Biologically Inspired Approaches to Computer Security

Matthew M. Williamson
Information Infrastructure Laboratory
HP Laboratories Bristol
HPL-2002-131
June 12th , 2002*

E-mail: matthew_willimason@hp.com

computer, security immune system, biologically inspired

Biologically inspired approaches to computer security are appealing for two reasons. Firstly our computers are under attack, and biological species have been attacking each other for millions of years. Secondly as computers become more complex, traditional approaches to security are unlikely to scale and biological metaphors become increasingly powerful. Given the overlap in the domains, the hope is that taking ideas from biology and applying them to computer security will bear fruit.

The particular ideas that researchers have applied to computer security include the immune system, the diversity of species, epidemiology and non-representational systems.

This paper examines the pros and cons of biologically inspired approaches. It provides both an introduction and review of work in this area, as well as making suggestions for future work.

* Internal Accession Date Only
Approved for External Publication

Biologically Inspired Approaches to Computer Security

Matthew M. Williamson
HP Labs Bristol, Filton Road, Stoke Gifford, BS34 8QZ, UK
matthew_williamson@hp.com

Abstract

Biologically inspired approaches to computer security are appealing for two reasons. Firstly our computers are under attack, and biological species have been attacking each other for millions of years. Secondly as computers become more complex, traditional approaches to security are unlikely to scale and biological metaphors become increasingly powerful. Given the overlap in the domains, the hope is that taking ideas from biology and applying them to computer security will bear fruit.

The particular ideas that researchers have applied to computer security include the immune system, the diversity of species, epidemiology and non-representational systems.

This paper examines the pros and cons of biologically inspired approaches. It provides both an introduction and review of work in this area, as well as making suggestions for future work.

1 Introduction

Biologically inspired approaches to computer security are appealing because of the obvious analogies between the security of computer systems and the survival of biological species. Over evolutionary time, organisms have competed for scarce resources, and have attacked and exploited weaknesses in other organisms. Our computer systems are also continually under attack, from determined hackers and malicious code. Both areas can be characterised as arms races between attacker and defender as each evolves or invents better ways to attack or defend. These similarities suggest that one might be able to build effective computer security systems by borrowing ideas from biological systems.

Indeed we often use biological language e.g. virus, worm, antidote, immune system to describe aspects of computer security.

A more subtle reading of the analogy is that biological systems have evolved to be remarkably robust to fairly major changes in themselves and their environments. The mechanisms that they use to defend themselves from attack are equally robust. For example, our immune system responds slowly to pathogens that it has not encountered before, but remembers the pathogen so that it can react quickly should the body be re-infected. It does this without attacking other "foreign" proteins such as food, foetuses or non-harmful bacteria.

A further factor is that computer systems are becoming more complex. The individual computers are complex, and they are connected together in larger networks. This makes security more and more difficult (the more complex, the more likely that something will be forgotten). Centralised approaches to security have difficulty scaling to this complex world, and the very homogeneity required to make such approaches scale can itself be a source of vulnerability. Biological systems, on the other hand are nearly always decentralised and distributed, so are obvious sources of inspiration.

There are a variety of ways that researchers have taken biological ideas and applied them to computer security. Some of them have been quite abstract, using the biological idea more as a metaphor, while others have been more detailed, attempting to build systems that mimic their understanding of the underlying biological mechanisms.

The paper begins by introducing the various facets of computer security, before looking at the biologically inspired approaches that have been taken: models of the human immune system, diversity, studying the propagation of viruses, and non-representational systems. The paper then discusses the overall approach, questioning its applicability given the differences between biological and computer systems, and concludes with some recommendations for further work.

2 Computer security background

The general approach of computer security is to counter threats by using a combination of prevention, detection and response (Schneier, 2000; Anderson, 2001). Attacks are prevented by ensuring that vulnerabilities are removed, machines are isolated from others using firewalls etc.. Security violations are detected when they occur, and are then responded to. Of the three, the most effort is put into prevention, less into detection, and little into response, which is nearly always a manual process.

The threats themselves can be divided into two broad classes: attacks on data; and attacks on infrastructure. Attacks on data are ones where data is stolen (e.g. credit card numbers from web sites), or where data is altered

either for fraudulent purposes, or to damage reputation (e.g. defacing of web sites (alldas.org, 2002)). These attacks are often the work of an individual, although the attacks themselves are increasingly automated. These attacks often involve "intrusions", where the attacker breaks into the computer system by exploiting a vulnerability in a piece of software and gains inappropriate access to data. Taking a traditional view on security, these attacks are prevented by using firewalls to protect computers from outside attack, detected using Intrusion Detection Systems (IDS), and the response is often a manual process.

Intrusion Detection Systems generally monitor either network traffic or the behaviour of individual machines/programs and fall into two types: those that look for patterns of known exploits (misuse detection); and those that attempt to detect new or previously unknown exploits (anomaly detection). The first type will miss new attacks until they have been characterised, and the second type is likely to be plagued with false alarms.

Attacks on infrastructure also fall into two categories: malicious mobile code, where self-replicating code causes damage to machines, or consumes resources (network bandwidth, disk space etc.); and denial of service attacks, where a flood of individually legitimate requests causes overload and failure of targeted systems.

The security response to viruses is generally to use virus scanning software. Viruses infect files on computers and the tell-tale changes that they make can be detected by scanning files looking for suspicious data. The most common scanners use a database of "virus signatures". Like the misuse detectors, these are vulnerable to new viruses until a signature has been developed and distributed, which is often a long and tedious process. This speed is becoming more significant, as viruses increasingly use Internet protocols to spread at alarming rates (eEye Security, 2001; Weaver, 2002).

For denial of service attacks, prevention is difficult (these are legitimate requests), detection is fairly straightforward (the machine is overloaded), but response is rather difficult, since the requests could be coming from many machines across the Internet, over which the victim has no control. This is a problem since denial of service attacks are remarkably common (Moore, Voelker, & Savage, 2001).

All of these attacks are serious issues for organisations, and all of them require significant manpower to counter and clean up afterwards.

3 Computer Immune Systems

3.1 Biological immune systems

This section considers the various ways that researchers have interpreted biological immune systems for security. The immune system is perhaps the most obvious "system" that would have an analogy for security. Its role is to defend against attack, and patch and clean up after an attack. It is thus appropriate for all of the threats.

There are many good overview textbooks for the immune system (e.g. Janeway *et al.* (2001)) but in brief antigens (foreign proteins) are recognised by antibodies (immune system detectors). The antibodies are highly specific, only binding to a small set of antigens. If they do bind, then a complex set of events occur that result in the foreign protein being destroyed.

Antibody cells are covered with antigen detectors, and they are as theoretically likely to match and destroy healthy cells as foreign proteins. This would obviously be undesirable, and in most cases does not happen. The immune system thus appears to be able to discriminate between "self", and "non-self". An alternate view is that the immune system does not discriminate self/non-self, but responds primarily to damage to cells, recognising foreign proteins that cause damage, not foreign proteins per se (Matzinger, 1994).

There are two ways to think about detecting the difference between self and non-self. One would be to model self, and detect differences from self; or the opposite, model non-self and detect non-self. The immune system chooses the latter approach, it's antibodies being detectors for non-self. This is somewhat counter-intuitive, as the space of possible antigens is much larger than the number of self proteins. Perelson & Weisbuch (1997) estimate that there are around 10^{15} antigens, and roughly 10^6 different proteins that make up the human body. Humans have around 10^{10} antibodies, of which there are roughly 10^7 different types (antibodies exist in multiple identical copies). There are thus more antibodies than different self proteins, but many less than there are different antigens.

The immune system also has the property that it can both react to antigens that it has not been in contact with before, and can also remember antigens for long periods of time (70+ years). The secondary response to an antigen is much quicker and more effective than the first. This property is exploited in vaccination.

3.2 Computer immune systems

Stephanie Forrest and her group at the University of New Mexico (Forrest, 2002) have applied models of the human immune system to the problem of intrusion detection. As described in Section 2, the idea is to look for changes in data, odd patterns in network traffic or odd behaviour of running programs that are evidence that an intruder is breaking in and gaining illegal access to data. On finding an intrusion, these systems alert system administrators who then deal with the problem as they see fit. While making these systems accurate is important (detecting all the intrusions), reducing the number of false alarms (normal behaviour classed as an intrusion)

is vital because of the human involvement.

Forrest makes the analogy between self in the body, and normal behaviour of a computer system (non-self is thus abnormal behaviour). Self is represented as a set of strings (with a variety of different representations depending on the domain), and antibodies or detectors are also represented as strings. The binding between the strings is modelled by a matching function, the most common one being r-contiguous bits, which returns true when two strings match in more than r contiguous positions. This allows detectors to match a variety of strings. In some of the papers (Forrest *et al.*, 1994; Hofmeyr, 1999) they use detectors for non-self (which is directly analogous to the immune system), and in others (Forrest *et al.*, 1996; Somayaji & Forrest, 2000) they use detectors for self.

By casting the problem as differentiating between normal and abnormal, the systems are capable of detecting novel attacks. They also turn the security problem into a classification problem: given a training set of normal examples, but no abnormal examples, learn a set of detectors that given a new string will predict whether it is normal or abnormal. The labels on the data are generally provided by a human.

An immune system inspired approach to this problem is not to model normal (self), but to generate detectors for abnormal (non-self), by generating strings that do not match any of the normal strings. A new string can then be matched against all of the detectors, and if any of them match, then the the new string can be classified as abnormal.

Detectors for non-self are preferred over detectors for self because they produce less false alarms, as shown in Figure 1. If non-self detectors are used and there is inadequate coverage, then the errors that result are false negatives (missing intrusions). On the other hand, using self detectors will results in false alarms, which are problematic. From a performance point of view, a test string need only be compared against non-self detectors until one matches, while it would need to be compared to all self detectors to ensure that the test string is non-self. However, if self is more common than non-self (which is a reasonable assumption), using detectors for self may be more efficient: test strings which are really self will end up being checked against all of the non-self detectors, but only against self detectors until a match is found.

If detectors for non-self are used then the probability of a false negative error P_f (missing an abnormal string) will go down as the number of detectors N_R increases. In addition, the number of detectors required to cover the whole of non-self (which is generally much larger than the size of self) will depend on how many strings each detector can detect. This can be represented by the probability P_m that two random strings will match. The error rate P_f is given by the probability that a random

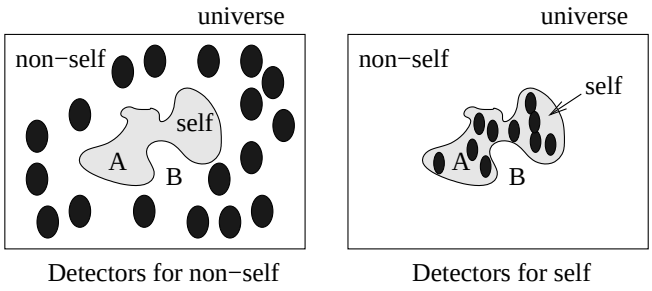

Figure 1: Differences between self and non-self detectors with incomplete coverage. The universe of strings is represented by the box, with self strings represented by the grey blob, and non-self surrounding self. Detectors are indicated by solid ellipses. The left hand figure shows the effect of using non-self detectors. Considering the test strings A and B, A will be accurately classified as self (it does not overlap any detectors), but B will be a false negative (an anomaly that will be missed). The right hand picture shows the situation when detectors for self are used. A is now a false positive (it is self classified as non-self), while B is accurately classified.

string will not match any of the detectors (Forrest *et al.*, 1994)

$$P_f = (1 - P_m)^{N_R} \approx e^{-P_m N_R} \tag{1}$$

so rearranging

$$N_R \approx \frac{-\ln P_f}{P_m} \tag{2}$$

This means that the number of detectors needed is proportional to $\ln P_f$, which is good, and is inversely proportional to P_m. P_m is the probability that two random strings will match, and is thus very sensitive to the representation used. For realistic applications, with many pieces of information encoded into the strings, P_m is likely to be small, making N_R large. It is worth noting that N_R is independent of the size of the self set, defined as N_S.

Since these detectors are different from all of the self set, they can be quite difficult to find. The simplest algorithm is the generate-and-test or negative selection algorithm (Forrest *et al.*, 1994), shown in Figure 2. Detectors are generated at random, checked against all of the self set and any that do not match are kept. There is no attempt made to ensure that there is even coverage of the non-self space, although other immune models (e.g. Farmer, Packard, & Perelson (1986)) do include this factor.

The number of detectors that need to be generated can be estimated by considering that the probability of a detector not matching any of the self strings is $(1 - P_m)^{N_S}$, so if N_{RO} is the number of detectors that need to be tested, $N_R = N_{RO}(1 - P_m)^{N_S}$. Rearranging this gives

$$N_{RO} = \frac{N_R}{(1 - P_M)^{N_S}} \approx N_R e^{P_m N_S} \tag{3}$$

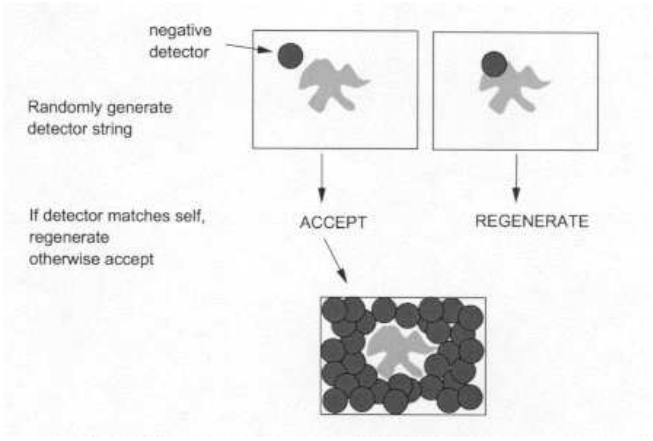

Figure 2: Negative selection. Detectors are generated at random, and compared against the self set (represented by the light grey blob). If they match, they are rejected, but if they do not match, then they are kept as detectors of non-self. The end result is that non-self is covered with detectors. Figure taken from Hofmeyr (1999).

Thus N_{RO} is proportional to N_R (so has the same dependency on P_f and P_m), and is also exponential in the size of N_S. This means that it might take a very long time to generate enough detectors[1]. Forrest *et al.* (1994); Hofmeyr (1999) argue that this is not a significant problem, although this was not the experience of Kim & Bentley (2001) when applying this algorithm in an intrusion detection application.

Given the number of detectors required, and the difficulty of generating them, why is such a simple algorithm used when there are other classification algorithms (e.g. neural networks) available? There are two reasons, firstly that by being simple, it is possible to change the model of non-self easily by deleting and regenerating detectors. This is important for computer systems where self/normal may change frequently. Secondly there are no examples for abnormal with which to train a more sophisticated model, and generating those examples is costly regardless of the model used.

Other researchers have used immune system ideas in a similar vein, although mostly for intrusion detection (Kim & Bentley, 2001; Dasgupta, 1999; Williams *et al.*, 2001). Dasgupta & Attoh-Okine (1997) is a more general review of immune system based approaches.

Overall, exploiting the immune system results in security systems with many simple detectors, that allow the detection of previously unknown events, and give control over the type of errors that will occur. On the other hand, a lot of simple detectors are needed to cover even small amounts of data, and even more have to be generated to find those.

[1] D'haeseleer, Forrest, & Helman (1996) shows a greedy algorithm that can generate detectors in linear time, but it is still a significant computational load.

3.3 Anti-virus immune system

An alternative to having many simple detectors is to have fewer more complex detectors. This has been the approach of the anti-virus group at IBM (2002a). Their "immune system" analogy concentrates more on the memory aspect of the human immune system, and neglects the details of antibodies etc.

The IBM work has at its core the idea of signature based scanning for virus detection and elimination, which was discussed in Section 2. The antibodies are the signatures, and they are stored in a memory (database). Files are scanned comparing them against the memory to determine if the computer is infected with a virus. Virus signatures are difficult to generate, although in a different way from the previous detectors. In order to generate a virus signature, the action of the virus in a system needs to be understood, and represented such that it can be checked by scanning.

Kephart (1994) is an early paper presenting this approach. Special "decoy" programs (whose only role is to sit on machines and wait to be infected) are used to "trap" viruses, and provide data for the automatic generation of the virus signature. This signature is then propagated in a self-replicating manner to the neighbours of the infected machine, as shown in Figure 3. This has the advantage that the antidote can be applied exactly where it is needed, at the centre of the virus spread.

A later paper (White *et al.*, 1998) shows the development of this work into a "commercial grade immune system". Interestingly this is a much more engineering based approach, removing nearly all of the detailed biological analogies: the virus signature is extracted from a central server to which users submit "virus reports", and the distribution of signatures is also centralised. This is probably because generating the virus signature is a computationally expensive task, and also one that might need manual intervention. This is easier to manage if it is centralised. Once you have a central signature processor, centralised distribution of signatures also makes sense. What is lost compared to their original system is speed of response (due to communication delays), and the ability to pinpoint the antidote/signature at the virus outbreak.

This research shows that the work required to accurately detect and defend against viruses is considerable. The centralised approach is one way to allow most machines to not be affected by this computational load. It also shows the complexity of immune type systems, whether they consist of many simple detectors or fewer more complex ones.

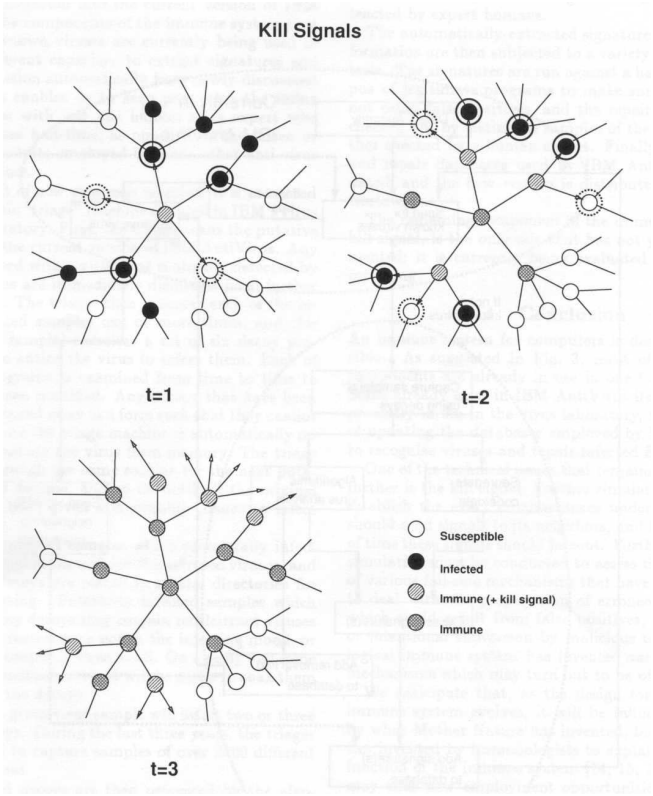

Figure 3: Since each host has its own signature-extraction engine, once they have calculated the signature for the virus that has infected them, they can immunise themselves, and spread the antidote to their neighbours. This means that the antidote can be applied to infected hosts (black circles with extra rings), and immunise hosts that are not yet infected (white circles with dotted rings). This fights the self-replication of the virus with the self-replication of the antidote. Figure taken from Kephart (1994).

3.4 Danger theory

In the two previous examples, the immune system is understood as differentiating normal and abnormal behaviour, but how can they be differentiated? The definition of normal in the Forrest case is determined either by a human labelling the data, or in Hofmeyr (1999) by the system asking a human for confirmation. In the IBM work, although decoy programs are used so that viruses can be found automatically, in their industrial system the detection of normal/abnormal is carried out by humans. In the human immune system, the idea of self/non-self discrimination defers the question of what is self to some other system.

Self/non-self and normal/abnormal are convenient labels for what these detection systems attempt to do. However, looking at the data without a human labeller, it is impossible to differentiate between the two. Even if the decoy programs of the IBM scanner catch viruses, they cannot detect whether the virus is doing any *harm*. This is a problem of symbol grounding (Harnad, 1990). What is needed is a grounding signal that is within the system and indicates damage.

The Danger theory of immunology (Matzinger, 1994) suggests that damage to cells is the key signal that enables a biological antibody response. This provides the grounding signal, that damage is abnormal. This theory explains why the body can choose to accept some foreign proteins (e.g. food, foetuses), but not others (e.g. harmful antigens), because it is only the proteins that cause harm that elicit an immune response.

The computer immune system described by Burgess (1998) describes an implementation of this idea, albeit for fault tolerance rather than security. He used the signals generated by dying computer processes as his grounding signal, allowing him to detect when some fault had occurred. While this work is an interesting first step, his "danger signals" are rudimentary. If computers were endowed with more ways of detecting their state (or health), and more ways of affecting that state, then building systems like these would be easier.

4 Diversity

A tactic that biological systems use to thwart attack is diversity. By making each individual slightly different (both genetically, and through experience), the overall species is robust to changes in predators, food supply, environmental conditions etc. A monoculture is much more vulnerable to being completely wiped out by these changes.

The diversity in nature contrasts strongly with the monoculture of computer systems. While there are variations of operating system type, version, and applications, the overall diversity is low. Indeed versions of an application are designed to behave identically. While this facilitates management of computer systems, it does make computer systems vulnerable to attack. This vulnerability was pointed out in the aftermath of the first Internet worm (Eichin & Rochlis, 1989).

Forrest, Somayaji, & Ackley (1997) present a good discussion of this topic, and make the point that diversity does not *remove* vulnerabilities, rather it makes each computer vulnerable in a *different* way. There is only so much diversity that is possible because of the layered architecture of computer systems: each layer must honour its contract/interface with adjacent layers. Some diversity can be introduced by making aspects of the operating system that are arbitrary, random. This will disrupt exploits that rely on any underlying consistency.

For example, a common attack method is a "buffer-overflow" attack, where the attacker overwrites the stack of the target machine, so that their code rather than the normal program is run (Aleph One, 1996). Buffer overflow exploits are sensitive to the exact layout of the stack,

but since compilers generally produce identical stack layouts, the exploits are general—a single exploit will be successful against all installations of the software. While there are ways of making stacks that are harder to attack (e.g. Cowan *et al.* (1998)), Forrest, Somayaji, & Ackley (1997) suggest an approach based on diversity. They produced a compiler that padded the stack with random amounts of blank space. This does not affect the normal running of the program, and does not reduce the total number of vulnerable machines, but does reduce the number of machines that are vulnerable to any one exploit. It also makes the exploit slightly harder to write, since the exact stack layout cannot be known in advance[2].

Pu *et al.* (1996) also suggest a method for adding diversity in operating systems, and Linger (1998) suggests ways to obfuscate code. Other suggestions are polymorphic code, where several teams code components independently, and select different combinations at run or compile time.

There are a number of issues with all these approaches:

- To be effective, the diversity needs to make machines vulnerable in different ways. However, since the different layers must honour their interfaces, adding diversity may not have any effect. For example, if the vulnerability is in the specification, any implementation (diverse or not) will be vulnerable.

- Software, (particularly enterprise software) is frequently brittle, and often written in ways that exploit "arbitrary" parts of the operating system. Changing those "arbitrary" parts can cause programs to fail. While it is not good practise to write programs in this way, many significant ones are.

- For other systems, removing diversity is key to reliability and portability. Early games software interacted directly with the computer hardware, with many bugs and incompatible systems. Microsoft's Windows 95 operating system contained a device management layer which masked the diversity of the computer hardware, making games more robust.

- In all security areas, there is a trade-off between the risk associated with a threat, and the costs incurred to defend against it. Using diversity as a counter measure definitely increases costs, but it is not clear how effective it would be at reducing the risk. This is because it does not remove vulnerabilities, it just makes them less uniform. A better understanding of the effect of diversity on risk would be helpful in understanding this trade-off.

[2]Unfortunately, there are methods for writing buffer-overflow attacks that are somewhat insensitive to the exact stack layout (Aleph One, 1996).

In summary, diversity is a nice idea for biological systems, but it is plagued with economic and software compatibility issues. It highlights some of the tensions between computer and biological systems. Computer systems are designed with a narrow set of criteria of which manageability is one. Diversity is a good idea, but only for systems without this kind of demand. Diversity for security will only become a good idea if the criteria under which we evaluate our computer systems are changed.

5 Virus Epidemiology

Viruses are a significant threat to computer security (Grimes, 2001). They are named after biological viruses: they are generally small pieces of code that "infect" a host, and use its resources (processor, memory, bandwidth) to damage the host and propagate itself.

The proposition of the biological work in this area is that by studying the propagation and epidemiology of biological viruses, the behaviour of computer viruses can be understood in more detail, and better countermeasures can be devised.

The field of mathematical epidemiology (modelling the spread of disease mathematically) has a long history (see Kephart & White (1991) for a review). Most of the work on computer viruses has concentrated on the effect of topology, the characteristics of the network over which the virus spreads. This is because unlike biological viruses that are transmitted when individuals are close to one another, computer viruses spread over a network. For example, an email virus spreads over the network formed by the address books of email users.

A standard epidemiological model for virus infection is the Susceptible $\rightarrow$ Infected $\rightarrow$ Susceptible (SIS) model. In this model individuals start susceptible, become infected at a rate λ, and are cured at a rate δ whereupon they can be re-infected. If this system is modelled, the system behaviour depends on the ratio $\gamma = \delta/\lambda$. If $\gamma > 1$, the cure rate is greater than the infection rate, and the virus does not spread. If $\gamma < 1$, then the virus does successfully propagate, reaching a steady state prevalence (or occurrence) of $(1 - \gamma)$ of the population.

Kephart & White (1991) examined the interaction of this model with various types of graphs, and found that as the connectivity of the graph was reduced, the virus was less and less likely to propagate, e.g. it needed a much lower value of γ. They used this to suggest that virus scanning software could be effectively used to prevent virus propagation, since that software essentially increases the cure rate δ, so increasing γ. Interestingly they also pointed that real computer viruses have much lower prevalences than might be expected i.e. lower than $(1 - \gamma)$.

The topologies that Kephart considered were appropriate to the threats at the time—boot and file viruses transmitted by floppy sharing. Modern viruses use In-

ternet protocols, which appear to have "small-world" or scale-free topologies (Albert, Jeong, & Barabási, 1999; Faloutsos, Faloutsos, & Faloutsos, 1999). These types of networks have a small but significant number of nodes with many connections, which makes it easy for viruses to spread quickly.

More recent work by Pastor-Satorras & Vespignani (2001) has found that when viruses spread on scale-free networks, there is no threshold on the value of γ i.e. all viruses spread, regardless of the value of γ.

As yet, this field has not generated any significant advances for combating viruses, although this newer work does hold some promise. It would be interesting to apply more sophisticated models of the individuals e.g. the Susceptible $\rightarrow$ Infected $\rightarrow$ Resistant (SIR) model more accurately models present day anti-virus software. In addition, these models mostly assume random propagation across the network, but viruses such as Code Red II (eEye Security, 2001) have sophisticated propagation strategies, designed to find vulnerable machines both local to and far from the infected machine.

6 Non-representational approaches

In computer security, the usual paradigm can be described as prevent, detect, and respond (Schneier, 2000), usually involving a human to understand the output of the detection and determine an appropriate response. A more biologically plausible approach is to tightly couple the detection and response phases rather like a control loop, so that overall the system does the right thing, but with no easy human understanding. This difference is illustrated in Figure 4.

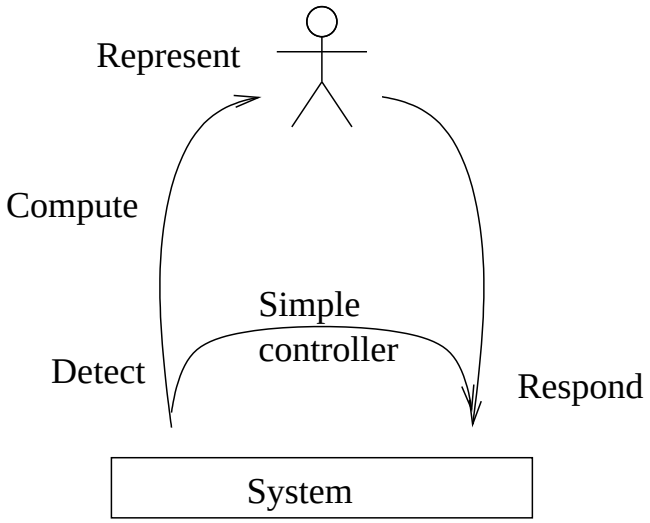

Figure 4: The figure shows the generic control of a system either by a simple control system that detects and responds, or with a human. For the human to understand the system, a human-interpretable representation needs to be computed. This is hard to calculate without errors, and any delay in the system (from computing the representation, or from the human), makes the chances of an inappropriate response higher.

When a human is in the loop, a representation of the system needs to be computed, and effort taken to reduce errors. The human is then responsible for deciding which (generally discrete and irrevocable) response is made. Computing an error-free representation is hard, as the discussion of errors in Section 3.2 highlights. In addition, any delay in the system (either from the human, or computing the representation) increases the likelihood that the planned response will be inappropriate. If a simple controller is used, there is no need to compute a complex representation; the loop is fast so responses are corrective actions rather than large changes; and the system is tolerant to detection errors, since these will be corrected by the loop before they generate a response that is inappropriate.

There are two issues with applying this idea to computer security, firstly being the lack of things to measure. Processor load, memory usage etc. are available, but are not direct measurements of "damage" or illegal activity. In addition corrective actions are typically binary and irrevocable e.g. "switch the machine off", "patch it", "shut down the service" etc. This means that the detection phase needs to be accurate, because the responses are so definite. In order to relax the need for accurate detection, the responses need to be made more benign e.g. "slow down the service".

Somayaji & Forrest (2000) is a good example of this approach. Their system is used to automatically respond to intrusions caused by e.g. an attacker exploiting a buffer overflow vulnerability in sendmail, a common mail program. The detection system looks for anomalous behaviour of running programs, by building a database of short sequences of system calls (these are requests to the operating system for resources e.g. memory access). The actual system calls made by a running process can be checked against this database. Previous work (Forrest *et al.*, 1996) showed that attacks on programs manifest themselves as a burst of abnormal system call sequences (since the program is doing something that it would not normally do).

The response part of the system is to insert delays between the system calls, in proportion to the number of abnormal patterns produced by the program. If too many abnormal patterns occur the process is terminated. The system thus degrades the performance of programs that are under attack, because in practise the attacks generate so much abnormal activity that the programs are shut down. At the same time, the system is tolerant to false alarms, since these are rare enough to not have a significant effect on the normal operation of the system.

Other complete systems (detection—response) such as

described by Dasgupta (1999), offer the same kind of automatic system, but are not in the same spirit, trying to use multiple detectors to improve the detection accuracy, because they are using binary (on/off) responses.

A better example would be Mahajan *et al.* (2001) where the flow of traffic through an Internet router is examined, and packets are preferentially dropped from the most congested flows (a flow is defined as all the packets to the same destination). Some Internet protocols interpret dropped packets as signalling congestion, and will "back off", reducing the rate at which they send packets. This system allows the router to deal with over-congestion from both flash floods (genuine over congestion) and denial of service attacks.

This approach to security has some promise because experience in other domains e.g. robotics (Brooks, 1986) has suggested that low level feedback loops can be more robust than using higher-level representations. However the difficulties of designing appropriate detectors and responses in computer systems remains a challenge.

7 Discussion

Looking at the work on biologically inspired approaches to computer security, one is struck that firstly there is not much of it, and secondly that none of it (with the exception of the IBM virus scanner) is wildly successful. If the analogies are so good, why has there not been more progress? This section looks at some of the counter arguments, asking whether the differences between biological and computer systems are too great, whether the time-scales that they operate under are too different, and whether the concept of "survival of the gene not the individual" is really appropriate for computer systems.

7.1 Fundamental differences

The first question is whether the differences between computer and biological systems are too great for any useful analogies to be drawn.

The most obvious difference is that biological systems are massively parallel and our computer systems are not. Taking for example our immune systems, with 10^{10} antibodies being recycled at the rate of 10^7 per day, our immune system uses significant amounts of the body's resources. It is only because all of the reactions can occur in parallel that this does not affect e.g. the speed of the brain. Biologically inspired approaches largely assume this level of parallelism, and while they can be implemented in serial computers, they are less effective. This progression is evident from the IBM virus scanner discussed in Section 3.3. Their first system had individual hosts computing virus signatures, and distributing them locally. Because this is too computationally expensive, the "commercial-grade" solution is centralised. This might be more effective, but loses some advantages: the response is slower because of communication delays, and the antidote is not immediately applied in the most affected areas. If the virus signature could be calculated without affecting the performance of the individual hosts, it might have been a different story.

This issue is compounded by the criteria with which we evaluate successful computer designs. Unlike biological systems, where the criteria is to survive and reproduce, computer systems are evaluated against a much narrower set of benchmarks. The result is biological systems which are massively parallel, redundant and robust, and computer systems which are serial, optimised and brittle. The driving characteristics of computing systems are performance and cost, and serialised biologically inspired approaches do not fare will against those criteria. Neither for that matter does any attempt to make computers secure, fault tolerant or robust.

As computers get more complex, the need for robustness should change the criteria (some moves in this direction are argued by IBM (2002b)), and this should create opportunities for all types of solution, biologically inspired or not.

Even without changing the criteria there are still ways that parallel systems could be incorporated. A good analogy would be the graphics hardware in computers, which handles the computationally expensive job of generating the display. Perhaps "immune chips" are needed that are tightly integrated with the processor and operating system but run in parallel. Alternatively dedicated machines could be used.

A further difference between biological and computer systems is the granularity of "sensors" and "actuators" available. Our skin is plastered with sensory receptors, and our muscles are incredibly finely controlled, with each muscle fibre having its own nerve supply. The kind of sensory information available in our computers e.g. processor load, memory allocation, etc. are more akin to measuring heart-rate than nerve signals. In addition, as pointed out in Section 6, having a variety of benign responses is important. Adding these features would allow the creation of more interesting automatic systems.

7.2 Different time-scales

The second issue is that of time-scales, and is more directly relevant to security than the previous section. The argument goes that biological viruses evolve slowly, changing randomly until they become a problem. Biological defences are optimised for this slow threat. Computer viruses, on the other hand are designed to be deadly by intelligent people, and thus biologically-inspired defences are inappropriate (Schneier, 2000).

There are two parts to this issue, the first being the suggestion that evolution cannot provide defences against attacks created by intelligent hackers. While evolution is slow, it is also parallel and thorough, and

evolved systems contain a cumulative memory of past attacks (their forebears have survived attacks in the past, and those defence mechanisms are passed on). There is no reason why evolved systems would be any worse than human designed systems on countering intelligent attack, and they could be better.

On the other hand, evolved systems are not immune to step changes, in fact they often cause mass extinction and death (cf. the environmental changes and dinosaurs). But human engineered systems are just as vulnerable. For example, virus scanners had to be greatly enhanced when virus writers started producing encrypted viruses. These had a different "footprint" on every machine, so the then-current scanning technology could not detect them (Grimes, 2001).

It would be fair to say that defence systems, biological or not, have problems with large changes in the skills of the attacker. However, biological system have been involved in these kinds of arms races for much longer than computer systems, and so should provide useful inspiration.

7.3 Survival of the individual

The final argument against biological approaches is that they often stress the survival of the gene at the expense of the individual. This is fine, as long as it is not an important computer that is the unlucky individual!

This could or could not be a problem depending on at what level the analogy is drawn: is an individual a computer? a computer component? etc. For example, if individuals are Internet routers, then the overall species performs routing. The routing system is already robust to failures of individual routers, so applying this sort of security solution is not a problem. On the other hand, having the routing system as an individual is not sensible.

This argument stems from a traditional view of computer security where the world is viewed in binary terms, computers are either infected or uninfected. Even in the security community there has been a move towards thinking about the survival of the mission, not the computing infrastructure (Ellison *et al.*, 1997). More information on this work can be found in DISCEX (2000, 2001).

7.4 Summary

The main stumbling blocks for biologically inspired approaches to security are the architectures of computers and the criteria under which we evaluate them. However, while these factors make it more difficult, they are not insolvable.

8 Conclusion

Biological approaches to computer security have been tried by a number of different research groups, in a variety of different security areas. Models of the human immune system have been used for intrusion detection, combating viruses and building fault-tolerance. The diversity of biological species has inspired the creation of compilers that create diverse code. The propagation of disease has inspired researchers to model the spread of computer viruses. The non-representational systems observed in nature have inspired others to build automatic systems to control congestion on networks and prevent intrusions. Some of these have been successful (e.g. virus scanning), others less so (e.g. the issues surrounding diversity), and for others it is too soon to tell (e.g. virus propagation).

The general approach is made more difficult by the fact that biological systems are inherently parallel, while computer systems are generally centralised. This, together with the focus of computer systems being high performance and low cost, makes the computational cost of biologically inspired approaches difficult to justify. Although to be fair, all approaches to security suffer from this criteria.

To address this issue, more work needs to be done either to parallelise computers, or to ensure that these security functions can be run in parallel. In addition, more information about the state of a machine is needed, as well as more ways to influence that state without shutting down the system.

There are also some areas, particularly in fighting Internet denial of service attacks, where there is a desperate need for better technology. In this case, the distributed nature of the system should make biological analogies a good place to look for inspiration.

To summarise, biological approaches to computer security is an interesting area that does show promise, although a relaxing of the constraints of the problem would help progress in this area.

9 Acknowledgements

The author would like to thank Dave Cliff, Paul Layzell, Stefek Zaba and the BICAS group at HP Labs (http://www.hpl.hp.com/research/bicas/).

References

Albert, R.; Jeong, H.; and Barabási, A.-L. 1999. The diameter of the world wide web. *Nature* 401(130).

Aleph One. 1996. Smashing the stack for fun and profit. *Phrack* 7(49). Available at http://immunix.org/StackGuard/profit.html.

alldas.org. 2002. alldas.org defacement archive. Available at http://defaced.alldas.org.

Anderson, R. J. 2001. *Security Engineering: a guide to building dependable distributed systems.* John Wiley & Sons.

Brooks, R. A. 1986. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation* RA-2:14–23.

Burgess, M. 1998. Computer immunology. In *Proceedings of the 12th Systems Administration Conference (LISA '98)*, 283–298.

Cowan, C.; Pu, C.; Maier, D.; Walpole, J.; Bakke, P.; Beattie, S.; Grier, A.; Wagle, P.; Zhang, Q.; and Hinton, H. 1998. StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks. In *Proceedings of the 7th USENIX Security Conference*, 63–78.

Dasgupta, D., and Attoh-Okine, N. 1997. Immunity-basedsystems: A survey. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics.*

Dasgupta, D. 1999. Immunity-based intrusion detection system: A general framework. In *Proceedings of the 22th National Information Systems Security Conference.*

D'haeseleer, P.; Forrest, S.; and Helman, P. 1996. An immunological approach to change detection: Algorithms, analysis and implications. In *Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy*, 110–119. IEEE Computer Society Press. Oakland, CA.

2000. *2000 DARPA Information Survivability Conference (DISCEX 2000)*, Hilton Head, SC: IEEE Computer Society.

2001. *2001 DARPA Information Survivability Conference (DISCEX 2001)*, Anaheim, CA: IEEE Computer Society.

eEye Security. 2001. .ida code red worm, http://www.eeye.com/html/research/advisories/al20010717.html.

Eichin, M. W., and Rochlis, J. A. A. 1989. With microscope and tweezers: An analysis of the internet virus of November 1988. In *Proceedings of the 1989 IEEE Computer Society Symposium on Security and Privacy.*

Ellison, R. J.; Fisher, D.; Linger, R. C.; Lipson, H. F.; Longstaff, T. A.; and Mead, N. R. 1997. Survivable network systems: An emerging discipline. Technical Report CMU/SEI-97-TR-013, Software Engineering Institute, Carnegie Mellon University.

Faloutsos, M.; Faloutsos, P.; and Faloutsos, C. 1999. On power-law relationships of the internet topology. In *ACM SIGCOMM*, 251–262.

Farmer, J. D.; Packard, N. H.; and Perelson, A. S. 1986. The immune system, adaptation and machine learning. *Physica D* 22:187–204.

Forrest, S.; Perelson, A. S.; Allen, L.; and Cherukuri, R. 1994. Self-nonself discrimination in a computer. In *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, 202–212. IEEE Computer Society Press.

Forrest, S.; Hofmeyr, S. A.; Somayaji, A.; and Longstaff, T. A. 1996. A sense of self for Unix processes. In *Proceedinges of the 1996 IEEE Symposium on Research in Security and Privacy*, 120–128. IEEE Computer Society Press.

Forrest, S.; Somayaji, A.; and Ackley, D. H. 1997. Building diverse computer systems. In *Workshop on Hot Topics in Operating Systems*, 67–72.

Forrest, S. 2002. http://www.cs.unm.edu/~immsec/.

Grimes, R. A. 2001. *Malicious Mobile Code: Virus Protection for Windows.* O'Reilly & Associates, Inc.

Harnad, S. 1990. The symbol grounding problem. *Physica D* 42:335–346.

Hofmeyr, S. A. 1999. *A Immunological Model of Distributed Detection and its Application to Computer Security.* Ph.D. Dissertation, Department of Computer Science, University of New Mexico.

IBM. 2002a. http://www.research.ibm.com/antivirus/.

IBM. 2002b. http://www.research.ibm.com/autonomic/.

Janeway, C. A.; Travers, P.; Walport, M.; and Shlomchik, M. 2001. *Immunobiology: The Immune System in Health and Disease.* Garland Science Publishing, fifth edition.

Kephart, J. O., and White, S. 1991. Directed graph epidemiological models of computer viruses. In *Proceedings IEEE Symposium on Security and Privacy.*

Kephart, J. O. 1994. A biologically inspired immune system for computers. In Brooks, R. A., and Maes, P., eds., *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, 130–139.

Kim, J., and Bentley, P. J. 2001. An evaluation of negative selection in an artificial immune system for network intrusion detection. In *Proceedings of Genetic and Evolutionary Computation Conference 2001 (GECCO-2001)*, 1330–1337.

Linger, R. C. 1998. Systematic generation of stochastic diversity in survivable systems software. Available from http://www.cert.org/research.

Mahajan, R.; Bellovin, S. M.; Floyd, S.; Ioannidis, J.; Paxson, V.; and Shenker, S. 2001. Controlling high bandwidth aggregates in the network. Technical report, AT&T Center for Internet Research at ICSI (ACIRI). Draft.

Matzinger, P. 1994. Tolerance, danger and the extended family. *Annual Review of Immunology* 12:991–1045.

Moore, D.; Voelker, G. M.; and Savage, S. 2001. Inferring internet denial-of-service activity. In *Proceedings of the 10th Usenix Security Symposium.*

Pastor-Satorras, R., and Vespignani, A. 2001. Epidemic spreading in scale-free networks. *Physical Review Letters* 86(14):3200–3203.

Perelson, A. S., and Weisbuch, G. 1997. Immunology for physicists. *Reviews of Modern Physics* 69(4):1219–1267.

Pu, C.; Black, A.; Cowan, C.; and Walpole, J. 1996. A specialization toolkit to increase the diversity of operating systems. In *Proceedings of the 1996 ICMAS Workshop on Immunity-Based Systems.*

Schneier, B. 2000. *Secrets and Lies.* John Wiley & Sons Inc.

Somayaji, A., and Forrest, S. 2000. Automated response using system-call delays. In *Proceedings of the 9th USENIX Security Symposium*, 185–197.

Weaver, N. C. 2002. Warhol worms: The potential for very fast internet plagues. Available from http://www.cs.berkeley.edu/ nweaver/warhol.html.

White, S. R.; Swimmer, M.; Pring, E. J.; Arnold, W. C.; Chess, D. M.; and Morar, J. F. 1998. Anatomy of a commercial-grade immune system. Available from http://www.research.ibm.com/antivirus/SciPapers.htm.

Williams, P. D.; Anchor, K. P.; Bebo, J. L.; Gunsch, G. H.; and Lamont, G. D. 2001. CDIS: Towards a computer immune system for detecting network intrustions. In *Proceedings of Recent Advances in Intrusion Detection (RAID 2001)*, volume 2212, 117–133.