WCIS: A Prototype for Detecting Zero-Day Attacks in Web Server Requests

Melissa Danforth
Department of Computer and Electrical Engineering and Computer Science
California State University, Bakersfield
melissa@cs.csubak.edu or mdanforth@csub.edu

Abstract

This work presents the Web Classifying Immune System (WCIS) which is a prototype system to detect zero-day attacks against web servers by examining web server requests. WCIS is intended to work in conjunction with more traditional intrusion detection systems to detect new and emerging threats that are not detected by the traditional IDS database. WCIS is at its core an artificial immune system, but WCIS expands on the concept of artificial immune systems by adding a classifier for web server requests. This gives the system administrator more information about the nature of the detected threat which is not given by a traditional artificial immune system. This prototype system also seeks to improve the efficiency of an artificial immune system by employing back-end, batch processing so that WCIS can detect threats on higher capacity networks. This work shows that WCIS is able to achieve a high rate of accuracy at detecting and classifying attacks against web servers with very few false positives.

Tags: Research, Security, Web, Artificial Immune System

1 Introduction

Traditional intrusion detection systems (IDS) are very efficient at detecting known threats and even some emerging variants, but are not as effective at detecting zero-day attacks. Artificial immune systems (AIS) are appealing for detecting zero-day attacks because they are inspired by the adaptive concepts of biological immune systems. Biological immune systems are alluring to the computer security realm because they can innately adapt to new pathogens or variations on previously seen pathogens, something which even modern intrusion detection systems struggle to do. The primary goal of an artificial immune system is to apply these biological principles to the problem of distinguishing normal traffic or data from abnormal traffic or data, even if the abnormal traffic corresponds to a completely new attack.

This work presents a variation of the artificial immune system concept called Web Classifying Immune System (WCIS). WCIS is intended to work in concert with a traditional IDS, scanning the traffic that the IDS has labeled as normal to see if there is a zero-day attack, or even just a new, unknown variant of an existing attack, present in the traffic. As the name implies, WCIS focuses on attacks conveyed in web server requests. While the concepts can apply to other problem domains, this work focuses on web server requests as a "proof of concept".

There are limitations to the traditional AIS model that WCIS seeks to overcome. Most traditional artificial immune systems only provide this binary classification of traffic or data as "normal" or "attack". For many problem domains, particularly the problem domain of malicious web server requests, this simple classification is not sufficient. There are a variety of web server attacks ranging from simple information gathering via HEAD or OPTIONS requests to attacks that attempt to execute code on the web server. The administrative response to an attack will vary based on the type of attack. The prototype system presented in this work overcomes this limitation by adding classifications to a traditional AIS.

Since WCIS classifies the attacks as they are detected, this provides the web administrator with more information about the nature of the attack than a simple alert would provide. For example, an attack which has a directory traversal component would require different configuration changes than a CGI or PHP script with a buffer overflow. By providing classifications along with alerts, WCIS can help direct the administrative response to a zero-day attack more effectively. The administrators might not know the name of the attack, but if they know it's a buffer overflow on index.ida, that will allow them to focus their response far more than they could with an "attack detected" alert provided by traditional artificial

immune systems.

Another limitation of traditional artificial immune systems is the training of the immune system "antibodies", e.g. the sensors for detecting attacks. The traditional AIS model assumes a continual process of evolution occurring in real-time as it sees and classifies network traffic. Most evolutionary algorithms require extensive memory and CPU cycles to operate. This leads to two main issues using AISes when high-volume, real-time detection is desired: the sensors take a long time to train, during which they are not capable of accurately labeling traffic, and sensor refinement after initial training can cause a CPU and/or memory bottleneck that limits the volume of traffic that the sensors can process.

WCIS seeks to minimize these issues by separating the evolutionary processes from the detection process. The evolutionary processes, pre-deployment training and sensor refinement, occur "offline" on a back-end system. The detection process, monitoring the network traffic, occurs "online" in real-time on the network. The "offline" evolutionary processes produce a set of sensors, which essentially detect patterns in the traffic, that are deployed to monitor the network traffic in real-time. It should be noted that the "online" mode of WCIS is intended to work in conjunction with a traditional IDS by scanning the traffic which the traditional IDS has not alerted upon. WCIS however does not produce traditional IDS rules as those rules would be unable to gather the statistics at the sensor, classification population and overall population levels that are needed for sensor refinement.

In order to maintain one highly desirable feature of an AIS, the customization of the sensors for that particular network's traffic, WCIS uses a system profile to train the sensors in the pre-deployment phase. These profiles include a sampling of normal traffic for the network which will be used to train the AIS and a set of labeled attacks that will be used to "prime" the classifier. The prototype implementation of WCIS takes Apache logs as the source of these two datasets, which makes customization of the datasets very easy. One simply has to copy log entries over into the appropriate dataset file and rerun the pre-deployment phase of WCIS.

To enable "offline" sensor refinement, the "online" WCIS sensors record statistics about their detection and classification rates at the individual sensor, classification population and overall sensor population levels. This information can be sent to a back-end system, which will enable WCIS to run the sensor refinement process as a batch process on the back-end system while the live sensors keep detecting. Once the batch process is complete, the live sensors can be replaced with the newly refined ("next generation") sensors. The current prototype does not yet implement this aspect as the prototype could not be run on live network traffic due to policies and bureaucratic limitations about collecting data that may contain personal or confidential information at the university. However, it is already supported by the internal structure of the sensors and merely requires a live network (or isolated network) test environment to implement and fully test this feature.

In summary, WCIS is a variation of an artificial immune system that is intended to work in conjunction with a traditional intrusion detection system to detect attacks that the IDS cannot yet detect. WCIS seeks to overcome the usability limitations of traditional artificial immune systems by adding a classifier to provide more information about detected attacks. Additionally, WCIS seeks to optimize the scalability of the AIS concept by separating the evolutionary processes from the detection process. This allows the resource intensive aspects of an AIS to occur "offline" on a back-end system rather than on the detection system.

Section 2 provides an overview of artificial immune systems and the biological principles that inspired them. Related work in the area of artificial immune systems and classifiers is presented in Section 3. The methodology used to add classifications to an artificial immune system is described in Section 4. Section 5 describes how WCIS models web server requests. The results of running WCIS on sample datasets is presented in Section 6 and conclusions drawn from these results are given in Section 7. Finally, future avenues of research and improvement for WCIS are discussed in Section 8.

2 Artificial Immune Systems

An artificial immune system (AIS) is a type of anomaly-based intrusion detection system (IDS) inspired by the adaptive nature of the biological immune system. A biological immune system has to be responsive to new and unknown pathogens while also recalling previously defeated pathogens to prevent a recurrence of illness. While not 100% effective at this task (e.g. auto-immune disorders and other immune system malfunctions), the biological immune system is more adaptive to new pathogens and variants of known pathogens than the analogous anomaly-based IDSes.

Using biological methods to create a better IDS is the core concept behind artificial immune systems. The goal of an AIS is to distinguish normal traffic (called "self" data) from abnormal traffic (called "non-self" data). It does so by creating immune "sensors" as analogs to biological immune cells. These sensors use pattern matching functions to determine if data is "non-self". Several key features of a biological immune system that serve as inspiration for an AIS are affinity maturation, negative selection and peripheral tolerance. Other features of bi-

ological immune systems can also be incorporated, but these are far more weighty concepts that are beyond the scope of this paper.

Affinity relates to pattern matching. Each immune cell (antibody) has a set of proteins on its surface that form a three dimensional "lock" pattern which can match the "key" pattern of proteins on the surface of a pathogen (also called an antigen). Affinity measures how "tightly" the lock and key patterns fit together, with a higher affinity meaning a tighter bond between the antibody and pathogen exists. Affinity maturation is the process of refining an antibody's lock pattern until it can tightly bind to a specific pathogen. This allows the body to "memorize" specific pathogen patterns, e.g. learn a "signature" for that pathogen. This is the basis of immunizations in biological immune systems. For AISes, affinity maturation allows generic immune system sensors to develop "signatures" for novel attacks or new variants of old attacks. This is accomplished by training the sensors against attack data in the pre-deployment phase and by refining the sensors during deployment using an evolutionary technique, such as a genetic algorithm.

Negative selection is a process for creating new immune cells that do not react to the body's own proteins ("self"). Most of the artificial immune system works focus on this feature of biological immune systems. The immune cells are initially created with a random pattern of "lock" proteins. The cells are then tested against a random sampling of "self" proteins and structures. If the immune cell has too high of an affinity for "self", it is destroyed. For AISes, this means the immune sensors are initialized with random patterns and each sensor is tested against a sample of "normal" data. Those which react too strongly to normal data are removed and replaced with a new randomly generated and tested sensor. A negative selection phase can be used along with affinity maturation to be sure that the sensors do not start reacting to normal data while they are developing an affinity for attack data.

Since negative selection uses a random sampling of "self" proteins to test new immune cells, there is a possibility that cells which are reactive to self will survive negative selection. Auto-immune disorders are caused by such cells. In an AIS, such sensors would lead to false positives, where normal traffic is labeled as an attack. The immune system has some protection against this by using peripheral tolerance. Peripheral tolerance deactivates or destroys immune cells that are too reactive to self proteins. Not many AISes explore the use of peripheral tolerance in their systems since it is hard to detect false positives automatically. One technique might be to have a human verify each alert and deactivate any sensor which is noted to have an excessive number of false positives. In WCIS, the person can also modify the sensor's internal statistics to mark the sensor as "bad", which will prevent the sensor from being used to refine the sensors during the next sensor refinement phase. This essentially removes the sensor from the "genetic pool" used for sensor refinement.

3 Related Work

The research group of Stephanie Forrest at the University of New Mexico has produced several pioneering works in the field of artificial immune systems. Forrest, *et al.* [9] focused on distinguishing self from non-self and laid the foundations for the negative selection algorithm. Somayaji, Hofmeyer and Forrest [16] explored the application of these concepts to computer security. This work ultimately resulted in the production of the LYSIS [12, 13] immune system for TCP connections. LYSIS monitored the TCP/IP headers of SYN packets to detect abnormal traffic.

Williams, *et al.* [22] expanded LYSIS to monitor TCP, UDP and ICMP traffic. This system, called CDIS, also monitored all packets instead of just TCP SYN packets. Each AIS sensor in the system monitored a random subset of features from the packet headers. The pattern matching function used by CDIS used a mix of binary, discrete and real value features.

Gonzales, Dasgupta and Gomez [10] showed that the negative selection algorithm is very sensitive to the type of matching function used. Ultimately, one hopes that negative selection results in sensors with a wide coverage of the non-self space, as this represents potential attacks. But [10] showed that the algorithms of Forrest, *et al.* [1, 9, 12, 13, 16] and Farmer, *et al.* [8] resulted in restricted coverage of the non-self space. These algorithms work best with binary and discrete data. Of the algorithms tested, the real value matching function used by Gonzales, Dasgupta and Kozma [11] had the best coverage of the non-self space.

In Dasgupta, Yu and Majumdar [5], a multilevel immune learning algorithm was introduced, in part to overcome deficiencies in simple negative selection algorithms. This system used collaborations and interactions between various types of sensors, analogous to the various types of immune system cells in a biological immune system. By requiring collaborations between sensors to label data as non-self, the experiments showed the AIS achieved better results than simple negative selection.

The first version of WCIS that was published [2, 4] was built off the work of CDIS and LYSIS to monitor web server requests. As with CDIS, the sensors monitored a random subset of features and the pattern matching function used a mix of binary, discrete and real value features. The system also incorporated basic collaboration between sensors to reduce the false positive rate.

Table 1: The classification scheme used for the web server attacks.

Class	Instances	Description
info	5	Gathers information about server (read only)
traversal	37	Directory traversal attempt (read only)
sql	4	SQL injection attack
buffer	7	Buffer overflow attack
script	86	Cause a script to do something malicious (execute)
xss	40	Cross site scripting

This first version of WCIS was simply an AIS for web server requests and included none of the enhancements that this work covers. The concept of adding a classifier to WCIS was explored in [3], but the classifier used in that version was prone to overfitting and poor classifications and was deemed unsuitable. Neither previous version separated the evolutionary processes from the detection process.

Watkins, Timmis and Boggess [17, 18, 19, 20, 21] proposed an artificial immune recognition system for for supervised learning and reinforcement learning. The proposed AIS functioned as a classifier. As with [5], it modeled a variety of immune cells working in collaboration to classify data. It required the features be represented as a vector of real value ranges and used vector mathematics to calculate affinity and distance between cells. A variation on k-nearest neighbors was used to calculate the class of unknown data once the cells had been trained. While this method worked well on datasets that can be modeled as a feature vector, its mathematical approach limits its application to other feature sets that cannot be easily modeled as a vector.

4 Methodology for the Classifying AIS

Previously [2, 3, 4], WCIS was defined as an AIS for web server attacks and a rudimentary, but poor, classifier was implemented. The scheme for fingerprinting web server requests, detailed in Section 5, was developed in those works. The classifier developed in [3] was prone to overfitting and misclassification. A better classifier was developed, which is the focus of this section. The simple classification scheme given in Table 1 was preserved from [3] however, as the classification scheme was not the issue with the previous classifier.

The classification scheme in Table 1 was developed based on several common groups of web server request attacks that can be found encoded in URIs. The "info" classification covers various information gathering attacks that do not alter the server. Likewise, the "traversal" category solely covers the attacks which utilize directory traversal, but do not attempt to execute anything on the web server, such as attempting to read /etc/passwd. If the traversal tries to execute a program, it is instead labeled "script". The "script" class also covers other attempts to maliciously execute a program or script on the web server. The "sql" class covers SQL injection attacks. The "buffer" class covers buffer overflow attacks, which may also result in commands being executed. Finally, the "xss" class covers cross site scripting attacks. Table 2 lists some examples for each class except buffer overflow attacks as those examples were too long to easily fit into the table.

The classification training occurs during the pre-deployment stage where the field of potential sensors is trained against a system profile. The system profile consists of a normal dataset (Apache log entries from non-malicious web requests) and an attack dataset (Apache log entries from actual attacks on a web server). Each attack in the attack dataset was hand inspected and labeled with a classification. One main issue faced while developing the attack dataset was obtaining sufficient examples of each classification of attack. Attack examples were gleaned from Bugtraq [15], live Apache web servers and an un-networked machine where selected attacks were run against a local web server. As seen from Table 1, most of the examples fell into the category of traversal, script or xss. To prevent the sensors from becoming biased towards those classes, each sensor tracks the percentage of the class that it is able to detect rather than a raw count.

To add classification to WCIS, each sensor not only tracks the percentage of each category it reacted to during pre-deployment training, it also has a desired category for which it should develop affinity. Previously in [3], WCIS did not have this second feature and it was discovered that the population of sensors optimized for the "script" and "traversal" classes. To prevent this from happening, the sensors were divided into groups and each group was tasked with optimizing affinity for a particular classification. This is a niching algorithm, which is intended to develop "specialists" for all classification labels.

To optimize affinity, the sensors must be trained and matured. This is accomplished with a typical artificial immune system lifecycle conducted during the pre-deployment and sensor refinement phases. The lifecycle is an iterative process which repeatedly applies the affinity maturation steps. This results in a set of trained sensors that have higher affinity towards attacks than the initial sensors. The steps for the lifecycle are detailed in the following subsections.

The primary difference between the pre-deployment

Table 2: A sample of requests in the attack dataset.

Class	URL
info	GET x HTTP/1.0
traversal	GET ../../../boot.ini HTTP/1.0
traversal	GET %2E%2E/%2E%2E/%2E%2E/%2E%2E/%2E%2E/winnt/win.ini HTTP/1.0
sql	GET /scripts/test.asp?var=foo';EXEC master.dbo.xp_cmdshell'cmd.exe' HTTP/1.0
script	GET /scripts/..%%35%63../winnt/system32/cmd.exe?/c+cmd.exe HTTP/1.0
script	GET /ans.pl?p=../../../../bin/command%20argument|&blah HTTP /1.1
xss	GET /<script>alert('Vulnerable')</script> HTTP/1.1
xss	GET /javascript:void%20window.open(HTTP/1.0

Table 3: A sample of requests in the normal dataset.

GET /00master/hqafgate.gif HTTP/1.0
GET /Copy%20of%2010.gif HTTP/1.0
GET /faq/web/viewfaq.php3 HTTP/1.0
GET /forums/newmsg.php?fid=2&pid=30 HTTP/1.1
GET /index.html?browsePage=commands.html HTTP/1.1
GET /index.html?browsePage=kb/item_detail.php&id=19 HTTP/1.1
GET /index.html?secure=1&PHPSESSID=db80c486ee8cef8090a532b93619cd7a HTTP/1.1
GET /%7E930www/Images/front_y2k_logo02.jpg HTTP/1.0
GET /ADTracker.asp?linkid=AHCX030&linktype=Room&RID=8 HTTP/1.0
GET /CGI-BIN/centralad/getimage.exe/19980714243?GROUP=default_buttons HTTP/1.0

and sensor refinement phases is the source of the statistics used for training. In the pre-deployment phase, training statistics come purely from the sensor's reaction to the system profile datasets. In the sensor refinement phase, statistics come from the sensor's reaction to live traffic, with negative selection against the normal dataset also conducted to prevent sensors from reacting to normal traffic.

Before going into the details of the pre-deployment phase, some key terminology should be reviewed. The sensor **population size** is the number of unique sensors being processed. Each individual sensor within the population has its own data structure to store its pattern, classification label and statistics. Patterns may be repeated in multiple individual sensors within the population. This is called a loss of diversity or **overfitting** which essentially leads to redundancy (e.g. multiple sensors have the same "signature"). The sensor **lifecycle** is the process of creating, refining and perhaps destroying individual sensors within the population. Throughout the lifecycle, the population size remains constant. Every destroyed sensor is replaced with exactly one sensor. The sensors that exist in each iteration through the lifecycle process are called a **generation** of the population. Each new generation is generated by the affinity maturation process, which uses a genetic algorithm to refine the sensor population *as a whole*. The sensor's **chromosome** is a method to represent the sensor's pattern by using data structures that can be manipulated by a genetic algorithm. The chromosome contains all possible features that a pattern in WCIS may use (see Section 5 for a description of the features), the current values for each feature and a flag to indicate if the sensor is using that feature in its pattern (e.g. if the feature is **expressed** in that particular sensor). The **fitness** of a sensor is determined by its statistics and is used to gauge its accuracy at detecting attacks in its classification label. The most fit sensors contribute more "genetic information" to the next generation than the less fit sensors.

4.1 Lifecycle

In pre-deployment training, a normal dataset, samples of which can be seen in Table 3, and the labeled attack dataset are given as input to the lifecycle function. The pre-deployment lifecycle begins by randomly generating a population of sensors for each classification group. The random generation process selects a subset of features for each sensor's matching pattern and randomly assigns values to those features. In the sensor refinement phase, the lifecycle function would instead begin with copies of the existing sensors and any sensors which have been deactivated by the system administrator (peripheral tolerance) will be discarded and replaced by a random sensor.

For both phases, the iterative affinity maturation process is then entered, which refines the sensors over a series of generations. It is important to note that affinity maturation occurs within each population for a classi-

fication label, not across all classification label populations. The goal of affinity maturation is to produce sensors which specialize in detecting attacks for that particular classification label, so each population is kept distinct.

4.2 Negative Selection Phase

The affinity maturation process begins with negative selection. The population of sensors is compared to the normal dataset. Any sensor that has too strong of an affinity to requests in the normal dataset is discarded and replaced with a random sensor. The replacement is likewise tested against the normal dataset and is not allowed to replace the discarded sensor until its random feature set (e.g. pattern) does not have strong affinity towards the normal dataset. The exact level of affinity towards the normal dataset that is tolerated in this phase is tunable in WCIS.

4.3 Training Phase

After negative selection, the sensors enter two phases of training. During the first phase of training, the sensors are compared to all of the attack requests and a random subset of normal requests. If a sensor has affinity to an attack, it records the classification of that attack. At the end of the first phase, each sensor will know the percentage of attacks in each category it can detect. It then sees which classification it is best at detecting and marks that classification as its class. The sensor may mark itself as a different classification than what its group is supposed to be optimizing for. This simply means the sensor is not as good at detecting the desired classification as it is at detecting a different classification.

During the second phase of training, the sensors make a second pass over the attack dataset. For each attack, the sensors which can detect it vote on the classification of the attack. The accuracy of each group of sensors at detecting its desired classification is recorded. This second phase is purely for computing the accuracy statistics and does not affect the affinity maturation process. The accuracy of the sensors during experimental testing is given in Section 6.

4.4 Genetic Algorithm Phase

After training, the sensors move on to the genetic algorithm phase. This phase first "breeds" the sensors to create the next generation of sensors and then mutates the next generation. The breeding phase uses a single-objective genetic algorithm which optimizes for a single fitness metric (multi-objective algorithms allow optimization for multiple fitness metrics). The fitness of each sensor for this phase is its ability to classify the attacks in the desired classification for its population. For example, if a "script" sensor can detect 70 of the 86 script attacks, it would have a fitness of 0.814 even if it could also detect 100% of the "traversal" attacks. A secondary fitness value is also computed for each sensor but is not directly used by the genetic algorithm. This fitness value measures how well the sensor can detect attacks without excessive false positives. The secondary fitness ranges in value from -2 (all of its alerts are on normal requests instead of attack requests) to +2 (all of its alerts are attack requests).

Rank selection with elitism using the primary fitness value is used to select the "parent" sensors. Rank selection chooses the most fit sensors to be parent sensors. Elitism allows a percentage of highly fit parent sensors to survive into the next generation. The exact percentage is tunable in WCIS. Once two parent sensors are selected, single point crossover on the parents' chromosomes is used to create the chromosomes for the "children" sensors. The chromosome is the complete feature set, a subset of which will be expressed in each parent. The expressed feature set for each child sensor is the intersection of the expressed feature sets of the parent sensors. Additionally, a feature that only one parent expresses will be randomly expressed in the child. Even if the feature is not expressed, the child will still inherit the values for that feature from the parent. It just will not be used by the child to match against requests. But this preserves the genetic information in a dormant state in case future offspring randomly choose to express that feature. Finally, if a child exits this expressed feature selection phase with less than two features expressed, it randomly chooses features to add to its expressed feature set until the set size is two.

Besides the children sensors created by crossover, randomly selected parent sensors are also be chosen as survivors during the elitism process. The population for the next generation of the affinity maturation process is the combination of the children and the survivors. Additionally, to prevent overfitting, breeding ceases when the population for a specific class achieves 100% accuracy at detecting that class. In that case, the next generation consists entirely of survivors.

After breeding is completed, mutation is performed on the next generation. A subset of sensors is selected randomly from the population. A random expressed feature in the sensor's chromosome is selected for mutation. If the feature is binary or discrete, a bit is flipped. If the feature is a real value, the value is altered by a random number.

4.5 Sensor Deployment and Refinement

The lifecycle continues by iterating through the negative selection, training and genetic algorithm phases until a maximum number of generations is reached. At this point, the sensors are considered trained (or refined), although they may not have perfect accuracy for their classification. In the pre-deployment phase, the sensors with a secondary fitness greater than 0.5 will become the live sensors. In the sensor refinement phase, those sensors would replace the existing live sensors, as the "next generation" of sensors. The threshold of secondary fitness may be refined to trade off between covering potential attacks and generating too many false positives.

Live deployment of the sensors could not be tested due to bureaucratic issues obtaining the appropriate authorization for live monitoring of the department network. Since live network traffic could contain personally identifying or confidential information, the campus requires assurance that WCIS will protect such information from unauthorized view before granting authorization. As of this time, the authorization is still pending.

Since this bureaucratic restriction prevented the live deployment of sensors to test the concept, the sensors are instead presented with unlabeled data to see how they perform in a real-world scenario. Any sensor with a secondary fitness less than the above threshold is not used for this phase as it has difficulty distinguishing normal requests from attack requests. The sensors determine if each unlabeled request is an attack or a normal request. If the request is labeled an attack by a sensor, the classification of the sensor is recorded. After passing the unlabeled request past all sensors, the classification with the highest "vote" count is chosen as the class label for the request. Those results are then hand-verified to see their accuracy. The results of testing the sensors against unknown data are given in Section 6.

The bureaucratic restriction also made it difficult to fully test the scalability of the pre-deployment, detection and sensor refinement phases. In particular, this made it difficult to fully implement the back-end processing aspects of the sensor refinement phase, as there were no deployment and back-end systems to communicate between. While WCIS contains the algorithmic components of sensor refinement, the practical aspects of deploying sensors, recording statistics, communicating those statistics back to the back-end system, refining sensors on the back-end system and re-deploying the next generation of sensors could not be fully investigated.

The department is currently in the process of building an isolated network. The sensors can be deployed on the isolated network since the data will be simulated, which means campus authorization is not required. This will allow testing of the sensor refinement phase. Scalability testing can also be conducted. Based on the promising results presented in Section 6, it is expected that WCIS will perform well in a simulated live environment. While this is still not an ideal scenario, it will allow continued development and testing of WCIS while the attempts to get campus authorization for live deployment continue.

Table 4: The special characters used in the fingerprinting method.

Character	Description
%	Used by various encoding methods such as hex encoding
,	Used by SQL injection attacks
+	Interpreted by Microsoft IIS as a space
..	Used in directory traversal attacks
\	Used in directory traversal attacks since URIs contain only /
(	Used in cross site scripting attacks
)	Used in cross site scripting attacks
<	Used in cross site scripting attacks
>	Used in cross site scripting attacks
//	Used in proxy attempts or to exploit an old Apache vulnerability

5 Fingerprinting URIs

In order to adapt the AIS method to detect malicious web server requests in WCIS, the web request data must be converted into a pattern consisting of binary, discrete and real value features. The chromosome in each sensors would then seek to match these features. The features from the web request chosen for WCIS are the Uniform Resource Identifier (URI), the HTTP command (GET, POST, HEAD, etc) and the HTTP version. Additional features from the request, such as headers, referrer, and so on, could also be added as features, although they are not supported at this time in WCIS due to the nature of the Apache logs available for data processing. Due to the restrictions imposed by the campus, WCIS has had to run off of Apache logs rather than the live network and the logs are not always configured to log these features. Additionally, WCIS does not look at the IP address of the client or the return code as it is not concerned with detecting the activities of unique clients or whether an attack failed or succeeded. It is concerned with discovering patterns that indicate a zero-day attack has been attempted.

The HTTP command is converted into a discrete bitmap where each set bit refers to a specific command. For example, bit 0 is set for GET, bit 1 is set for POST and so on. The HTTP protocol is likewise converted into

a discrete bitmap, although it could also be modeled as a real value. The length of the URI is converted into a real value feature. Likewise, the number of variables in the URI is also converted into a real value feature. The URI is then parsed to develop a fingerprint of special characters used in the URI. Table 4 summarizes the special characters modeled in the fingerprint. These characters were chosen based on the whitepapers published online at cgisecurity.com [23, 24] and based on the inspection of later web server attacks. Each special character or character sequence listed in Table 4 is modeled as a real value feature.

Real value features are all modeled as a pair of values: [base, offset]. The sensor will match a URI if the URI value is within the range of base to base+offset. When mutating a real value feature, a random value may be added or subtracted from the base, the offset or both. The base can only be altered by a value from -2 to +2. The offset can only be altered by a value of -4 to +4. This prevents mutation from wildly changing the range that a feature detects.

A sensor is considered to match a web request when all of its expressed features matches the features in the web request. For binary features, the feature matches when the corresponding bit to the feature is set in the sensor. For real value features, a feature matches when its value falls within the range of values in the sensor. Note that the web request may contain additional features that the sensor does not check. The matching is driven by the feature set that the sensor expresses, not the feature set in the request.

6 Experimental Results

WCIS was tested using an attack dataset, a normal dataset and an unknown dataset, as described in Section 4. The attack dataset consists of 179 labeled attacks gathered from Bugtraq, live web server logs and tests run on an un-networked machine. The normal dataset consists of 52977 regular requests gathered from the Lincoln Laboratory DARPA dataset [14] and live web server logs.

Obviously, the preferred method of testing WCIS would have been actual live requests to a web server, as this would best approximation of the real-world performance of WCIS. Unfortunately, as described in previous sections, the regulations at this university have made it difficult to do such testing on live web servers due to privacy concerns. Instead, the Apache `access.log` repository for the Computer Science department web server was used for the unknown dataset. 11659 random requests were pulled from the logs and placed into the unknown dataset.

Besides the datasets, WCIS has many parameters that tune its performance. These parameters are:

`pop` The population size for each classification category. A larger population size creates a larger pool of initial random sensors and thus a greater likelihood of randomly creating a "good" sensor.

`gen` The maximum number of generations for the affinity maturation process. The higher this value is, the more likely it is that affinity maturation can derive "good" sensors even if the random initial sensors are only mediocre.

`xover` The percentage of the next generation that comes from breeding. The remaining percentage of the next generation will be survivors.

`mut` The mutation rate for the next generation. A higher value introduces more random change in each generation, which can be beneficial, harmful or benign.

`thresh` The threshold for affinity when doing negative selection. Sensors with affinity above this threshold are destroyed.

`agree` The number of sensors that must agree a request is an attack before it is labeled as an attack. For classification, `2 * agree` must label an unknown data as an attack before it will be classified.

WCIS was tested with population sizes of 25, 50 and 75 for each classification category. Each sensor in a population is analogous to a rule in an IDS in that it looks for a specific pattern in the web request. Note that the actual total number of sensors tested in each tested run of WCIS was `pop*number_of_classifications`. References to "population size" in this section refers to the number of sensors for each classification category (`pop`), not the total number of sensors tested (`pop*number_of_classifications`).

The maximum number of generations tested were 10, 20, 30, 40 and 50. The mutation rates tested were 1%, 2.5%, 5% and 10%. The value for `xover` was 0.6, the value for `threshold` was 0.0002 and the value for `agree` was 3, as prior testing has shown these values yield good results.

6.1 Runtime

One of the first concerns with any method that uses evolutionary computation, such as genetic algorithms, is how long it takes the algorithm to complete. This is one of the motivations behind separating the operation of WCIS into phases: pre-deployment, detection and sensor refinement. Only the pre-deployment and sensor refinement phases will need to run the genetic algorithm.

To test the runtime for the pre-deployment phase, WCIS was tested on a Xeon E5410 2.33GHz machine

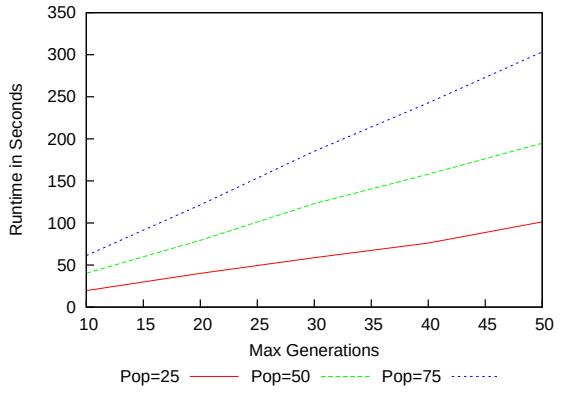

Figure 1: Average runtime for the pre-deployment phase of WCIS for the three tested population sizes for each classification label. Note that the actual total number of sensors is $pop * 6$ since there were 6 classification labels tested. This is purely the pre-deployment phase runtime, not the detection or sensor refinement phase runtime. The detection phase runtime was 0.23 to 0.61 seconds.

with 4GB of RAM. The pre-deployment phase was coded as a single-threaded process. The population for each classification label was processed in a series, using round-robin scheduling (e.g. it processed the first generation of the info class, then the first generation of the traversal class, and so on). With sufficient memory to hold multiple copies of the normal and attack datasets, WCIS could easily be changed to a multi-threaded program with a thread for each population, which would lead to a substantial decrease in the runtime and increase in scalability. The current prototype was also coded in C++, which could be changed to a more efficient programming language in future versions to provide additional scalability.

These changes were not made because the point of this test was to run the genetic algorithm under less than ideal conditions to illuminate choke-points in the underlying algorithms. These choke-points might not be apparent if the code runs too quickly for any differences between input parameters to become significant. This might leave inefficient areas in the underlying algorithms that could affect future scalability. Additionally, if the runtime for WCIS is reasonable under these less than ideal, and easily remedied, coding conditions, then we can be reasonably assured that there are not choke-points in the underlying algorithms.

As shown in Figure 1, even with the largest population sizes and number of generations, WCIS trained the sensors in the pre-deployment phase in under six minutes. This is very reasonable for an evolutionary algorithm, so it is unlikely that there are hidden scalability issues in the underlying algorithms. Converting WCIS to a multi-threaded program in a more efficient programming language should yield even faster results. The sensor refinement phase is expected to have a similar runtime as it needs to run through a similar lifecycle. These results also emphasize why it is important to separate off the evolutionary phases as back-end processes on a separate system from the deployment system. It would be unacceptable to wait 6 minutes for the sensors to refine themselves on a live system, but the separation allows the deployed sensors to continue monitoring live traffic while the back-end system refines the sensors.

While it was not possible at this time to test the detection phase with live data due to the previously described issues, presenting WCIS with the 11659 unknown requests to emulate the detection phase took from 0.23 to 0.61 additional seconds on average, including the extra I/O time to load the unknown dataset from disk, log classifications and log the classification statistics that are presented in the remaining results. There seemed to be little correlation between population size and the additional time required for WCIS to test the unknown requests. For example, the population size of 50 had the lowest average time, while the population size of 25 had the highest average time. This suggests most of variance in the time to test the unknown dataset was due to I/O latency, particularly since the test system had only a consumer-grade SATA drive.

More testing will need to be done to determine the realistic traffic rates that WCIS can handle during the detection phase. These can be conducted once the department's isolated network is completed.

6.2 Accuracy at Classification

Since the primary fitness function was the accuracy at classifying the attack dataset, let us look at the best accuracy for each population in the test runs. Five separate populations for each classification label were tested for each possible combination of variables. The best performing population for each classification and combination was examined.

The best performing populations when the population size was 25 had a maximum number of generations of 40 and a mutation rate of 1%, as shown in Figure 2. The small population size means that WCIS starts with less random diversity. This means the affinity of the initial sensors might be quite low for their desired classification, whereas with a larger population there is a higher chance of randomly generating an antibody with moderate to strong affinity for the class. Because of this low affinity in early generations, the small population size needs more generations for affinity maturation. In par-

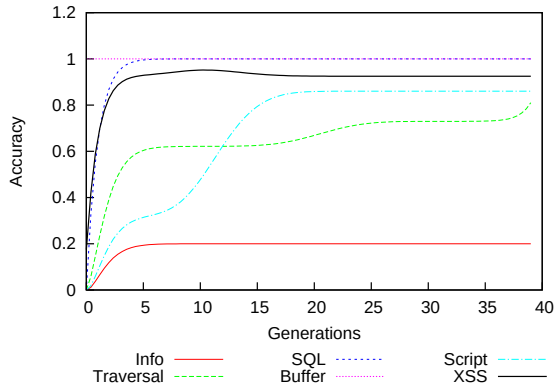

Figure 2: Detection accuracy for each class when the population size for each classification is 25, the maximum generations is 40 and the mutation rate is 1%. This was the best performing population when the population size for each classification was 25.

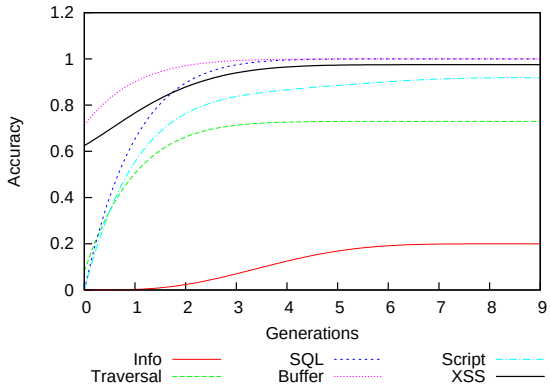

Figure 3: Detection accuracy for each class when the population size for each classification is 50, the maximum generations is 10 and the mutation rate is 2.5%. This was the best performing population when the population size for each classification was 50.

ticular, Figure 2 shows that the "script" and "traversal" classes took the longest number of generations to plateau in accuracy. However, increasing the maximum generation to 50 actually led to overfitting, where the fitness started to decrease in the final generations. This population size also needed the lowest mutation rate of the best tested population sizes. While mutation can help increase the likelihood that the appropriate feature(s) for that classification are affected in a beneficial way, there is also the possibility that mutation might negatively affect the accuracy. A small population is less able to recover from a negative mutation than a large population.

The best performing populations when the population size was 50 had a maximum number of generations of 10 and a mutation rate of 2.5%, as shown in Figure 3. Since WCIS starts off with a larger random population, it is better able to withstand negative mutations and a higher mutation rate can also increase the likelihood of beneficial mutations. This population size also does not need as many generations to achieve good accuracy at classification since it starts with a larger random pool of sensors and there is a greater likelihood of a good sensor being randomly generated in the initial generation. As with a population size of 25, too many generations led to overfitting and a decrease in accuracy, as shown in Figure 4 where the maximum number of generations is 30.

The best performing populations when the population size was 75 had a maximum number of generations of 20 and a mutation rate of 5%, as shown in Figure 5. While most of the classification accuracies plateaued in early generations, the slightly higher rate of mutation allowed for the "info" and "traversal" classifications to randomly find the right combination of features to increase accuracy in later generations. As noted with the other population sizes, a higher number of maximum generations led to overfitting.

Several trends were noticed across all combinations of variables tested. First, regardless of population size, maximum generations and mutation rates, the populations had great difficulty correctly identifying the "info" class of attacks, as shown in Figures 2 through 4. This is not surprising as the "info" class is the hardest to distinguish from normal data. Information gathering attacks are also hard to distinguish from innocent mistakes, such as a typo in the URI.

Second, as noted above, overfitting and loss of accuracy is seen in all tested combinations of variables when the number of generations is high. This is a general problem in single-objective, single-crossover genetic algorithms. This is caused by a loss of diversity within the population. In essence, the sensors become too specialized for specific attack instances and lose the ability to detect more generalized attacks or attacks which lay on the peripheral of the non-self space. It may be the case that another genetic algorithm would be better suited to this problem domain. For example, a multi-objective genetic algorithm, such as NSGA-II [6, 7], is designed to maintain diversity by balancing multiple fitness objectives.

Overall however, the classification scheme employed by WCIS achieves a high rate of accuracy, particularly in the classifications with a large set of attack instances such as "traversal", "script" and "xss". While no population was able to obtain 100% accuracy in those categories, this may be due to the diversity issue. Even so, the accuracy for "traversal" was 81% in many popula-

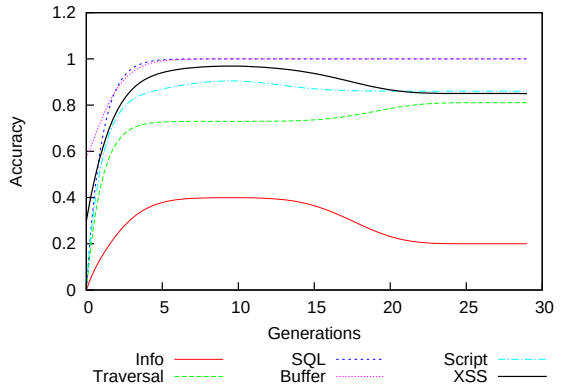

Figure 4: Detection accuracy for each class when the population size for each classification is 50, the maximum generations is 30 and the mutation rate is 2.5%. Note that the extra generations do not yield better results than Figure 3. In fact, overfitting occurs within several classification populations.

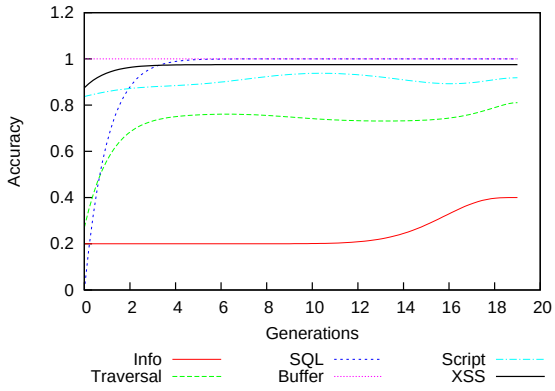

Figure 5: Detection accuracy for each class when the population size for each classification is 75, the maximum generations is 20 and the mutation rate is 5%. This combination of variables had the best classification accuracies of all tested parameters.

tions, the accuracy for "script" was 92% in many populations and the accuracy for "xss" was 92 – 97% in many populations.

6.3 Labeling Unknown Data

After inspecting the accuracy rates, next let us look at how well WCIS could label unknown data gleaned from Apache access logs. The access logs for the Computer Science web server are rotated on a monthly basis, with data going back for several years. Random entries were selected out of two months of access logs. This created an unknown dataset with 11659 entries in it.

After each population finished affinity maturation, it was presented this dataset to label. This emulated a live scan of web traffic. While this test was sufficient to evaluate the effectiveness of WCIS at detecting zero-day attacks, it does not provide metrics for the scalability of WCIS. That would require live testing on networks with various traffic capacities. Unfortunately, due to the previously described challenges with conducting this research in our campus environment, that was not possible at this time. So this test purely focuses on gauging WCIS's ability to detect zero-day attacks and attack variants and its false positive rate when given a large dataset of unlabeled web requests.

It quickly became apparent when looking at the alerts that WCIS raised that someone had tried to attack the web server repeatedly during the time frame covered by the Apache logs. Table 5 shows a subset of the attacks detected by the best population of size 25. Table 6 shows a subset of the attacks detected by the best population of size 50. Finally, Table 7 shows a subset of the attacks detected by the best performing population of size 75.

As seen in the sample of detected attacks, someone attempted to access the /proc/self/environ file, which contains a list of environmental variables, using various directory traversal attempts. This particular attack is actually associated with getting a shell on poorly configured web servers using a combination of directory traversals and shellcode or shell commands injected via the User-Agent field. Even though WCIS does not include the User-Agent field in its feature set, and thus didn't see the actual shellcode that was attempted, it still detected this attack as it appeared in the unknown dataset.

WCIS did have difficulty deciding whether this attack was a "traversal" or a "script" attack since this attack uses directory traversals to access /proc/self/environ to execute code. The attack dataset does classify such attacks as "script" even though they contain features of a "traversal". As pointed out in Table 1, only the attacks which were read-only (such as retrieving the password file) were labeled as "traversal" in the attack dataset. If the directory traversal resulted in an attempt to execute code, it was labeled as a "script" in the attack dataset. Thus, it is not unexpected to see that WCIS has difficulty determining if these attacks were a "script" or a "traversal" since its feature set does not, as of now, include the portion of the attack (the User-Agent field) that would have made it clear it was a "script" attack.

Additionally, looking at the voting data, the attacks labeled as "traversal" also had votes for "script", with many cases having only a difference of one or two votes between the two labels. So the "script" population was detecting these attacks, just not quite as vigorously as the

Table 5: A sample of unknown requests detected as attacks in the unknown dataset for the population in Figure 2.

Class	URL
script	GET /*.php?option=com_dump&controller=..//..//..//..//..//..//..//..///proc/self/environ%0000 HTTP/1.1
traversal	GET /.php?index=../../../../../../../../../../../../../../proc/self/environ%00 HTTP/1.1
traversal	GET /courses/ls290//index.php?p=../../../../../../../../../../../../../..///proc/self/environ%0000 HTTP/1.1

Table 6: A sample of unknown requests detected as attacks in the unknown dataset for the population in Figure 3.

Class	URL
script	GET /////?option=com_dump&controller=..//proc/self/environ%0000 HTTP/1.1
script	GET /cs150/index.php?p=../../ HTTP/1.1
traversal	GET /*.php?option=com_dump&controller=..//..//..//..//..//..//..//..///proc/self/environ%0000 HTTP/1.1

"traversal" population.

Being able to detect attacks is desirable, but one also wants an IDS to have a low rate of false alarms. WCIS did not falsely alarm on any of the normal requests in the unknown dataset. This may be due to the fact that some of the requests in the normal dataset were also gleaned from the department Apache access logs. However, this is a good result since it shows that WCIS is easily tuned to the normal traffic for a specific website by using a sampling of that normal traffic to generate the normal dataset.

7 Conclusions

This paper presented a method of detecting zero-day attacks on web servers via malicious requests that is based on artificial immune systems. This prototype system, called Web Classifying Immune System (WCIS), is intended to augment the capabilities of an existing intrusion detection system (IDS) by detecting attacks that are not detectable by the existing IDS. WCIS is a modified artificial immune system (AIS) that adds classification. WCIS also seeks to improve the efficiency of an AIS by separating tasks into the pre-deployment phase, detection phase and sensor refinement phase instead of requiring all these tasks to take place within a single AIS lifecycle. This allows the detection phase to focus on low-resource, speedy sensors while the more costly evolutionary computation associated with the other phases occurs on a separate back-end system.

Notably, WCIS is able to achieve a high rate of accuracy at detecting most classes of attacks in the attack dataset, with the exception of the "info" attacks, which are difficult to distinguish from normal requests. When tested against unlabeled data from Apache access logs, WCIS is able to identify attacks within the requests without falsely alerting on normal traffic. WCIS does have some difficulty choosing between the "traversal" and "script" classifications when the "script" attack uses some elements of directory traversal in its URI. This is likely due to the fact that WCIS only models the HTTP method, URI and HTTP protocol. However, even with this limitation, WCIS is able to detect that an attack containing elements of a directory traversal has occurred.

In summary, WCIS is able to achieve a high rate of accuracy at detecting and classifying attacks against web servers without falsely alarming on normal traffic when properly trained on the normal traffic patterns of the network. WCIS can be easily trained on the normal traffic patterns by giving it a sampling of web server logs, such as Apache logs. The ability to classify the attacks is particularly noteworthy as it allows an administrator to rapidly focus on the initial mitigation and response techniques. It might also lead to integration with an automated response engine, although that has not yet been explored for WCIS.

8 Future Work

The next phase of development for WCIS will focus on creating an appropriate test bed. The department has recently secured a Department of Education grant that is funding the expansion of research laboratory space. A portion of this grant is being used to develop an isolated network. This can be used to test WCIS (and other security tools) without concern about running afoul of the campus privacy regulations. This is not a perfect solution, as it will still be a simulated environment instead of a live environment, but it will permit the full testing of the sensor refinement phase, which has been hampered by the campus regulations. This will also allow scalability testing, although the isolated network funding currently limits the test bed to Gigabit Ethernet instead of 10 Gigabit Ethernet, so there will be limitations to testing the scalability to high capacity networks.

Table 7: A sample of unknown requests detected as attacks in the unknown dataset for the population in Figure 5.

Class	URL
script	GET /.../ports_labeled.jpg HTTP/1.1
traversal	GET /index2.php?option=com_dump&controller=..//..//..//..//..//..//..//..///proc/self/environ%0000 HTTP/1.1
traversal	GET /index.php?t=1&p=technical_info/howto//index.php?filename=../../../../../../../../../../../../../../../../proc/self/environ%00 HTTP/1.1
script	GET /faculty/interests/..\\index.html HTTP/1.1

Another area of future development is expanding the feature set used by WCIS sensors. Currently, the feature set of WCIS only models the request line from the request, consisting of the HTTP method, URI and HTTP version. It does not model the general headers, request headers, entity headers or message body specified by Hypertext Transfer Protocol version 1.1 for the HTTP request. This limitation arose because WCIS had to be run on Apache access logs, instead of live data, due to policy restrictions at the institution. The available Apache logs did not consistently record any header fields. However, attackers are using the header fields as a part of their attacks so WCIS should expand its feature set to model these aspects of malicious web server requests. The isolated network test bed should enable the incorporation of these fields into the sensor feature set, since WCIS will no longer be constrained by the formatting of the Apache logs.

Additionally, as noted in Section 6, the genetic algorithm currently being used by WCIS may not be the best algorithm for this problem domain. It suffers from a loss of diversity, which leads to overfitting and a decreased accuracy at detecting and classifying attacks as the generations progress. Another avenue of future research is to explore how other genetic algorithms such as multi-objective genetic algorithms can improve diversity in the sensor population. This diversity will also be useful in detecting novel attacks that do not clearly fall under one of the existing classification categories.

References

[1] BALTHROP, J., ESPONDA, F., FORREST, S., AND GLICKMAN, M. Coverage and generalization in an artificial immune system. In *Proc. Genetic and Evolutionary Computation Conference (GECCO 2002)* (July 2002), pp. 3 – 10.

[2] DANFORTH, M. *Models for Threat Assessment in Networks*. PhD thesis, University of Califonia, Davis, Davis, CA, USA, June 2006.

[3] DANFORTH, M. Towards a Classifying Artificial Immune System for Web Server Attacks. In *Proceedings of the International Conference on Machine Learning a nd Applications (ICMLA 2009)* (Miami, FL, USA, December 2009).

[4] DANFORTH, M., AND LEVITT, K. Immune System Model for Detecting Web Server Attacks. In *Proceedings of the International Conference on Machine Learning and Applications (ICMLA 2003)* (Los Angeles, CA, USA, June 2003), pp. 161 – 167.

[5] DASGUPTA, D., YU, S., AND MAJUMDAR, N. S. MILA - multilevel immune learning algorithm. In *Proc. Genetic and Evoluntionary Computation Conference (GECCO 2003)* (Chicago, IL, USA, July 2003), pp. 183 – 194.

[6] DEB, K., AGRAWAL, S., PRATAB, A., AND MEYARIVAN, T. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. In *Proceedings of the Parallel Problem Solving from Nature VI Conference* (Paris, France, 2000), Springer. Lecture Notes in Computer Science No. 1917, pp. 849–858.

[7] DEB, K., PRATAB, A., AGRAWAL, S., AND MEYARIVAN, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation 6*, 2 (April 2002), 182 – 197.

[8] FARMER, J. D., PACKARD, N. H., AND PERELSON, A. S. The immune system, adaptation, and machine learning. In *Physica D* (1986), pp. 22:187 – 204.

[9] FORREST, S., PERELSON, A. S., ALLEN, L., AND CHERUKURI, R. Self-nonself discrimination in a computer. In *Proc. 1994 IEEE Symposium on Research in Security and Privacy* (Los Alamitos, CA, USA, 1994), pp. 202 – 214.

[10] GONZALES, F., DASGUPTA, D., AND GOMEZ, J. The effect of binary matching rules in negative selection. In *Proc. Genetic and Evoluntionary Computation Conference (GECCO 2003)* (Chicago, IL, USA, July 2003), pp. 195 – 206.

[11] GONZALEZ, F., DASGUPTA, D., AND KOZMA, R. Combining negative selection and classification techniques for anomaly detection. In *Proc. 2002 Congress on Evoluntionary Computation (CEC 2002)* (Honolulu, HI, USA, May 2002), pp. 705 – 710.

[12] HOFMEYER, S., AND FORREST, S. Architecture for an artificial immune system. *Evolutionary Computation Journal 8*, 4 (2000), 433 – 473.

[13] HOFMEYER, S. A., AND FORREST, S. Immunity by design: An artificial immune system. In *Proc. Genetic and Evolutionary Computation Conference (GECCO)* (San Francisco, CA, USA, 1999), pp. 1289 – 1296.

[14] MIT LINCOLN LABORATORY. Darpa intrusion detection evaluation, 1999. http://www.ll.mit.edu/IST/ideval/.

[15] SECURITYFOCUS. Bugtraq mailing list, 2002 – present. http://www.securityfocus.com/archive/1.

[16] SOMAYAJI, A., HOFMEYER, S., AND FORREST, S. Principles of a computer immune system. In *Proc. New Security Paradigms Workshop (NSPW-97)* (Langdale, UK, 1997), pp. 75 – 82.

[17] WATKINS, A. *A resource limited artificial immune classifier*. PhD thesis, Mississippi State University, 2001.

[18] WATKINS, A., AND BOGGESS, L. A new classifier based on resource limited artificial immune systems. In *IEEE Congress on Evolutionary Computation* (Honolulu, HI, USA, May 2002), pp. 1546 – 1551.

[19] WATKINS, A., AND BOGGESS, L. A resource limited artificial immune classifier. In *IEEE Congress on Evolutionary Computation* (Honolulu, HI, USA, May 2002), pp. 926 – 931.

[20] WATKINS, A., AND TIMMIS, J. Artificial Immune Recognition System (AIRS): Revisions and refinements. In *Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS)* (University of Kent at Canterbury, UK, 2002), pp. 173 – 181.

[21] WATKINS, A., TIMMIS, J., AND BOGGESS, L. Artificial Immune Recognition System (AIRS): An Immune-Inspired Supervised Learning Algorithm. *Genetic Programming and Evolvable Machines 5*, 3 (2004), 291 – 317.

[22] WILLIAMS, P. D., ANCHOR, K. P., BEBO, J. L., GUNSCH, G. H., AND LAMONT, G. D. CDIS: Towards a computer immune system for detecting network intrusions. In *Proc. Recent Advances in Intrusion Detection (RAID 2001)* (Davis, CA, USA, October 2001), pp. 117 – 133.

[23] ZENOMORPH (ADMIN@CGISECURITY.COM). Fingerprinting port 80 attacks: A look into web server, and web application attack signatures. Whitepaper, November 2001. http://www.cgisecurity.com/papers/fingerprint-port80.txt.

[24] ZENOMORPH (ADMIN@CGISECURITY.COM). Fingerprinting port 80 attacks: A look into web server, and web application attack signatures: Part 2. Whitepaper, March 2002. http://www.cgisecurity.com/papers/fingerprinting-2.txt.