An Overview of Biologically Inspired Computing in Information Security

Wim Hordijk
Karunya Institute of Technology and Sciences
Karunya Nagar, Coimbatore – 641 114
wim@santafe.edu

Abstract

Traditional computing methods and systems rely on a central processing unit or central server, and process information mostly serially. They are non-robust and non-adaptive, and have limited scalability. In contrast, biological systems process information in a parallel and distributed way, without a central control. They are highly robust, adaptive, and scalable. This paper gives a brief overview of how ideas from biology have been used to design new computing methods and systems that also have some of the advantageous properties of biological systems. In addition, some examples are given of how these methods can be used in information security applications.

1. Introduction

The power and popularity of current computing systems is largely due to faster and faster CPUs and more and more memory availability at low cost. However, these "traditional" computing methods, architectures, systems, and networks mostly rely on a central processing unit or a central server, they process information serially, and they depend on humans to be programmed and told what to do (and how). This has some serious drawbacks. First, the systems are not very robust. If one part of a system breaks down, the entire system is useless. Second, they are not adaptive. Most computing systems do not learn (or have only limited learning capability), and cannot adjust or adapt to new or unexpected situations without human intervention. Third, there is only limited scalability. The larger the system becomes, or the more nodes are added to the network, the higher the workload of the central processor or server becomes, until it cannot process all instructions or service requests in a reasonable time anymore.

In contrast, most biological systems process information in a parallel and distributed way, without the existence of a central control. They usually consist of a large number of relatively simple individual units, which act in parallel and interact only locally. For example, the brain consists of a large number of simple neurons (more or less equivalent to on-off switches), each of which is connected only to a relatively small portion of all other neurons. Yet an enormous amount of information processing is going on in the brain, where each neuron performs only part of the processing, but they all do so in parallel. In social insect colonies, such as ants and bees, a large number of relatively simple individuals manage to build intricate nests or find the shortest path between the nest and a food source, again in a parallel and distributed way. The human immune systems is another example, where (simple) individual immune cells perform only part of the complete task, but there are many of them working together in parallel.

This parallel and distributed processing method makes these systems highly robust. If some individual units in the system break down, the system as a whole will still function. In fact, it is easy to repair or replace broken units without having to "shut down" the entire system.

Furthermore, these systems are highly scalable. As many individual units can be added as desired, since there are only local interactions involved, and there will be no overload on one particular part of the system. Finally, most systems in nature are adaptive, either through learning (in individual organisms) or through evolution (at the level of populations or species). They can adjust to changing situations or even cope with entirely new situations. So, there are many advantages in biological systems that would be desirable to have in our computing systems.

In this paper, a brief overview is given of how ideas from biology have been used to design new computing methods. This is generally referred to as *biologically inspired computing* [1]. These methods overcome some of the disadvantages of traditional computing, making them more robust, adaptive, and scalable. In particular, three examples are reviewed: (1) genetic algorithms, (2) neural networks, and (3) artificial immune systems. Furthermore, for each of these three methods, some actual applications in the area of information security are also described, in particular in cryptography, biometrics for security, and computer and network security. The biological concepts and ideas underlying the methods described here can be found in any standard textbook on biology, such as [2] and [3].

2. Genetic Algorithms in Cryptography

Genetic algorithms were developed in the 60s and 70s by John Holland and his colleagues and students. They were used both as simple models of evolution and adaptation, and as new computer algorithms to find good solutions to difficult optimization problems. Subsequently, they became very popular as a general optimization tool, and they have been applied successfully to a wide range of problems. This section gives a brief overview of the algorithm (more details can be found in [4], [5], [6], and [7]), and some specific applications of genetic algorithms in cryptography and coding are described.

2.1 Genetic Algorithms

A *genetic algorithm* (GA) is a stochastic search method based on principles from genetics and natural evolution and selection. It is one of a number of computational methods generally referred to as *evolutionary computation* (EC). Instead of trying to directly solve a problem, a solution is *evolved* over time by maintaining a population of (initially random) candidate solutions, creating subsequent generations by recombining different parts of the current best solutions in the population. This way, new candidate solutions are sampled based on the current sample, where the search is guided by a selection process which favors the (currently) best solutions in the population to use for creating new ("offspring") solutions.

Given some optimization problem, first a suitable encoding for candidate solutions needs to be found. Usually, this encoding takes the form of character strings such as bit strings (i.e., strings of 0s and 1s). This is analogous to the biological distinction between the *genotype* (the genetic encoding) and the *phenotype* (the actual shape and appearance) of an organism. For example, in graph problems where some optimal subset or partition of the nodes needs to be found (such as a minimum cover or maximum cut set), a bit string encoding can be used where each bit position corresponds to one particular node in the graph. Construction of the actual candidate solution (phenotype) from a given bit string (genotype) is done as follows. For each bit with value 1, the corresponding node in the graph is included in the candidate subset (or put on one side of the candidate partition), and for each bit with value 0, the corresponding node is not included in the subset (or put on the other side of the partition). This way, the GA can directly search the (much simpler) space of bit strings instead of the space of actual candidate solutions, just as natural evolution happens at the level of genotypes.

Next, a *fitness function* needs to be designed which can be used to evaluate candidate solutions. The main idea is that this function takes as its input an encoded candidate solution (e.g., a bit string), translates this into an actual candidate solution (e.g., a partition of the nodes of a graph), and returns a number according to how good this candidate solution is for the given problem (e.g., the total number of edges between nodes from different sides of the partition for the maximum cut problem). This number, or fitness value, indicates the "goodness" of a candidate solution: higher fitness values mean better solutions. This way, the GA can perform selection based on these fitness values, just as natural selection happens at the level of the phenotypes.

Given a suitable encoding and fitness function (which have to be designed separately for each optimization problem that is considered), the actual algorithm is relatively simple. Assuming a bit string encoding is used, the basic GA works as follows (the selection and crossover & mutation operators are explained below):

1. Initialize the population with N random bit strings ("individuals"), calculate their fitness values, and set *gen*=1.
2. Create a "mating pool" by selecting (with replacement) N individuals from the current population based on fitness.
3. While still individuals in the mating pool, do:
    a. Remove the next pair of individuals ("parents") from the mating pool.
    b. With probability p_c perform crossover between the parents to create two "children".
    c. With probability p_m perform mutation on the children.
    d. Place the children in the new population.
4. Replace the previous population with the new population, calculate the fitness of all individuals, and set *gen*=*gen*+1.
5. If *gen* < M go to step 2, otherwise stop.

There are various ways in which the selection operator can be implemented, but the main idea is that individuals with higher fitness values, compared to the rest of the population, have a higher chance of being selected than individuals with lower fitness values (i.e., fitness proportionate selection). In other words, the mating pool will (on average) contain multiple copies of the best individuals in the current population and no (or just a few) copies of the worst individuals.

The crossover operator literally chops up the genotypes of the parent individuals and recombines them to create offspring genotypes. The most basic method is one-point crossover, in which a random crossover point is first chosen (somewhere between the first and last bit), and the first part of the first parent is recombined with the second part of the second parent to create the first child (and vice versa for the second child). Usually crossover is done with a certain probability p_c (often set in the range [0.6;0.95]) for each pair of parents. If crossover is not performed, the children will be identical to their parents. Finally, with a usually very low probability p_m, mutation is performed, where a bit is flipped at random. Examples of (one-point) crossover and mutation are shown below. In the crossover example, the crossover point is (randomly) chosen between the 3rd and 4th bit. In the mutation example, the 0 at the 9th position is mutated into a 1 (shown in bold).

```
        crossover                       mutation

0000000000  →  0001111111       0101010101  →  010101011
1111111111     1110000000
```

Finally, the creation of new generations of candidate solutions by selection and crossover & mutation is repeated for a set number `M` of generations. Other stopping criteria are possible, of course, such as reaching a certain level of fitness or a maximum amount of computing time. In short, the main idea of the algorithm is to evolve better and better solutions by repeatedly selecting the best candidate solutions from the current population and recombining parts of their genotypes to create subsequent generations of candidate solutions.

2.2 Applications of Genetic Algorithms in Cryptography and Coding

A concise overview of the state of the art and of still open problems in using evolutionary computation techniques (such as genetic algorithms) in cryptography is given in [8]. In cryptography, it is important to know how difficult it is to "break" an encryption method. Obviously, methods that are very difficult to break are preferred over methods that are more easily broken. *Cryptanalysis* is all about analyzing (or "attacking") encryption methods to find out how easy or difficult they are to break. Genetic algorithms have been used successfully in this area, for example in attacking substitution ciphers [9], [10] and transposition ciphers [11]. Although this does not directly give rise to better ciphers, it does show where their weaknesses are, which in turn can help in improving them. Furthermore, in [12] a genetic algorithm was used successfully to find Boolean functions with good cryptographic properties, thus showing how these techniques can also be used directly for constructing encryption methods.

An important method that is often used in cryptography is that of generating pseudo random numbers. Here, the objective is to generate a stream of numbers (by some deterministic method) that is "as random as possible", and which has a high period (i.e., it will not repeat itself quickly). An interesting approach, using an evolutionary technique similar to GAs, was introduced in [13], where cellular automata (simple parallel and distributed computing devices) were evolved to produce pseudo random numbers with a high degree of randomness.

As a final example, consider coding techniques for data transmission. Next to providing data security through encryption, it is also important that information loss is minimized during transmission of encrypted messages. Genetic algorithms have been used successfully to optimize so-called "turbo codes" [14]. In this case, the GA was able to find a slightly better code than what was available at the time.

These applications are just a selection of the many possibilities of applying genetic algorithms and other evolutionary computation techniques in the area of information security. Next, an overview of neural networks, another biologically inspired computing method, is presented.

3. Neural Networks in Biometrics for Security

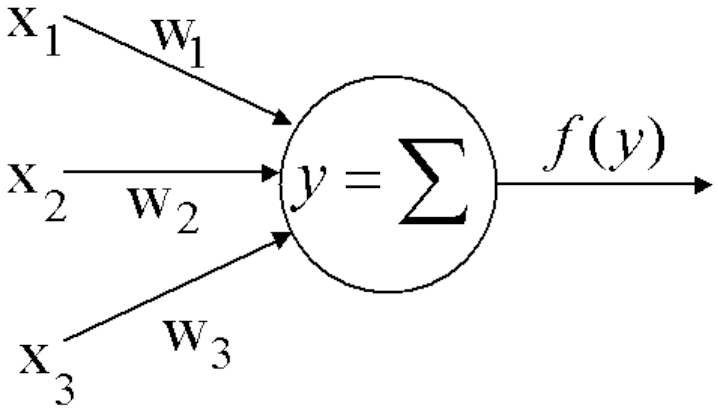
The research on neural networks was pioneered by McCulloch and Pitts in 1943 [15]. They presented a logical (mathematical) model of a simple neuron, and showed that a suitably constructed network of such "artificial neurons" can, in principle, compute any computable function. Thus, a neural network is equivalent (in terms of computational power) to a universal Turing machine, but with a very different architecture. In this section, first the concept of neural networks is briefly reviewed. A good introduction to computing with neural networks is provided in [16], and more detailed information can be found in any standard textbook on neural networks, such as [17] and [18]. Next, an example of an application of neural networks in biometrics for security is described.

3.1 Neural Networks

A *neural network* (NN) is a *parallel distributed processing* (PDP) architecture that is modeled after the working of the brain. It can perform computations, in particular classification of inputs, and provides an example of an alternative model of computation compared to the serially and centrally based computations of standard computing systems.

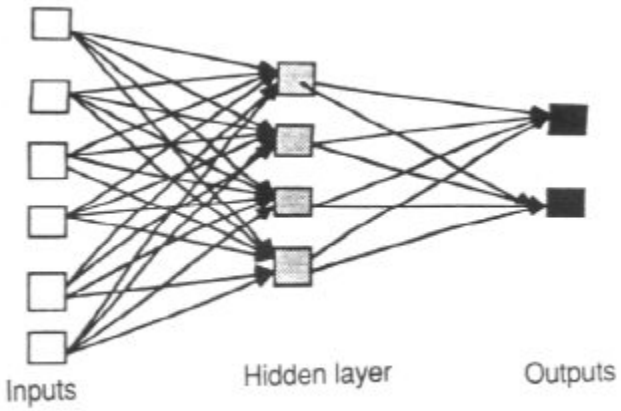
Our brains consist of many (about 10 billion) simple cells called *neurons*. Each neuron consists of a cell body, an *axon* (an elongated "transmission line" through which chemical signals can travel), and many *dendrites* (a tree-like structure of many branching "tentacles"), which end in *synapses* which form connections with the axons of other neurons. Simply put, each neuron receives inputs (the presence or absence of signals) from other neurons through the synaptic connections, which travel down the dendrites to the cell body. Here, the inputs are "added up", and if a certain threshold is reached the neuron sends out a signal itself through its axon, which is then forming an input to yet other neurons which are connected to its axon. However, not all synaptic connections are equal. Some are stronger than others, and so some inputs have a higher "weight" than others. Learning is achieved by adjusting the strengths (weights) of existing synaptic connections, or by creating new or deleting old connections.

A simplified model of a real neuron is illustrated in the figure below. A neuron receives inputs (x_i) from other neurons, which are weighted (w_i) and then added (y). The output of a neuron is a function $f(y)$ of this weighted sum of inputs, and can in turn form the input to other neurons. In the simplest case, each input can be either 0 (absence of signal) or 1 (presence of signal), and the output function is a step function such that the output is 0 if the weighted sum of inputs is below a certain threshold value, and 1 if it is above the threshold value. In more realistic cases, the inputs and outputs are real valued numbers within some range, and the output function is for example a sigmoidal shape.



Any number of such neurons can be connected to each other to form an artificial neural network. A standard network architecture that is often used is a *feed forward network*. In such a NN, there is one layer of input neurons, one or more layers of "hidden" neurons, and one layer of output neurons, as illustrated in the figure on the next page. The neurons in the input layer are initialized with some input pattern, and the outputs from this layer go "forward" and serve as inputs to the first hidden layer. These neurons then produce their outputs which serve as inputs to the next hidden layer (if present), until the final, or output, layer is reached. The state of the neurons in the output layer can then be interpreted as the "answer". For example, in classification problems, if the state of the first output neuron is 1 and that of the second one is 0, the input belongs to one

class. If their final states are reversed (i.e., 0 and 1, respectively), then the input belongs to the other class (assuming there are two classes into which to partition the inputs). Other network architectures are of course also possible, such as recurrent networks, where connections can feed back to previous layers as well, or grid networks, where the neurons are arranged in a grid with connections between neighboring neurons.



Given some network architecture, it is not directly obvious how to set the weights on the connections to get a certain network behavior. However, several *training algorithms* have been designed to optimize these weights. The main idea of these algorithms is to repeatedly present the network with example inputs for which the correct answer is known. The weights in the network are then updated depending on the amount of error between the correct answer and that given by the network. This is repeated until no more errors are made, or the amount of error falls below a certain threshold. The network can now be said to have *learned* the given task. At the next stage, the network can be used to perform the task on new inputs which it might not have seen before.

3.2 Applications of Neural Networks in Fingerprint Recognition

One area where neural networks have become very popular is image processing, such as pattern recognition and classification, noise filtering, edge detection, etc. As an application in biometrics for security, they can be used successfully for fingerprint recognition. Fingerprint recognition is often split up in two stages: (1) feature extraction, and (2) classification. In the first stage, certain features from a fingerprint image are extracted, such as ridge directions, arches and whorls, delta points, etc. (for a more detailed overview, see for example [19]). In the second stage, these features are used to recognize (or classify) the given fingerprint image.

Neural networks have been applied successfully in both of these stages, often giving rise to high correct classification rates and low false rejection rates, and frequently outperforming more traditional methods (see for example [20], [21], [22], [23], and [24]). Furthermore, neural networks can be used similarly for other image recognition tasks in biometrics security, such as retina or iris scan classifications, or for voice recognition.

Finally, as a last example of biologically inspired computing in the area of information security, a brief overview of artificial immune systems for computer security is presented in the next section.

4. Artificial Immune Systems for Computer Security

A very recent idea that is still being developed is that of building a computer immune system. The task of such a system is to provide computer and network security based on the workings of the human immune system. This section first presents a high-level and somewhat simplified overview of the human immune system. A good introduction to this topic can be found in [25]. Next, an example of an implementation of a simple computer immune system is given to illustrate the applicability of the idea.

4.1 The human Immune System

The human immune system is a complex and multi-layered system. The part that is of most interest here is the *adaptive immune response*. A brief overview of this is given below, with many details left out. However, the general properties of this part of the immune system serve as a starting point for the design of an artificial immune systems for computer and network security.

The human body consists of many different types of molecules (mostly proteins), which are referred to as "self". Everything else, including things that make us ill, is referred to as "non-self". So, the main task of the immune system is to distinguish "non-self" from "self", and trigger a response whenever "non-self" proteins are detected. However, this is not an easy task as there are an estimated 10^{16} "non-self" proteins that the immune systems needs to recognize, compared to about 10^{5} "self" proteins. The way the immune system solves this problem is by using a dynamic and distributed system.

At any time, many "detector" cells, including so-called T-cells, circulate through our bodies. These cells mature in an organ called the thymus, where they are exposed to most of the "self" proteins that make up our bodies. If any of the maturing T-cells binds to any of these "self" proteins, that T-cell is eliminated. So, the only T-cells that leave the thymus are those that do not bind to "self" proteins. Consequently, if a matured T-cell does bind to a protein, it means this must be a "non-self" protein, and an appropriate immune response will be triggered. However, not all T-cells are able to bind to (or "recognize") all possible "non-self" proteins, but some T-cells bind to some "non-self" proteins, other T-cells to others, etc. In this way, the immune system is a distributed system.

It is also dynamic, as T-cells are continuously replaced through a genetic process including variation (or random "mutations"). This way, the set of "non-self" proteins that the immune system is able to recognize, changes over time. Since it is impossible to recognize all possible "non-self" proteins at any one time, this dynamic system is the next best solution. Furthermore, because of this, no two individuals will have exactly the same set of T-cells at any given time, so what might make me sick, my neighbor might be immune to, and vice versa.

Finally, the immune system also has a "memory". It is capable of remembering illness-causing "non-self" proteins (antigens), so that the next time an individual gets infected with the same antigen, it is recognized immediately and an appropriate immune response can be triggered, preventing the actual illness from occurring again.

4.2 Computer Immunology

Forrest and students were some of the originators of using principles from the human immune systems to design an intrusion detection system for computers and networks [26], [27]. In particular, in [28] they show the results of a basic implementation based on scanning short sequences of system calls. Briefly, the idea is as follows. In the first stage, a database of system call sequences during "normal" behavior is built. This database thus contains the sequences that constitute "self". In the next stage, system call sequences are scanned during system operation that might contain intrusion attempts. These sequences are then compared to the available database, and any sequence that is not present in the database ("non-self") triggers an "alarm". This way, abnormal behavior can be easily detected, and appropriate actions can be performed if necessary.

Obviously, the databases containing normal behavior have to be updated frequently. For example, adding new users or software and hardware to the system will change the normal behavior, or a user's behavior might change over time (different tasks, different priorities, etc.). However, with this design, the intrusion detection system becomes more adaptive, as it is capable of recognizing abnormal behavior that has not been observed before. In other words, the system can identify, for example, new viruses or new attacking mechanisms, without the need for downloading new virus "signatures" from some central server first. Furthermore, different computers will have different databases of "self" behavior, so a virus that infects one computer, might not be able to infect every other computer. This way, the network as a whole also has a better (distributed) protection.

The (small-scale) examples and simulation that have been implemented so far indicate the viability of these ideas, and show a promising future. Currently, the ideas and designs are still being developed further, and are also being picked up by others [29], [30]. Computer immunology and artificial immune systems are now an active area of research.

5. Summary

Traditional computing methods have several disadvantages, such as a lack of robustness and adaptability, and limited scalability. In contrast, biological systems, being mostly parallel distributed processing systems, are highly robust, adaptable, and scalable. Biologically inspired computing involves the design, implementation, and application of new computer methods and systems that incorporate these advantageous properties of biological systems. In this paper, a brief overview of biologically inspired computing has been presented, with some specific examples of how these methods can be used in information security in particular. Many of these methods have already been applied successfully, such as genetic algorithms and neural networks, and some are still being further developed, such as computer immunology. It is clear that the area of information security can benefit greatly from these new and exciting computing methods.

Acknowledgements

The author would like to thank Dr. K. Anbumani for the invitation to contribute a paper to this conference, and the Karunya Institute of Technology and Sciences for providing an enjoyable working environment.

References

[1] L. N. de Castro and F. J. Von Zuben, *Recent Developments in Biologically Inspired Computing*. Idea Group Publishing, 2005.

[2] S. Alters, *Biology: Understanding Life*. Mosby, 1996.

[3] N. A. Campbell, L. G. Mitchell, and J. B. Reece, *Biology: Concepts & Connections,* 2nd edition. Benjamin Cummings, 1997.

[4] J. H. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

[5] J. H. Holland, "Genetic Algorithms," *Scientific American*, vol. 267 (1), pp. 66-72, 1992.

[6] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[7] M. Mitchell, *An Introduction to Genetic Algorithms*. MIT Press, 1996.

[8] P. Isasi and J. C. Hernández, "Introduction to the applications of evolutionary computation in computer security and cryptography," *Computational Intelligence*, vol. 20 (3), pp. 445-449, 2004.

[9] R. Spillman, M. Janssen, B. Nelson, and M. Kepner, "Use of a genetic algorithm in the cryptanalysis of simple substitution ciphers," *Cryptologica*, vol. 17 (1), pp. 31-44, 1993.

[10] A. Clark and E. Dawson, "Optimisation heuristic for the automated cryptanalysis of classical ciphers," *Journal of Combinatorial Mathematics and Combinatorial Computing*, vol. 28, pp. 63-86, 1998.

[11] R. A. J. Matthews, "The use of genetic algorithms in cryptanalysis," *Cryptologica*, vol. 17 (2), pp. 187-201, 1993.

[12] W. Millan, A. Clark, and E. Dawson, "An effective genetic algorithm for finding Boolean functions," in *Proceedings of the International Conference on Information and Communications Security*, 1997.

[13] M. Sipper and M. Tomassini, "Co-evolving parallel random number generators," in *Proceedings of the Parallel Problem Solving from Nature Conference*, 1996, pp. 950-959.

[14] N. Durand, J.-M. Alliot, and B. Bartolomé, "Turbo codes optimization using genetic algorithms," in *Proceedings of the Congress on Evolutionary Computation*, 1999.

[15] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115-133, 1943.

[16] R. P. Lippmann, "An introduction to computing with neural nets," *IEEE ASSP Magazine*, vol. 4, pp. 4-22, 1987.

[17] J. A. Anderson, *Introduction to Neural Networks*. MIT Press, 1995.

[18] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Prentice-Hall, 1999.

[19] N. Ratha and R. Bolle, Eds., *Automatic Fingerprint Recognition Systems*. Springer-Verlag, 2004.

[20] P. A. Hughes and A. D. P. Green, "The use of neural network for fingerprint classification," in *Proceedings of the 2nd International Conference on Neural Networks*, 1991, pp. 79-81.

[21] M. Kamijo, "Classifying fingerprint images using neural networks: Deriving the classification state," in *Proceedings of the 3rd International Conference on Neural Networks*, 1993, pp. 1932-1937.

[22] C. L. Wilson, G. T. Candela, and C. I. Watson, "Neural network fingerprint classification," *Journal of Artificial Neural Networks*, vol. 1 (2), pp. 203-228, 1994.

[23] H. V. Neto and D. L. Borges, "Fingerprint classification with neural networks," in *Proceedings of the 4th Brazilian Symposium on Neural Networks*, 1997, pp. 66-72.

[24] A Ceguerra and I. Koprinska, "Automatic fingerprint verification using neural networks," in *Proceedings of the International Conference on Artificial Neural Networks*, 2002, pp. 1281-1286.

[25] C. A. Janeway and P. Travers, *Immunobiology: The Immune System in Health and Disease*. Current Biology Ltd., 2nd edition, 1996.

[26] S. Forrest, S. A. Hofmeyr, and A. Somayaji, "Computer Immunology," *Communications of the ACM*, vol. 40 (10), pp. 88-96, 1997.

[27] A. Somayaji, S. A. Hofmeyr, and S. Forrest, "Principles of a computer immune system," in *New Security Paradigms Workshop*, 1998, pp. 75-82.

[28] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for Unix processes," in *Proceedings of the IEEE Symposium on Computer Security and Privacy*, 1996.

[29] J. Kim and P. Bentley, "The human immune system and network intrusion detection," in *Proceedings of the 7th European Conference on Intelligent Techniques and Soft Computing*, 1999.

[30] J. Kim and P. Bentley, "Towards an artificial immune system for network intrusion detection: An investigation of dynamic clonal selection," in *Proceedings of the Congress on Evolutionary Computation*, 2002, pp. 1015-1020.