Analogies with immunology represent an important step toward the vision of robust, distributed protection for computers.

Comput Immu

Stephanie Forrest, Steven A. Hofmeyr, and Anil Somayaji

NATURAL IMMUNE SYSTEMS PROTECT ANIMALS from dangerous foreign pathogens, including bacteria, viruses, parasites, and toxins. Their role in the body is analogous to that of computer security systems in computing. Although there are many differences between living organisms and computers, the similarities are compelling and could point the way to improved computer security.[1] Improvements can be achieved by designing computer immune systems with some of the important properties of natural immune systems, including multilayered protection; highly distributed detector, effector, and memory systems; diversity of detection ability across individuals; inexact matching strategies; and sensitivity to most new foreign patterns. Some of these properties are well known but seldom implemented successfully; other properties are less well known. The immune system provides a persuasive example of how they might be implemented in a coherent system.

The immune system comprises cells and molecules.[2] Recognition of foreign protein, called *antigen*, occurs when immune system detectors, including T cells, B cells, and antibodies, bind to antigen. Binding between detector and antigen is determined by the physical and chemical properties of their binding regions. Binding is highly specific, so each detector recognizes only a limited set of structurally related antigen. When a detector and antigen bind, a com-

[1]The authors gratefully acknowledge support from the National Science Foundation (IRI-9157644), the Office of Naval Research (N00014-95-1-0364), Defense Advanced Research Projects Agency (N00014-96-1-0680), Interval Research Corp., the MIT Artificial Intelligence Laboratory, and the Santa Fe Institute.

[2]A good source for basic immunology is [6]; a computer scientist's overview of immunology is given at http://www.cs.unm.edu/~steveah/imm-html/immune-system.html.

Trend Micro, Inc.
EX1019-1

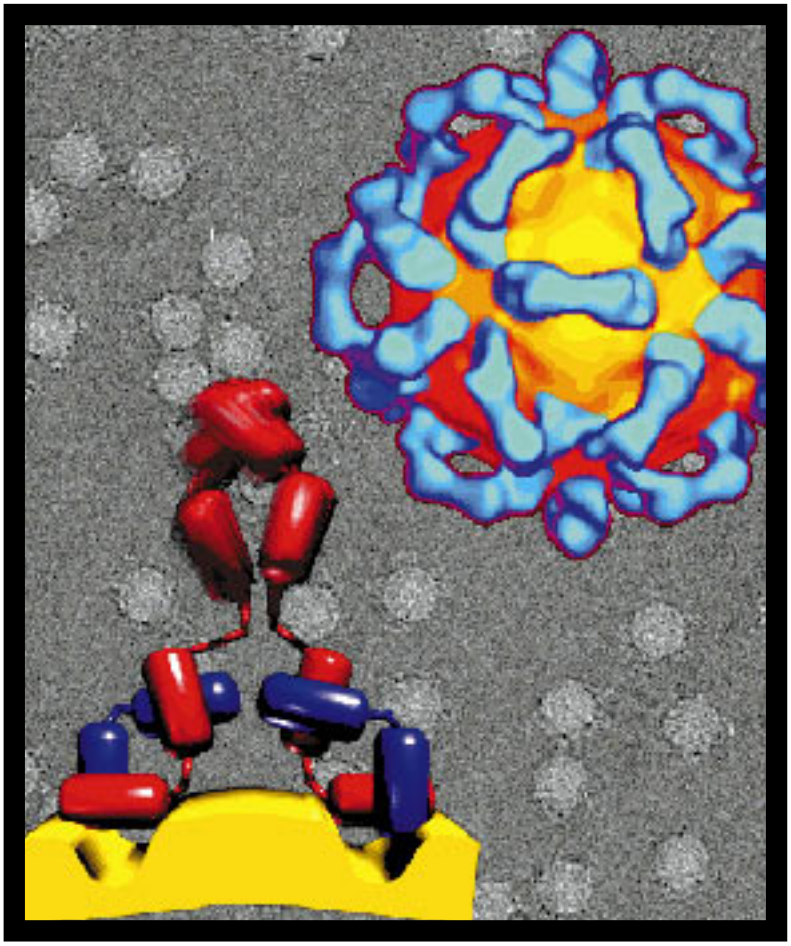

IMAGE COURTESY OF THOMAS SMITH AND TIMOTHY BAKER AT PURDUE UNIVERSITY

er nology

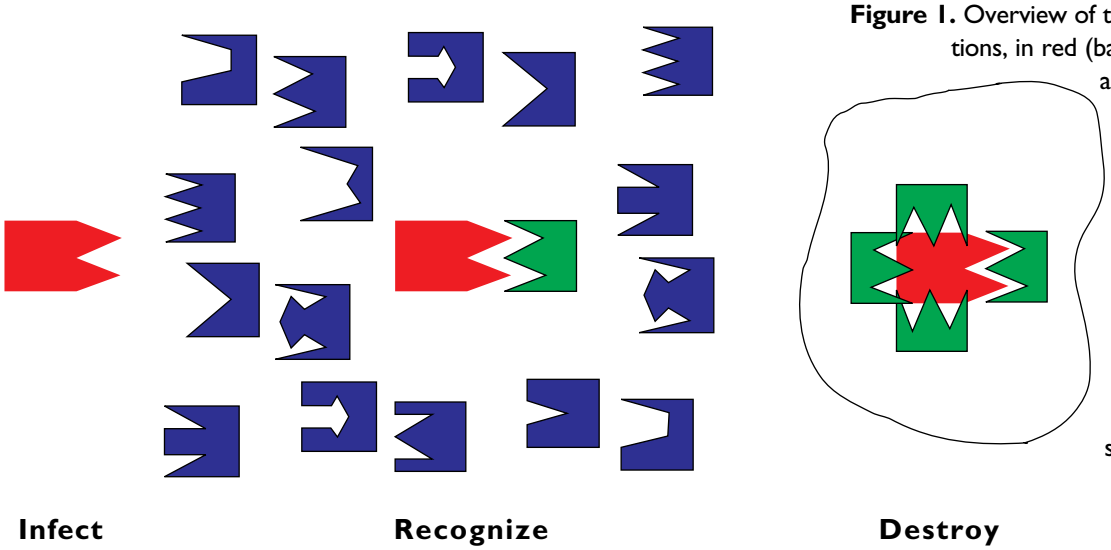

Figure 1. Overview of the immune system. Infections, in red (bacteria, viruses, parasites), are recognized by immune system detectors, in blue and green (T cells, B cells, and antibodies), when molecular bonds form between them. Infections are eliminated by general-purpose scavenger cells (macrophages), indicated by the thin line surrounding the detector/antigen complex.

Source: Figure prepared by R. Hightower.

Trend Micro, Inc.
EX1019-2

plex set of events takes place, usually resulting in elimination of the antigen by scavenger cells called macrophages. (How antigen is bound and cleared depends on the type of detectors involved.) Figure 1 outlines this highly simplified view of the immune system. A striking feature of the immune system is that the processes by which it generates detectors, identifies and eliminates foreign material, and remembers the patterns of previous infections are all highly parallel and distributed. This is one reason immune system mechanisms are so complicated, but it also makes them highly robust against failure of individual components and to attacks on the immune system itself.

The analogy between computer security problems and biological processes was recognized as early as 1987, when the term "computer virus" was introduced by Adelman [1]. Later, Spafford argued that computer viruses are a form of artificial life [12], and several authors investigated the analogy between epidemiology and the spread of computer viruses across networks [7, 10]. However, current methods for protecting computers against viruses and many other kinds of intrusions have largely failed to take advantage of what is known about how natural biological systems protect themselves from infection. Some initial work in this direction included a virus-detection method based on T-cell censoring in the thymus [4] and an integrated approach to virus detection incorporating ideas from various biological systems [8]. However, these early efforts are generally regarded as novelties, and the principles they illustrate have yet to be widely adopted.

Immunologists have traditionally described the problem solved by the immune system as that of distinguishing "self" from dangerous "other" (or "nonself") and eliminating other.[3] Self is taken to be the internal cells and molecules of the body, and nonself is any foreign material, particularly bacteria, parasites, and viruses. The problem of protecting computer systems from malicious intrusions can similarly be viewed as the problem of distinguishing self from dangerous nonself. In this case, nonself might be an unauthorized user, foreign code in the form of a computer virus or worm, unanticipated code in the form of a Trojan horse, or corrupted data.

Distinguishing between self and nonself in natural immune systems is difficult for several reasons. First, the components of the body are constructed from the same basic building blocks, particularly proteins, as nonself. Proteins are an important constituent of all cells, and the immune system processes them in various ways, including in fragments called peptides, which are short sequences of amino acids. Second, the size of the problem to be solved is large with respect to the available resources. For example, it has been estimated that the vertebrate immune system needs to be able to detect as many as 10^{16} patterns, yet it has only about 10^5 different genes from which it must construct the entire immune system (as well as everything else in the body). The difficulty of this discrimination task is shown by the fact that the immune system can make mistakes. Autoimmune diseases provide many examples of the immune system confusing self with other.

The computer security problem is also difficult. There are many legitimate changes to self, like new users and new programs, and many paths of intrusion, and the periphery of a networked computer is less clearly defined than the periphery of an individual animal. Firewalls attempt to construct such a periphery, often with limited success.

The natural immune system has several distinguishing features that provide important clues about

[3]The modern view emphasizes the immune system's role in eliminating infection in addition to its tolerance of self, an emphasis that is similarly important in the computer security problem.

Trend Micro, Inc.
EX1019-3

how to construct robust computer security systems, including:

- Multilayered protection. The body provides many layers of protection against foreign material, including passive barriers, such as skin and mucous membranes; physiological conditions, such as pH and temperature; generalized inflammatory responses; and adaptive responses, including both the humoral (B cell) and cellular (T cell) mechanisms. Many computer security systems are monolithic, in the sense that they define a periphery inside which all activity is trusted. When the basic defense mechanism is violated, there is rarely a backup mechanism to detect the violation. A good example is a computer security system that relies on encryption to protect data but lacks a way for noticing whether the encryption system has been broken.
- Distributed detection. The immune system's detection and memory systems are highly distributed; there is no centralized control that initiates or manages a response. Its success arises from highly localized interactions among individual detectors and effectors, variable cell division, and death rates, allowing the immune system to allocate resources (cells) where they are most needed, and from the ability to tolerate many kinds of failures, including deletion of entire organs, such as the spleen.
- Unique copies of the detection system. Each individual in a population has a unique set of protective cells and molecules. Computer security often involves protecting multiple sites, including multiple copies of software and multiple computers on a network. In these environments, when a way is found to avoid detection at one site, all sites become vulnerable. A better approach would be to provide each protected location a unique set of detectors or even a unique version of software. Thus, if one site were compromised, other sites would likely remain secure.
- Detection of previously unseen foreign material. An immune system protecting us from only those diseases against which we had been vaccinated would be much less effective than one that was able to recognize new forms of infection. Immune systems remember previous infections and mount a more aggressive response against those seen before; immunologists call this a secondary response. However, in the case of a novel infection, the immune system initiates a primary response, evolving new detectors specialized for the infection. This process is slower than a secondary response but provides an essential capability lacking in many computer security systems. Many virus- and intrusion-detection methods scan only for known patterns (e.g., virus signatures), leaving systems vulnerable to attack by novel means. Some exceptions include anomaly intrusion-detection systems [2] and cryptographic checksums.
- Imperfect detection. Not all antigen are well matched by a preexisting detector. The immune system uses two strategies to confront this problem—learning (during the primary response) and distributed detection (within a single individual and across populations of individuals). Thus, high systemwide reliability is achieved at relatively low cost (in time and space) and with minimal communication among components.

A Computer Immune System

What would it take to build a computer immune system with some or all of these features? Such a system would have much more sophisticated notions of identity and protection than those afforded by current operating systems, and it would provide general-purpose protection to augment current computer security systems. It would have at least the following basic components: a stable definition of self, the ability to prevent or detect and subsequently eliminate dangerous foreign activities (infections), memory of previous infections, a method for recognizing new infections, autonomy in managing responses, and a method of protecting the immune system itself from attack.

If we want to cast the problem of computer security in the framework of distinguishing self from nonself, the first task is to define what we mean by self and nonself. Do we want to define self in terms of memory access patterns on a single host, TCP/IP packets entering and leaving a single host, the collective behavior of a local network of computers, network traffic through a router, instruction sequences in an executing or stored program, user behavior patterns, or even keyboard typing patterns? The immune system has evolved its recognition machinery to focus on peptides (protein fragments), but it must consider many different paths of intrusion. For example, there are two quite different recognition systems in the immune system: cell-mediated response, aimed at viruses and other intracellular infections, and humoral response, primarily directed at bacteria and other extracellular material. For computers, self also likely needs to be presented in multiple ways to provide comprehensive protection.

We want our definition of self to be tolerant of

Trend Micro, Inc.
EX1019-4

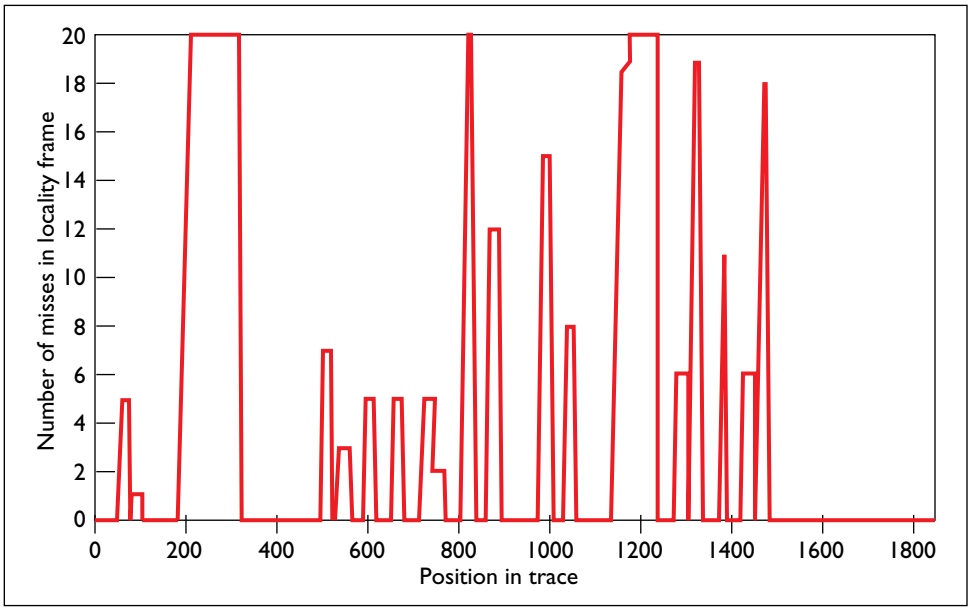

Figure 2. Anomalous signature for successful `syslog` exploit of `sendmail` under SunOS4.1.4. The normal database was generated with sequences of six system calls. The *x*-axis measures the position in the anomalous trace in units of system calls. The *y*-axis shows how many mismatches were recorded when the anomalous trace was compared with the normal database. The *y*-axis unit of measure is the total number of mismatches over the past 20 system calls in the trace (called the *locality frame*). That is, for position *i* in the trace, the locality frame records how many mismatches were observed in positions ($i - 19$) through *i*.

Source: Data generated with the help of L. Rogers and T. Longstaff of the Computer Emergency Response Team at the Software Engineering Institute in Pittsburgh.

many legitimate changes, including those in files due to editing, new software, new users, new user habits, and routine activities of system administrators. At the same time, we want it to notice unauthorized changes to files, viral software, unauthorized users, and insider attacks. In computer security parlance, we want a system with low rates of false-positives and few false-negatives. It is generally not possible to get perfect discrimination between legitimate and illegitimate activities. Given our bias toward multilayered protection, adaptive responses, and autonomous systems, we are more willing to tolerate false-negatives than false-positives, because false-negatives for one layer could well be true positives for another.

Two examples of how we are applying ideas from immunology to today's computer security problems are intrusion-detection method and distributable change detection. These examples highlight an important question about how analogies between biology and computer science can be applied. In one case, the analogy is much more direct than in the other. Yet both examples incorporate the basic principles we elucidated earlier and support the overall vision guiding our work. The analogy between immunology and computer security is a rich one and goes well beyond these two examples.

For example, Kephart et al. [8] exploit a similar analogy in quite different ways. In the quest for computer security, the analogy with immunology contributes an important point of view that can potentially lead to systems built with quite different sets of assumptions and biases from those used in the past. It is more important that the underlying principles be correct than that the surface-level analogy be obvious.

Intrusion Detection for System Processes

As an initial step toward defining self in a realistic computing environment, we are developing an intrusion-detection system for networked computers [5]. Discrimination must be based on some characteristic structure that is both compact and universal in the protected system. The immune system's "choice" of basing discrimination on patterns of peptides limits its effectiveness. For example, it cannot protect the body against radiation. However, proteins are a component of all living matter and generally differ between self and nonself, so they provide a good distinguishing characteristic.

What is the most appropriate way to define self in a computer? Most earlier work on intrusion-detection monitors the behavior of individual users, but we concentrate instead on system processes [9]. Our computer "peptide" is defined in terms of short sequences of system calls executed by privileged processes in a networked operating system. Preliminary experiments on a limited testbed of intrusions and other anomalous behavior show that short

Trend Micro, Inc.
EX1019-5

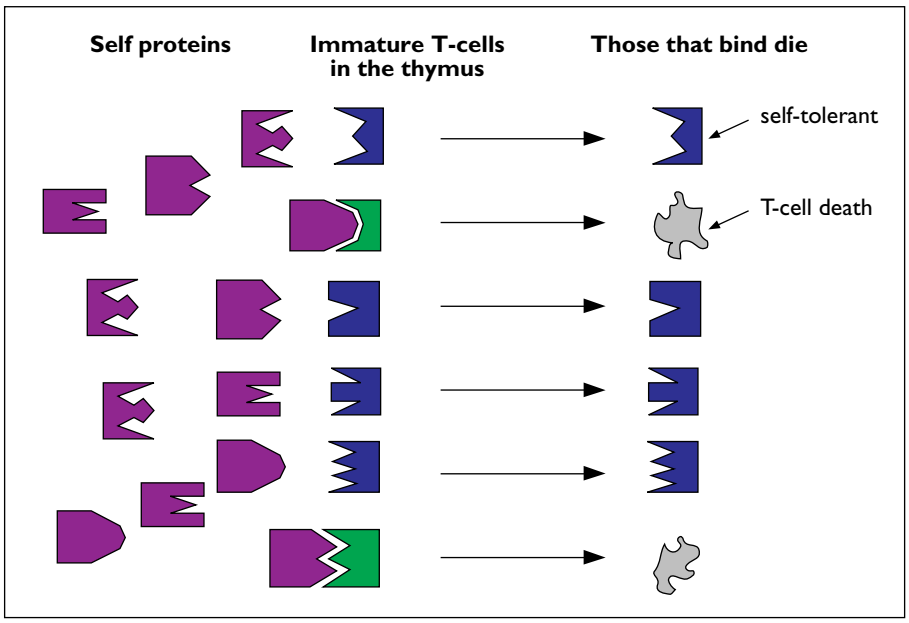

Figure 3. Censoring T cells in the thymus

sequences of system calls (currently, sequences of six system calls) provide a compact signature for self, distinguishing normal from abnormal behavior.

The strategy for our intrusion-detection system is to build up a database of normal behavior for each program of interest. Each database is specific to a particular architecture, software version and configuration, local administrative policies, and usage patterns.

When a stable database is constructed for a given program in a particular environment, the database can be used to monitor the program's behavior. The sequences of system calls form the set of normal patterns for the database, and sequences not found in the database indicate anomalies. In terms of the immune system, one host (or small network of hosts) would have several different databases defining self (one for each type of program being protected). Having several different databases is analogous to the many types of tissue in the body, each expressing a somewhat different set of proteins. That is, the patterns comprising self are not uniformly distributed throughout the protected system.

The proposed system involves two stages. In the first, we scan traces of normal behavior and build up a database of characteristic normal patterns, or observed sequences of system calls, (see the sidebar "A Database of Normal Patterns"). Parameters to system calls are ignored by the system, and we trace forked subprocesses individually. In the second, we scan traces that might contain abnormal behavior, matching the trace against the patterns stored in the database. If a pattern is seen that does not occur in the normal database, it is recorded as a mismatch. In our current implementation, tracing and analysis are performed off-line. Mismatches are the only observable characteristic we use to distinguish normal from abnormal. We observe the number of mismatches encountered during a test trace and aggregate the information in several ways. For example, Figure 2 shows the mismatch rate over time during a successful intrusion.

Do the normal databases allow the system to discriminate between normal and abnormal behavior? To date, we have constructed databases of normal behavior for three Unix programs: `sendmail`, `wu.ftpd` (a Linux version of `ftpd`), and `lpr`. When comparing the normal database for one program (e.g., `sendmail`) with traces of normal behavior of a different program (e.g., `ls`), we observed 40%–80% mismatches over the length of the foreign (e.g., `ls`) trace. We also observed clear detection of several common intrusions for the three programs (mismatch rates generally ranged from 1%–20% for the length of the trace). These results suggest that short sequences of system calls do provide a compact signature for normal behavior and that the signature has a high probability of being perturbed during intrusions.

Although this method does not provide a cryptographically strong or completely reliable discriminator between normal and abnormal behavior, it is much simpler than other proposed methods and could potentially provide a lightweight, real-time tool for continuously checking executing code. Another appealing feature is that code running frequently will be checked frequently, and code that is seldom executed will be checked infrequently. Thus, system resources are devoted to protecting the most relevant code segments. Finally, given the large variability in how individual systems are configured, patched, and used, we conjecture that databases at different sites would likely differ enough to meet the principle of diversity discussed earlier. Diversity is important for another reason—it could provide a behavioral signature, or identity, for a computer that is much more difficult to falsify than, say, an IP address. However, our results are quite preliminary, and a great deal of additional testing and develop-

Trend Micro, Inc.
EX1019-6

ment is needed before a system built on these ideas could be deployed.

Distributed Change Detection

The second example of applying immunology to computer security borrows more closely from mechanisms in the immune system. T cells are an important class of detector cells in the immune system. There are several different kinds of T cells, each playing a role in the immune response. However, all T cells have binding regions that can detect antigen fragments (peptides). These binding regions are created through a pseudo-random genetic process, which can be viewed as analogous to generating random strings. Given that the binding regions, called receptors, are created randomly, there is a high probability that some T cells will detect self peptides, causing an autonomous response. The immune system solves this problem by sending nonfunctional T cells to an organ called the thymus to mature.

There are several stages of T-cell maturation, one of which is a censoring process whereby T cells that bind with self proteins circulating through the thymus are destroyed. T cells failing to bind to self are allowed to mature, leave the thymus, and become part of the active immune system, a process called *negative selection* (see Figure 3). Once it is in circulation and when a T cell binds to antigen above a threshold, a *recognition event* is said to have occurred, triggering the complex set of events leading to elimination of the antigen.

The T-cell censoring process can be thought of as defining a protected collection of data (the self proteins) in terms of its complementary patterns (the nonself proteins). We can use this principle to design a distributed change-detection algorithm with interesting properties. Suppose we have a collection of digital data we call self that we wish to monitor for changes. This might be an activity pattern, as in the intrusion-detection algorithm described earlier, a compiled program, or a file of data. The algorithm works as follows:

1. Generate a set of detectors that fail to match self
2. Use the detectors to monitor the protected data
3. When a detector is activated, recognize that a change must have occurred and know the location of the change.

Before we have an implementable algorithm, we must answer several questions:

- How are the detectors represented?
- How is a match defined?
- How are detectors generated?
- How efficient is the algorithm?

These questions are explored in detail in [3] and [4], but we give some highlights here.

In our computer immune system, binding between detectors and foreign patterns is modeled as string matching between pairs of strings. Self is defined as a set of equal-length strings (e.g., by logically segmenting the protected data into equal-size substrings), and each detector is defined as a string of the same length as the substring. A perfect match between two strings of equal length means that at each location in the string the symbols are identical. However, perfect matching is rare in the immune system. Partial matching in symbol strings can be defined using Hamming distance, edit distance, or a more immunologically plausible rule called r-contiguous bits [11] based on regions of contiguous matches. The rule looks for r contiguous matches between symbols in corresponding positions. Thus, for any two strings x and y, we say that $match(x,y)$ is true if x and y agree at no less than r contiguous locations.

Detectors can be generated in several ways. A general method (that works for any matching rule) is also the one apparently used by the immune system. It goes like this: Simply generate detectors at random and compare them against self, eliminating those that match self. For the "r-contiguous-bits" rule, the random generating procedure is inefficient—often exponential in the size of self.[4] However, more efficient algorithms based on dynamic programming methods allow us to generate detectors in linear time for certain matching rules [3]. The total number of detectors required to detect nonself (using the r-contiguous-bits matching rule) is the same order of magnitude as the size of self.[5]

The algorithm has several interesting properties. First, it is easily distributed because each detector covers a small part of nonself. A set of negative detectors can be split up over multiple sites, reducing the coverage at any given site but providing good systemwide coverage. Achieving similar systemwide coverage with positive detection is much more expensive; either a nearly complete set of positive detectors is needed at every site, resulting in multiple copies of the detection system, or the sites

[4]Interesting to note is that in the body, only 2% of the immature T cells entering the thymus complete the maturation process and become functioning T cells. It is not known how much of this deletion can be attributed to negative selection, but the proportion is thought to be sizable.

[5]This is a gross simplification. The actual number is heavily dependent on the organization of the self set, the false-negative rate we are willing to tolerate, and the choice of matching rule. See [3] and [4] for a more careful analysis.

Trend Micro, Inc.
EX1019-7

must communicate frequently to coordinate their results. A second point about this algorithm is that it can tolerate noise, depending on the details of how the matching function is defined.

Consequently, the algorithm is likely to be more applicable to dynamic or noisy data, like the intrusion-detection example, than for, say, cryptographic applications in which efficient change-detection methods already exist. The algorithm's feasibility was originally shown for the problem of computer virus detection in DOS environments [4] in which the protected data was DOS system files; the self set was generated by logically segmenting .com files into equal-size substrings of 32 (binary) characters; detectors (32-bit strings) were generated randomly; the *r*-contiguous-bits matching rule was used with thresholds ranging from 8 to 13 contiguous positions; and infections were generated by various file-infector viruses. For example, one self set consisted of 655 self strings and was protected with essentially 100% reliability by as few as 10 detectors. Similar results were later obtained with boot-sector viruses.

Conclusions

An intrusion-detection system could be part of a multilayered system, possibly sitting behind cryptographic and user authentication systems. It could be distributed among sites, possibly using the negative-selection algorithm. Because the databases of normal behavior are generated empirically, based on local operating conditions, each different site would have unique protection, conferring diversity of protection across sites. Finally, by focusing on anomaly intrusion detection, the intrusion-detection system trivially meets the requirement of being sensitive to new attacks.

Besides the five principles of multilayered protection, distributability, diversity, sensitivity to new intrusions, and inexact matching, other useful orga-

A Database of Normal Patterns

To build a database of normal patterns, we first collect a trace of system calls emitted by a normally running process. We then slide a window of size *k* across the trace, recording each unique *k*-symbol sequence. This technique goes by various names, including "time-delay embedding" and "n-gram analysis" and is shown in the Figure on a trace of seven system calls and a window size of three symbols, resulting in a database of four unique sequences. This method differs slightly from the one described in [5] and gives better results.

After the database of normal behavior has been constructed, new behavior can be monitored for anomalies by tracing the system calls and checking them against the existing database, as shown in the Figure). Here, a one-symbol change (`read` to `mmap`) in one position causes two mismatches in the checking procedure.

An important consideration is how to choose the normal behavior used to define the normal database. We have experimented with two methods—synthetic and actual. In the former, we trace a running process while exercising it via a set of synthetic commands. For example, we have a suite of 112 artificial email messages we use to exercise `sendmail`, resulting in a highly compact database of 891 different sequences, each of six system calls. This test suite is useful for replicating results, comparing performance in different settings, and during other kinds of controlled experiments. We are also collecting traces of normal behavior in live user environments. These data are difficult to collect and evaluate, but they provide important information about how our system is likely to perform in real-world settings, including resource-stressed environments and data on false-positive rates. For example, in initial studies of `lpr` in a "live" environment, we observed some growth in database size but less than we anticipated; during a four-month trial, the database roughly doubled in size, from 171 distinct sequences after one month to 354 after four, mostly due to network access errors under loaded conditions.

In preliminary experiments, we constructed databases of normal behavior for three different processes: `sendmail`, `wu.ftpd`, and `lpr`. The databases are generated by exercising each process under different operating conditions (like our suite of 112 messages that cause `sendmail` to exhibit a wide variety of behavior). These databases of normal behavior are surprisingly compact. For example, the `sendmail` database consists of 891 different sequences, `ftpd` consists of 663, and `lpr` consists of 182 in which each sequence is six system calls. Compactness suggests that the normal behavior of a running process is a small subset of the range of its possible behaviors.

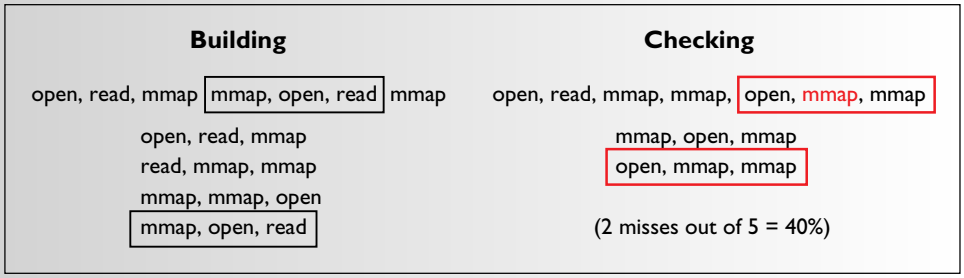

Figure. Sequence databases for normal behavior in Unix processes. This example trace of seven system calls produces a database of normal patterns containing four unique sequences. An example anomalous trace is constructed by replacing a `read` with an `mmap` (shown in red). This anomaly is detected in the checking phase because the anomalous trace contains two subsequences that do not appear in the normal database.

Trend Micro, Inc.
EX1019-8

nizing principles suggested by the immune system include:

- Disposability. No single component is essential.
- Automated response and self-repair.
- No secure components. Mutual protection among components compensates for lack of a secure code base.
- Dynamically changing coverage. In resource-constrained environments, changing coverage over time compensates for incomplete coverage at any single instant.

The two examples explored here of how these principles can be incorporated into a computer security framework represent some initial steps toward the larger intellectual vision of robust and distributed protection systems for computers. However, we ignored many important complexities of the immune system, some of which will have to be incorporated before we achieve our goal. For example, it is difficult to imagine how we could implement truly distributed protection without adopting the immune system strategy of self-replicating components or emulating some of the complex molecular signaling mechanisms (e.g., interleukins) used to control the immune response. Another aspect of the analogy not yet specified involves the circulation pathways through which immune cells migrate in the body. More generally, many other biological mechanisms have been incorporated into computational systems, including evolution, neural models, viruses, and parasites, many of which might be relevant to the computer security problem. In the near future, we hope to integrate the negative-selection algorithm with our intrusion-detection work and then begin augmenting the system with other immune system features.

Although we stress the similarities, there are also many important differences between computers and living systems. In the case of immunology and computer security, probably the most important difference is that the immune system is not concerned with the important problems of protecting secrets, privacy, or other issues of confidentiality. The success of the analogy ultimately rests on our ability to identify the correct level of abstraction, preserving what is essential from an information-processing perspective and discarding what is not. This task is complicated by the fact that natural immune systems process cells and molecules, but computer immune systems would handle other kinds of data. In the case of a computer-vision or speech-recognition system, the input data is in principle the same as that processed by the natural system—photons or sound waves. Deciding exactly how to draw the analogy between immunology and computation is a difficult task, and there are many different strategies that could be tried. We model peptides as sequences of system calls and binding as string matching. There are many other possible choices, some of which we hope to explore in future work.

References

1. Cohen, F. Computer viruses. *Comput. Secu.* 6 (1987), 22–35.
2. Denning, D.E. An intrusion-detection model. *IEEE Trans. Software Eng. 2*, 222 (Feb. 1987), 118–131.
3. D'haeseleer, P., Forrest, S., and Helman, P. An immunological approach to change detection: Algorithms, analysis, and implications. In *Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy* (Oakland, Calif., May 6–8, 1996). IEEE Press, Los Alamitos, Calif., 1996, pp. 110–119.
4. Forrest, S., Perelson, A.S., Allen, L., and Cherukuri, R. Self-nonself discrimination in a computer. In *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy* (Oakland, Calif., May 16–18, 1994). IEEE Computer Society Press, Los Alamitos, Calif., 1994, pp. 202–212.
5. Forrest, S., Hofmeyr, S., Somayaji, A., and Longstaff, T. A sense of self for Unix processes. In *Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy* (Oakland, Calif., May 6–8, 1996). IEEE Press, Los Alamitos, Calif., 1996, pp. 120–128.
6. Janeway, C.A., and Travers, P. *Immunobiology: The Immune System in Health and Disease*, 2d ed. Current Biology Ltd., London, 1996.
7. Kephart, J.O., White, S.R., and Chess, D.M. Epidemiology of computer viruses. *IEEE Spectrum 30*, 5 (May 1993), 20–26.
8. Kephart, J.O., Sorkin, G.B., Arnold, W.C., Chess, D.M., Tesauro, G.J., and White, S.R. Biologically inspired defenses against computer viruses. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1995.
9. Ko, C., Fink, G., and Levitt, K. Automated detection of vulnerabilities in privileged programs by execution monitoring. In *Proceedings of the 10th Annual Computer Security Applications Conference* (Dec. 5–9, 1994), pp. 134–144.
10. Murray, W.H. The application of epidemiology to computer viruses. Comput. Secur. 7 (1988), 139–150.
11. Percus, J.K., Percus, O., and Perelson, A.S. Predicting the size of the antibody-combining region from consideration of efficient self/nonself discrimination. In *Proceedings of the National Academy of Science 90* (1993), pp. 1691–1695.
12. Spafford, E.H. Computer viruses: A form of artificial life? In *Artificial Life II*, C.G. Langton, C. Taylor, J.D. Farmer, and S. Rasmussen, eds. Addison-Wesley, Redwood City, Calif., 1992, pp. 727–745.

Stephanie Forrest (forrest@cs.unm.edu) is an associate professor in the Department of Computer Science at the University of New Mexico in Albuquerque and wrote this article while on sabbatical at the MIT Artificial Intelligence Laboratory.

Steven A. Hofmeyr (steveah@cs.unm.edu) is a Ph.D. student in the Department of Computer Science at the University of New Mexico in Albuquerque.

Anil Somayaji (soma@cs.unm.edu) is a Ph.D. student in the Department of Computer Science at the University of New Mexico in Albuquerque.

Permission to make digital/hard copy of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage; the copyright notice, the title of the publication, and its date appear; and notice is given that copying is by permission of ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.

© ACM 0002-0782/97/1000 $3.50

Trend Micro, Inc.
EX1019-9