Published in Artificial Life IV, Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, Rodney A. Brooks and Pattie Maes, eds., MIT Press, Cambridge, Massachusetts, 1994, pp. 130-139.

A Biologically Inspired Immune System for Computers

Jeffrey O. Kephart
High Integrity Computing Laboratory
IBM Thomas J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598

Abstract

Computer viruses are the first and only form of artificial life to have had a measurable impact on society. Currently, they are a relatively manageable nuisance. However, two alarming trends are likely to make computer viruses a much greater threat. First, the rate at which new viruses are being written is high, and accelerating. Second, the trend towards increasing interconnectivity and interoperability among computers will enable computer viruses and worms to spread much more rapidly than they do today.

To address these problems, we have designed an immune system for computers and computer networks that takes much of its inspiration from nature. Like the vertebrate immune system, our system develops antibodies to previously unencountered computer viruses or worms and remembers them so as to recognize and respond to them more quickly in the future. We are careful to minimize the risk of an auto-immune response, in which the immune system mistakenly identifies legitimate software as being undesirable. We also employ nature's technique of fighting self-replication with self-replication, which our theoretical studies have shown to be highly effective.

Many components of the proposed immune system are already being used to automate computer virus analysis in our laboratory, and we anticipate that this technology will gradually be incorporated into IBM's commercial anti-virus product during the next year or two.

1 Introduction

Unique among all forms of artificial life, computer viruses have escaped their playpens and established themselves pervasively throughout the world's computing environment. Of the roughly 100 to 200 million PC and Macintosh users in the world, at least several hundred thousand, and perhaps over a million, have been afflicted at one time or another. Computer viruses have found a niche on all of the world's continents, including Antarctica [1][1], and most of its countries.

[1]The "Barrote" virus was discovered at Spanish and Argentinian scientific bases in Antarctica when it triggered on January 5th, 1994. Machines booted on or after that date displayed a pattern of jail-like bars with the legend "Virus Barrote" (Spanish for "Virus Jail"), and would halt the PC (and thus any scientific experiments that were being conducted).

A sufficiently amoral artificial life enthusiast might view the success of these artificial creatures in the real world as amazing, amusing, and admirable, but most responsible citizens regard computer viruses (and those who write them) with abhorrence. Even though just a small minority of viruses are intentionally harmful, the vast majority of them are poorly-written, poorly-tested, buggy pieces of software that create problems that are often time-consuming to diagnose. According to a Dataquest survey [2] and spokesmen for several different insurance companies [3], a virus spreading among several PC's in a company costs (on average) several thousands of dollars in down-time and data lossage; one company interviewed by Dataquest reported a $2 million dollar loss due to a single incident. At least one insurer offers a $100,000/year policy for damage due to computer virus infection [3].

Computer viruses are serious business. They have engendered an entire anti-virus industry, consisting of hundreds of researchers and developers who are employed by dozens of companies around the world. At least one such company, devoted almost exclusively to anti-virus software, is traded on the Nasdaq stock exchange.

Currently, the arms race between virus authors and anti-virus developers is roughly even. During any particular moment, it is typical for a few viruses to be increasing in prevalence, and other formerly prevalent ones to be on the decline [4]. However, two alarming trends threaten to turn the balance in favor of virus authors:

1. The rate at which new viruses are being written is quite high, and appears to be accelerating. Human experts who analyze and find cures for viruses are already swamped, and their ability to keep pace with the large influx of new viruses is being questioned.

2. The continuing increase in interconnectivity and interoperability among the world's computers enhances the ability of any particular virus to spread, and the rapidity with which it does so. The current strategy of periodically distributing updates to anti-virus software from a central source will be orders of magnitude too slow to keep up with the spread of a new virus.

In the near future, computers will somehow need to automatically recognize and remove previously unknown viruses on the spot soon after they are discovered. Fortunately for us, Nature has already invented a remarkably effective mechanism for recognizing and responding rapidly to viruses and other undesired intruders, even in cases where the intruder has never been seen before: the vertebrate immune system. The success of the vertebrate immune system in protecting its host from a wide array of viruses and other undesirables that are continually mutating and evolving has inspired us to design and implement an immune system for computers that is founded on similar principles. Various components of the immune system are already being used to automate the task of computer virus analysis in the laboratory. Over the next year or two, the immune system will be phased gradually into IBM's anti-virus software.

This paper is organized as follows. Section 2 briefly discusses the two trends mentioned above, and why they threaten to overwhelm current anti-virus technology. Appealing to biological analogy, section 3 motivates and presents a biologically inspired design for an immune system for computers and computer networks. Section 4 concludes with a brief discussion of important issues that remain to be resolved.

2 Why current anti-virus techniques are doomed

There are a variety of complementary anti-virus techniques in common usage [5, 6]. *Activity monitors* alert users to system activity that is commonly associated with viruses, but only rarely associated with the behavior of normal, legitimate programs. *Integrity management systems* warn the user of suspicious changes that have been made to files. These two methods are quite generic, and can be used to detect the presence of hitherto unknown viruses in the system. However, they are not often able to pinpoint the nature or even the location of the infecting agent, and they often flag or prevent legitimate activity, and so can disrupt normal work or lead the user to ignore their warnings altogether.

Virus scanners search files, boot records, memory, and other locations where executable code can be stored for characteristic byte patterns that occur in one or more known viruses. They tend to be substantially less prone to false positives than activity monitors and integrity management systems. Scanners are essential for establishing the identity and location of a virus. Armed with this very specific knowledge, *repairers*, which restore infected programs to their original uninfected state, can be brought into play. The drawback of scanning and repair mechanisms is that they can only be applied to known viruses, or variants of them; this requires that scanners and repairers be updated frequently.

Debates over the relative merits of the various anti-virus techniques have largely subsided, and many of the major anti-virus vendors now offer packages that usefully integrate scanners and repairers with activity monitors and integrity management systems.

In the remainder of this section, I shall describe the typical method by which scanners and repairers are updated, and demonstrate why it can be expected to become untenable in the near future, given projected trends in viral influx and increased interconnectivity among computers.

2.1 Virus scan/repair updates

Whenever a new virus is discovered, it is very quickly distributed among an informal, international group of virus collectors who exchange samples among themselves. Many such collectors are in the anti-virus software business, and they set out to obtain information about the virus which enables:

1. detection of the virus whenever it is present in a host program, and

2. restoration of an infected host program to its original uninfected state (which is usually possible.)

Typically, a human expert obtains this information by disassembling the virus and then analyzing the assembler code to determine the virus's behavior and the method that it uses to attach itself to host programs. Then, the expert selects a "signature" (a sequence of perhaps 16 to 32 bytes) that represents a sequence of instructions that is guaranteed to be found in each instance of the virus, and which (in the expert's estimation) is unlikely to be found in legitimate programs. This "signature" can then be encoded into the scanner, and the knowledge of the attachment method can be encoded into the repairer.

Such an analysis is tedious and time-consuming, sometimes taking several hours or days, and even the best experts have been known to select poor signatures — ones that cause the scanner to report false positives on legitimate programs.

2.2 Viral influx and its consequences

One reason why current anti-virus techniques can be expected to fail within the next few years is the rapid, accelerating influx of new computer viruses. The number of *different* known DOS viruses over the last several years can be fit remarkably well by an exponential curve. [2] Currently, it is approximately 2000, with two or three new ones appearing each day — a rate which already taxes to the limit the ability of anti-virus vendors to develop detectors and cures for them. Were this trend to hold up (Fig. 1), there would be approximately 10 million different DOS viruses by January, 2000 — about 100,000 new ones per day! Of course, curve extrapolation of a phenomenon that depends largely on human sociology and psychology should be regarded very skeptically, but it is not impossible that virus writers could be so prolific. To do so, they would have to automate both the writing and the distribution of viruses. Already, the beginnings of a trend towards automated virus-writing is evinced

[2] Note that is *not* the same as the growth in prevalence of any particular viral strain. Even for the minority of viruses that are successful in any degree, the growth in prevalence is strongly sub-exponential, perhaps even roughly linear.

by the Virus Creation Laboratory, a menu-driven virus toolkit circulating among virus writers' bulletin boards. Even if the rate at which new viruses appear were to suddenly plateau at a level not much higher than what it is today, the number of different DOS viruses could easily reach the tens of thousands by the year 2000, and the burden on current anti-virus techniques to detect and eradicate so many viruses would be severe.

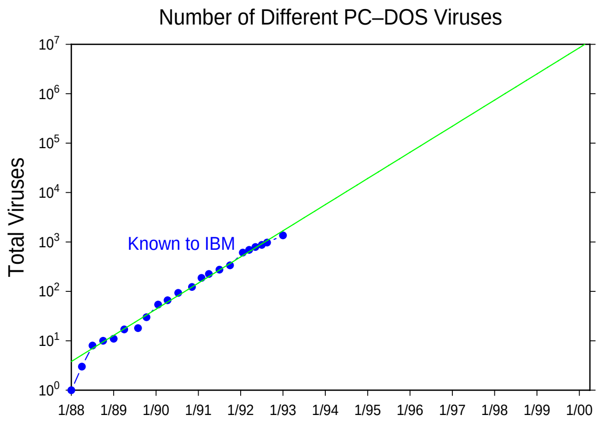

Figure 1: Number of different known DOS viruses vs. time (logarithmic scale). Straight line is the best exponential fit of the data through mid-1993. **Warning:** Extrapolation of the exponential trend beyond 1993 should be regarded *very* skeptically.

2.3 Interconnectivity and its consequences

It is unfortunate, but hardly surprising, that increased interconnectivity and interoperability among computers — designed to facilitate the flow of desirable information — also facilitates the flow of computer viruses. Biological diseases have always taken advantage of technological advances which enhance man's mobility [8]; it is natural that computer diseases should make opportunistic use of advances in the mobility of information.

One can expect increased networking to be reflected in increases in two important epidemiological parameters: the overall rate at which a given infected individual computer spreads a virus and the number of partners with which that individual has potentially infectious contacts. The first factor is related to one of the most fundamental results of classical mathematical epidemiology [10]. If the average rate at which infection can spread from one individual to another is sufficiently low, widespread infection is impossible. Above a well-defined critical threshold, however, epidemics can occur. As a simple way of explaining the existence of a sharp threshold, imagine that an individual has the flu. If, during that individual's period of contagion, he or she can be expected to infect 0.9 other people, the strain of flu will sooner or later die out. However, if that individual can be expected to infect 1.1 other people, there is likely to be a flu epidemic. The second factor, increased promiscuity, has apparently been given little attention by theoretical epidemiologists until our own study of it [7, 11]. We have found that a topology in which each individual has several "neighbors" to which it can spread infection is more conducive to epidemics than one which is sparsely connected — even when the infection rate along each link is adjusted so as to keep the total the same in the two cases.

Thus, to the extent that technological advances will increase the contact rate and promiscuity among computers, we can expect computer virus epidemics to become more likely, to spread faster, and to affect more computers. Experience with the Internet worm, which spread to hundreds or perhaps thousands of machines across the world in less than one day in 1988 [9], shows that even today's computing environment is vulnerable to a spread rate that is about two orders of magnitude faster than the typical timescale of monthly updates. While it is true that updates might be made somewhat more frequently, this would not solve the problem. The updates must be distributed to customers, and the customers must install them. Given the time, money, and effort involved, it is not surprising that many customers blissfully continue to use anti-virus software that is more than a year out of date.

3 An immune system for computers

Imagine that, every time a new strain of the common cold began to make its rounds, researchers at the Center for Disease Control had to race to find a cure for it. They would have to make sure that the cure worked properly for all sorts of people, and did not cause any allergic or other adverse reactions. The problem of distributing the cure to billions of people worldwide would be overwhelming.

This scenario is clearly ludicrous — we could not have survived as a species if we relied on a central agency to defend us against every disease. Yet this is precisely how we defend ourselves against computer viruses today! Time is running out on this approach, and a different alternative is sought.

Rather than relying on a central authority to protect them from all ills, humans and other vertebrates carry around their own individual immune systems. The vertebrate immune system exhibits some remarkable properties, including [12]:

1. Recognition of known intruders.
2. Elimination/neutralization of intruders.
3. Ability to learn about previously unknown intruders.
   - Determine that the intruder doesn't belong.
   - Figure out how to recognize it.
   - Remember how to recognize it.
4. Use of selective proliferation and self-replication for quick recognition and response.

Phrased in this way, it is evident that these fundamental properties are desirable for computers as well. The remainder of this section describes how each of these functions are being implemented in our design of the computer immune system, and compares our implementation with Nature's implementation of the vertebrate immune system. At the end of the section, the various elements will be assembled into a complete sketch of the proposed computer immune system.

3.1 Recognizing Known Intruders

The vertebrate immune system recognizes particular antigens (viruses and other undesirable foreign substances) by means of antibodies and immune cell receptors which bind to epitopes (small portions of the antigen, consisting of at least 4 to 6 amino acids).

It is interesting to note that an *exact* match to the entire antigen is not attempted; in fact, it is almost certainly a physical impossibility. No antibody molecule or immune-cell receptor could be perfectly specific to a given antigen because matching occurs at surfaces, not throughout volumes. T cell receptors can see the inner portions of antigen, but only after the antigen has been consumed by a macrophage or other cell, which then presents pieces of the antigen on it surface, where they can be seen by other cells.

Similarly, in the computer immune system, a particular virus is not recognized via an exact match; rather, it is recognized via an exact or fuzzy match to a relatively short sequence of bytes occurring in the virus (a "signature", as described in section 2). Although matching to a small portion of the virus is not necessitated in this case by the laws of chemistry, it has some important advantages. In particular,

1. it is more efficient in time and memory, and

2. it enables the system to recognize variants.

The issues of efficiency and variant recognition are relevant for biology as well.

For both biological and computer immune systems, an ability to recognize variants is essential because viruses tend to mutate frequently. If an exact match were required, immunity to one variant of a virus would confer no protection against a slightly different variant. Similarly, vaccines would not work, because they rely on the biological immune system's ability to synthesize antibodies to tamed or killed viruses that are similar in form to the more virulent one that the individual is being immunized against.

3.2 Eliminating Intruders

In the biological immune system, if an antibody meets up with an antigen, the two bind together, and the antigen is effectively neutralized. Thus recognition and neutralization of the intruder occur simultaneously. Alternatively, a killer T cell may encounter a cell that exhibits signs of being infected with a particular infecting agent, whereupon it kills the host cell. This is a perfectly sensible course of action. A biological virus co-opts its host cell's machinery, matter and energy into synthesizing viral proteins that are assembled into copies of the virus. Eventually, the host's cell wall is ruptured, resulting in the death of the host and the release of hundreds or thousands of viruses into the intercellular medium. By killing an infected host cell, a killer T cell is merely hastening the execution of a cell that was slated to die anyway , and it prevents the virus from completing the replication process.

If the computer immune system were to find an exact or fuzzy match to a signature for a known virus, it could take the analogous step of erasing or otherwise inactivating the executable file containing the virus. This is a valid approach. However, an important difference between computer viruses and biological viruses raises the possibility of a much gentler alternative.

From the body's point of view, cells are an easily-replenished resource. Even if biological viruses didn't destroy infected cells, an infected host cell would hardly be worth the trouble of saving; there are plenty of other cells around that can serve the same function. In contrast, each of the applications run by a typical computer user are unique in function and irreplaceable (unless backups have been kept, of course). A user would be likely to notice any malfunction. Consequently, it would be suicidal for a computer virus to destroy its host program, because the ensuing investigation would surely lead to its discovery and eradication. For this reason, all but the most ill-conceived computer viruses attach themselves to their host in such a way that they do not destroy its function. The fact that host information is merely rearranged, not destroyed, allows one to construct repair algorithms for a large class of non-destructive viruses for which one has a precise knowledge of the attachment method.

3.3 Learning to Recognize Unknown Intruders

When the biological immune system encounters an intruder that it has never seen before, it can immediately recognize the intruder as non-self, and attack it on that basis. Over the course of days or weeks, through a process of mutation and selective proliferation (see the next subsection), it "learns" to fabricate antibodies and B- and T cell receptors capable of recognizing that particular intruder very efficiently. By some unknown means, the immune system is able to "remember" the antigen (*i. e.* it retains immune cells with the proper receptors for recognizing that antigen) for decades after the initial encounter, and thus it is ready to respond much more quickly the next time that antigen is encountered.

To be effective, an antibody or receptor for a particular antigen must bind to that antigen (or close variants of that antigen) with high efficiency, and it must *not* bind to self proteins — otherwise, the host would be likely to suffer from an auto-immune disease. The biological immune system reduces the chances of recognizing self by subjecting immature immune cells to a training period in the thymus, during which those possessing self-recognizing receptors are eliminated.

Unfortunately, the notion of "self" in computers is somewhat problematic. We can not simply regard the "self" as the set of software that was pre-loaded when the computer was first purchased. Computer users are continually updating and adding new software. It would be unacceptable if the computer immune system were to reject all such modifications and additions out of hand on the basis that they were different from anything else that happened to be on the system already. While the biological immune system can usually get away with presuming the guilt of anything unfamiliar, the computer immune system must presume that new software is innocent until it can prove that it is guilty of containing a virus.

The thorny issue of what constitutes "self" for computer software, interesting as it is, can be regarded as a side-issue. The actual problem that both the vertebrate and the computer immune system must solve is to distinguish between harmful and benign entities. Due to the high degree of stability of body chemistry in individual vertebrates during their lifespans, their immune systems can replace the difficult problem of distinguishing between benign and harmful entities by the much simpler one of distinguishing self from non-self. This is a nice hack, because "self" is much easier to define and recognize than "benign". The immune system can simply implement the strategy "know thyself" (and reject all else). Although this errs on the side of false positives (*i.e.* falsely rejecting benign entities), rejection of foreign benign entities is generally not harmful (except in cases of blood transfusion or organ transplantation, which have been introduced much too recently to have affected the course of evolution).

By contrast, false rejection of legitimate software is extremely harmful. It worries users unnecessarily, and can cause them to erase perfectly legitimate programs — leading to hours or days of lost productivity. After such an experience, users are often tempted to stop using anti-virus software, leaving themselves completely unprotected. Thus a false positive indentification of a virus may be much more harmful than the virus itself. For this reason, self/non-self discrimination is not by itself an adequate means for distinguishing between harmful and unharmful software.

The process by which the proposed computer immune system establishes whether new software contains a virus has several stages. Integrity monitors, which use checksums to check for any changes to programs and data files, have a notion of "self" that is as restrictive as that of the vertebrate immune system: any differences between the original and current versions of any file are flagged, as are any new programs.[3] However, evidence of a non-self entity is not by itself enough to trigger an immune response. Mechanisms that employ the complementary strategy of "know thine enemy" are also brought into play. Among these are activity monitors, which have a sense of what dynamic behaviors are typical of viruses, and various heuristics, which examine the static nature of any modifications that have occurred to see if they have a viral flavor.

In the computer immune system, integrity monitors and generic know-thine-enemy heuristics are periodically or continually on the lookout for any indications that a virus is present in the system. If one of the virus-detection heuristics is triggered, the immune system runs the scanner to determine whether the anomaly can be attributed to a known virus. If so, the virus is located and removed in the usual way. If the anomaly can *not* be attributed to a known virus, either the generic virus-detection heuristics yielded a false alarm, or a previously unknown virus is at large in the system.

At this point, the computer immune system tries to lure any virus that might be present in the system to infect a diverse suite of "decoy" programs. A decoy program's sole purpose in life is to become infected. To increase the chances of success in this noble, selfless endeavor, decoys are designed to be as attractive as possible to those types of viruses that spread most successfully. A good strategy for a virus to follow is to infect programs that are touched by the operating system in some way. Such programs are most likely to be executed by the user, and thus serve as the most successful vehicle for further spread. Therefore, the immune system entices a putative virus to infect the decoy programs by executing, reading, writing to, copying, or otherwise manipulating each of them. Such activity tends to attract the attention of many viruses that remain active in memory even after they have returned control to their host. To catch viruses that do not remain active in memory, the decoys are placed in places where the most commonly used programs in the system are typically located, such as the root directory, the current directory, and other directories in the path. The next time the infected file is run, it is very likely to select one of the decoys as its victim. From time to time, each of the decoy programs is examined to see if it has been modified. If one or more have been modified, it is almost certain that an unknown virus is loose in the system, and each of the modified decoys contains a sample of that virus. These virus samples are stored in such a way that they will not be executed accidentally.

The capture of a virus sample by the decoy programs is somewhat analogous to the ingestion of antigen by macrophages or B cells [12]. It allows the intruder to be processed into a standard format that can be parsed by some other component of the immune system, and provides a standard location where information on the intruder can be found. In the biological immune system, the T cells that recognize the antigen are selected according to their ability to bind to fragments of the antigen that are presented on the surface of cells that have ingested (or been infected by) the antigen. Likewise, in the computer immune system, the infected decoys are then processed by another component of the immune system — the signature extractor — so as to develop a recognizer for the virus. The computer immune system has an additional task that is not shared by its biological analog: it must attempt to extract from the decoys information about how the virus attaches to its host, so that infected hosts can be repaired (if possible).

Unfortunately, the proprietary nature of our methods for deriving a virus's means of attachment to its host forbid any discussion of them here. Briefly, the algo-

[3]An interesting alternative to traditional integrity monitoring via checksums, inspired by the detailed mechanisms by which the vertebrate immune system learns to recognize "self", has been studied recently by Forrest, Perelson, Allen, and Cherukuri [13].

rithms extract from a set of infected decoys information on the attachment pattern of the virus, along with byte sequences that remain constant across all of the captured samples of the virus.

Next, the signature extractor must select a virus signature from among the byte sequences produced by the attachment derivation step. The signature must be well-chosen, such that it avoids both false negatives and false positives. In other words, the signature must be found in each instance of the virus, and it must be very unlikely to be found in uninfected programs.

First, consider the false negative problem. The samples captured by the decoys may not represent the full range of variable appearance of which the virus is capable. As a general rule, non-executable "data" portions of programs, which can include representations of numerical constants, character strings, work areas for computations, *etc.* are inherently more likely to vary from one instance of the virus to another than are "code" portions, which represent machine instructions. The origin of the variation may be internal to the virus (*e.g.* it could depend on a date). Alternatively, a virus hacker might deliberately change a few data bytes in an effort to elude virus scanners. To be conservative, "data" areas are excluded from consideration as possible signatures. Although the task of separating code from data is in principle somewhat ill-defined, there are a variety of methods, such as running the virus through a debugger or virtual interpreter, which perform reasonably well.

The false positive problem is more interesting. In the biological immune system, false positives that accidentally recognize self cause auto-immune diseases. In both traditional anti-virus software and the proposed computer immune system, false positives are particularly annoying to customers, and so infuriating to vendors of falsely-accused software that it has led to at least one lawsuit against a major anti-virus software vendor. (So one could say that health is also an issue in this case!)

Briefly, the automatic signature extractor examines each sequence of S contiguous bytes (referred to as "candidate signatures") in the set of invariant-code byte sequences that have presented to it, and for each it estimates the probability for that S-byte sequence to be found in the collection of normal, uninfected "self" programs. Typically, S is chosen to be 16 or 24. The probability estimate is made by

1. forming a list of all n-grams (sequences of n bytes; $1 \leq n \leq n_{max}$) contained in the input data (n_{max} is typically 5 or 8),

2. calculating the frequency of each such n-gram in the "self" collection (in the case of signatures that are to be distributed worldwide, we use a half-gigabyte corpus of ordinary, uninfected programs),

3. using a simple formula to combine the n-gram frequencies into a probability estimate for each candidate signature to be found in a set of programs similar in size and statistical character to the corpus, and

4. selecting the signature with the lowest estimated false-positive probability.

Characterizations of this method show that the probability estimates are poor on an absolute scale, due to the fact that code tends to be correlated on a longer scale than 5 or 8 bytes. However, the relative ordering of candidate signatures is rather good, so the method generally selects one of the best possible signatures. In fact, judging from the relatively low false-positive rate of the IBM AntiVirus signatures (compared with that of other anti-virus vendors), the algorithm's ability to select good signatures is *better* than can be achieved by typical human experts.

Having automatically developed both a recognizer and a repair algorithm appropriate to the virus, the information can be added to the corresponding databases. If the virus is ever encountered again, the immune system will recognize it immediately as a known virus. A computer with an immune system could be thought of as "ill" during its first encounter with a virus, since a considerable amount of time and energy (or CPU cycles) would be expended to analyze the virus. However, on subsequent encounters, detection and elimination of the virus would occur much more quickly: the computer could be thought of as "immune" to the virus.

3.4 Self Replication and Selective Proliferation

In the biological immune system, immune cells with receptors that happen to match a given antigen reasonably well are stimulated to reproduce themselves. This provides a very strong selective pressure for good recognizers, and by bringing a degree of mutation into play, the immune cell is generally able to come up with immune cells that are extremely well-matched to the antigen in question.

One can view this as a case in which self-replication is being used to fight a self-replicator (the virus) in a very effective manner. One can cite a number of other examples in nature and medical history in which the same principle has been used very successfully. The self-replicator need not itself be a virus. In the case of the worldwide campaign against smallpox, those who were in close contact with an infected individual were all immunized against the disease. Thus immunization spread as a sort of anti-disease among smallpox victims [10].

We propose to use a similar mechanism, which we call the "kill signal", to quell viral spread in computer networks. When a computer discovers that it is infected, it can send a signal to neighboring machines. The signal conveys to the recipient the fact that the transmitter was infected, plus any signature or repair information that might be of use in detecting and eradicating the virus. If the recipient finds that it is infected, it sends the signal to *its* neighbors, and so on. If the recipient is not infected, it does not pass along the signal, but at least it has received the database updates — effectively immunizing it against that virus (see Fig. 2).

Theoretical modeling has shown the kill signal to be extremely effective, particularly in topologies that are

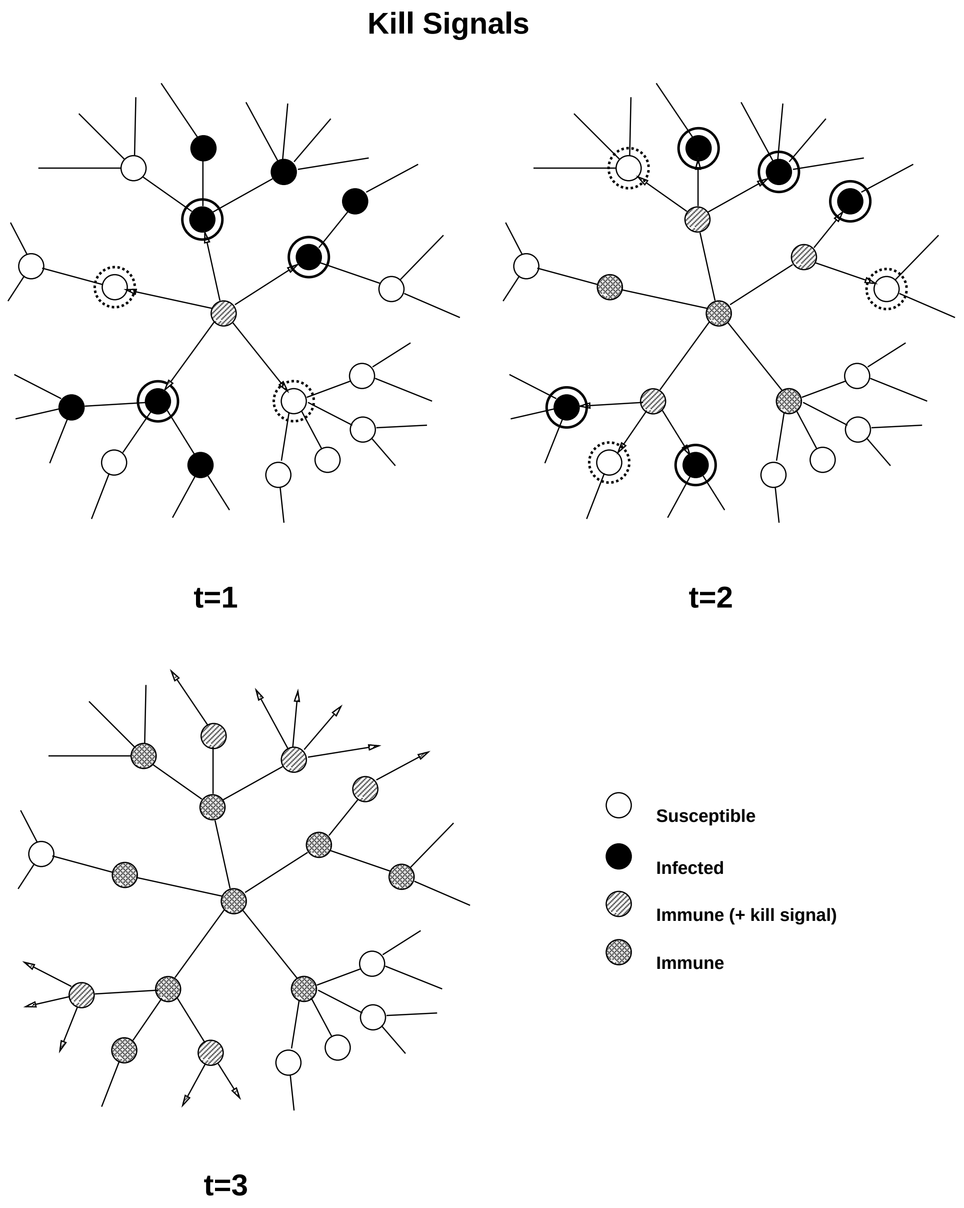

Figure 2: Fighting self-replication with self-replication. When a computer detects a virus, it eliminates the infection, immunizes itself against future infection, and sends a "kill signal" to its neighbors. Receipt of the kill signal results in the immunization of uninfected neighbors; infected neighbors are both immunized and prompted to send kill signals to their neighbors. Thus detection of a virus by a single computer can trigger a wave of kill signals that propagates along the path taken by the virus, destroying the virus in its wake.

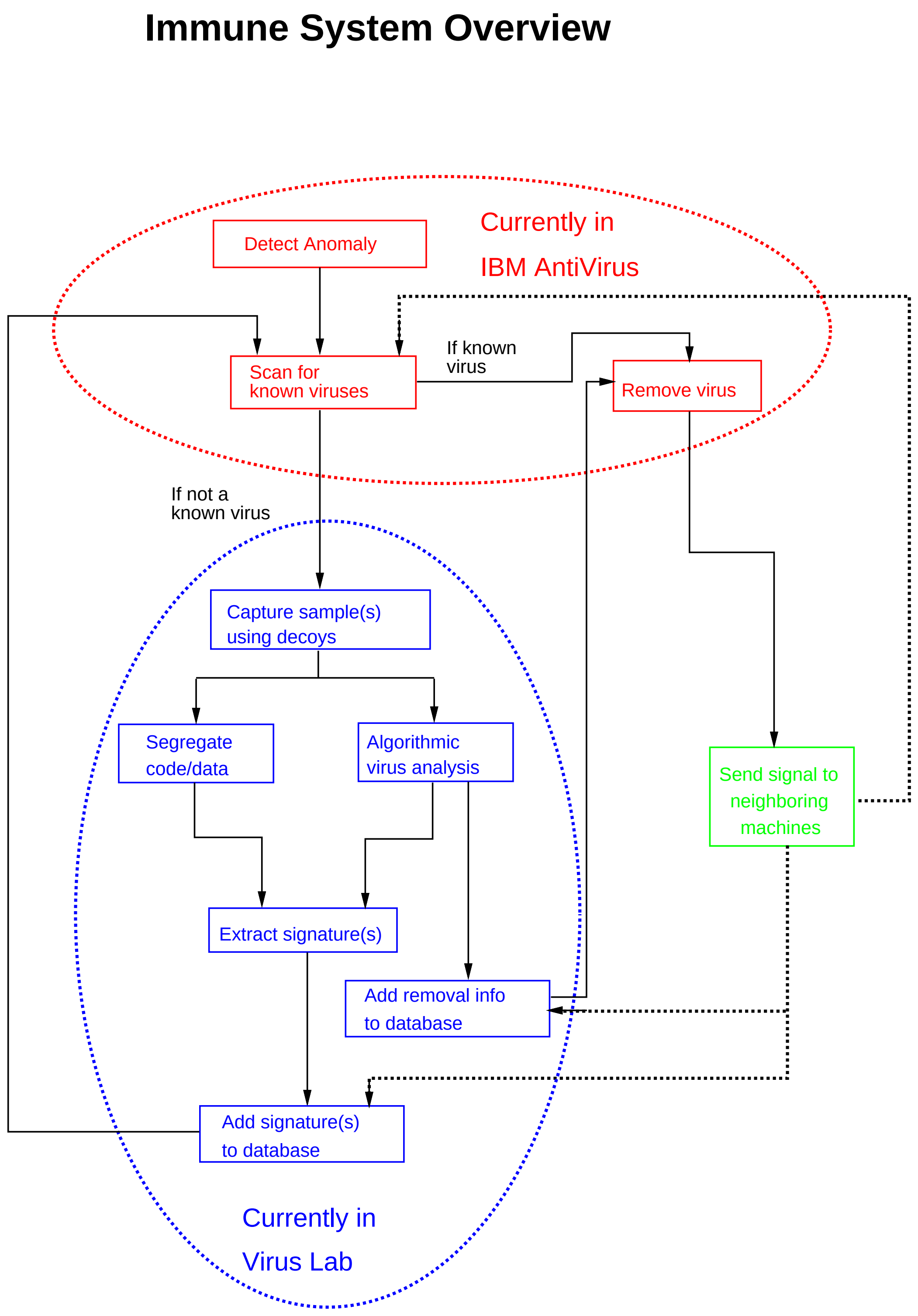

Figure 3: The main components of the proposed immune system for computers and their relationship to one another.

highly localized or sparsely connected [4, 11].

3.5 Computer Immune System: Schematic and Implementation

Fig. 3 sketches the relationship among various components of the proposed computer immune system. Some are already integrated into the current version of IBM AntiVirus. The components of the immune system that deal with unknown viruses are currently being used in a slightly different capacity: to extract signatures and repair information automatically from newly-discovered viruses. This enables us to keep pace with the influx of new viruses with just one human virus expert who analyzes viruses half-time, as opposed to the dozen or more virus analysts employed by some other anti-virus software vendors.

When a raft of new viruses is received, it is presented to an automatic "triage" machine situated in IBM's virus isolation laboratory. First, the triager scans the putative viruses using the current version of IBM AntiVirus. Any samples infected with a virus that is already detected by IBM AntiVirus are immediately dismissed from further consideration. The triager then executes each of the remaining infected samples one or more times, and (for each infected sample) exercises a set of six decoy programs so as to entice the virus to infect them. Each of the decoy programs is examined from time to time to see if it has been modified. Any decoys that *have* been modified are stored away in a form such that they cannot be executed, and the triage machine is automatically rebooted to eliminate the virus from memory. The triage script goes through the same routine for the next putatively infected sample, and so on until all the original samples have been given a reasonable chance to infect the decoys.

Putatively infected samples which successfully infect decoys are placed in the archive of confirmed viruses, and the infected decoys are placed in special directories for further processing. Putatively infected samples which fail to infect any decoys may contain recalcitrant viruses that for some reason were not in the infecting mood, or they may not contain a virus at all. On a rainy day some weeks hence, another attempt will be made to coax them into infecting the decoys.

Typically, a given virus sample will infect two or three of the six decoys. During the last three years, the triager has been used to capture samples of over 2000 different PC/DOS viruses.

The infected decoys are then processed by the algorithmic virus analyzer, which extracts information that is useful in repairing viruses. Still in early prototype, the analyzer is able to supply useful information for about 90% of the viruses that it has seen. A debugger is used to execute each infected decoy; any executed instructions are obviously code (rather than unexecutable data), and as such are eligible for consideration as part of a signature for the virus.

Next, the automatic signature extractor takes as input all byte sequences which appear in each infected decoy and which have been established as code, selects a signature, and provides an estimate of the maximum number of mismatches between scanned data and the signature that can be considered a match. During three years of constant improvements, the automatic signature extractor has been used to extract signatures for roughly 1500 different PC/DOS viruses. In addition, it has been used to evaluate several hundred signatures that had been extracted by expert humans.

The automatically-extracted signatures and repair information are then subjected to a variety of independent tests. The signatures are run against a half-gigabyte corpus of legitimate programs to make sure that they do not cause false positives, and the repair information is checked out by testing on samples of the virus, and further checked by a human expert. Finally, the detection and repair databases used by IBM AntiVirus are updated, and the new version is distributed to customers worldwide.

The remaining component of the immune system, the kill signal, is the only one that has not yet been implemented; it is currently being evaluated via theoretical modeling.

4 Conclusion

An immune system for computers is desirable and feasible. As suggested in Fig. 3, most of the necessary components are already in use in one form or another. Some already exist in IBM AntiVirus itself. Others are presently in use in the virus laboratory, for the purpose of updating the databases employed by IBM AntiVirus to recognize viruses and repair infected files.

One of the technical issues that remains to be explored further is the kill signal. Further simulation will help to establish the exact circumstances under which a node should send signals to its neighbors, and for what length of time these signals should be sent. Further analysis and simulation must be conducted to assess the effectiveness of various fail-safe mechanisms that have been proposed to deal with the propagation of erroneous kill signals, which could result from false positives, software bugs, or intentional subversion by malicious users. The biological immune system has invented various inhibitory mechanisms which may turn out to be of use to us.

We anticipate that, as the design for our computer immune system evolves, it will be influenced, not just by what Mother Nature has invented, but also by theories invented by immunologists to explain the observed function of the immune system [14, 15, 16]. In fact, we may offer new employment opportunities for theoretical immunologists, because our criteria for success are different: a proposed mechanism need not be a correct description of biology; it only has to work!

Acknowledgments

I would like to thank all the members of IBM's High Integrity Computing Laboratory, particularly Steve White, Bill Arnold, Dave Chess, and Greg Sorkin, for numerous discussions about automatic virus analysis and the design of the immune system. Bill Arnold and Dave Chess invented the idea of using decoy programs. Bill

Arnold implemented the decoy-infection routines and the viral-code identifier on DOS machines in the virus isolation laboratory, and invented "kill signals" (and coined their name). Greg Sorkin invented the algorithms that produce a concise description of a virus's method of attachment to its host.

References

[1] Amodio, Jorge. Submission to Virus-L digest 7:4. January 18, 1994.

[2] Dataquest. 1991. *Computer Virus Market Survey for National Computer Security Association.* San Jose.

[3] Sulit, Beth K. Nothing to sneeze at. *Risk & Insurance,* August 1992, 1.

[4] Kephart, Jeffrey O. and Steve R. White. Measuring and modeling computer virus prevalence. *Proceedings of the 1993 IEEE Computer Society Symposium on Research in Security and Privacy.* Oakland, California, May 24–26, 1993, 2–15.

[5] Spafford, E. H. 1991. Computer viruses: A form of artificial life? In D. Farmer, C. Langton, S. Rasmussen, and C. Taylor, eds., *Artificial Life II. Studies in the Sciences of Complexity*, 727–747. Redwood City: Addison-Wesley.

[6] Kephart, Jeffrey O., Steve R. White, and David M. Chess. *Computers and epidemiology.* IEEE Spectrum, May 1993, 20–26.

[7] Kephart, Jeffrey O. and Steve 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, 343–359.

[8] McNeill, William H. 1977. *Plagues and Peoples.* New York: Doubleday.

[9] Spafford, E. H. 1989. The Internet worm program: an analysis. *Computer Comm. Review* 19, 1.

[10] Bailey, Norman T. J. 1975. *The mathematical theory of infectious diseases and its applications,* second edition. New York: Oxford University Press.

[11] Kephart, Jeffrey O. 1994. How topology affects population dynamics, in C. Langton, ed., *Artificial Life III. Studies in the Sciences of Complexity*, 447–463. Redwood City: Addison-Wesley.

[12] Paul, William E., ed. 1991. *Immunology: Recognition and Response ... Readings from Scientific American.* New York: W. H. Freeman and Company.

[13] Forrest, Stephanie, Alan S. Perelson, Lawrence Allen, and Rajesh Cherukuri. Self-nonself discrimination in a computer. *Proceedings of the 1994 IEEE Computer Society Symposium on Research in Security and Privacy,* Oakland, California, May 16–18, 1994.

[14] Perelson, Alan S., ed. 1988. *Theoretical Immunology, Parts One and Two.* Redwood City: Addison-Wesley.

[15] Forrest, Stephanie and Alan S. Perelson. 1990. Genetic algorithms and the immune system. *Proceedings of Workshop on Parallel Problem Solving from Nature.* New York: Springer-Verlag.

[16] Seiden, P. E. and F. Celada. 1992. A model for simulating cognate recognition and response in the immune system. *J. Theor. Biology* 158, 329.