Coverage and Generalization in an Artificial Immune System*

Justin Balthrop
judd@cs.unm.edu
Computer Science Dept.
University of New Mexico
Albuquerque, NM 87131

Fernando Esponda
fesponda@cs.unm.edu
Computer Science Dept.
University of New Mexico
Albuquerque, NM 87131

Stephanie Forrest
forrest@cs.unm.edu
Computer Science Dept.
University of New Mexico
Albuquerque, NM 87131

Matthew Glickman
glickman@cs.unm.edu
Computer Science Dept.
University of New Mexico
Albuquerque, NM 87131

Abstract

LISYS is an artificial immune system framework which is specialized for the problem of network intrusion detection. LISYS learns to detect abnormal packets by observing normal network traffic. Because LISYS sees only a partial sample of normal traffic, it must generalize from its observations in order to characterize normal behavior correctly. A variation of the r-contiguous bits matching rule is introduced, and its effect on coverage and generalization is studied. The effect of representation diversity on coverage and generalization is also explored by studying permutations in the order of bits in the representation.

1 Introduction

The natural immune system uses a variety of evolutionary and adaptive mechanisms to protect organisms from foreign pathogens and misbehaving cells in the body. Artificial immune systems (AISs) seek to capture some aspects of the natural immune system in a computational framework, either for the purpose of modeling the natural immune system or for solving engineering problems. In either form, the fundamental problem solved by most AISs can be thought of as learning to discriminate between "self" (the normally occurring patterns in the system being protected, e.g., the body) and "non-self" (foreign pathogens, such as bacteria or viruses, or components of self that are no longer functioning normally). Almost any set of patterns that can be expressed as strings of symbols can be placed into this framework, for example, the set of normally occurring TCP connections in a local area network (LAN) and the set of TCP connections observed during a network attack [Hofmeyr, 1999, Kim and Bentley, 2001]. This is the example on which we will focus in this paper.

We are interested in the question of representation—how well a set of AIS detectors covers the set of normally occurring patterns (or conversely, how well it can detect the set of abnormal patterns). Because AIS detectors are typically generated on-line in a fluctuating environment, they are highly unlikely to be exposed to every possible normal pattern during training. Consequently, it is important for detectors to generalize from the set of observed normal patterns to the set of expected normal patterns. The generalization properties of the AIS affect both false positives (mistakenly identifying normal patterns as abnormal) and false negatives (mistakenly identifying abnormal patterns as legitimate). These are known as Type I and Type II errors respectively in the statistical decision theory literature.

There are several components of the AIS that affect how well it represents its environment and how well it generalizes. The first of these is the mapping from the domain to detectors, or what information is presented to the AIS. Here we will use the 49-bit compressed representation of TCP SYN packets, introduced by Hofmeyr [Hofmeyr, 1999, Hofmeyr and Forrest, 1999, Hofmeyr and Forrest, 2000]. In this representation each detector is a 49-bit string. Detectors are matched against the compressed 49-bit SYN packets (see Figure 1) using a partial matching rule which scores how closely they match. Choosing an appropriate mapping for a given problem in the AIS context has all the same complications as choosing a representation for a genetic algorithms problem. Some representations are clearly better than others, but it is difficult to formalize criteria by which one can choose a good

*This paper appears in the proceedings of the 2002 Genetic and Evolutionary Computation Conference (GECCO 2002).

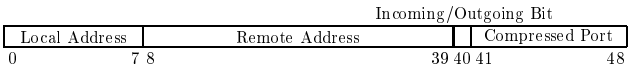


Figure 1: The 49-bit compression scheme used by LISYS to represent TCP SYN packets. Strings are compressed in two ways. First, it is assumed that one of the IP addresses is always internal, so only the final byte of this address needs to be stored. The port number is also compressed from 16 bits to 8 bits by re-mapping the ports into several different classes.

one in a particular instance. The 49-bit representation chosen by Hofmeyr works surprisingly well, although it contains a minimal amount of information and the information is arranged in an arbitrary ordering.

The second component is the match rule that is used to assess how well an AIS detector matches a particular pattern. A perfect *match* between a detector and a compressed SYN packet means that at each location in the 49-bit string, the symbols are identical. However, perfect matching (binding) is rare in the immune system and improbable between strings of any significant length. We use a matching rule known as r-contiguous bits [Percus *et al.*, 1993]. This 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 (match) in at least r contiguous locations. We also introduce a variant of this rule which we refer to as *r-contiguous templates*, or more simply, *r-chunks*. Both r-contiguous bits and r-chunks are related to genetic algorithms and classifier systems in interesting ways.

A third component is the permutation mask, also introduced by Hofmeyr. Permutation masks are a mechanism for introducing diversity of representation, crudely analogous to MHC diversity in the natural immune system. The idea behind this form of diversity is that different representations will match different patterns, and that the union of a set of different representations will have greater detection ability than any single representation. This insight is complicated by the form of our problem, in which detecting more patterns is not always better (because patterns detected in error lead to false positives). Permutation masks simply store a different permutation of the 49-bit mapping, one permutation for each detector set[1]. This, combined with r-contiguous bits matching, causes different permutations to discover different correlations among bits in the representation.

[1]Permutation masks are one possible means of generating *secondary representations*. A variety of alternative schemes are explored in [Hofmeyr, 1999].

2 LISYS

The following summary of LISYS is largely drawn from [Balthrop *et al.*, 2002]. LISYS is situated in a local-area broadcast network and used to protect the LAN from network-based attacks. In contrast with switched networks, broadcast LANs have the convenient property that every location (computer) sees every packet passing through the LAN. In this domain, *self* is defined to be the set of normal pairwise connections (at the TCP/IP level) between computers, and *non-self* is the set of connections, which are not normally observed on the LAN and are likely to be correlated with network intrusions. A connection is defined in terms of its "data-path triple"—the source IP address, the destination IP address, and the service (or port) by which the computers communicate [Mukherjee *et al.*, 1994, Heberlein *et al.*, 1990].

LISYS consists of sets of *detectors*, where each detector is a 49-bit string and a small amount of local state. The detectors can be distributed across multiple hosts, and they can perform their function with virtually no communication. The detectors assigned to a particular host are referred to as a *detector set*.

LISYS uses *negative detection* in the sense that valid detectors are those that fail to match the normally occurring behavior patterns in the network. LISYS generates random detectors, censors them against self, and eliminates those that match self (negative selection). The censoring process, known as the *tolerization period*, lasts for a few days during which time the detector is matched against every SYN packet occurring in the network. More efficient detector generation algorithms are described in [D'haeseleer *et al.*, 1996, Wierzchon, 2000, Wierzchon, 2001]. However, when generating detectors asynchronously for a dynamic self set, such as the network setting, these methods are not directly applicable and random generation seems to work well.

Detectors in LISYS have a finite lifetime. The expected lifetime of a mature detector is a parameter of the system. Detectors can die in several ways, through negative selection, old age, or lack of co-stimulation (see [Hofmeyr, 1999]). The finite lifetime of detectors, when combined with detector re-generation and tolerization, results in *rolling coverage* of the self set.

Each independent detector set has its own *permutation mask*, as described above. A permutation mask defines a permutation of the bits in the string representation of the network packets. Each detector set (network host) has a different, randomly-generated permutation mask. One feature of the negative-selection algorithm

as originally implemented is that it can result in undetectable patterns called *holes* [D'haeseleer *et al.*, 1996, D'haeseleer, 1996], or put more positively *generalizations* [Esponda and Forrest, 2002]. Holes can exist for any symmetric, fixed-probability matching rule, but permutation masks effectively change the match rule and thus the distribution of holes. Using a different permutation on each host allows us to control how much the system generalizes in the vicinity of self, and thus gives us more control over the undetectable holes [Esponda and Forrest, 2002].

The original LISYS system uses several other mechanisms, such as *activation thresholds*, *sensitivity levels*, and *co-stimulation* to reduce false positives, and *memory detectors* to increase true positives. For details on the full system, the reader is referred to [Hofmeyr, 1999, Hofmeyr and Forrest, 2000].[2]

3 Data Set

The experiments reported in this paper use the data set described in [Balthrop *et al.*, 2002]. Our data collection strategy was to control the data set as much as possible while still collecting data in a realistic context. The data set was collected from an internal restricted network of computers in a small university research group. The six internal computers in this network connected to the Internet through a single Linux machine that acted as a firewall, router and masquerading server for the internal machines. The internal network was set up as a broadcast network, so we were able to monitor the traffic of all the computers easily.

This scenario provided a data set that satisfied both objectives. The internal restricted network was much more controlled than the external university or departmental networks. In this environment, we can understand all of the connections that occur, and we can be relatively certain that there were no attacks during the normal training periods. Moreover, this environment is realistic. Many corporations have intranets in which activity is somewhat restricted and external connections must pass through a firewall. This environment could also model the increasingly common home network that connects to the Internet through a cable or DSL modem and has a single external IP address. Attacks are a reality in environments such as these, and the attack scenarios corresponded to plausible occurrences in this class of environment.

[2]The programs used to generate the results in this paper are available from http://www.cs.unm.edu/~immsec. The programs are part of LISYS and are found in the LisysSim directory of that package.

The normal network data in our data set consist of two weeks of data collected in November, 2001. In these data, there are a total of 22,329 TCP SYN packets, and roughly 55% of this is web traffic. Thus, there was an average of approximately 1600 packets per day during the normal period. Because the network data being produced is dependent on a small number of users, two weeks seemed to be the shortest period of time that could possibly give a reasonable characterization of self. Attack data were generated over the course of two days near the end of the collection period. The attacks took place about one week after the normal period ended, and consisted of 76,179 TCP SYN packets.

In [Hofmeyr, 1999], network connections to web servers are removed from the data by filtering out all connections to port 80. Instead of completely removing web connections, the data set simulates the behavior of a proxy server. All outgoing connections to port 80 (http) or port 443 (https) are re-mapped to port 3128 on the proxy machine. This is very close to what the traffic would have been like if we were using the web proxy cache SQUID.

All of the attacks, with the exception of the denial-of-service attack, were performed using a laptop connected to the internal network. The firewall machine was configured as a DHCP server, so the laptop was able to acquire a dynamic IP address because it had a physical connection to the internal network. We used the free security scanner Nessus to perform the attacks. A total of eight attacks were run, including denial of service (from an internal computer to an external computer), a firewall attack against the firewall/gateway machine, an ftp attack against an internal machine, an ssh probe against several internal machines, an attack probing for certain services, a TCP SYN scan, an nmap tcp connect() scan against several internal computers, and a full nmap port scan.

4 r-Chunks Matching

In this section we introduce a variant of the r-contiguous bits matching rule, which we refer to as "r-chunks." We will show in section 7 that r-chunks matching performs better than *full-length* r-contiguous bits matching for our data set. However, r-chunks matching also has the virtue of being more amenable to mathematical analysis than full-length matching [Esponda and Forrest, 2002]. r-Chunks matching is reminiscent of the $\{1, 0, \#\}$ matching rule for classifier systems [Holland *et al.*, 1986], with the additional restrictions that all detectors have a constant number of defined bits (the r parameter) and that all the de-

fined bits are located in contiguous positions. Matching with both r-chunks and full-length detectors is related to the crossover operator in genetic algorithms [Holland, 1975].

In r-chunks detectors, only r contiguous positions of the detector are specified (known as the *window* of the detector); the remaining bit positions can be thought of as "don't cares." Alternatively, an r-chunks detector can be thought of as a string of r bits together with a specification of the window to which it refers. An r-chunks detector d is said to match a string x if all the bits of d are equal to the r bits of x in the window specified by d.

The relation between full-length detectors and r-chunks is shown in the following figure for $l = 4$ and $r = 2$. A single full-length detector can be decomposed into $l - r + 1$ (the number of windows) r-chunks detectors. Let d_{fl} be the full-length detector and d_{c1}, d_{c2}, d_{c3} the r-chunks detectors into which it can be decomposed:

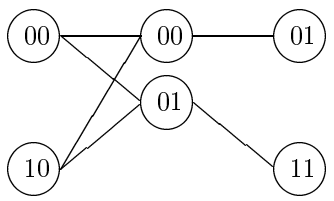
$$\begin{array}{ll} d_{fl}: & \boxed{1}\boxed{0}\boxed{1}\boxed{1} \\ d_{c1}: & \boxed{1}\boxed{0} \\ d_{c2}: & \quad\boxed{0}\boxed{1} \\ d_{c3}: & \qquad\boxed{1}\boxed{1} \end{array}$$

An important difference between the two match rules is in the number of undetectable strings they induce. We refer to these strings as "holes" and the set of holes for a given self set S as H. For full-length matching there are two sources of holes: crossover holes and length-limited holes.

Hence, a crossover hole is a string h not in S, for which all windows in h are crossovers of adjacent windows in S, according to the restricted crossover operation defined below. A crossover occurs in this context between two adjacent windows $W_i = v_i..v_{i+r-1}$ and $W_{i+1} = u_{i+1}..u_{i+r}$ whenever bits $v_j = u_j \ \forall_j : i+1 \leq j \leq i+r-1$. There is an example of this type of hole at the end of this section.

The second source of holes arises because in full-length detectors, all the bit positions are specified. This can induce holes h which are strings that have at least one window of r bits not present in S, but for which a detector still cannot be generated. For instance, let $S = \{110, 010\}, l = 3, r = 2$ and let $h = 011$ be a string that has r contiguous bits not exhibited in any string in S. A full-length detector for h must either start with the pattern 01 and/or end with the pattern 11 but any detector starting with 01 will match self and hence can not be generated. Similarly, if a potential detector for h ends with pattern 11 the two possible strings 011 and 111 match a string in S as well, therefore a detector for h cannot be generated.

r-Chunks detection does not induce length-limited holes, because a detector can always be generated for a pattern of length r which is not present in S. Thus, the only holes induced by r-chunks matching are crossover holes. This greatly simplifies the task of characterizing and managing holes. For example, the generalization of a set S, for r-chunks, can be depicted as a directed acyclic graph (DAG) with as many nodes as there are distinct bit patterns for each window (each node labelled as the bit pattern it represents) where two nodes are connected together if the windows they refer to crossover. Consider, for example, a self set S comprised of the following two strings $S = \{0001, 1011\}$ with $l = 4$, $r = 2$:



Following all the paths, starting from the leftmost nodes, yields the strings $\{0001, 0011, 1001, 1011\}$ which constitute the generalization of the r-chunks matching rule, out of which $\{0011, 1001\}$ are crossover holes. We refer to the holes plus the self strings that induced them as the *crossover-closure* [Helman, 2002].

5 The Experimental Setup

In LISYS, new detectors are generated when the system is initialized. Thereafter, new detectors are generated whenever another detector dies, usually through negative selection or old-age. Detectors are generated, trained, tested, and killed asynchronously throughout a LISYS run. Consequently, different detectors are tolerized at different times and are thus exposed to different samples of self.

Although this rolling coverage is desirable for dynamically changing self sets and to make evasion by an adversary more difficult, it also complicates analysis. Accordingly, for the experiments reported in this paper, we trained all detectors on the identical set of self strings (training set), and tested them subsequently against the identical set of test strings. We did not kill off detectors due to old age. In all of the experiments

the initial tolerization period was set to 15,000 packets, corresponding to approximately 8 days. Among these 15,000 initial packets, there were 131 unique strings to which the immature detectors were exposed.

6 The Effect of Permutation Masks

The goal of the first experiment was to assess how different permutations affect the performance of the system. Because performance is measured in terms of true and false positives, this experiment also tests the effect of permutations on the system's ability to generalize (because low false positive rates correspond to good generalization).

100 sets of detectors were tolerized using the 131 unique strings derived from the first 15,000 packets in the data-set (the *training set*), and each detector set was assigned a random permutation mask. Each detector set had exactly 5,000 mature detectors at the end of the tolerization period and an r-value of 10. These numbers were chosen on the basis of previous experiments [Balthrop *et al.*, 2002] which showed that 5,000 detectors provide maximal coverage (i.e. adding more detectors does not improve subsequent matching) for this data set and r threshold.[3] Each set of detectors was then run against the remaining 7,329 normal packets, as well as against the simulated attack data. In these data (the *test sets*), there are a total of 476 unique 49-bit strings. Of these 476, 50 also occur in the training set and are thus undetectable (because any detectors which would match them are eliminated during negative selection). This leaves 426 potentially detectable strings, of which 26 come from the normal test set and 400 are from the attack test set. The maximal possible coverage by a detector set is thus 426 unique matches.

An ideal detector set would achieve zero false positives on the normal test data and a high number of true positives on the attack data. Thus, a perfect detector set would match the 400 unique attack strings, and fail to match the 26 unique normal strings in the test set, thus generalizing from the self observed during training. Note that because network attacks rarely, if ever, produce only a single anomalous packet, we don't need to achieve perfect true-positive rates at the packet level in order to detect all attacks against the system.

[3] The use of 5,000 detectors to protect 131 unique strings is clearly a somewhat artificial situation. This arises from the small size of our data set and the decision to provide maximal coverage of non-self. In general, once the number of self strings increases above a certain threshold, the number of detectors needed to cover non-self through negative detection becomes less than that required for positive detection (see [Esponda and Forrest, 2002] for the exact tradeoff). And, for most applications, complete coverage of non-self is an overly strict requirement.

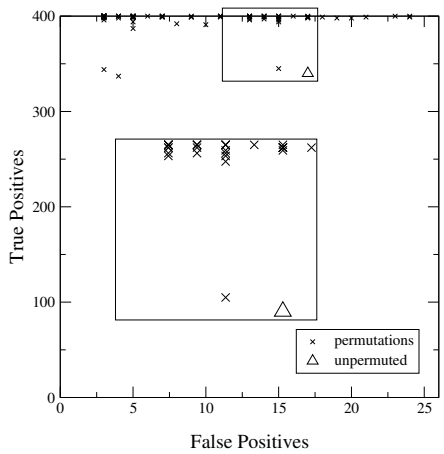


Figure 2: LISYS performance under different permutations. Each plotted point corresponds to a different permutation, showing false positives (x-axis) and true positives (y-axis). The inset shows a zoomed view of the same data.

Figure 2 shows the results of this experiment. The performance of each detector set is shown as a separate point on the graph. Each detector set has its own randomly generated permutation of the 49 bits, so each point shows the performance of a different permutation. The numbers on the x-axis correspond to the number of unique self-strings in the test set which are matched by the detector set, i.e. the number of false positives (up to a maximum of 26). The y-axis plots the corresponding value with respect to the attack data, i.e. the number of unique true positive matches (up to a maximum of 400). The graph shows that there is a large difference in the discrimination ability of different permutations. Points in the upper left of the graph are the most desirable, i.e. they correspond to permutations which minimize the number of false positives and maximize the number of true positives; points toward the lower right corner of the graph indicate higher false positives and/or lower true positives.

Surprisingly, the performance of the original (unpermuted) mapping is among the worst we found, suggesting that the results reported in [Balthrop *et al.*, 2002] are a worst case in terms of true vs. false positives. Almost any other random permutation we tried outperforms the original mapping. Although we don't yet have definitive proof, we believe this behavior arises in the following way.

The LISYS design assumes that there are certain predictive bit-patterns that exhibit regularity in self, and that these can be the basis of distinguishing self from non-self. As it turns out, there are also deceptive bit-patterns which exhibit regularity in the training set (observed self), but the regularity does not generalize to the rest of self (the normal part of the test set). These patterns tend to cause false positives when self strings that do not fit the predicted regularity occur.

We believe that the identity permutation is bad because the predictive bits are at the ends of the string, while the deceptive region is in the middle. Under such an arrangement, it is difficult to find a window that covers many predictive bit positions without also including deceptive ones. It is highly likely that a random permutation will break up the deceptive region, and bring the predictive bits closer to the middle, where they will appear in more windows.

7 r-Chunks vs. Full-Length Detectors

In this section we compare the performance of r-chunks matching to that of r-contiguous bits matching with full-length detectors on our data set. The essential difference between full-length detectors and r-chunks lies in the holes which they induce, as discussed earlier. Holes are desirable to the extent that they prevent false positives (strings which are close to self and represent legitimate but novel behavior of the network)[4]; holes are undesirable to the extent which they lead to false negatives (a failure to match strings which correspond to attempted intrusions). Although both representations are subject to crossover holes, full-length detectors are additionally subject to length-limited holes. Therefore, we are interested in knowing if in practice length-limited holes generalize over true positives or false positives.

For this experiment, we generated one set of r-chunks detectors for each value of r, ranging from 1 to 12. Because there are only $2^r \times (l - r + 1)$ possible r-chunks detectors, we generated all of them, and then eliminated through negative selection any detector that matched a string in the training set. Full-length detectors were generated according the the procedure described in Section 5.

The results of this experiment are shown in Figure 3. As in the initial permutation-mask experiment, the number of false positives is plotted on the x-axis and the number of true positives on the y-axis. There are two sets of points, each connected by lines. One set indicates the results obtained with r-chunks for values of r ranging from 1 to 12. The second set, shows the results of using full-length detectors for $r = 8, 10,$ and 12.

[4]This is the sense in which holes can be thought of as generalizations.

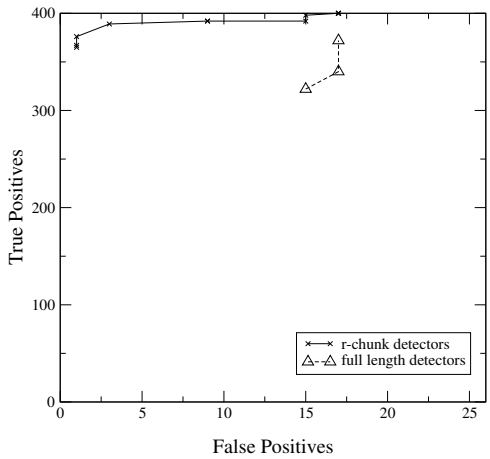


Figure 3: LISYS performance under different r-values. For r-chunks we plotted r =1..12 and for full length detectors we plotted r =8, 10 and 12 (the points for r-chunks and those for full-length detectors are each connected via a line to indicate the ordering in terms of r). Each point shows false positives (x-axis) and true positives (y-axis).

Section 4 tells us that for any self set, a given value of r will always achieve equivalent-or-greater overall coverage (i.e. a greater sum-total of true and false positives) when using r-chunks than with using full-length detectors. This follows from the fact that there are no holes induced by r-chunks which are not also induced using full-length detectors. The experiment shows whether or not this additional coverage is helpful. Figure 3 shows that for this data set r-chunks outperforms full-length detectors. The greater coverage achieved by r-chunks more often results in the detection of true positives than false positives. In fact, for any value of r shown using full-length detectors, there exists some value of r for which r-chunks achieve a higher rate of true positives while incurring an equal or lesser number of false positives.

Another property of r-chunks illustrated by the graph is that for a given value of r, equivalent-or-greater overall coverage will always be achieved using $r + 1$ rather than r. This is because any string detected using r can be detected using $r+1$. For this reason, as we increase r, while the number of true and/or false positives may increase or remain constant, neither value can decrease.

A surprising result is how well r-chunks performs as r becomes low (e.g. even for $r = 1$). An explanation for this phenomenon is discussed below. This is surprising in part because of the difficulty reported by Kim and Bentley [2001] in finding detectors using r-contiguous bits and negative selection, a result explained in part by their choice of a low value for r [Balthrop *et al.*, 2002].

7.1 r-Chunks and the Magic Bit

We were interested in how r-chunks could perform so well, especially for $r = 1$. A closer examination of the data revealed that the DHCP (Dynamic Host Configuration Protocol) configuration on the internal network was set up in such a way that dynamic IP addresses were always assigned with the final byte in the range 128-254, while static IP addresses were always in the range of 1-127 for the same byte. This is not an unusual DHCP configuration. As it happened, however, no hosts connected to the network using DHCP during the normal data collection period. When we ran the attacks, the attacking laptop did use DHCP to connect to the network, and the majority of the attacks were launched from this laptop (the Denial-of-Service attack is the only one that wasn't).

As a consequence, the majority of our attack data had the first bit of the 49-bit string (the internal IP is at the start of the string) set to one, while none of the normal data had this bit set. In other words, there was a single "magic bit" that identified approximately 84% of the attack SYN packets. r-chunks was able to detect this magic bit and take advantage of it. Thus, even the smallest possible window $r = 1$ could take advantage of the magic bit, and because $r + 1$ can detect everything that r can detect, all of the other r values can use the magic bit as well.

Although artifacts such as these are not unlikely occurrences in real data, we were curious to see what the results would be without the presence of a magic bit. Would r-chunks (and full-length detectors) still perform well? To answer this question we eliminated the magic bit from our data by systematically changing the internal address of the computer from which the attacks originated to look like the address of another internal computer. This scenario is also realistic, because the attacks could as easily have originated from an internal computer as from a malicious laptop, and such an internal attack might be more difficult to detect.

We repeated the r-chunks experiments with this modified data set. The results are shown in Figure 4. From this figure, we can see that r-chunks did not perform as well as before. In particular, the low r-values did not yield results as dramatically positive as before. Removing the magic bit also hurt the performance of r-contiguous bits for $r = 10$ and $r = 12$, although the effect was not as significant as for r-chunks. However, r-chunks without the "magic bit" still outperforms full-length detectors with the magic bit for all the r-values we tested ($r = 8, 10, 12$).

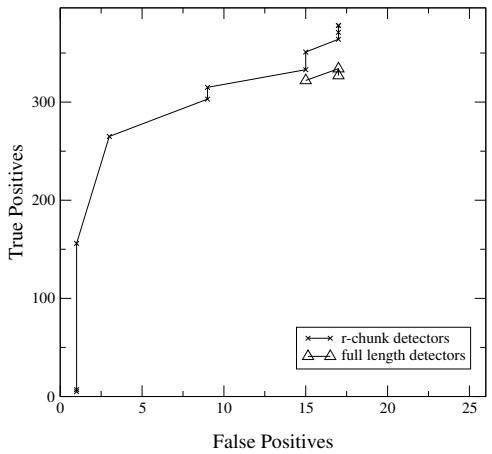


Figure 4: LISYS performance under different r-values after the magic bit has been removed. For r-chunks we plotted r =1..12 and for full length detectors we plotted r =8, 10 and 12. Each point shows false positives (x-axis) and true positives (y-axis).

8 Conclusions

In this paper we introduced a new matching rule, r-chunks, and showed that it performs better than full-length r-contiguous bits matching on one data set. r-Chunks is appealing because it is easier to analyze mathematically [Esponda and Forrest, 2002] and it scales well as the length of l increases (both in terms of efficiency of matching and in terms of number of detectors that are required for a given level of coverage). This second property is essential if AIS frameworks such as LISYS are to be used for real applications. We also studied the effect of different permutations on the ability of LISYS to generalize from an initial sample self. This form of generalization is important for controlling false positives. The results reported here show that some permutations perform much better than others, and we have given an informal explanation for why that is true.

The r-chunks detection scheme is intriguing because it solidifies the connection between r-contiguous bits matching and crossover. Although we have shown that

the crossover-closure is a good generalization for this data set, we still don't know whether it will carry over to related problems. However, the connection is tantalizing, and one that we plan to explore in future work.

It is important to emphasize that the results presented here are empirical and are based on one small data set. An important avenue for further work is to conduct experiments on other applications and to develop a mathematical understanding of the properties of this system. A second caveat concerns the simplified version of LISYS used to conduct these experiments. In the future, it will be important to confirm how well permutations and r-chunks perform in the context of the complete LISYS system.

Acknowledgments

The authors gratefully acknowledge the support of the National Science Foundation (grants IRI-9711199, CDA-9503064, and ANIR-9986555), the Office of Naval Research (grant N00014-99-1-0417), Defense Advanced Projects Agency (grant AGR F30602-00-2-0584), the Intel Corporation, and the Santa Fe Institute.

References

[Balthrop *et al.*, 2002] J. Balthrop, S. Forrest, and M. Glickman. Revisiting lisys: Parameters and normal behavior. In *CEC-2002: Proceedings of the Congress on Evolutionary Computing*, 2002.

[D'haeseleer *et al.*, 1996] P. D'haeseleer, S. Forrest, and P. Helman. An immunological approach to change detection: algorithms, analysis and implications. In *Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy*. IEEE Press, 1996.

[D'haeseleer, 1996] P. D'haeseleer. An immunological approach to change detection: theoretical results. In *Proceedings of the 9th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 1996.

[Esponda and Forrest, 2002] F. Esponda and S. Forrest. Defining self: Positive and negative detection. Technical Report TR-CS-2002-03, University of New Mexico, 2002.

[Heberlein *et al.*, 1990] L. T. Heberlein, G. V. Dias, K. N. Levitte, B. Mukherjee, J. Wood, and D. Wolber. A network security monitor. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEE Press, 1990.

[Helman, 2002] Paul Helman, 2002. Personal communication.

[Hofmeyr and Forrest, 1999] S. Hofmeyr and S. Forrest. Immunity by design: An artificial immune system. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1289–1296, San Francisco, CA, 1999. Morgan-Kaufmann.

[Hofmeyr and Forrest, 2000] S. Hofmeyr and S. Forrest. Architecture for an artificial immune system. *Evolutionary Computation Journal*, 8(4):443–473, 2000.

[Hofmeyr, 1999] S. Hofmeyr. *An immunological model of distributed detection and its application to computer security*. PhD thesis, University of New Mexico, Albuquerque, NM, 1999.

[Holland *et al.*, 1986] J.H. Holland, K.J. Holyoak, R.E. Nisbett, and P. Thagard. *Induction: Processes of Inference, Learning, and Discovery*. MIT Press, 1986.

[Holland, 1975] John H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, MI, 1975.

[Kim and Bentley, 2001] J. Kim and P. J. Bentley. An evaluation of negative selection in an artificial immune system for network intrusion detection. In *GECCO-2001: Proceedings of the Genetic and Evolutionary Computation Conference*, 2001.

[Mukherjee *et al.*, 1994] B. Mukherjee, L. T. Heberlein, and K. N. Levitt. Network intrusion detection. *IEEE Network*, pages 26–41, 1994.

[Percus *et al.*, 1993] J. K. Percus, O. Percus, and A. S. Perelson. Predicting the size of the antibody combining region from consideration of efficient self/non-self discrimination. *Proceedings of the National Academy of Science*, 90:1691–1695, 1993.

[Wierzchon, 2000] S. T. Wierzchon. Discriminative power of the receptors activated by k-contiguous bits rule. *Journal of Computer Science and Technology*, 1(3):1–13, 2000.

[Wierzchon, 2001] S. T. Wierzchon. Deriving concise description of non-self patterns in an artificial immune system. In S. T. Wierzchon, L. C. Jain, and J. Kacprzyk, editors, *New Learning Paradigm in Soft Computing*, pages 438–458, Heidelberg New York, 2001. Physica-Verlag.