An Immunological Approach to Change Detection: Theoretical Results

Patrik D'haeseleer
Department of Computer Science
University of New Mexico
Albuquerque, New Mexico, 87131
patrik@cs.unm.edu
http://www.cs.unm.edu/~patrik

Abstract:

This paper examines some of the theoretical foundations of the distributable change detection method introduced by Forrest et al. in [10], including fundamental bounds on some of its parameters. A short overview is given of the reasoning behind this method, its immunological counterpart and its computer implementation. The amount of information that is lost by splitting a data stream into unordered strings can be estimated, and this estimate can be used to guide the choice of string length. A lower bound on the size of the detector set is derived, based on information-theoretic grounds. The principle of holes (undetectable nonself strings) is illustrated, along with a proof of their existence for a large class of matching rules. The influence of holes on the achievable failure rate is discussed, along with guidelines on how to avoid them.

1. Introduction

One of the main problems in the field of computer security is how to distinguish *self* (legitimate users, authorized actions, original source code, uncorrupted data, etc.) from *nonself* (intruders, computer viruses, spoofing, Trojan horses, etc.) Nature, more specifically the natural immune system, has been solving a similar problem for millions of years and may have some clues about how to approach this.

The body's immune system comprises a whole range of mechanisms, the interactions between which are still being unraveled. These mechanisms are often categorized as either "specific" or "nonspecific" depending on whether they offer a specialized protection against a known type of intrusion (such as antibodies reacting against one specific kind of antigen), or a more general protection against anything coming in from the outside (such as the skin, or inflammatory responses against cell damage). Likewise, computer security measures can be divided into specific (virus checking with signatures, security analysis tools [9], etc.) and nonspecific (good code hygiene, firewalls, encryption, etc.). However, most of the nonspecific measures are passive, in the sense that they only *prevent* nonself from intruding on the system but don't detect intrusions in progress, whereas the specific ones have a hard time keeping up with the new attacks continually being developed or discovered by malicious agents.

In earlier work by Forrest et al. [10], a change detection algorithm was developed based on one of the mechanisms used by the immune system: the generation of T-lymphocytes in the thymus (see also [15] for another possible computer security mechanism borrowed from the immune system). T-lymphocytes, or T-cells, are one of the many kinds of specialized cells in the immune system. The surface of a lymphocyte is covered with receptors that can bind to antigens (foreign proteins). Each lymphocyte has one specific kind of receptor, and each receptor binds to a small group of structurally related antigens. The receptors on T-lymphocytes are constructed by a pseudo-random process. As the T-cells mature in the thymus, they undergo a censoring process called *negative selection*, in which those T-cells that bind self proteins are destroyed [14, 17]. After censoring, T-lymphocytes that do not bind self are released to the rest of the body and provide the basis for our immune protection against foreign antigens. This mechanism in the immune system is very robust because of its distributed nature, and remarkably efficient. The repertoire of T-cells produced in this way seems to cover most of the antigen space (estimated at around 10^{16} different foreign molecules), while tolerating an estimated 10^{6} different body proteins as "self" [13].

The algorithm presented in [10] works in a similar way. Detectors are generated to match anything that does not belong to self (for this reason, it is also sometimes called the "negative selection method"). In operation, these detectors are used in much the same way as signatures are used to scan for specific computer

To appear at the 9th IEEE Computer Security Foundations Workshop, Dromquinna Manor, County Kerry, Ireland. 10-12 June 1996.

viruses. However, a virus scan program only checks for a small number of known signatures and needs to be updated continuously with signatures for newly emerging viruses. Our method relies on having a large enough set of random detectors that virtually all nonself strings will be detected. In that sense it can be classified as an active, yet nonspecific protection mechanism. In immunology, T-cells are counted under the specific immune response, because each individual T-cell responds to a specific antigen. However, the generation of a repertoire of randomly generated T-cells covering almost all of the antigen space may be viewed as a nonspecific mechanism.

This method (as most other change detection methods) relies on having an accurate and stable description of self.. In this sense it is very static (although some of the more dynamic mechanisms in the immune system could be used to counteract this). However, its application is not limited to purely static data sets such as data files. Current research is focusing more on applying it to monitoring patterns of activity. For instance, the pattern of "normal" sequences of system calls for certain running processes has been shown to be quite stable [11]. A lot of work has been done on making this method more useful in practice. One of the difficulties in implementing it is that there are many degrees of freedom in choosing the parameters. [8] describes some practical constraints on a number of the parameters involved and gives guidelines on how to choose them. The work presented here is an attempt at a theoretical analysis of the method and its properties, and derives some theoretical bounds for these parameters.

Section 2 gives a short overview of the change detection method and its main features. In Section 3, an estimate is derived of the amount of information that is lost by cutting a data stream into an unordered set of strings of a given length, and offer suggestions on how to choose the string length appropriately. Section 4 shows the derivation of a lower bound on the number of detectors needed to achieve a certain level of reliability of detection, based on mutual information between the protected set and the detector set. Section 5 shows that for most matching rules there may be nonself strings which cannot possibly be covered by any detectors, and the consequences of this. The basis for the material in these last three sections can also be found in [7]. Section 6 presents some conclusions and outlines further directions of study.

2. The change detection method

To date, our research on this new change detection method has focused largely on the case where both the detectors and the data to be protected consist of fixed length strings, over a given alphabet of symbols (usually binary). Similar to the generation of T-cells in the thymus, detector strings can be generated at random. The ones that match any of the self strings to be protected (according to a specified matching rule) are eliminated. The ones that don't match any of the self strings are stored in the detector set (or "repertoire") R. This process is repeated until we have enough detectors to ensure the desired protection level (defined in terms of the probability of success in detecting any nonself strings with at least one detector in R). This generate-and-test method, illustrated in Figure 1, requires sampling a number of candidate detectors which is exponential in the size of the self set [5]. Using a dynamic programming approach, two new algorithms were developed [12, 6, 7] that can generate a set of detector strings for the "r-contiguous-bits" matching rule (in which two strings are said to match if they are identical in at least r contiguous positions) in linear time with respect to the size of the self set.

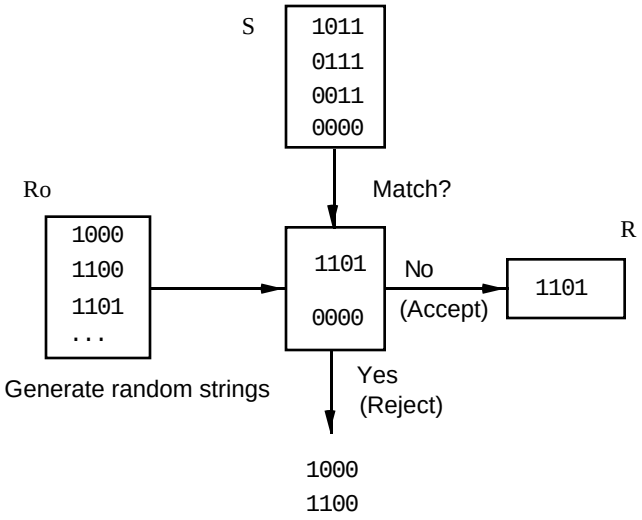


Figure 1: Constructing a set of detectors R for a set of self strings *S*, using generate-and-test. The matching rule used here is "r-contiguous-bits" with *r*=2. The strings in R_0 are generated at random. The ones that match any of the strings in *S* are rejected.

The negative selection method has a number of features which distinguish it from most other methods:

- Since this is a general change-detection method, no prior knowledge of intrusions is necessary. This is in contrast with signature-based virus scanners.
- Detection is probabilistic, but tunable. That is, in general we will not be generating a *complete* repertoire of detectors (i.e. a set of detectors that covers all possible nonself strings). Instead we will content ourselves with matching all but a small fraction P_f of nonself strings, in exchange for a smaller set of detectors. The desired probability of success in detecting random nonself strings with a single set of detectors can be weighed against the cost of generating, storing and checking detectors.

- The detection scheme is inherently distributable: small sections of the protected object can be checked separately, different sites can have independently created detector sets for the same object, and the detectors in the detector set can themselves be run independently. No communication between detectors or detector sets is needed until a change is detected. This makes it a promising tool for computer security in networked or distributed computer environments, or in a scheme with autonomous agents such as the one presented in [4].
- The set of detectors at each site can be unique. This means that if one site should be compromised, others would still be protected. For independent detector sets, the *system-wide* failure probability decreases exponentially with the number of sites protected. This may allow us to choose a fairly high P_f (and thus small detector sets) for the individual systems.
- The set of self strings and the detector set are mutually protective, meaning that the detector set protects the self set against change and vice versa. Also, detection is local: we can pinpoint the individual string (or detector) which has been changed.

3. Splitting data into strings

The change detection method described above works on an unordered set of strings. In order to apply the method to a real data set (such as a file on disk, an audit trail, the representation of the behavior of a process, a sequence of system calls of a running program, etc.), we will usually have to preprocess the data by splitting it into strings of length l over an alphabet of size m. Reducing a continuous data stream to an unordered set of strings destroys some of the internal structure of the data that could be used to make the detection proceed more efficiently. When trying to distinguish self from nonself, we will want to make use of as much information as possible about what constitutes self. Loss of information about self may imply a smaller achievable success rate, therefore we would like to minimize this information loss. In the following analysis we will assume a binary alphabet (m=2), but extension to larger alphabets is straightforward.

The information lost by splitting up a stream $\hat{S}$ into a set of unordered strings S is equal to the amount of information (in bits) we would need to be given in order to reconstruct the original stream from the unordered set, or alternatively the amount of information we would gain by being given the original stream $\hat{S}$ if we already knew S. This can be denoted by $H(\hat{S}\,|\,S)$ (i.e. the conditional entropy of $\hat{S}$ given S). In information theory, this quantity is often called the Doubt, since it is the amount of information about $\hat{S}$ that is missing in the encoding S. For a given string length l, a stream $\hat{S}$ of L_S bits will be split up into a set S of N_S l-bit self strings. Usually, not all of these strings will be unique, so S is really a multiset. Let's say we get k unique l-bit strings, where each string s_i appears N_i times:

$$\sum_{i=1..k} N_i = N_S \,.$$

Given such a set of N_S strings, the original stream could have been one of

$$\binom{N_S}{N_1, N_2 \cdots N_k} = \frac{N_S!}{N_1! \cdot N_2! \cdots N_k!}$$

possible rearrangements of these strings. If we assume that each possible rearrangement is equally likely (i.e. we have no prior information about the nature of the data stream), then the amount of information lost is:

$$\Delta I_1 = H\left(\hat{S}|S\right) = \log_2 \binom{N_S}{N_1, N_2 \cdots N_k} \text{ bits.}$$

From Stirling's approximation for the factorial function, or by a reasoning similar to the one in [3], it can be shown that (1):

$$\binom{N_S + k - 1}{N_S}^{-1} 2^{N_S \cdot H(N_i/N_S)} \le \binom{N_S}{N_1, N_2 \cdots N_k} \le 2^{N_S \cdot H(N_i/N_S)}$$

where $H(N_i/N_S)$ stands for the entropy associated with the relative frequencies of occurrence of the strings in S. Taking the logarithm base 2, this becomes:

$$\begin{aligned} & N_S \cdot H\left(\frac{N_i}{N_S}\right) - \log_2 \binom{N_S + k - 1}{N_S} \\ \le\ & \log_2 \binom{N_S}{N_1, N_2 \cdots N_k} \\ \le\ & N_S \cdot H\left(\frac{N_i}{N_S}\right) \end{aligned}$$

or

$$\log_2 \binom{N_S}{N_1, N_2 \cdots N_k} = O\left(N_S \cdot H\left(N_i/N_S\right)\right) .$$

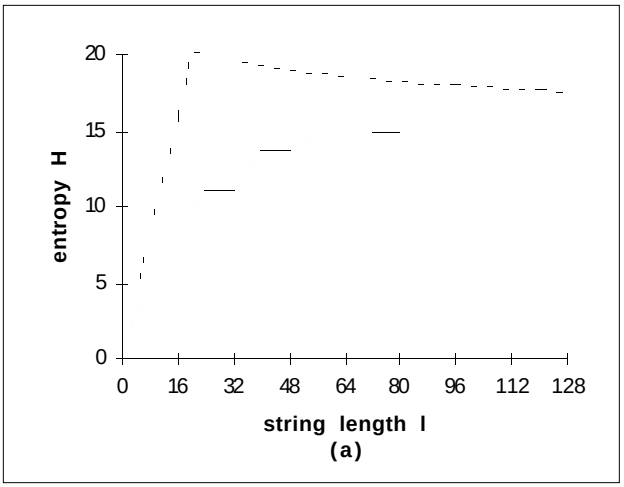
Therefore, as a bound on the amount of information lost, we get:[1]

$$\Delta I_1 = O\left(N_S \cdot H\left(N_i/N_S\right)\right) = O\left(L_S \cdot H\left(N_i/N_S\right)/l\right) .$$

This analysis suggests that if we measure the average entropy per bit of the strings in S for different string

[1] Interestingly, it can be shown that the average Kolmogorov complexity of a member of a simply described ensemble is almost identical to its entropy. More formally (see [2, 16, 1]), $H(p(s_i)) \le \sum p(s_i) K(s_i) < H(p(s_i)) + K(s_i, p(s_i)) + O(1)$, where $K(s_i)$ is the Kolmogorov complexity of string s_i, and $K(s_i, p(s_i))$ is the complexity of the ensemble of strings. This would mean that $\Delta I = O\left(\sum N_i \cdot K(s_i)\right)$ if $K(s_i, p(s_i))$ is small: the information lost by splitting a data stream into strings is of the order of the sum of the Kolmogorov complexities of the resulting strings!

lengths l, we could minimize the information loss by choosing the value of l for which the entropy per bit is minimal. In general, this will only be one of many factors that need to be considered in the choice of l. Choosing a small l tends to reduce the entropy of the strings because it reduces the size of the total string space. The entropy per bit will tend to go up with decreasing string length though. Also, as l gets smaller S will occupy a larger and larger fraction of this space, which will adversely affect the behavior of the detection algorithm itself. On the other hand, increasing l beyond the point where most of the strings in S are unique will reduce entropy because it reduces the number of strings in S, but choosing a large l will make generation of detectors difficult in terms of space and time requirements.



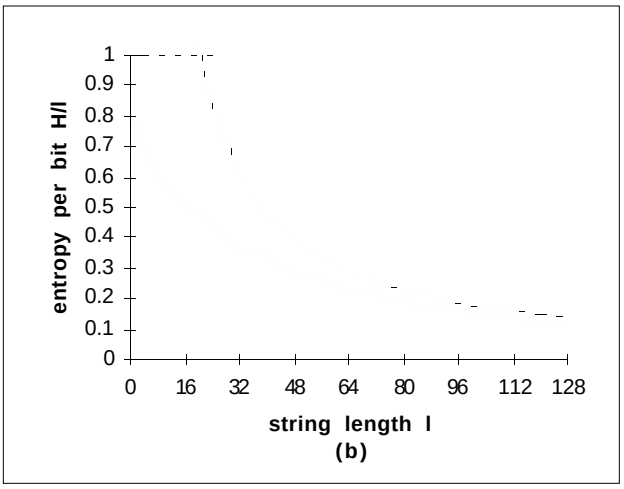


Figure 2: Entropy of the set of strings obtained by splitting an emacs binary into *l*-bit strings. The dotted lines indicate the envelope within which *H* must fall, delimited by $H \le l$ and $H \le log_2(L_S/l)$ (corresponding to, respectively, having all *l*-bit strings being present in equal frequency in *S*, and all strings in *S* being unique). (a) shows the entropy $H(N_i/N_S)$ of the resulting set of strings. (b) shows the entropy per bit of these strings, which is proportional to ΔI_1.

Figure 2 illustrates how $H(N_i/N_S)$ changes with respect to l for a real data file. The data chosen was an emacs binary (GNU emacs v19.25.2 SGI binary, 3.2MB). The file was split up into strings of length l=1 (single bits), 2, 4, and multiples of 8 (i.e. at byte boundaries) up to 16 bytes long. For each value of l, the entropy of the resulting set of strings was calculated. Figure 2(a) shows clearly that there are minima in the entropy at multiples of 4 bytes, corresponding to the 32-bit RISC instruction set this binary is compiled for. The values plotted here at 8-bit intervals are themselves minima as well.: for intermediate values of l the entropy would be much higher, because this corresponds to cutting up the data at non-byte boundaries, ignoring the natural byte structure present in this data. The entropy per bit plotted in Figure 2(b) is proportional to ΔI_1 (apart from a constant factor L_S). This graph is uniformly decreasing, so in order to minimize information loss for this data we would want to choose l as high as our other constraints allow, with a preference for multiples of 4 bytes.

This calculation of the information lost by splitting up the data at l-bit boundaries does not only hold for the change-detection method described here. In fact, this is a general result for any method which transforms a stream of data into an unordered set of strings. In particular, it would also hold for methods that take into account the frequency of occurrence of the self strings. However, in our method there is a second source of information loss, which is that the change-detection algorithm does not encode how many duplicates of each unique l-bit string s_i occur in S. If we assume N_S is known, then if we are given the set of k unique strings in S $(s_1, s_2 ... s_k)$, there are

$$\binom{N_S - 1}{k - 1}$$

possible ways of assigning the values for N_i (such that they sum up to N_S), and thus of reconstructing the unordered multiset S. If each of those assignments is equally likely (again, assuming we have no prior knowledge of the contents of the data stream), the amount of information (in bits) lost by ignoring the frequencies of the strings is given by:

$$\begin{aligned}
\Delta I_2 &= H(S|s_i) \\
&= \log_2 \binom{N_S - 1}{k - 1} \\
&\le (N_S - 1) \cdot H\left(\frac{k-1}{N_S - 1}\right) \qquad \text{(see [3])} \\
&< N_S \, .
\end{aligned}$$

In general, ΔI_2 will be small compared to ΔI_1 and can be ignored. If we do want to minimize ΔI_2, minimizing N_S by choosing a large string length l would be the obvious way to do it. This will tend to reduce the number of duplicate strings and thus the amount of information lost by ignoring duplicates, although this may in some cases increase ΔI_1 because of an increase in $H(N_i/N_S)$.

4. Choosing the number of detectors

The number of detectors needed to achieve a certain acceptable failure rate P_f is an important factor in comparing different parameter settings, because this affects the time needed to generate the detector set, the storage space needed for the detector, the amount of nonself space covered by each detector, etc. It is possible to derive a theoretical lower bound on the number of detectors needed, again based on the amount of information about *S* that can be stored in *R*.

A set of detectors *R* constructed for a given set of self strings *S* can be viewed as a message encoding information about that self set. Given a "perfect" detector set *R´*, that is a set of detector strings that exactly recognizes all nonself strings, it is possible to reconstruct the original self set *S* (by checking for each string in the string space whether it is detected by any detector in *R´*). Therefore, this set *R´* would have to contain at least the same information as the original set *S*. By calculating the information content of the original file of self strings it is possible to get an absolute lower bound on the information content, and thus on the size, of the detector set. If the self set *S* consists of independently chosen self strings, its information content will be of the same order as its size in bits. This means that the perfect detector set *R´* would have to be of approximately the same size (in bits) as the self set *S*. For large self sets this may not be acceptable.

We can alleviate this problem by allowing a certain amount of error in the self-nonself detection performed by the detectors. For the purpose of this analysis, we will assume that a fraction P_f of the nonself strings are not matched by the detectors. In other words, we allow at most $P_f(N_U - N_S)$ (where *U* is the total string space, and $N_U = |U|$) false negatives (nonself strings not detected) but no false positives (no self strings classified as nonself). Not requiring an exact encoding of the self set *S* means that we can get away with a much smaller information content, and thus a much smaller size, for the detector set.

If we choose the self strings from the set of all possible strings *U*, then there are

$$M = \binom{N_U}{N_S}$$

possible self sets of size N_S. If we assume (i) that the set *S* of self strings is independent, i.e. that each string is chosen at random out of the set of all possible strings, then each of those sets is equally likely, so the average information content (i.e. entropy) of a self set of size N_S is:

$$H(S) = \log_2 \binom{N_U}{N_S}.$$

Given a detector set *R* constructed for this self set *S* we could try to reconstruct *S*. Since *R* has been constructed to allow a certain amount P_f of false negatives, we won't be able to reconstruct *S* exactly from *R*. Instead we would get a set *S´* consisting of the N_S original self strings plus $P_f(N_U - N_S)$ unmatched nonself strings.

Assumption (i) says that the self strings are independent of each other. We will make a second assumption (ii) that the unmatched nonself strings are also independent, and independent from the self strings. In other words, given *S´* we have no knowledge of which subset of N_S strings constitutes the original set *S*. The amount of information about *S* that is missing in the encoding *R* is then $H(S|R)$ (i.e. the amount of information we would gain by being given *S* if we already knew *R*). Assumptions (i) and (ii) tell us that any subset of size N_S, taken from the set *S´* of size $N_S + P_f(N_U - N_S)$, is equally likely to be the original self set *S*, so:

$$H(S|R) = \log_2 \binom{N_S + P_f(N_U - N_S)}{N_S}.$$

The difference between *H(S)* and $H(S|R)$ (i.e. the information in S, minus the information about *S* that is missing in *R*) represents the amount of information about *S* that is preserved in the encoding *R* and is called the Mutual Information of *S* and *R* (denoted by $I(S;R)$):

$$\begin{aligned}
I(S;R) &= H(S) - H(S|R) \\
&= \log_2 \binom{N_U}{N_S} - \log_2 \binom{N_S + P_f(N_U - N_S)}{N_S} \\
&= \log_2 \left[\binom{N_U}{N_S} \Big/ \binom{N_S + P_f(N_U - N_S)}{N_S} \right] \\
&= \log_2 \left[\frac{N_U!}{N_S!\cdot(N_U - N_S)!} \Big/ \frac{(N_S + P_f(N_U - N_S))!}{N_S!\cdot(P_f(N_U - N_S))!} \right] \\
&= \log_2 \left[\frac{N_U}{N_S + P_f(N_U - N_S)} \cdot \frac{N_U - 1}{N_S + P_f(N_U - N_S) - 1} \cdot \right. \\
&\qquad \left. \cdots \frac{N_U - N_S + 1}{N_S + P_f(N_U - N_S) - N_S + 1} \right]
\end{aligned}$$

The next two steps hold under the assumption (iii) that $N_U >> N_S$ and $P_f N_U >> N_S$. Further simplifying we get:

l	N_S	N_R ($P_f = 0.1$)	N_R ($P_f = 0.01$)
16	62500	12977	25953
20	50000	8305	16610
24	41667	5768	11535
28	35715	4238	8474
32	31250	3245	6489
36	27778	2564	5127
40	25000	2077	4152

Table 1: lower bounds on N_R for a 1Mbit long data set, for different string lengths.

$$I(S;R) \approx \log_2\left[\frac{N_U}{P_f N_U}\cdot\frac{N_U-1}{P_f N_U-1}\cdots\frac{N_U-N_S+1}{P_f N_U-N_S+1}\right]$$
$$\approx \log_2\left[\left(1/P_f\right)^{N_S}\right]$$
$$\approx N_S\cdot\log_2\left(1/P_f\right).$$

(Similar results can be obtained using (1)). Since R needs to contain at least this much information, this is a lower bound on the size of the detector set, expressed in bits (the detector set may contain spurious information introduced by the stochastic generation of the set). If the detectors are strings of length l in an alphabet of size m, the number of detectors N_R is given by:

$$N_R \geq \frac{N_S\cdot\log_2\left(1/P_f\right)}{l\cdot\log_2(m)}\ . \qquad (2)$$

Table 1 illustrates this theoretical lower bound for N_R as a function of string length l for a data stream of one million bits. Note that for a tenfold decrease in failure rate we only need a twofold increase in the number of detectors.

If the self strings are not independent of each other (as is often the case in reality), this will affect both assumptions (i) and (ii). Assumption (i) (self strings are independent) led us directly to the formula for $H(S)$ because we were allowed to assume that all self sets were equally likely. In reality this will not be the case. Real data sets tend to have more similarity between the self strings than sets of totally random strings. This may affect $H(S)$ because real data will tend to form only a subset of all possible sets, and thus any single real data set may contain less information than a random one. However, given only one single self set it is impossible to judge how likely this set is, and therefore how much information it contains. Obviously, we would need to use more high-level knowledge about the nature of the ensemble of "normal" self sets in order to be able to estimate the entropy of a single set. In any case, $H(S)$ will probably be smaller than the value used above, allowing for a smaller encoding.

It is unclear what effect non-independent self strings would have on assumption (ii). Clearly, if the $P_f\left(N_U - N_S\right)$ undetected nonself strings in S' are independent of each other but the N_S self strings in S' are dependent, we may some information about which subset N_S of strings are more likely to be the original self file S and therefore $H(S \mid R)$ can be smaller. However, the character of the undetected nonself strings will largely depend on the matching rule and repertoire construction algorithm chosen. With our current matching rules and algorithms, we would expect the undetected nonself strings to lie approximately in the same region of space as the self strings. This may mean that assumption (ii) still holds. Of course, in practice it is recommended to check the real failure probability achieved with the generated detector set, by sampling nonself strings or expected intrusion strings, if the distribution of these is known.

Assumption (iii) states that $P_f N_U >> N_S$. In general it will be reasonable to assume at least $N_U >> N_S$ (otherwise almost all possible detector strings would match at least one of the self strings, and most of the nonself strings would in fact be undetectable holes). For failure rates that are not excessively small, assumption (iii) will usually still hold. For instance, with $P_f = 0.1$, every single nonself string has 10% chance of escaping detection by the detectors. If an intrusion in the self file consists of one single changed string, 10% may be quite a large error rate. However, usually we will have to deal with intrusions consisting of several different nonself strings and it is sufficient to catch only one of these to sound the alarm. If we have to use a much smaller value for P_f, our approximation may no longer hold.

Although the above analysis was done with the change detection algorithm in mind (and bit strings for detectors and self strings), it is clear that this approach is a lot more general than that. In fact, the result above holds for any possible encoding of a set S with zero false positives and P_f false negatives. In particular, these are some of the methods covered:

- generating a detector set for Hamming distance based matching rules
- matching rules with variable radius for each detector
- matching rules with symbolic descriptions of S and/or R
- matching rules based on complementarity of 3-D matching sites on antigen and detector proteins
- self/nonself recognition using some positive selection method (i.e. by recognizing what is self, instead of what is nonself) such as Neural Networks etc.
- other methods (such as k-nearest-neighbors) for self-nonself detection

A second lower bound for N_R particular to the negative selection method can be derived, given the average matching probability P_m (probability that a ran-

domly chosen string and detector match according to the specified matching rule) [18]. At best, we can spread the detectors apart such that no two detectors match the same nonself string. The amount of string space covered by N_R detectors is then $P_m \cdot N_R \cdot N_U$. This needs to be at least as big as $(1-P_f)N_U$, in order to cover enough nonself strings for a failure probability of P_f (ignoring N_S with respect to N_U), so:

$$N_R \geq (1-P_f)/P_m \ .$$

5. The existence of holes

Having found a lower bound for the number of detectors needed to achieve a certain failure probability, we would like to know the best achievable failure probability P_f. This depends largely on the specific matching rule used. One that has been examined in detail is the "r-contiguous-bits" matching rule (two l-bit strings match each other if they are identical in at least r contiguous positions). Efficient detector generating algorithms [12, 5] have been developed for this rule, making it possible to generate a *complete* detector repertoire, in the sense that it covers all nonself strings that *can* be covered. As shown in [8], it turns out that for this matching rule there may be some nonself strings, called "*holes*," for which it is impossible to generate valid detectors. For clarity, I have reproduced the example presented there:

If S contains two strings s_1 and s_2 that match each other over (r-1) contiguous bits, they may induce two other strings h_1 and h_2 that cannot be detected because any candidate detector would also match either s_1 or s_2, as shown below:

$$\begin{aligned} s_1 &: \ a_1 ... a_k \ b_{k+1} ... b_{k+r-1} \ c_{k+r} ... c_l \\ s_2 &: \ a'_1 ... a'_k \ b_{k+1} ... b_{k+r-1} \ c'_{k+r} ... c'_l \\ &\Downarrow \\ h_1 &: \ a_1 ... a_k \ b_{k+1} ... b_{k+r-1} \ c'_{k+r} ... c'_l \\ h_2 &: \ a'_1 ... a'_k \ b_{k+1} ... b_{k+r-1} \ c_{k+r} ... c_l \end{aligned}$$

where a_i, a'_i, b_i, c_i, and c'_i are single bits.

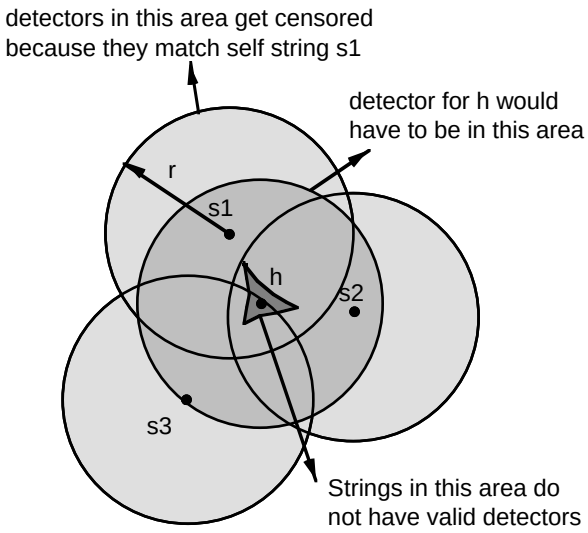


Figure 3: if self strings are presented as points in a plane and the matching rule is Euclidean distance, three self strings s_1, s_2, s_3 may induce a hole *h* for which no valid detector can be generated.

A similar argument shows that we also can have holes using a Hamming distance matching rule (where two strings match if their Hamming distance is less than or equal to a fixed radius r). Figure 3 shows a similar situation that occurs when self strings are represented as points in a Cartesian plane and a match between a string and a detector is said to occur if the Euclidean distance between the corresponding points is smaller than a certain radius r.

It can be shown that all practical matching rules with a constant matching probability (i.e. each detector matches equally many strings and vice versa) can exhibit holes even with a non-trivial set of self strings.

Let U be the set of all strings, S the set of self strings to be protected, $M(d)$ the strings matched by detector d, and $M'(s)$ the detectors matching string s, then $h \in U$ is a $hole$ iff $\forall d \in M'(h), \exists s \in S : s \in M(d)$ (3). Given a string h and a matching rule M with constant matching probability P_m, it suffices to show that we can construct a non-trivial self set S such that h is a hole. Starting with the empty set, for each detector d matching h, we pick one string s_d matched by d and add this string to S. Then, by construction, for each possible detector for h there exists a self string matching that detector, such that (3) holds. The set S constructed in this way will be non-trivial if it does not need to contain a significant portion of the total string space U in order for h to be a hole.[2] If we are using a matching rule with a constant matching probability P_m, a self set of size $N_S = |M'(h)| = P_m \cdot N_U$ always suffices to induce holes. Of course we may already get holes for smaller self sets than that. As we have shown earlier, two self strings matching over r-1 contiguous bits are sufficient for the r-contiguous-bits matching rule.

The existence of holes imposes a lower bound on the failure probability P_f we can achieve with a detection method because it will always fail to detect holes (see Table 2). If we calculate the required number of detectors to achieve a certain acceptable P_f without taking holes into account, the real P_f achieved with this detector set may be substantially higher than expected. Further, the failure probability associated with the holes themselves does not improve by distributing the algorithm if we use the same matching rule at all the sites. Although the detector set on each machine may be generated independently from the other machines, the

[2] An example of a trivial set S inducing holes would be $U - \{h\}$. For this self set, h would be a hole for almost all matching rules.

L_S (a)	N_S (b)	l (c)	r (d)	P_m (e)	number of holes (f)	lowest possible P_f (g)
500B	250	16	10	0.00391	634	0.0097
			9	0.00879	4438	0.0677
			8	0.01953	21076	0.3216
1KB	250	32	11	0.00562	2649	6.1676e-07
			10	0.01172	24911	5.8000e-06
			9	0.02441	2150714	0.0005
			8	0.05078	5.1815e+08	0.1206
	500	16	11	0.00171	882	0.0135
			10	0.00391	3854	0.0588
			9	0.00879	24937	0.3805
2KB	500	32	12	0.00269	4787	1.1145e-06
			11	0.00562	52318	1.2181e-05
			10	0.01172	2420706	0.0006
			9	0.02441	4.6564e+08	0.1084
	1000	16	12	0.00073	1353	0.0206
			11	0.00171	5428	0.0828
			10	0.00391	23933	0.3652
4KB	1000	32	13	0.00128	8475	1.9732e-06
			12	0.00269	85798	1.9976e-05
			11	0.00562	3991790	0.0009
			10	0.01172	5.2296e+08	0.1218

Table 2: Number of holes and best achievable P_f for different configurations. These results were calculated on randomly generated test files, with sizes 500B, 1KB, 2KB and 4KB. (a): size of test data. (b), (c), (d): parameters chosen for the matching rule (r-contiguous-bits). (e): corresponding matching probability P_m. (f): number of holes present. (g): resulting best achievable failure rate P_f .

same holes occur on all machines. It may therefore be useful to try spread the self strings apart as much as possible, reducing the number of holes between them. This could be done by encrypting each self string in some way.

Another option is to try to eliminate the holes altogether. This can be achieved either locally or in a distributed fashion. Locally, this can be achieved by using a matching rule which does not exhibit holes. For instance with the r-contiguous-bits rule, we can choose a different value of r for each detector, between $r=1$ (detector matches almost the entire string space) and $r=l$ (detector only matches a single string: itself). The value of r will have to be stored with each detector. This method also has the advantage that very large sections of string space can be covered with only a few detectors. Note that although this seems to be a much more efficient way to cover the nonself string space, the size of the detector set still has to conform to equation (2).

In a distributed setting on the other hand, with several sites running their own detector sets we can choose a different matching rule (same rule with different parameters or different rule altogether) for each machine. Each will have a different set of holes which hopefully will be covered by some other machines.

Note that the existence of holes can be a blessing instead of a curse. Since holes are generated by interaction between a number of self strings, holes will tend to be "close to" (according to the matching rule used) these self strings. In some applications, we may not wish to detect strings which are that close to self. For instance, if this method is used to monitor the behavior of a process, small deviations from the known normal behavior may be acceptable. On the other hand, if the method is being used to detect viral code in a binary program file, both the self strings and the viral strings will consist of short machine code segments. Pieces of viral code will then be more similar to the valid self strings than to a totally random string of the same length, and thus be more likely to be obscured by holes. In either case, it may be wise to calculate P_f based on the a priori probabilities of expected changes (if known) instead of on random nonself strings.

6. Conclusions

Previous work on this new immunological approach to change detection has shown its feasibility. As a very general-purpose change detection method it can be a useful supplement to more specific, and therefore more brittle, protection mechanisms. Non-specific yet active methods such as this are needed to intercept those intrusions that evade the specific mechanisms. The work presented here provides a good start at understanding the theoretical foundations of this method and its properties. Furthermore, some of the analysis (specifically Sections 2 and 3) is more general in scope and could be of importance to other domains.

A number of issues we have touched on only briefly here may be interesting avenues for further research:

- It should be feasible to come up with more general formulas for systems in which a certain amount P_f' of false positives is allowed as well.
- It may be interesting to investigate further the relationships between the Kolmogorov/Chaitin complexity of the self strings and the minimal repertoire size, or the information loss due to splitting up the data into strings.
- The result derived in Section 3 assumes that we split a file of L_S bits into L_S/l strings of l bits. In practice, we may want to use all possible l-bit strings present in S by using a sliding l-bit window. This would give us $L_S - l + 1$ partially overlapping self strings and a much smaller information loss. It should be feasible

to get an estimate for the amount of information lost in that case.

- It may be possible to derive approximate formulas for non-random data by looking at some measures of the self strings (entropy, number of unique self strings, etc.).
- In order to eliminate the negative effect of holes on the system-wide failure probability, one would want to have a family of matching rules with the same basic parameters. This would allow the parameters to be chosen optimally for the task, while still allowing for an effectively different matching rule at each site. For example, each site could perform a different permutation on the bits in the bit strings before applying the r-contiguous-bits matching rule, or all strings could be encrypted with a site-dependent key. The combination of the permutation (or encryption) plus the r-contiguous-bits rule generates an entire family of matching rule, one for each different permutation (or encryption key).

7. Acknowledgments:

The author thanks Stephanie Forrest, Ron Hightower, Andrew Kosoresow, Derek Smith, and Dipankar Dasgupta for their useful suggestions and critical comments. The idea of a computer immune system grew out of a collaboration with Dr. Alan Perelson through the Santa Fe Institute. This work is supported by grants from the National Science Foundation (grant IRI-9157644), Office of Naval Research (grant N00014-95-1-0364), Interval Research Corporation, General Electric Corporate Research, and Development and Digital Equipment Corporation.

References:

[1] C. H. Bennett, “The thermodynamics of computation - a review”, in *International Journal of Theoretical Physics*, v.21 nr.12, 1982.

[2] G. Chaitin, “On the length of programs for computing finite binary sequences”, in *Journal of the ACM* 13, 1966.

[3] T. M. Cover, J. A. Thomas, *Elements of Information Theory*, a Wiley-Interscience Publication, 1991

[4] M. Crosbie and G. Spafford, "Defending a Computer System using Autonomous Agents", in Proceedings of the 18th National Information Systems Security Conference, 1995.

[5] R. J. De Boer and A. S. Perelson, “How diverse should the immune system be?” in *Proceedings of the Royal Society* London B, v.252, London, 1993.

[6] P. D’haeseleer, “Further efficient algorithms for generating antibody strings”, Technical Report CS95-3, The University of New Mexico, Albuquerque, NM, 1995.

[7] P. D’haeseleer, “A change-detection algorithm inspired by the immune system: Theory, algorithms and techniques”, Technical Report CS95-6, The University of New Mexico, Albuquerque, NM, 1995.

[8] P. D’haeseleer, S. Forrest and P. Helman, “An Immunological Approach to Change Detection: Algorithms, Analysis and Implications”, accepted to *the 1996 IEEE Symposium on Security and Privacy*, 1996.

[9] D. Farmer, W. Venema, "Improving the Security of Your Site by Breaking Into it", ftp.win.tue.nl pub/security/admin-guide-to-cracking.101.Z, 1993.

[10] S. Forrest, A. S. Perelson, L. Allen and R. Cherukuri, “Self-nonself discrimination in a computer”, in *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, Los Alamitos, CA: IEEE Computer Society Press, 1994.

[11] S. Forrest, S. A. Hofmeyr, A. B. Somayaji and T. A. Longstaff, “A sense of self for UNIX processes”, submitted to *the 1996 IEEE Symposium on Security and Privacy*, 1995.

[12] P. Helman and S. Forrest, “An efficient algorithm for generating random antibody strings”, Technical Report CS-94-07, The University of New Mexico, Albuquerque, NM, 1994.

[13] J.K. Inman, “The antibody combining region: Speculations on the hypothesis of general multispecificity”, in *Theoretical Immunology*, M. Dekker, New York, NY, 1978.

[14] J.W. Kappler, N. Roehm, P. Marrack, "T cell tolerance by clonal elimination in the thymus." in *Cell*, 49:273-280, 1987.

[15] J. O. Kephart, “A biologically inspired immune system for computers”, in *Proceedings of Artificial Life IV*, MIT Press, Cambridge, MA, 1994.

[16] L. A. Levin, “Various measures of complexity for finite objects (axiomatic description)”. in *Soviet Mathematics Doklady* 17, 1976.

[17] W. E. Paul, Ed., *Fundamental Immunology*, Raven Press Ltd. New York, 88-90, 1989.

[18] A. S. Perelson and G. F. Oster, “Theoretical Studies of Clonal Selection: Minimal Antibody Repertoire Size and Reliability of self-nonself Discrimination” in *Journal of Theoretical Biology*, 1979.