An Artificial Immune System-Inspired Multiobjective Evolutionary Algorithm with Application to the Detection of Distributed Computer Network Intrusions

Charles R. Haag Gary B. Lamont Paul D. Williams Gilbert L. Peterson
Department of Electrical and Computer Engineering
Graduate School of Engineering and Management
Air Force Institute of Technology (AFIT)
2950 Hobson Way, Bldg. 640
WPAFB, OH, 45433-7765
charles.haag@langley.af.mil,[gary.lamont, paul.williams, gilbert.peterson]@afit.edu

ABSTRACT

Today's signature-based intrusion detection systems are reactive in nature and storage-limited. Their operation depends upon catching an instance of an intrusion or virus and encoding it into a signature that is stored in its anomaly database, providing a window of vulnerability to computer systems during this time. Further, the maximum size of an Internet Protocol-based message requires the database to be huge in order to maintain possible signature combinations. In order to tighten this response cycle within storage constraints, this paper presents an innovative *Artificial Immune System*-inspired *Multiobjective Evolutionary Algorithm*. This distributed intrusion detection system (IDS) is intended to measure the vector of tradeoff solutions among detectors with regard to two independent objectives: best classification fitness and optimal hypervolume size. Our *antibody* detectors promiscuously monitor network traffic for exact and variant abnormal system events based on only the detector's own data structure and the application domain truth set, responding heuristically. Applied to the MIT-DARPA 1999 insider intrusion detection data set, our software engineered algorithm correctly classifies normal and abnormal events at a high level which is directly attributed to a detector affinity threshold.

Categories and Subject Descriptors

D.1.5 [**Programming Techniques**]: Object-oriented Programming.
D.2.11 [**Software Engineering**]: Software Architectures – *Patterns*.
I.5.4 [**Pattern Recognition**]: Applications – *text processing*.
I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search – *Heuristic methods*.
I.6.8 [**Simulation and Modeling**]: Types of Simulation – *Distributed*.

General Terms

Algorithms, Measurement, Performance, Design, Experimentation, Security, Theory.

Keywords

Intrusion detection, computer networks, computer security, evolutionary computation, artificial immune system, multiobjective, distributed computing.

1. PROBLEM MOTIVATION

Signature-based intrusion detection systems (IDS) detect attacks by discovering exact matches between incoming data and a database of known attack string signatures. Their reactive nature and storage-limited database allows for only a subset of the 256^{65535} harmful signature combinations to be catalogued for future detection, after an attack. These constraints define our algorithm's application domain as the intrusion detection (ID) problem domain. Compounding these constraints, the trend of exact and variant nefarious signatures since 2001, through at least mid-2004, indicate exponential growth [1]. Thus, a more efficient IDS approach is required.

The IDS method of signature string evaluation is analogous to the Boolean Satisfiability Problem (SAT): the enumeration (or exhaustion) of a search space of *n* variables against a function to determine which variables return true from that function [2]. The SAT defines our formal problem classification to be Nondeterministic Polynomial time (NP)-Hard, preventing it from being solved in polynomial time [3]. Inability to solve in polynomial time leads to the inability to solve ID deterministically as required in real-time IDS operation.. Because signature-based IDSs are deterministic, they become infeasible and we must consider a stochastic approximation solution, such as an evolutionary algorithm (EA) that possesses the unique ability to generate solutions in polynomial time.

In particular, we chose to integrate a multiobjective EA (MOEA) with an artificial immune system resulting in an innovative software engineered IDS [4]. In presenting this development, the project goals are defined in Section 2, the IDS design is discussed in Section 3, with experimental results and analysis presented in Section 4.

2. HYPOTHESIS

One evolutionary algorithm (EA) of choice is the artificial immune system (AIS) because the IDS architecture is similar to the human biological immune system (BIS): a parallel and distributed adaptive system [5] for detecting and destroying antigens. In addition, we pursue a multiobjective approach because the consolidation of information into a single aggregated objective tends to lose data granularity and offer the decision-maker only one solution vice a set of trade-off solutions. The solution approach develops a new proof-of-concept algorithm that advances the existing work of two AIS-motivated algorithms: Edge, Lamont and Raines' retrovirus algorithm (REALGO) [6] for its ability to escape local optima, with regard to string matching, and Coello and Cortés' multiobjective immune system algorithm (MISA) [5] for its employment of an AIS in a multiobjective EA context.

Our hypothesis is that the integration of REALGO and MISA, applied to an ID data set, validates an AIS-inspired multiobjective evolutionary algorithm (MOEA). This MOEA provides a set of tradeoff solutions with regard to the measurement of two independent objectives seeking a global minimum: highest network traffic classification fitness and optimal detector hypervolume. This hypothesis is composed of four measurable objectives:

1. *Validate the migration of C language-based REALGO and MISA into their Java equivalents*;
2. *Attain the highest correct classification rate possible.* A heuristic-based positive value is assessed for any outcome, increasing the fitness score of the first objective of highest classification fitness. The higher a detector's effectiveness, the lower this objective's sum score is;
3. *Identify a known optimal detector hypervolume.* Research shows detector effectiveness is impacted by hypervolume of a particular size [7,8]. We seek detectors that do not stray from the pre-determined *negative selection* affinity threshold; hence, the lowest deviation score for our second objective;
4. *Validate AIS cooperative communication within a distributed environment.* As AIS detectors are rewarded for correct classification and detection, the AIS broadcasts its fittest detectors to listening AISs, for possible inclusion into their population. Validation is observation of this operation.

Upon termination of our algorithm, two vectors compose each objective's score for the set of surviving detectors in the population, which each detector is a solution point. By definition, multiobjective algorithms produce multiple solutions which may not be optimal for each objective [10]. By adjusting one solution for greater optimality, we risk decreasing the desired value of one or more other solutions. Thus, we desire a set or subset(s) of *nondominated solutions* through *Pareto Optimality* (*P**). A solution point is considered *Pareto*-optimal (in a global-minimum context) if each of its objective's values is less-than-or-equal to all solution point objective values and it has at least one value, in any objective, smaller than any other solution point's value for that objective [9]. This set of *Pareto*-optimal points represents the *Pareto Front* (*PF**) of fittest (*non-dominated*) detectors that is provided to the decision-maker, giving that person more options in selecting solution points for future ID domain employment based on the classification-to-hypervolume tradeoff depicted by the *true PF**.

3. HIGH AND LOW LEVEL DESIGN

Our IDS algorithm is the methodical integration of the foundational AIS framework, a MOEA and a modification of that EAs standard operators by REALGO and MISA [4]. The design approach employs good software engineering practice. Timmis and De Castro define a standardized AIS framework where the engineered solution is *application domain*-specific (see Figure 1) [10]. This framework can be thought of as a layered approach. The basis for an AIS begins with the pre-defined *application* (problem) *domain*, which governs the method of *representation*. Once chromosome data structure representation (e.g., bit string, real-valued vector, length, etc.) is decided, one or more affinity measures are used to quantify interactions of the system's elements; e.g., *Hamming distance* measurements applies to bit string representation while *Euclidian distance* is applied to real-valued vectors. The top layer, *immune algorithms*, encompasses those functions that govern the behavior (dynamics) of the system; e.g., method of mutation, selection, evaluation, etc. Addressing these layers leads to a engineered domain-specific solution.

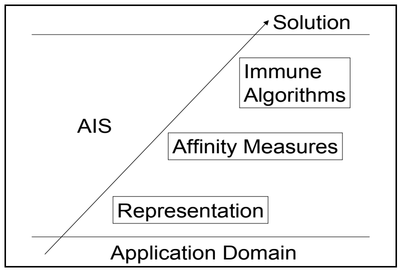


Figure 1. Timmis' AIS framework [10]

3.1 Application Data Domain

The application domain is composed of week one and two of the MIT-DARPA (LL) 1999 insider intrusion detection data set [11] because it currently constitutes the largest publicly available benchmark of network traffic [12]. The first week consists of normal (*self*)-only traffic, facilitating *negative selection*. The second week totals 7.2 million packets composed of 99.25% *self* and 0.75% labeled attacks (*non-self*), of which we evaluate the entire week.

3.2 Antigen and Antibody Representation

The data structure that composes each *antigen* (Ag) data set record and *antibody* (Ab) detector chromosome is a fixed integer array with binary allele values that define the chromosome's location in the search space. We chose this data structure because it was (conveniently) the same type employed by both REALGO and MISA.

The Abs for network intrusion are generated and trained in the same manner as in anti-virus detectors [13]. However, network intrusion Ags are longer and segregated because they utilize the IP packet structure for its template. For this reason, we constrain our ID domain to encode Ags from network packets wrapped in the three most common IP protocols utilized by *non-self*: TCP, UDP and ICMP. Decimal-value header information from each incoming Ag's packet header field is encoded into its binary equivalent and concatenated to the end of its string. Hence, each Ag's IP $\cup$ (TCP $\vee$ UDP $\vee$ ICMP) header results in a Ag TCP, UDP or ICMP binary DNA chromosome, respectively (Figure 2).

Ab chromosomes are composed of three parts: its DNA (binary), RNA (binary) and seven state attributes (integer) (see Figure 3). Its DNA is generated by *negative selection* and the only portion of the Ab to be computed against the Ag. Its RNA is its DNA replica, facilitating the REALGO method of escaping local optima through *RNA reverse transcription* modeling [5]. If the mutated DNA results in a higher fitness than last time, its DNA is replicated to its personal memory space called RNA. If the fitness is worse, the DNA reverts to its last best fitness RNA to purpose mutation "in a different direction." Finally, there are seven parameters: $\lambda \leftarrow$ name, $\alpha \leftarrow$ number of false detections, $\rho \leftarrow$ (true positive + true negative) fitness score, $\phi \leftarrow$ (false positive + false negative) fitness score, $\eta \leftarrow$ deviation from *negative selection*-defined affinity threshold (determining volume), $\beta \leftarrow$ broadcasted (yes/no), $\psi \leftarrow$ number of Abs that *Pareto*-dominate this Ab.

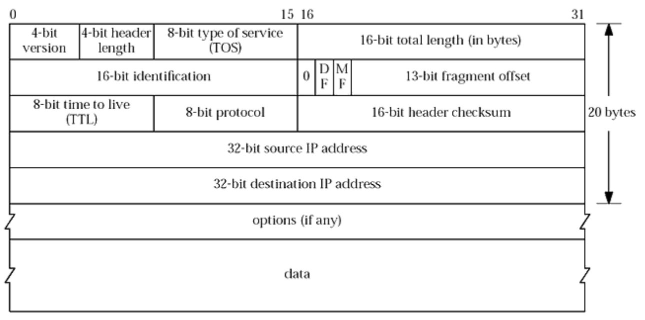


Figure 2. Ag chromosomes are formed through IP (shown), TCP, UDP and ICMP packet header field decimal values.

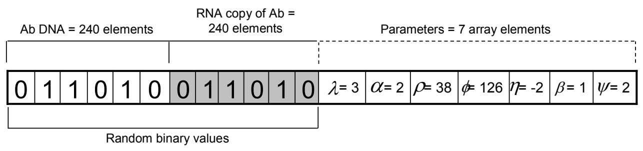


Figure 3. Ab chromosome (DNA $\cup$ RNA $\cup$ parameters).

Because our DNA is composed of a bit string data structure, our *affinity measure* is Hamming distance (Equation 1). We choose Hamming distance over the r-contiguous bit rule or other measures because we're pattern-matching the entire context of each Ag packet vs. particular contiguous IP fields [11]:

$$H = \sum_{i=1}^{L} \delta_i \text{ where } \delta_i = \begin{cases} 1 \text{ if } Ab_i \neq Ag_i \\ 0 \text{ otherwise} \end{cases} . \qquad (1)$$

3.3 Immune Algorithm

Integrating REALGO's RNA transcription into the *evaluation* operator and the MISA framework, including its *evaluation*, *selection* and *mutation* operators, we have the pseudocode for our algorithm we call jREMISA: the *Java retrovirus-inspired MISA* (Algorithm 1). REALGO and MISA do not employ crossover because of the sufficiency of mutation to move Abs throughout the search space. In addition, crossover can break good building blocks.

```
1   procedure jREMISA
2   begin
3   repeat
4       Randomly generate initial TCP, UDP, ICMP Populations ($P_p$)
5       Initialize empty secondary Population ($P_s$)
6   negative_selection($P_p$,data_set$_{clean}$,threshold) /* Eval 1 */
7   until (end of data_set$_{clean}$)
8   repeat
9       fitness function (ag) /* evaluation_2 */
10      mutationCauchy($P_p$)
11      P_optimality() /* evaluation_3 */
12      clonalSelection(0.05)
13      mutationUniform($P_s$)
14      $P_p \leftarrow P_s$  /* copy best of $P_s$ as next gen's $P_p$ */
15      if (networking)
16          broadcast($P_s$)  /* offer nondominateds to all AISs */
17          processReceived()  /* Any captured Abs? */
18      endif
19  until (end of data_set$_{attack}$)
20  end
```

Algorithm 2. jREMISA AIS-inspired MOEA.

Algorithm 2, lines 3-7, which equates to Algorithm 1, lines 3-5, consist of *negative selection* where Abs are randomly generated and evaluated against every Ag of a *self*-only day of traffic, given a predefined affinity threshold percentage. If an Ab reacts to *self*, it is discarded without replacement. In doing this, we are assured that the remaining population at algorithm termination does not react to a single packet of the day's traffic. Post-*negative selection* consists of Algorithm 2, lines 8-19, which equates to Algorithm 1, lines 6-12. These lines are summed up in Figure 4 where each Ag entering the evaluation window represents a new generation. We partition our primary population (pop$_p$) by IP protocol for two reasons: efficiency is increased by evaluating the Ag only against a subset of pop$_p$ and pattern-matching becomes more relevant. E.g., TCP, UDP and ICMP don't all share the same chromosomal structure, as their IP fields differ; hence, it doesn't make sense to compare a TCP Ab to a non-TCP Ag.

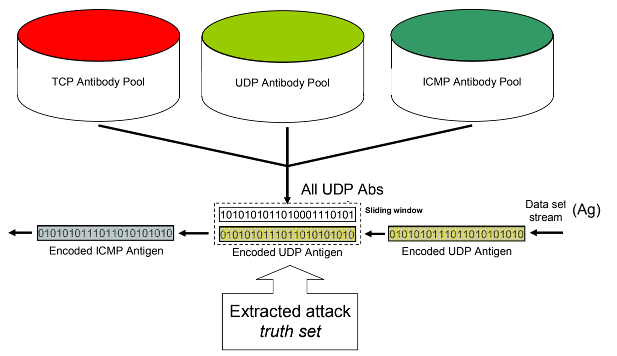


Figure 4. Transient (data set) Ags evaluated against its IP protocol-matching Ab.

When the Ag enters the window, the following occurs (per Algorithm 2):

- FITNESS FUNCTION (line 9): Hamming distance *H* is computed between the Ab and Ag DNA signature. Combined with the affinity threshold and *truth set* informing whether this Ag is *self* or *non-self*, one of four outcomes results:
    - True negative (Ag←Ab←*self*): penalize first objective (obj_1) fitness += *H*, copy DNA to RNA, reward second objective (obj_2) += 1%;
    - True positive (Ag←Ab←*non-self*): penalize obj_1 += ($Ag_{length} - H$), copy DNA to RNA, reward obj_2 += 1%;
    - False negative (Ag←self, Ab←non-self): falseDetections++, revert DNA to RNA, penalize obj_2 -= 1%;
    - False positive (Ag←non-self, Ab←self): falseDetections++, revert DNA to RNA, penalize obj_2 -= 1%.

  Both true negative and positive outcomes are penalized because, if true negative, *H* should ideally equal zero. If not, obj_1's value is increased by the number of complimentary alleles (*H*). If true positive, then *H* should ideally equal the length of the Ag. If not, obj_2's value is increased by the number of non-complementary alleles;
- CAUCHY MUTATION (line 10): REALGO suggests Cauchy mutation to move Abs around the landscape. We perform it upon all penalized Ab alleles;
- *P**-TEST (line 11): all Abs undergo a *Pareto*-optimality test to determine the number of Abs each is dominated by. *Quicksort* then ascending-sorts them;
- MISA's CLONAL SELECTION PRINCIPLE (lines 12-14): elitist selection copies the top 5% of nondominated Abs from their pop_p to their respective secondary (or external) population (pop_s) meant to contain only nondominated Abs. Copied Abs are cloned to 600% the pop_s size. All copied and cloned Abs are then mutated in *n*-random allele positions, where *n* ← is the number of objectives (two) plus their *Pareto*-dominance value. Following, any pop_p Abs lost to reaching the max number of false detections is replaced by the fittest Abs from the pop_s in order to return the pop_p to its original size. Finally, the pop_s is culled for only nondominated Abs;
- AD-HOC NETWORKING (optional): if enabled, newly discovered nondominated Abs copied to the pop_s are broadcast to the subnet where listening jREMISAs capture and add to their pop_s only if it dominates their entire pop_s (see Figure 5).

This process defines one generation and recurs for the number of data set packets.

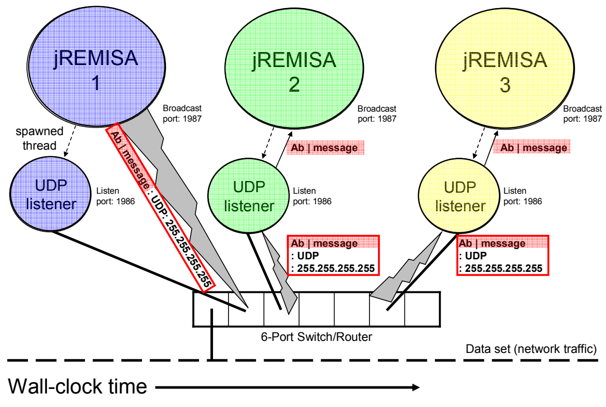


Figure 5. jREMISA distributed communication architecture.

The software engineering of jREMISA in design and implementation used various paradigms; Model-View-Controller, UML Class Diagram, and Design Patterns[4]. jREMISA was developed in the *Eclipse*[2] open-source Integrated Development Environment (IDE).

4. EXPERIMENTATION AND ANALYSIS

Based upon the objectives of Section 2, the experiments are divided into three parts:

1. validation of the C-to-Java migration of REALGO and MISA;
2. jREMISA effectiveness measurement against the LL data set in 13 scenarios: 10 standalone, involving all days of week two, and three *distributed island model* executions in a two-, three- and four-jREMISA configuration;
3. statistically comparing jREMISA against other algorithms applied to the same data set.

In the distributed configuration, four computers with Windows XP Professional 2002, Service Pack 2 were used:

- "PC1" ← Dell Inspiron 710m laptop, 2 GHz Pentium-M, 2 GB of RAM;
- "PC2" ← Dell XPS laptop, 3.4 GHz Pentium-4 HyperThreading, 1 GB RAM;
- "PC3" ← Dell Precision laptop, 1.8 GHz Pentium-4, 512 MB RAM;
- "PC4" ← Dell Optiplex GX270 tower, 2.6 GHz Pentium-4, 512 MB RAM.

The C-to-Java migration experiment is straightforward, yet we are concerned about the impact of a different programming language and random number generator on the output. Figure 6 compares the output between REALGO and jREALGO, employing REALGO's Yao and Liu's test function [6]where the optimal value is -12569.5. Figure 7 compares the output between MISA

[2] The Eclipse project, www.eclipse.org.

and jMISA, employing MISA's Kita-proposed two-variable test function [5].

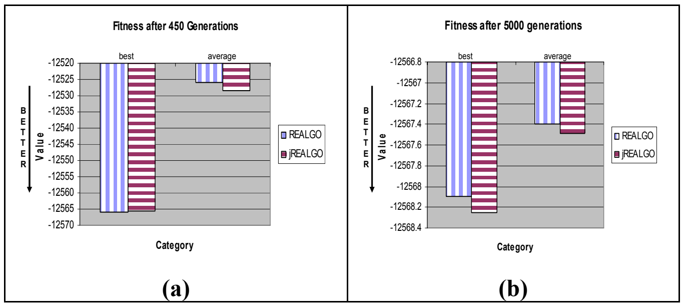


Figure 6. REALGO vs. jREALGO: (a) after 450 generations; (b) after 5000 generations.

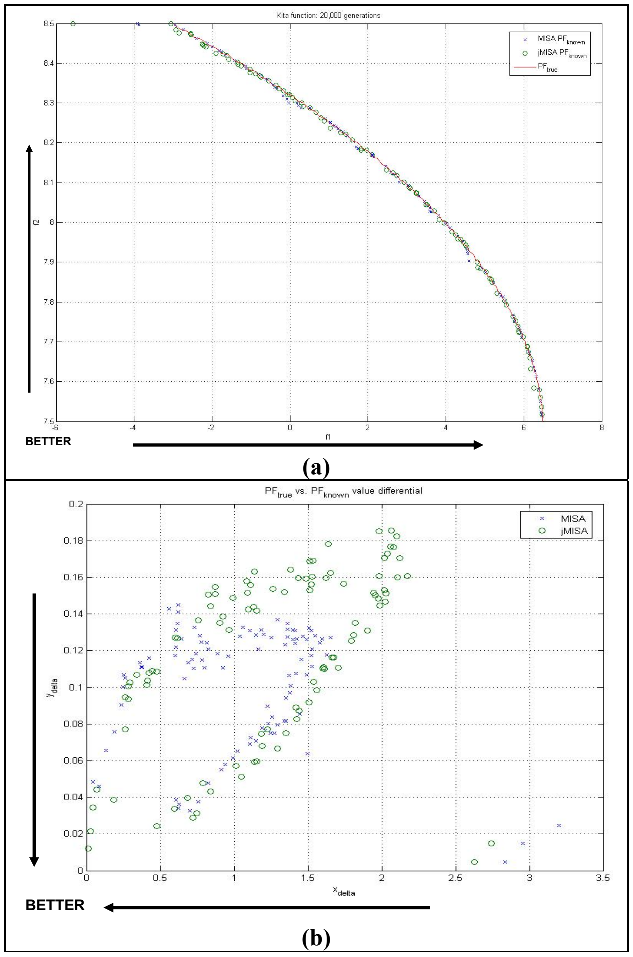


Figure 7. (a) MISA and jMISA *PF* vs. MISA's true *PF; (b) Euclidian distance measure of MISA and jMISA points from MISA's true *PF**.**

While Figure 7(a) shows MISA and jMISA's known *PF* to be almost identical, Figure 7(b)'s Euclidian distance measurement of each point shows that while MISA has the preponderance of points closer to true *PF**, jMISA possesses the few points closest to true *PF**. Therefore, based on the comparison of REALGO to jREALGO and MISA to jMISA, we conclude this experiment validated.

Mapping the LL truth set of labeled-attacks to the *Ethereal*[3]-analyzed five days of labeled attacks, 16 context-based attacks are extracted (Figure 8).

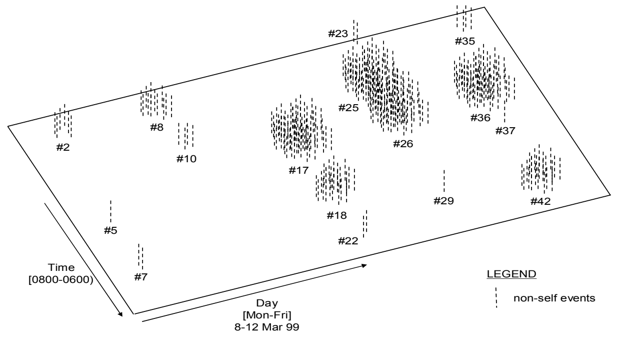


Figure 8. LL 1999 week-two insider data set landscape with labeled attacks.

In comparing our results to Williams' *Warthog* [14], the surviving TCP pop_ps should be in the range of *Warthog*'s experimentally-employed Abs: between 32 and 2048. Table 1 shows that this Ab range requires the affinity threshold between 37-42%. In Table 2, scenarios 1-6 show our Ab effectiveness based on each threshold. Since scenario 4 has the combined highest classification and lowest false detection rate, we use that 39% threshold to compare it to the remaining days of the week (scenarios 7-10). In Table 3, we employ this same threshold to the distributed tests, differing only by the number of partitions in the data set decomposition.

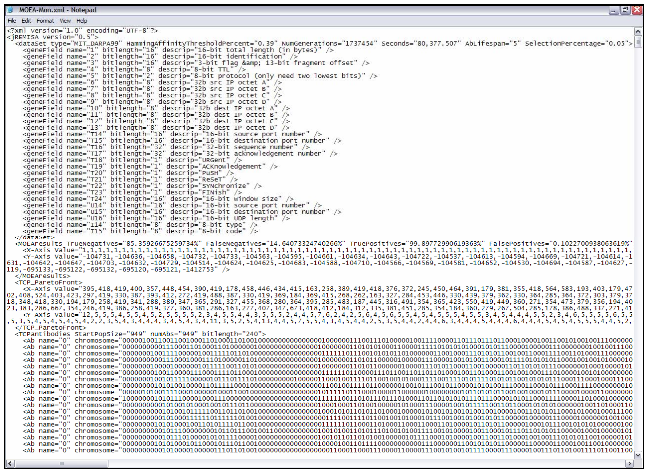
Figure 10(a) shows the scenario-four TCP pop_s. This PF* is provided for all three populations for each MOEA execution. Its intent is to show the tradeoff between each Abs fitness score and affinity threshold deviation. For all five days of the week, all three pop_s *PF** had Abs in the +(4-5)% range 73% of the time. In addition, Abs were concentrated in this same range 87% of the time. Figure 10(b) shows an *attack graph* where each Ab plots its classification declaration for each attack. The x-axis indicates the outcome while the y-axis represents the packet number, mapping the attack. Here, there are no points on the false positive side for two attacks, indicating a 0% false positive rate for LL attack #26 and #29. While not shown, jREMISA also optimally discovered LL attacks #7 and #22.

Figure 11 depicts a snapshot of PC1 in scenario 11 where PC1 has broadcast Abs and decided to reject an incoming Ab received from PC2. Figure 15 shows how our algorithm does better than *Warthog* with regard to the false detection rate as the number of Abs are increased for the week-two Thursday data set. With *Warthog*, Williams declares that as the number of Abs increases, so does the false positive rate. However, our experiments show that as we increased our Abs, the false positive rate decreased. Therefore, we have shown how our algorithm is more effective with respect to the same benchmark data set.

[3] Ethereal: open-source network protocol analyzer, www.ethereal.com.

Figure 12 graphically depicts Table 2 with classification and false detection ratios having 5% variance due to the multiple trials run of the same scenario to maximize statistical accuracy. Figure 13 and Figure 14 summarize the results of the distributed phase of our experiments. In Figure 13, a graphical depiction of Table 3, we discover that the sharing of nondominated Abs among jREMISAs did not conclusively show a synergistic increase in effectiveness when compared to the standalone classification ratio, although the two-PC configuration fared better. Figure 14 shows the increase in efficiency as more jREMISAs participated. Upon the completion of either phase—*negative selection* or MOEA—jREMISA saves the resulting

population, *Pareto Front* and attack graph values to a formatted XML file (Figure 9)



```
<?xml version="1.0" encoding="UTF-8"?>
<jREMISA version="0.5">
  <dataSet type="MIT_DARPA99" HammingAffinityThresholdPercent="0.39" NumGenerations="1737454" Seconds="80,377.507" AbLifespan="5" SelectionPercentage="0.05">
    <geneField name="1" bitlength="16" descrip="16-bit total length (in bytes)" />
    <geneField name="2" bitlength="16" descrip="16-bit identification" />
    <geneField name="3" bitlength="16" descrip="3-bit flag &amp; 13-bit fragment offset" />
    <geneField name="4" bitlength="8" descrip="8-bit TTL" />
    <geneField name="5" bitlength="2" descrip="8-bit protocol (only need two lowest bits)" />
    <geneField name="6" bitlength="8" descrip="32b src IP octet A" />
    <geneField name="7" bitlength="8" descrip="32b src IP octet B" />
    <geneField name="8" bitlength="8" descrip="32b src IP octet C" />
    <geneField name="9" bitlength="8" descrip="32b src IP octet D" />
    <geneField name="10" bitlength="8" descrip="32b dest IP octet A" />
    <geneField name="11" bitlength="8" descrip="32b dest IP octet B" />
    <geneField name="12" bitlength="8" descrip="32b dest IP octet C" />
    <geneField name="13" bitlength="8" descrip="32b dest IP octet D" />
    <geneField name="T14" bitlength="16" descrip="16-bit source port number" />
    <geneField name="T15" bitlength="16" descrip="16-bit destination port number" />
    <geneField name="T16" bitlength="32" descrip="32-bit sequence number" />
    <geneField name="T17" bitlength="32" descrip="32-bit acknowledgement number" />
    <geneField name="T18" bitlength="1" descrip="URGent" />
    <geneField name="T19" bitlength="1" descrip="ACKnowledgement" />
    <geneField name="T20" bitlength="1" descrip="PuSH" />
    <geneField name="T21" bitlength="1" descrip="ReSeT" />
    <geneField name="T22" bitlength="1" descrip="SYNchronize" />
    <geneField name="T23" bitlength="1" descrip="FINish" />
    <geneField name="T24" bitlength="16" descrip="16-bit window size" />
    <geneField name="U14" bitlength="16" descrip="16-bit source port number" />
    <geneField name="U15" bitlength="16" descrip="16-bit destination port number" />
    <geneField name="U16" bitlength="16" descrip="16-bit UDP length" />
    <geneField name="I14" bitlength="8" descrip="8-bit type" />
    <geneField name="I15" bitlength="8" descrip="8-bit code" />
  </dataSet>
  <MOEAresults TrueNegatives="85.35926675259734%" FalseNegatives="14.64073324740266%" TruePositives="99.89772990619363%" FalsePositives="0.1022700938063619%"
    <X-Axis Value="1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
    <Y-Axis Value="-104731,-104636,-104658,-104732,-104733,-104563,-104595,-104661,-104634,-104643,-104722,-104537,-104613,-104594,-104669,-104721,-104614,-1
631,-104642,-104647,-104703,-104632,-104729,-104514,-104624,-104625,-104683,-104588,-104710,-104566,-104569,-104581,-104652,-104530,-104694,-104587,-104627,-
119,-695133,-695122,-695132,-695120,-695121,-1412753" />
  </MOEAresults>
  <TCP_ParetoFront>
    <X-Axis Value="395,418,419,400,357,448,454,390,419,178,458,446,434,415,163,258,389,419,418,376,372,245,450,464,391,179,381,355,418,564,583,193,403,179,47
02,408,524,403,423,297,419,330,387,393,412,272,419,488,387,330,419,369,184,369,415,268,262,163,327,284,453,446,330,439,379,362,330,364,285,364,372,303,379,37
18,348,418,330,194,179,258,419,341,288,389,347,365,291,327,455,368,280,364,395,285,483,187,445,316,491,354,365,423,550,419,449,360,271,354,473,379,356,194,40
23,383,286,667,354,246,419,386,258,419,377,360,381,286,163,277,407,347,673,418,412,184,312,335,381,451,285,354,184,360,279,267,504,285,178,386,488,337,271,41
    <Y-Axis Value="12,5,5,5,4,5,5,4,5,2,5,5,5,5,2,3,4,5,5,4,4,3,5,5,5,2,4,4,5,7,6,2,4,2,5,6,4,5,6,5,5,4,5,4,5,5,5,4,5,5,3,4,5,4,4,4,5,5,2,3,4,6,5,5,5,5,6,5,5
,5,3,5,4,5,4,5,4,5,4,2,2,3,5,4,3,4,4,4,3,4,5,4,3,4,11,3,5,2,5,4,13,4,4,5,7,5,5,4,3,4,5,4,4,2,5,3,5,4,4,2,4,4,6,3,4,4,4,5,4,4,4,6,4,4,4,5,4,5,4,5,5,5,4,4,5,2,
  </TCP_ParetoFront>
  <TCPantibodies StartPopSize="949" numAbs="949" bitlength="240">
    <Ab name="0" chromosome="00000100110011001000110100011010010000000000000000100000011[illegible]
    <Ab name="0" chromosome="00000000011100011010001101000000100000000000000000100000001[illegible]
    <Ab name="0" chromosome="00000010011110000010011111011010000000000000000000111111011[illegible]
    <Ab name="0" chromosome="00000000011100010001110100000110100000000000000000100000001[illegible]
    <Ab name="0" chromosome="00000001000010000000101111100110101000000000000000100000001[illegible]
    <Ab name="0" chromosome="00000000100110000111000111101100010000000000000000111111011[illegible]
    <Ab name="0" chromosome="00000001001011111000001011101111010000000000100000110001001[illegible]
    <Ab name="0" chromosome="00000000101010010000110111110001000000000000000000110010011[illegible]
    <Ab name="0" chromosome="00000101001111000100000100011001000000000000000000101111101[illegible]
    <Ab name="0" chromosome="10000000101011100001000111000000000000000000000000111111001[illegible]
    <Ab name="0" chromosome="00000000001010010100010010111011000000000000000000100010001[illegible]
    <Ab name="0" chromosome="00000000001010010111110011011010010000000000000000100010101[illegible]
    <Ab name="0" chromosome="00000000001010001111110111111010010000000000000000111100111[illegible]
    <Ab name="0" chromosome="00000000101010001001101011110110010000000000000000111111011[illegible]
    <Ab name="0" chromosome="00000000001011100000000101101110010011000000000000100101001[illegible]
    <Ab name="0" chromosome="00000000001011101000010101111000010000000000000000100101101[illegible]
    <Ab name="0" chromosome="00000000101010001011000101110110010000000000000000100001001[illegible]
    <Ab name="0" chromosome="00000000001010000100000111011010010000000000000000110001101[illegible]
```

Figure 9: Example post-MOEA XML output file

Table 1. Negative selection results for all pop$_p$ starting at 4096 for Friday data set (1,467,775 packets).

Affinity (%)	Runtime(mins)	End TCP	survived	End UDP	survived	End ICMP	survived
37	186.65	2663	65.015%	3737	91.235%	3707	90.503%
38	124.20	1563	38.159%	3372	82.324%	3513	85.767%
39	89.17	935	22.827%	2890	70.557%	3290	80.322%
40	45.27	357	8.716%	2275	55.542%	2700	65.918%
41	26.43	126	3.076%	2000	48.828%	2344	57.227%
42	16.28	34	0.830%	1431	34.937%	1997	48.755%

Table 2. MOEA run summary: standalone jREMISA.

								Self Events		*Non-self* Events	
Scen-ario	Day	Gener-ations	Affinity Threshold	TCP Pop	UDP Pop	ICMP Pop	Runtime	True Neg%	False Neg%	True Pos%	False Pos%
1	Thurs	1547710	42%	37	86	248	39.12 m	53.78	46.22	62.6	37.4
2	“	“	41%	106	116	284	52.48 m	67.44	32.56	68.33	31.67
3	“	“	40%	315	146	341	3.61 hrs	76.10	23.90	76.92	23.08
4	“	“	39%	966	361	810	18.21hrs	85.45	14.55	97.66	2.34
5	“	“	38%	1580	423	881	2.36 day	**86.48**	13.52	92.51	7.49
6	“	“	37%	2564	462	927	5.83 day	82.52	17.48	99.71	0.29
7	Mon	1737455	39%	969	349	846	20.02 hr	85.36	14.64	**99.90**	0.10
8	Tues	1571748	“	922	362	882	18.86 hr	84.61	15.39	97.35	2.65
9	Wed	995235	“	920	333	798	11.69 hr	83.37	16.63	98.26	1.74
10	Fri	1347393	“	964	376	829	13.43 hr	83.59	16.41	96.57	3.43

Table 3. MOEA run summary: distributed jREMISAs against Thursday data set.

						Self Events		*Non-self* Events	
jREMISA ID	Packet range (1547709 total)	TCP Pop	UDP Pop	ICMP Pop	Runtime	True Neg%	False Neg%	True Pos%	False Pos%
Scenario 11: 2 jREMISAs, 39% affinity threshold, Thursday attack data set									
PC1	1 – 773854	966	361	810	9.44hrs	**86.21**	13.79	98.10	1.90
PC2	773855 – 1547709	936	344	854	9.63hrs				
Scenario 12: 3 jREMISAs, 39% affinity threshold, Thursday attack data set									
PC1	1 – 515903	966	361	810	5.09hrs	84.31	15.69	97.94	2.06
PC2	515904 – 1031807	936	344	854	6.35hrs				
PC3	1031808 – 1547709	951	357	826	6.86hrs				
Scenario 13: 4 jREMISAs, 39% affinity threshold, Thursday attack data set									
PC1	1 – 386927	966	361	810	4.33hrs	84.94	15.06	**98.55**	1.45
PC2	386928 – 773854	936	344	854	4.63hrs				
PC3	773855 – 1160781	951	357	826	4.86hrs				
PC4	1160782 – 1547709	954	360	822	5.09hrs				

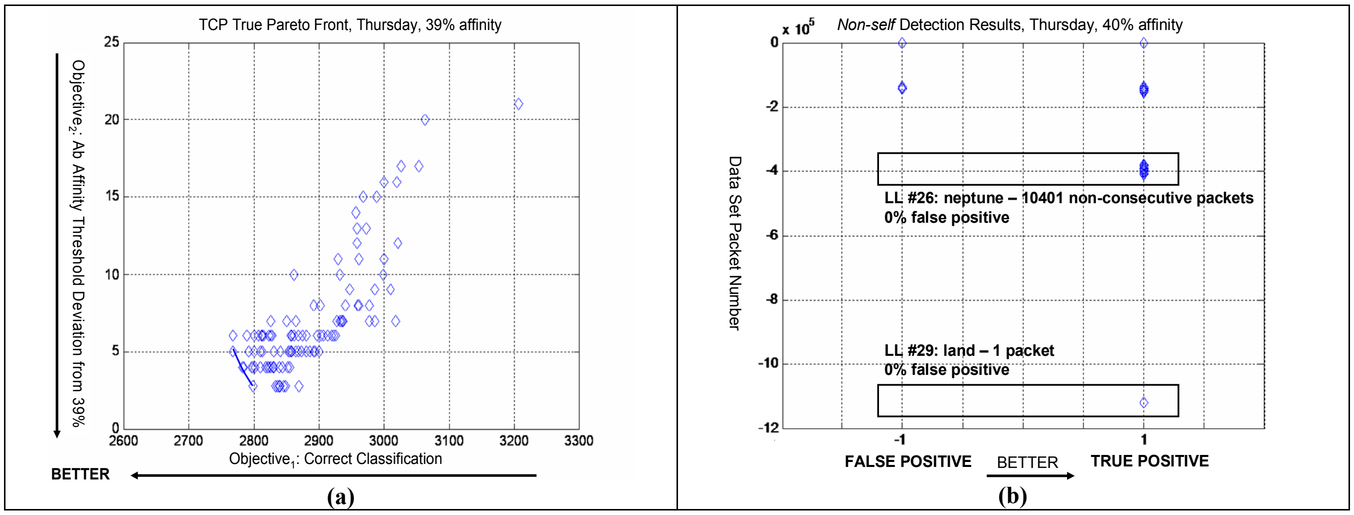
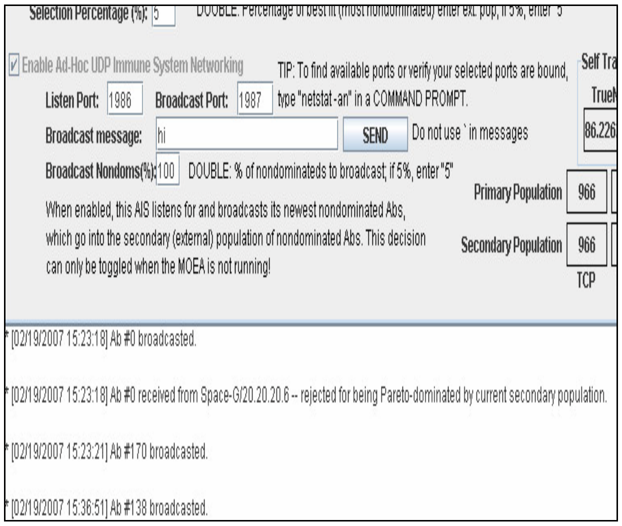


Figure 10. MOEA post-execution: (a) TCP pop$_s$ *PF; (b) attack graph: two attacks found with 0% false positive rate.**



Selection Percentage (%): 5 DOUBLE: Percentage of best fit (most nondominated) enter ext. pop; if 5%, enter "5"

☑ Enable Ad-Hoc UDP Immune System Networking TIP: To find available ports or verify your selected ports are bound, type "netstat -an" in a COMMAND PROMPT.

Listen Port: 1986 Broadcast Port: 1987

Broadcast message: hi SEND Do not use ` in messages

Broadcast Nondoms(%): 100 DOUBLE: % of nondominateds to broadcast; if 5%, enter "5"

When enabled, this AIS listens for and broadcasts its newest nondominated Abs, which go into the secondary (external) population of nondominated Abs. This decision can only be toggled when the MOEA is not running!

Self Tra
True
86.226
Primary Population 966
Secondary Population 966
TCP

```
* [02/19/2007 15:23:18] Ab #0 broadcasted.
* [02/19/2007 15:23:18] Ab #0 received from Space-G/20.20.20.6 -- rejected for being Pareto-dominated by current secondary population.
* [02/19/2007 15:23:21] Ab #170 broadcasted.
* [02/19/2007 15:36:51] Ab #138 broadcasted.
```

Figure 11. Snapshot of jREMISA distributed communication.

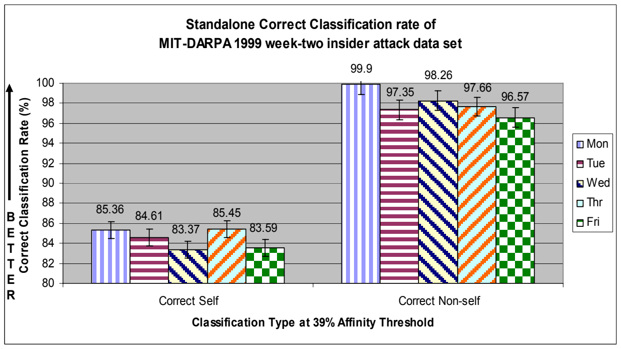


Figure 12. jREMISA standalone effectiveness against each day of the week-two insider attack data set.

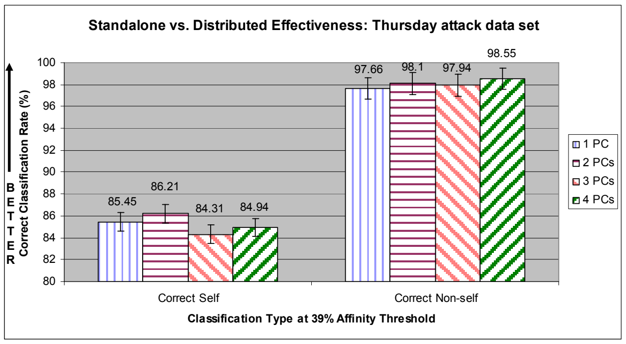


Figure 13. Standalone vs. distributed effectiveness against the week-two Thursday insider attack data set.

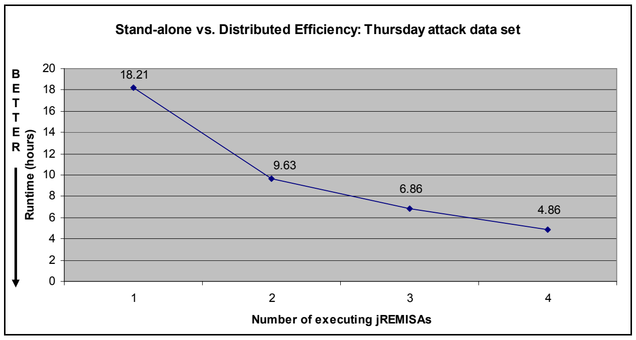
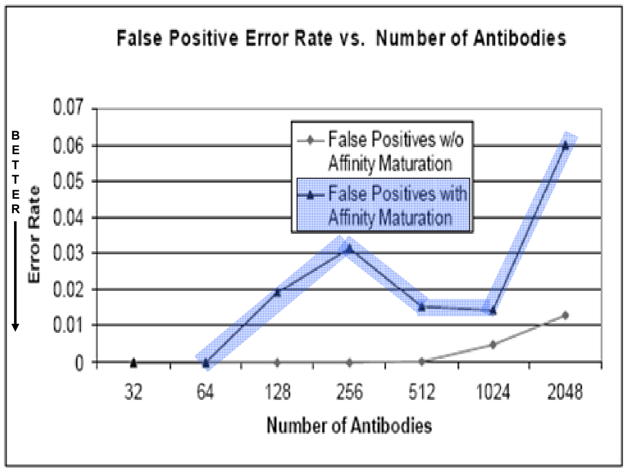
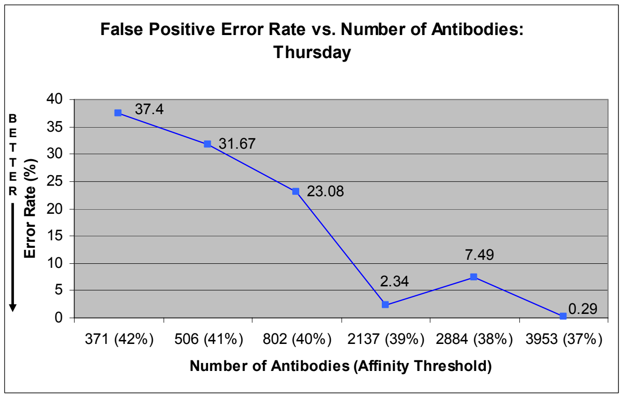


Figure 14. Data decomposition-based distributed execution: efficiency vs. number of executing jREMISAs.



(a)



(b)

Figure 15. false positive error rate trend: (a) Warthog [15] vs. (b) jREMISA.

5. CONCLUSION

The hypothesis was tested against four measurable goals which were all validated based on our experimentation. REALGO and MISA were successfully migrated to Java (and integrated via software engineering principles into jREMISA). Our algorithm achieved an average 83.37-85.45% *self* classification and 96.57-99.90% *non-self* classification rate for a 39% affinity threshold. We observed a patterned Ab hypervolume between 0-5% above this threshold (making their hypervolume 39-44%) and Ab broadcasting and receipt. In addition, we validated our algorithm as performing better in at least one way over another against the same data set. As a bonus, jREMISA detected four attacks, ranging from one to 10401 non-consecutive packets, with a 0% false positive rate. From this, we declare this IDS as possibly the first validated AIS-inspired MOEA applied to the ID problem domain.

Use of software engineering design paradigms provides an easily extendable IDS package. Future jREMISA advances include com pleting the KDD Cup 99 data set facilitation, evolving the Ab hypershape and exploring advanced *negative selection* methodologies.

6. REFERENCES

[1] Symantec Internet Security Threat Report, Trends for January 1, 2004 – June 30, 2004, Volume VI, September, 2004, eval.veritas.com/mktginfo/enterprise/white_papers/ent-whitepaper_symantec_internet_security_threat_report_vi.pdf

[2] Michalewicz, Z., Fogel, D., *How to Solve It: Modern Heuristics, Second Edition*, Springer 2004.

[3] J.. Dréo, A. Pétrowski, P. Siarry, and E. Taillard, *Metaheuristics for Hard Optimization: Methods and Case Studies*, Springer-Verlag, Berlin, Germany, 2006.

[4] Haag, C.R., *An Artificial Immune System-inspired Multiobjective Evolutionary Algorithm with Application to the Detection of Distributed Computer Network Intrusions*, M.S. Thesis, Graduate School of Engineering and Management, Air Force Institute of Technology, WPAFB, Dayton, OH, March, 2007

[5] Coello, C., Cortés, N., *Solving Multiobjective Optimization Problems Using an Artificial Immune System*, Genetic Programming and Evolvable Machines, Vol. 6, pp.163-190, 2005.

[6] Edge, K., Lamont, G., Raines, R., *A Retrovirus Inspired Algorithm for Virus Detection & Optimization*, GECCO '06, July 8-12, 2006.

[7] McGee, P., Building Better Antibody Therapeutics, Drug Discovery & Development, www.dddmag.com/ShowPR.aspx?PUBCODE=090&ACCT=1600000100&ISSUE=0701&RELTYPE=DEV&PRODCODE=00000000&PRODLETT=AG&CommonCount=0.

[8] Middlemiss, M., *Positive and Negative Selection in a Multilayer Artificial Immune System*, Information Science Discussion Paper Series, No. 2006/03, University of Otago, January, 2006.

[9] Coello, C., Van Veldhuizen, D., Lamont, G.B., *Evolutionary Algorithms for Solving Multi-Objective Problems*, Springer 2002.

[10] De Castro, L.N., Timmis, J., *Artificial Immune Systems: A New Computational Intelligence Approach*, Springer 2002.

[11] MIT Lincoln Laboratory–DARPA Intrusion Detection Evaluation, www.ll.mit.edu/IST/ideval/.

[12] Mahoney, M., Chan, P., *An Analysis of the 1999 DARPA/Lincoln Laboratory Evaluation Data for Network Anomaly Detection*, Technical Report CS-2003-02, Computer Science Department, Florida Institute of Technology, 2003.

[13] Harmer, P., Williams, P., Gunsch, G., Lamont, G.B., *An Artificial Immune System Architecture for Computer Security Applications*, IEEE Transactions on Evolutionary Computation, Vol. 6, No. 3, June 2002.

[14] Williams, P., *WARTHOG: Towards a Computer Immune System for Detecting "Low and Slow" Information System Attacks*, AFIT Master's Thesis, March, 2001