perhaps more people start updating their anti-virus software, eliminating the macro viruses that were so prominent just a year ago. But the viruses never become prevalent enough to reach equilibrium.[7] One consequence of this possibility is that environmental changes that are hostile to the current crop of virus are vital in controlling viral epidemics. Indeed, if it were not for these changes, viruses might soar to dizzying heights of prevalence. This theory, however, does not appear to explain why so few viruses have "gotten lucky," an observation that only seems consistent with the birth rate being close to the death rate.

So it would be interesting to find a model of virus spread that naturally explains today's low level of prevalence, and helps us understand the conditions under which higher levels of prevalence may be reached. Here are two possible paths to such a model.

Differences in Infection Rates

In the simple models that have been examined to date, all of the systems are assumed to have the same viral birth and death rates. Some systems are assumed to have connections to more systems than others, corresponding to users who share software or documents more than others. But even those more-connected users are assumed to be scattered randomly in the population. In the real world, some subpopulations are more prone to infection that others. Anecdotal evidence suggests that college campuses tend to have a high degree of sharing of programs and documents. It may be that there are subpopulations that are constantly infected, and constantly spreading the infection beyond them, even though the world as a whole has a very low infection rate.

Taking this into account in a model is a matter of assigning a higher birth rate, a lower death rate, or a higher connectivity to one or more subpopulations. In a model in which every system is equally likely to infect every other system, a single system that is constantly infected would provide an ongoing source of viruses to everyone else. But in more realistic models, in which spread is more localized, a single infected system or group of systems would only infect their immediate neighbors. They would not provide a source of ongoing spread for the entire population unless the population itself promoted pervasive virus spread, and that would bring us back to the same mystery as before.

Topologies that Limit Spread

Models that take into account more localized spread are more realistic. They show effects that we see in real viral spread, such as very gradual increases in viral prevalence over time. The more connected a subpopulation is, the easier it is for a virus to spread pervasively within it. You may exchange lots of documents within your office in Paris, for instance, and have constant virus infections because of it, even though you never exchange documents with people in New Zealand, and hence don't contribute directly to the infection rate there.

One possibility is that there are well-connected subpopulations that maintain ongoing virus spread, and act as sources of viruses to the rest of the world. This could be studied by modeling viral spread on hierarchically clustered graphs, which take locality into account in a uniform way. Preliminary studies on such graphs have been done, but a comprehensive understanding awaits further work.

Both May Be Needed

Simple hierarchical clustering may also not be enough. It is likely to fall victim to the same difficulty as simple differences in infection rates: even though a subpopulation may stay infected, the model provides no obvious way of explaining a low infection rate in the rest of the world.

It is likely that both mechanisms must play a part in a successful model. That is, it may be that we need to assume hierarchically clustering, but also assume that some fraction of systems within each subpopulation has a higher than normal birth rate or a lower than normal death rate. That might provide a mechanism for viruses to reach the many subpopulations, while not overwhelming any of them. Indeed, it might result in a model in which a fairly large change in infectivity of a virus causes only small changes in its prevalence. And this might, in turn, explain the stability of a low level of viral prevalence in the world.

Distributed Approaches to an Immune System

Along with our colleagues at Symantec, we at IBM Research are in the process of developing and deploying a digital immune system.[8] This system is capable of finding previously-unknown viruses and sending them safely over the Internet to an analysis center, where they are analyzed, detection and disinfection information is extracted from them, and the resulting cure is sent back to the infected systems, and then around the world. The entire process can occur automatically, and very quickly – more quickly than the virus itself can spread.

In our first implementation of this system, we have chosen to use a single, centralized analysis center. This choice was made for a number of good reasons, including security, simplicity and ease of modification. This is not, however, the only way to organize such a system, nor is it necessarily the best. The mammalian immune system, for instance, analyzes viruses in organs called lymph nodes, which are found in various places around the body. We could even consider a massively distributed system, in which there is a tiny analysis center associated with each computing device in the world.

There are good reasons for distributing the task of analysis, including increasing the amount of computing power available to the task, introducing redundancy of a critical component, reducing network latency and load balancing. With these advantages come challenges. How are the results of one analysis center communicated with the others? What are the effects of lack of synchronization between the analysis centers?

Until we have sufficient data from our initial deployment of the digital immune system, it will be difficult to judge many of these trade-offs in today's environment. However, it would be very worthwhile to characterize the conditions under which a centralized organization is optimal, and those in which more distributed approaches are best. Such a characterization would involve a detailed understanding of the system considerations listed above. It would also involve both an analytic understanding, and simulation results, of the response of such systems in a variety of important conditions.

A digital immune system must handle routine virus incidents, of course, speeding up the submission of a new virus sample to an anti-virus vendor and speeding up the response. But its more vital role is in dealing well with floods of viruses and related conditions – conditions under which the anti-virus industry's current methods of dealing with viruses simply break down.

As the Internet becomes the common vehicle for communication in the world, and as more and more people use it, digital communication will increase vastly in scope and speed. As this happens, new kinds of viruses will take advantage of this increase to spread much more broadly, and much more quickly, than present-day viruses. These viruses will spread to thousands of systems in a matter of minutes, and around the world in a matter of hours. When this happens, a digital immune system will be inundated with thousands of instances of an apparently new virus within the first hour. And all of these will have come from worried customers who want a solution as quickly as possible.

Similarly, we have become used to periodic virus "scares," in which stories of a newer virus make their way into the popular media, accompanied by warnings of certain and sudden doom. These warnings occur whether or not the virus has ever been seen, or will ever be seen, in the wild. You may remember Michelangelo Madness in 1992, one of the first such scares.[9] In such a scare, millions of people want to make sure that their anti-virus software can protect them from this allegedly deadly threat. This gives rise to "download storms," in which millions of requests for updated virus definitions must be handled in a very short time.

A simulation study of centralized versus distributed immune system approaches should help us understand when flooding conditions like these are handled properly in the various models.

Dealing with Worms

There have been thankfully few instances of worms - freestanding virus-like programs that spread themselves, without relying on the actions of people or other programs. The worms we have seen, such as the infamous Internet Worm of the late 1980's, spread through systems around the world in a matter of hours.[10] They can be a significant problem.

Almost all of our current anti-virus technology is directed towards viruses rather than worms, because we have mostly seen viruses and not worms. It continues to be possible to write worms in various environments today, and we believe these possibilities will increase in the future as Internet-based systems become more functional.

Worms present their own technological challenges, some of which are not addressed well by current technology.

One common thread in these problems is that worms incidents tend to be episodic. A given worm is created and unleashed. It spreads very quickly through its target systems, in many cases creating a much more substantial problem than a run-of-the-mill PC virus. Eventually, it is discovered and eliminated, and steps are taken to help ensure that that particular worm cannot recur. Unlike today's PC viruses, which often constitute a low-level, ongoing infection of the world's systems, worms tend to come and go and not come back. By itself, this is good. But it also means that every worm we see will be a new worm. Unlike PC viruses, from which you can be protected because I happened to find the virus months ago and create a cure, the anti-virus industry will not have time to craft a cure for a worm before you get it.

Another common thread is that worms can be written entirely as active processes, without having to create or modify files in the file system. Almost all of our current anti-virus technology relies on the virus being present in the file system. We examine suspicious changes in files, look in files for specific viruses, and eliminate viruses by changing the files they have infected.

We discuss three problems relating to worms, and suggest a course of investigation.

Detection

Detecting a worm runs into two difficulties. First, it is likely to be new, so techniques that detect previously known worms may not be helpful. Instead, detection heuristics must be developed that anticipate the kinds of worms that will be written. Second, it is likely to be a collection of active processes, and not be present in the file system. So detection methods will require access to the list of active processes, and to the active process space itself, in order to examine them.

Even so, detection may prove very difficult. It is common for processes on a modern system to spawn other processes, but that is precisely what worms do. If mobile code systems become commonplace, it will become common for processes on one system to spawn processes on another system. How can we distinguish this legitimate activity from the actions of worms?

One possibility is to build on the work that has been done in intrusion detection. The goal of intrusion detection is to determine when another user is using your system without permission, perhaps after guessing your password and logging on. Intrusion detection software often relies on making a model of your typical behavior and noticing when atypical actions occur. For instance, it would be quite common for me to use an email program at 3:00 PM on a Wednesday, or to play Quake at midnight on Friday, but it is almost inconceivable that I would be using a C compiler at 6:00 AM on a Saturday. Similarly, worms are likely to do very different things than individual users do. It may be possible to get a hint that something worm-like is going on by examining actions outside of a given user's profile.

Even that may be insufficient. It may be that it is only possible to detect the presence of a worm by examining the collective behavior of many systems at once. It may be common, for instance, for users in an organization to start programs on remote systems occasionally, but it may be wildly unlikely for thousands of such remote programs to be started by thousands of users within a few minutes of each other. That might point to a worm in action. It will be an interesting challenge to create a distributed system that can monitor collective behavior and make intelligent decisions about when the behavior is unusual.

Analysis

Once you believe a worm might be present on a system, just capturing a sample of it might be difficult. It is likely to be a collection of active processes. How do you capture its state? How do you know you have all of the relevant state? What if the processes are distributed among a number of systems, so you only have all the relevant pieces of the worm if you capture all the relevant active processes at the same time?

Once you have a sample, the next challenge is analyzing it automatically. (Remember that worms spread very quickly. To deal with them quickly enough, analysis will have to be automated.) You may have to create an instrumented, virtual environment that mimics the natural environment of the worm but allows you to watch what it is doing at the system level. You may have to do this for a great many different environments in which worms can exist.

Disinfection

Suppose you have successfully completed analysis for a particular worm. How do you get rid of it on all of the infected systems? At the very least, your disinfection software will require access to the active task list, and perhaps to the active processes themselves. What if the worm produces new processes to replace any of its processes that it notices have been terminated? What if the worm is distributed among a number of systems – how do you coordinate the disinfection across all of these systems?

A Course of Investigation

Clearly, this is a large and complex problem. It would be possible to spin a long list of things worms could do, and ways in which worm writers could make ever nastier and harder to find creatures. But we strongly advise against publishing work like this, as it only serves to make the problem worse.

Instead, it is clear that significant work needs to be done in each of the above areas to develop protective technologies. The space of possible worms is so large that we suggest focussing in on prototyping protective technologies for one system and one type of worm. Lessons learned from that exercise should be useful in generalizing the work to other systems and other types of worms.

Pro-Active Approaches to Controlling Viruses

Current anti-virus technology relies almost entirely on finding a particular virus before being able to deal with it well. As such, it is largely a reactive technology. Customers are required to update their anti-virus software periodically to deal with new threats. Customers (and anti-virus marketers!) have long desired anti-virus solutions that did not require constant updates. Some anti-virus vendors have gone so far as to claim that their products could detect all possible viruses, never make mistakes, and never need updates, a claim that can be easily shown to be mathematically impossible.

Somewhat surprisingly, this reactive approach has been very effective in dealing with viruses in the past, as they spread slowly and gave the anti-virus industry time to react. As discussed in this paper, we expect future generations of viruses to spread much more quickly, and we have developed digital immune system technology to react quickly to these threats. It is possible, however, to conceive of situations in which viruses spread even more quickly than could be handled by the fastest reactive technology. In these situations, it would be useful to have pro-active technology that could eliminate, or at least limit, the threat.

Previous work along these lines has been largely limited to the use of traditional access control mechanisms to limit viral spread. The idea was that limiting who could read, write or execute files could limit how far a virus could spread. In principle, it is possible to create systems in which strict mandatory access controls could prevent viruses from spreading. Unfortunately, experiments with real-world systems have demonstrated that this approach is virtually hopeless in practice. [11] There are, however, two approaches that may hold some promise.

Limited-Function Environments

If, in a given programming environment, it can be proven to be impossible to create or modify programs, then that programming environment is incapable of spreading viruses. If, for instance, a programming environment were limited to displaying dancing icons on the screen, and if it could not otherwise access the screen buffers, then it would not be capable of spreading viruses. [12] The Java “sandbox” model is a good example of this kind of limited function, as is the Web markup language HTML.[13]

Unfortunately, there is always a desire on the part of developers and customers to extend limited-function languages to be more functional. Witness the loosening of the original Java security model to include options that permit Java applets to access any part of the system they want. There is a good reason for this: fully functional languages can do lots of useful things. Of course, they can also support viruses. And that’s the problem.

Because of the continuing desire to make languages fully functional, we expect that most major programming environments will remain functional enough to support viruses. There will be, however, niches in which limited-function environments are useful and acceptable.

It would be very interesting to examine problem areas in which environments can be useful, but limited so as not to support viruses. This is particularly important in the area of mobile code on the Internet, in which programs can move from system to system. A way of preventing these programs from creating the obvious virus threat is vital. Classification of these functional areas and strategies for limiting environments would give system designers critical information in order to create these new environments safely.

Granular Execution Controls

Traditional access controls, focussing on permissions to read, write and execute, are not successful in preventing viruses in typical present-day systems. This may be because they do not offer sufficiently granular control. It may be necessary for you to execute a program that I own, but it is unlikely that the program must have full access to your entire system. Instead, it may be possible to limit its access to only those parts of the system, and only those actions, that are critical to its operation. Conversely, it may be possible for you to specify which parts of your system may be accessed, and what kind of access is permitted.

By itself, this is not very useful. What is important in controlling the spread of a virus is not the behavior of the virus on any individual system, but rather its global behavior. As with discretionary access controls, letting each system owner set their access controls independently typically results in sufficient connection between systems to permit global virus spread.

Instead, as with mandatory access controls, limiting global spread requires global control. Within a company, say, it may be possible to limit programs within a particular environment from creating or modifying other programs in that environment, unless those programs are authorized by a company authority. This prevents the guy down the hall from spreading a virus in that environment.

This approach differs from limiting the function of an environment in that particular designated authorities are permitted to make exceptions. The full function of the environment is still available to appropriately authorized programs, but only to those programs.

It would be interesting to understand what kinds of granular execution controls are useful in real life. How limiting can these controls be and still permit organizations to develop software, do workflow, write fancy Web applications, or any of the hundreds of other useful things that need to be done? In each case, is it possible to limit what programs can do so that you can prove that viruses cannot spread? Given that authorizing parties must be in place to make exceptions to these global restrictions, how do you make the trade off between convenience for end-users and control by the organization? Finally, it would be interesting to study what restrictions could be imposed on programs that come from outside the organization to ensure they are safe while still allowing them to be useful.

Conclusion

We have examined a few open problems in computer virus research. Our hope is to convince those who would like to do research in this area that there are significant problems yet to be addressed, and that these require dedicated work by clever people. The field is by no means complete, and easily anticipated problems in the relatively near future will require substantial new invention to avoid substantial problems with new viruses.

Beyond the near future lies a vast and ever-changing landscape, in which today's simple programs will evolve into complex ecologies of interrelated processes that span the planet, interacting with each other in ways we cannot anticipate. And inevitably, lurking within these ecologies, will be the conceptual descendents of today's viruses, spreading wherever they can. The battle between microbes on the one hand, and plants and animals on the other, has raged in the biological world for millions of years. There is every reason to believe that the battle in cyberspace will also be long-lived, and will continue to require insight and invention for a long time to come.

Acknowledgements

The author thanks William C. Arnold, David M. Chess, Jeffrey O. Kephart, John F. Morar and Gerald Tesauro for helpful conversations.

Notes and References

[1] This use of "heuristic" to apply only to methods of detecting previously unknown viruses ignores the fact that scanning methods are also inexact. They can miss instances of viruses, e.g. slowly-polymorphic viruses if the virus analyst failed to discover this property in a virus – the "false negative" problem. More commonly, they can falsely accuse a file of being infected merely because it contains a certain string of bytes – the "false positive" problem. Nevertheless, we use this common nomenclature here.

[2] Jeffrey O. Kephart and William C. Arnold, “Automatic Extraction of Computer Virus Signatures,” Proceedings of the Fourth Virus Bulletin International Conference, R. Ford (ed.), Jersey, UK, Sept. 7-8, 1994, pp. 179-194.

[3] Gerald Tesauro, Jeffrey O. Kephart and Gregory B. Sorkin, “Neural Networks for Computer Virus Recognition,” IEEE Expert, Vol. 11, No. 4, Aug. 1996, pp. 5-6.

[4] Gerald Tesauro, private conversation.

[5] J.O. Kephart and S.R. White, “Directed-Graph Epidemiological Models of Computer Viruses,” Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, California, May 20-22, 1991, pp. 343-359.

[6] J.O. Kephart, “How Topology Affects Population Dynamics,” Proceedings of Artificial Life 3, Santa Fe, New Mexico, June 15-19, 1992.

[7] David M. Chess suggested the possibility that current viruses have never reached equilibrium.

[8] Jeffrey O. Kephart, Gregory B. Sorkin, Morton Swimmer and Steve R. White, “Blueprint for a Computer Immune System,” Proceedings of the Virus Bulletin International Conference, San Francisco, California, October 1-3, 1997.

[9] Michael W. Miller, “’Michelangelo’ Scare Ends In an Anticlimax,'” The Wall Street Journal, March 9, 1992, pp. B5.

[10] E.H. Spafford, “The Internet Worm Program: An Analysis,” ACM Computer Communication Review, Vol. 19 No. 1, pp. 17-57, Jan 1989.

[11] Fred Cohen, “Computer Viruses: Theory and Experiment,” Computers & Security, Vol. 6, pp. 22-35, 1987.

[12] In principle, our dancing icon program could write information to the screen buffer that could be used by a program outside the environment to create a virus, if that other program could read the screen buffer. But no virus can be created in the dancing icon environment itself.

[13] John F. Morar and David M. Chess, "Web Browsers – Threat or Menace", Proceedings of the Virus Bulletin International Conference; Munich, Germany; October 1998. Preprint.