A Nonself Space Approach to Network Anomaly Detection

Marek Ostaszewski[1], Franciszek Seredynski[1,2,3], and Pascal Bouvry[4]

[1]University of Podlasie
Institute of Computer Science
Sienkiewicza 51, 08-110 Siedlce, Poland
marekostaszewski@o2.pl

[2]Polish-Japanese Institute
of Information Technology
Koszykowa 86,02-008 Warsaw, Poland
sered@ipipan.waw.pl

[3]Polish Academy of Sciences
Institute of Computer Science
Ordona 21,01-237 Warsaw, Poland
sered@ipipan.waw.pl

[4]Luxembourg University
Faculty of Sciences,
Technology and Communication
6 rue Coudenhove Kalergi,
L-1359 Luxembourg-Kirchberg, Luxembourg
pascal.bouvry@uni.lu

Abstract

The paper presents an approach for the anomaly detection problem based on principles of immune systems. Flexibility and efficiency of the anomaly detection system are achieved by building a model of network behavior based on self-nonself space paradigm. Covering both self and nonself spaces by hyperrectangular structures is proposed. Structures corresponding to self-space are built using a training set from this space. Hyperrectangular detectors covering nonself space are created using niching genetic algorithm. Coevolutionary algorithm is proposed to enhance this process. Results of conducted experiments show a high quality of intrusion detection which outperforms the quality of recently proposed approach based on hypersphere representation of self-space.

1 Introduction

Security of information channels is crucial in times, when so much depends on data exchange. Value of information is high and this means, that there will be someone, who will try to get hold on important data without permission. Internet is an environment, where fluent data exchange takes place, and where systems and information are vulnerable to attacks and exploits. Indicating activities, that may pose a possible threat is an important task, which applications dealing with security problems must face. A common approach to detection of unwanted activity is based on signatures of various attacks [9]. An application observes several parameters of network traffic, and raises an alarm, when values match with those in a signature, dealing with threats in reactive way. Only known intrusions can be detected, modified patterns of attack, if they are not in database of signatures, will be treated like normal traffic.

A solution that can be applied to overcome these problems is an artificial immune system (AIS) [3, 4, 10] - a set of methods adapted from natural immune systems of various species. One of recently proposed approaches to a network anomaly detection is based on a description of self behavior using hypersphere structures [2].We propose a hyperrectangle structure as more precise definition of a legitimate traffic, alon with coevolutionary-based mechanisms, which can be used to enhance the process of anomaly detection, similarly to vaccines used widely in medicine to boost efficiency of a human immune system.

The structure of the paper is the following. The next section contains short definition of an artificial immune system along with a description of mechanisms useful to solve stated problems. Principles of construction of an effective model of network behavior, called self space, are presented in section 3. Genetic algorithm used to generate detectors of nonself space, along with niching and coevolutionary techniques are described in section 4. Section 5 contains results of experiments which were carried out using sets of network data collected at MIT. Finally, last section brings forward conclusions and further possibilities of development of the presented approach in the network anomaly detection domain.

2 Immunological approach to anomaly detection problem

Mechanisms developed to protect organisms against variety of threats are highly effective in detecting abnormalities and have interesting computational features. Artificial immune systems (AIS) are attempt to adapt mechanisms and techniques of natural immune systems, and obvious conclusion is applying them into security domain. Features such as distributed and threshold detection, simplicity and uniqueness [4, 10] are very attractive from security point of view. Especially mechanisms of generalization and detection based only on self structures seem to be a good solution to problem of detecting various abnormalities without patterns of possible attacks. An immune system utilizes space of its own cells (self space) to generate detectors of space complementary to it. Detectors are matched against cells from organism during their generation, and those detectors matching are destroyed. This process has been adopted in the field of computer science and is called the negative selection algorithm [2, 3].

3 Proposed self space construction

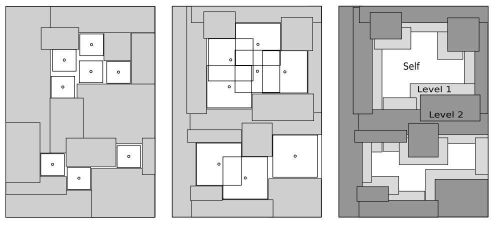
A description of a regular network traffic for the anomaly detection system is crucial to effectiveness of detection capabilities. More accurate model of a traffic behavior assures that anomalies are indicated with greater precision. A normal traffic can be characterized by a set of monitored parameters, and their values can be recorded to describe a regular behavior. Gathered data is used to construct self space, which contains all normal states and allows to begin a process of generation of detectors. For further analysis of this problem some definitions have to be assumed [2]:

- **Normalized space:** values of monitored parameters can have different ranges, therefore they are normalized to $[0.0, 1.0]$ range, and the set of n monitored parameters gives n-dimensional space $[0.0, 1.0]^n$ (S).
- **State of system:** a vector containing normalized values of monitored parameters, describing system in a given moment. A state of the system is defined as $\vec{x} = (x^1, \ldots, x^n) \in S$.
- ***Self* - *Nonself* classification:** Self space, henceforth called *Self* $\in S$ represents a space containing all recorded regular states is used to construct nonself space, henceforth called and defined *Nonself* $= S -$ *Self* .The classification process is defined as :

  $$classify(\vec{x})\begin{cases} Self, \ if \ \mu(\vec{x}) > st \\ Nonself, \ if \ \mu(\vec{x}) \leq st \ . \end{cases}$$

  where function $\mu : \vec{x} \rightarrow [0.0, 1.0]$, describes a certain level of similarity to states to *Self*, and st defines required level of similarity.
- **Anomaly detection:** it is based on assessing, if given state belongs to *Nonself*, with detectors constructed to cover all space excluding *Self*. Affinity can be adjusted by changing a value of threshold in borders of interval $[0.0, 1.0]$.

Self space construction is a process where a regular traffic, required for a generation process, influences the quality of a detector set. Every state recorded becomes automatically an element of *Self*. To implement threshold affinity and apply "similarity threshold" described above, a definition of variability parameter is needed. An assumption that certain states are similar to recorded normal states allows to define a level of deviation from given data. Figure 1 illustrates pro-



(a) (b) (c)

Figure 1. Self space and detector sets for (a) low and (b) high value of v, and (c) detection levels in summary

posed in the paper hyperdimensional space construction, where a dimension (in this case 2D) depends on a number of parameters taken under consideration. Variability parameter (henceforth called $\boldsymbol{v}$) describes the similarity degree, therefore *Self* can be defined as a subspace that contains all states with some level of resemblance to recorded normal states. For a given state of the system $\vec{x} = (x^1, \ldots, x^n)$ and variability parameter $\boldsymbol{v}$ the structure of self space for this vector takes form of two vectors, $low = (x^1 - \boldsymbol{v}, \ldots, x^n - \boldsymbol{v})$ and $high = (x^1 + \boldsymbol{v}, \ldots, x^n + \boldsymbol{v})$, where a pair low^j and $high^j$ describes interval for the parameter j considered as normal. A state of the system fitting in all intervals belongs to normal space. An analysis of generation process for *low* and *high* vectors allows to indicate that values of $\boldsymbol{v}$ may be different and dependant on a parameter they are applied to. Henceforth $\boldsymbol{v}$ is defined as a vector of values, $\boldsymbol{v} = (\boldsymbol{v}^1, \ldots, \boldsymbol{v}^n)$, and *Self* structures are constructed [7] using this vector, $low = (x^1 - \boldsymbol{v}^1, \ldots, x^n - \boldsymbol{v}^n)$ and $high = (x^1 + \boldsymbol{v}^1, \ldots, x^n + \boldsymbol{v}^n)$.

Figure 1(a) shows *Self* constructed for relatively small value of $\boldsymbol{v}$, and effects in large volume of *Nonself*. A small number of states similar to normal effects in greater possibility of false alarm, but also guarantees less risk of classifying an anomaly as a self state. Figure 1(b), on the other hand, illustrates *Self* offering grater both tolerance and risk of misjudging anomaly as a regular traffic element. Figure 1(c) presents a comparison of *Self* constructed for two $\boldsymbol{v}$ values, and shows, that detector sets are different, so as *Nonself* classification process. Therefore $\boldsymbol{v}$ allows to construct more than one model of security by applying different detector sets for different circumstances, including cases of many users or cycles in network traffic.

Applying multiple detector sets to achieve levels of security is done by assessing every state of the system with all generated sets and defining, which one of them raises an alarm [2, 7]. Levels of security have to be sorted by growing value of $\boldsymbol{v}$ used to construct *Self*. A level with highest number defines a set with largest volume of *Self* used to generation of detectors, thus most tolerant to abnormalities. With more than one set *Nonself* classification process is changed and range of values returned is widened with $classify(\vec{x}) = max(\{level(DSets)\} \cup \{0\})$, where $level(DSets)$ returns highest index of detector sets that raise an alarm.

System states used for *Self* construction are recorded once in defined period of time and create time series. The anomaly detection process is highly dependant on frequency of recordings as well as on processing of monitored values. Therefore, to improve the precision of *Self* construction process, sliding window method [2,7] is applied. Time series of recorded values $R = r_1, r_2, \ldots, r_n$ for monitored parameters with a certain frequency are transformed to window series $W = \{(r_1, \ldots, r_w), (r_2, \ldots, r_{w+1}), \quad \ldots \quad, (r_{n-w+1}, \ldots, r_n)\}$, and may also take the form $W = \{rw_1, rw_2, \ldots, (rw_m)\}$, where $rw_i = \frac{r_i + r_{i+1} + \cdots + r_{i+w}}{w}$ and $m = n - w + 1$ with w as a parameter describing a window size. Applying sliding window method allows to define notation of the system state as $\vec{x} = (rw^1, rw^2, \ldots, rw_i^n)$.

The values of parameters are measured every period of time and during the process two subsets of states are built, training set and testing set. The parameter $\boldsymbol{v}$ is calculated from the standard deviation taken from distances between states belonging to training set, and testing set is used to adjust *Self*. It is achieved by checking, if any of created structures intercepts states from testing set. If no, a new structure based on uncaught state is created, and the process is started from the beginning.

4 Detector set generation

An anomaly detection is a process of monitoring *Nonself* with detectors of hyperrectangular structure. *Self* elements are created from recorded states with $\boldsymbol{v}$ and their number is potentially large, when long period of time is used to model *Self*. Therefore, it is desirable to develop a set of structures (detectors) efficiently covering *Nonself*, with volume as large as possible and small overall number. The detector set is described by conditions [2] and takes form $DSet = \{D_1, \ \ldots \ , D_k\}$, where $D_i : \ if \ Condition \ then \ Nonself$, and $Condition = x^1 \ \in \ [low_i^1, high_i^1] \wedge \ \ldots \ \wedge \ x^n \ \in \ [low_i^n, high_i^n]$. A single detector classifies if a given state is captured by covered space, defined in conditional part by two vectors, *high* and *low*. The construction of a space from junction of intervals in every dimension is the same as in the case of *Self* elements (see section 3).

4.1 Niching genetic algorithm

Creating a detector set is a multimodal problem with many objectives, and to efficiently deal with it genetic algorithm (GA) was used [2, 7]. The goal is to cover the space as large as possible with detectors that avoid *Self* elements and cover unique space, without overlaying the same *Nonself*. A population of GA consists of individuals constructed from pair of vectors, reflecting *high* and *low* values of conditional part of detector. Therefore, real-coded GA [6] have to be applied to this problem, as long as vectors of real values (from $[0.0, 1.0]$ interval) are used. GA schema goes as follows [2, 7]:

```
DetectorSet ds=$null$; numAtt=0; numDet=0;
**while** numDet < maxDet **and** numAtt < maxAtt
  $runGA(W, \boldsymbol{v})$;
```

```
    $D \leftarrow$ *best evolved detector*;
    $fit = calculateFitness(D)$;
    **if** $fit >$ `minFit`
        $ds.addDetector(D)$;
        $numAtt = 0$;
    **else**
        $numAtt = numAtt + 1$;
**end while**
**return** ds,
```

where W is learning set consisting of *Self* states, $\boldsymbol{v}$ is variability parameter used for *Self* structures construction, minFit is minimal value of fitness expected from evolved detector, maxAtt and numAtt are, respectively, maximal and the current number of attempts to evolve single detector and maxDet and numDet are, respectively, maximal and the current number of detectors in the detector set.

GA have to deal with multiobjective problem and for that reason sequential niching algorithm [1] has been applied to provide GA with ability to cover different subspaces of *Nonself* with largest possible volume. Sequential niching GA goes as follows:

```
SolutionSet solSet = null;
**while not** solSet.*satisfied*()
Population = *runGA*();
**for each** Individual $\rightarrow$ ind **from** Population
    $Individual.calculateFitness()$;
    **for each** *Individual* $\rightarrow$ *sol* **from** *solSet*
        $sim = calculateSimilarity(ind, sol)$;
        $Individual.decreaseFitness(sim)$;
    **if** *ind.fitness*() $>$ `minFitness`
    $solSet.add(ind)$;
**end while**
**return** solSet.
```

According to the schema presented above, with every run of niching genetic algorithm (NGA) a fitness function is modified, to focus process of searching for new detectors on subspaces not covered by detectors already evolved. The fitness is calculated by taking under consideration three factors, as shown below:

- **Volume calculation:** a volume of a given detector is calculated as follows

$$Volume(D) = \prod_{i=1}^{n}(high^{i} \ - \ low^{i}),$$

  where *high* and *low* are elements of vectors on positions corresponding to parameters used.

- **Overlaying with *Self* structures:** a volume of the space overlayed with *Self* is calculated as follows

$$SelfOverlay(D) = \sum_{i=1}^{m}(D \ \cap \ xs_i),$$

  where xs_i is a structure created from a state taken from the learning set W and variability parameter $\boldsymbol{v}$, as described in Section 3.

- **Overlaying with already developed detectors:** a volume of the space overlayed with already developed detectors is calculated as follows

$$DetectorOverlay(D) = \sum_{i=1}^{k}(D \ \cap \ D_k).$$

The fitness of a single detector is calculated from the equation

$$Fitness(D) = Volume(D) - (SelfOverlay(D) + DetectorOverlay(D)) \ .$$

4.2 Coevolution in detector generation process

Coevolutionary algorithms are relatively new research area in the field of evolutionary computation. The basic idea is taken from the world of Nature, where two or more coexisting species are constraining one other to evolve better features. Among many coevolution models one seems to be useful to the detector generation problem. *Predator - prey* paradigm [8] describes a model, where individuals of one type (predators) are trying to catch another (preys). The population of the first species develops features that allow it to catch its prey easily, and attributes of the second one evolves to make escape from a predator possible. Some definitions [8] have to be assumed to apply coevolutionary algorithm to detector generation problem:

- **Constraint Satisfaction Problem (CSP):** a class of problems effectively solvable by coevolutionary algorithms. The first of two coevolving populations is a population of solutions (henceforth called *Solutions*), and the other one is a population of constraints (henceforth called *Constraints*) that *Solutions* have to fit. Because of their static nature, *Constraints* cannot evolve but their fitness can be also evaluated.

- **Encounter:** a confrontation between individuals from *Solutions* and *Constraints* results in the victory of one and the loss of the other. A *Solution* wins if it fits given *Constraint*, and loses if *Constraint* cannot be satisfied by a given solution.

- **LifeTime Fitness Evaluation (LTFE):** in opposite to the classic GA, every individual is tested multiple times and has a list of his encounters that changes, as it might be said, through its lifetime. A fitness is calculated on the basis of confrontations with individuals from coevolving population. LTFE regulates a number of confrontations, and thus, affects calculated fitness. Probability of choice to encounter depends on fitness, therefore, even if *Constraints* cannot evolve, winning ones are tested more frequently against *Solutions*.

To apply the coevolutionary algorithm to the detector generation problem, the second (coevolving) population has to be assumed. To define proper constraints for detectors, a set of anomalies have to be constructed. Similarity to *predator - prey* model resolves to treating detectors as individuals trying to intercept individuals from a set of anomalies. An anomaly (individual) is defined as a certain state from *Nonself* space in the form of a vector $a = (a_1, ..., a_n)$, where a_i is a value for corresponding parameter. Encounter between detector and anomaly resolves to assessment, if anomaly lays inside subspace covered by a detector. The detector wins the encounter if it intercepts a given anomaly, otherwise constraining state is the winner. A fitness of given detector is finally calculated after running the coevolutionary algorithm as follows:

$$Fitness(D) = Fitness(D) + EncounterHistory(D) * Fitness(D) \ ,$$

where $EncounterHistory(D)$ is a function returning summarized effect of all encounters for given detector.

5 Experimental results

A number of experiments have been performed to find out the effectiveness of nonself approach to the anomaly detection problem, based on hyperrectangular *Self* structures and involving the coevolutionary algorithm. *Self* space for this experiment was constructed using data gathered at MIT [5]. The first week of the collected network traffic using *tcpdump* were unaffected by anomalies, and for one of chosen computers with IP 172.16.114.50 (marx) states of network parameters was collected and used for *Self* structures development. Network traffic parameters used for the experiments are a number of bytes per second (P1), a number of packets per second (P2) and a number of ICMP packets per second (P3), and a state of the system takes form of the vector *state* $= \{P1,\ P2,\ P3\}$ involving values measured for given IP address. *Self* structures for GA were created from the first week data ina proportion of 70:30 of trining to testing sets.

5.1 Experiment #1 - Generation of detector set for different v values

The first experiment concerned running GA to develop self space and detector sets for different $\boldsymbol{v}$ levels. NGA was run with max. num. of runs equal to 20, max. num. of attempts to evolve rule equal to 15, a number of generations equal to 750 and a population size equal to 100. Following GA operators have been used [6]:tournament selection with tournament size 2, vector crossover, gaussian mutation with probability 0.1 and border mutation with border values 0 and 1, and probability 0.01.

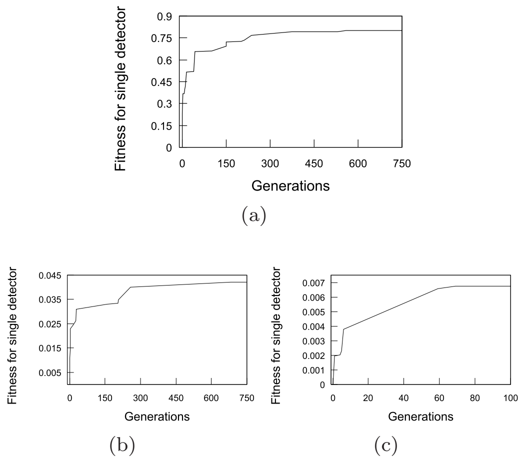
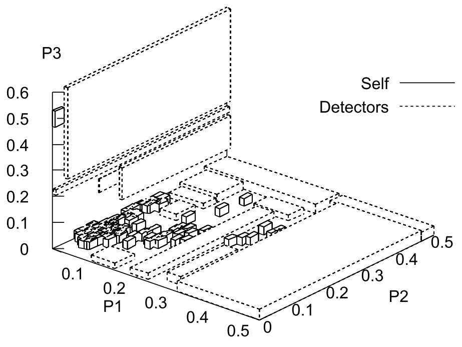
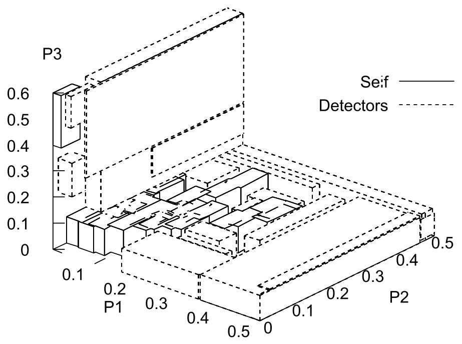


Figure 2. Fitness evaluation for (a) 1st, (b) 3rd and (c) 6th run of niching genetic algorithm

NGA characterizes itself with multiple runs and consequently modifies the range of values of the fitness function. Figure 2 shows changes of the fitness function during evolving single rules in 3 runs of GA. Figure 2(a) shows evolving a rule in the first run, Figure 2(b) in the third and Figure 2(c) in the sixth run. In each next run of NGA the space to be covered is smaller and individuals not overlaying already developed detectors are harder to find. One can notice (see Figure 2), that the range of fitness function corresponding to a volume of *Nonself* detector decreases in each run. *Self* was created for different values of $\boldsymbol{v}$ to compare space covered by detector sets generated using NGA. Figure 3 shows comparison between covered *Nonself* space by detectors developed in the case of two values of $\boldsymbol{v}$. Figures 3(a) and 3(b) show only a part of generated detectors, the total size of detector set in both cases is 15.



(a)



(b)

Figure 3. Self space and detector sets for (a) 0.3v and (b) 1.0v

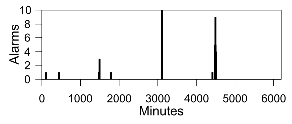
5.2 Experiment #2 - Anomaly detection process for different v and w values

Detector sets generated by NGA have been used to anomaly detection process on MIT data [5]. The second week contains five simulated attacks, one for every day of the network traffic, as shown in Table 1.

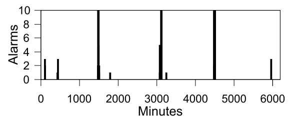
Table 1. MIT Second week attacks

Day	Name	Type	Start	Duration
1	Back	DoS	9:39:16	00:59
2	Portsweep	Probe	8:44:17	26:56
3	Satan	Probe	12:02:13	02:29
4	Portsweep	Probe	10:50:11	17:29
5	Neptune	DoS	11:20:15	4:00

Effects of monitoring the anomaly detection process are presented in Figures 4 and 5. Figure 4(a) presents anomalies detected by the set of detectors for 0.3$\boldsymbol{v}$ and w=1, and Figure 4(b) for w=3. Figure 5 presents detection effects for the set with 1.0$\boldsymbol{v}$ and w=1 (Figure



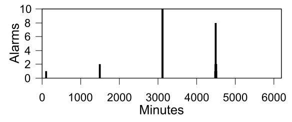
(a)



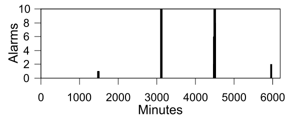
(b)

Figure 4. Attacks detected using 0.3v and w equal to (a) one and (b) three

5(a)), and w=3 (Figure 5(b)). The analysis of this Figure indicates greater sensitivity of detectors constructed for Self with w=3, what can be explained, if temporal patterns are taken under consideration. The window size greater than one can intercept time dependencies between preceding and succeeding states, what is impossible for detectors based on w=1.



(a)



(b)

Figure 5. Attacks detected using 1.0v and w equal to (a) one and (b) three

The tolerance of detectors depends on $\boldsymbol{v}$ level and higher values of it cause less number of raised alarms. On one hand, low level offers precise anomaly detection, but on the other hand, it causes false alarms and constructs inflexible model of the system. A high level of $\boldsymbol{v}$ assures more tolerant detector set, with loss of precision in the detection process. Nevertheless, Figure 5 indicates that both detector sets together found all of five attacks, though relatively high level of $\boldsymbol{v}$. As one may notice, some of peaks in these Figures (1485 and 4491 minutes in Figure 4(a), 4498 in Figure 5(a) and 1487 in Figure 5(b)) are groups of multiple lines. After the analysis of Table 1 it is possible to notice, that duration of attacks 2 and 4 was relatively long and the system raised more than one alarm during monitoring process. Those attacks on Figures 4(b) and 5(b) are displayed as groups of lines with alarm number equal to or more than 10, in effect they look like bold lines. An interesting fact can be observed after the comparison of alarms raised for each attack - obviously probe attacks are indicated stronger than DoS attacks. Additionally, a window size seems to have optimal values for every attacks, as it may be observed in Figure 5, where w=1 manages to capture the first attack, but misses the last one, and for w=3 the first attack remains unreported, but last one is signalized. Interesting case is the attack number three, indicated with a great strength in every parameter configuration, though relatively short duration time. It may be explained with dependencies between an attack type and the structure and parameters used in *Self* construction.

5.3 Experiment #3 - Anomaly detection process for spherical construction of *Self*

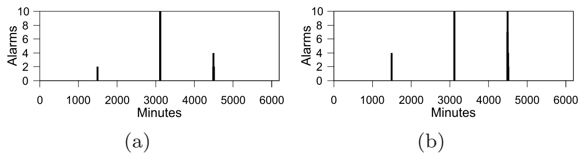


Figure 6. Sphere detectors efficiency for 0.5v and w equal to (a) one and (b) three

The approach presented in [2] is based on detector sets developed for *Self* constructed from hyperspheres, which are created using a given state of the system as a center and single value of $\boldsymbol{v}$ as a hyperradius. Experimental results presented in [2] show, that hyperspherical design of *Self* was sufficient to catch four attacks at most, with a window size equal to 3, and three with a window size equal to 1. Experiments carried out for this paper include also *Self* construction with hyperspheres, and the detector generation for this purpose. Figure 6 presents detected anomalies for $0.5\boldsymbol{v}$ with w=1 (Figure 6(a)) and w=3 (Figure 6(b)). Due to the different process of $\boldsymbol{v}$ calculation results differ from those presented in [2], nevertheless in both cases with hyperspherical *Self* construction system was unable to discover all five attacks. The comparison with detector sets based on hyperrectangular structure indicates, that the approach presented in this paper is more precise and offers better performance in the anomaly detection.

5.4 Experiment #4 - Coevolution effectiveness for randomly generated set of anomalies

Mechanism of coevolution has been tested to check, if there is a possibility of applying it to enhance the detector generation process. A set of 1000 randomly generated vectors from *Nonself* has been assumed as the second population coevolving with the population of generated detectors. After generation of a detector set using both coevolutionary and classic NGA, it has been tested against the set of anomalies, to check how many of them have been caught.

Table 2. Performance of detector sets for randomly generated set of anomalies

Generations	LTFE	Anomalies caught
100	5	866
	10	899
	20	**928**
	—	895
300	5	981
	10	962
	20	**982**
	—	972
500	5	963
	10	961
	20	973
	—	**987**

Table 2 presents the results of performed experiment, and the best of them has been highlighted with bold font. Detector sets have been developed for three different number of generations and for three different values of LTFE parameter. There has been generated additionally one detector set with standard NGA, without coevolution mechanism, marked with "—" symbol in LTFE column. A number of anomalies that have been caught differ, but an advantage of coevolutionary NGA is insignificant, and for 500 generations classic NGA surpasses one with coevolution. These results are possible to explain by the way of random generation of anomaly set. Distribution of states in *Nonself* for this case is regular, and classic NGA, while trying to cover largest space possible, intercepts also states generated without any specialization.

5.5 Experiment #5 - Coevolution effectiveness for specialized set of anomalies

For the detector set developed with classical NGA there has been generated an alternative set of anomalies, which were specialized by generating them in *Nonself* space and beyond subspace covered by this given detector set. The distribution of anomalies from the second set is irregular and elements from it are in areas, where detectors are harder to develop, for example, in small subspaces between *Self* structures. Results of conducted experiments are presented in Fig-

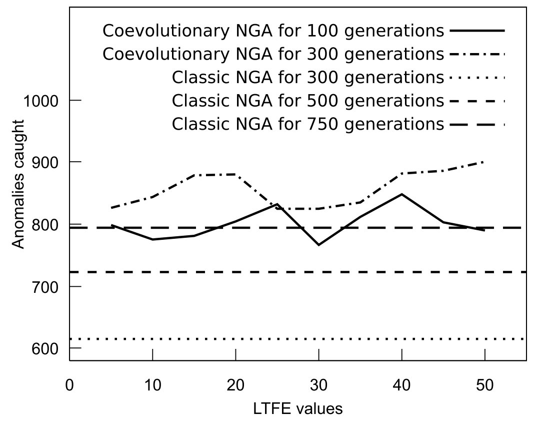


Figure 7. Copmarison of coevolution efficiency with classic NGA approach

ure 7. The second experiment including coevolution shows significant advantage of coevolutionary NGA and proves that the detector generation process can be stimulated by the additional population. Even relatively small number (100) of generations allowed the coevolutionary NGA to obtain better results than classic NGA with 750 generations and, consequently with less computational cost. Futhermore, one can notice, that the efficiency of coevolution is LTFE dependent, but also depends on number of generations, and results for more than 300 generations are worth further study. Another concern is a tradeoff between coevolution efficiency and detector fitness, calculated on the basis of various factors (see Section 4), which may cause worse performance of coevolutionary detectors development.

6 Conclusion

Results of conducted experiments indicate, that detectors generated with NGA proved to be effective, and hyperrectangular *Self* structures construction made precise detection process possible. Presented approach is efficient and allows to capture all five simulated attacks in MIT data. Hyperspherical design of *Self*, as presented in [2] made possible to catch four of five attacks, and applied for *Self* development and NGA presented in this paper, three of five. It also has been shown that coevolutionary mechanism can enhance the detectors generation process and in the result can make detection process more effective against given patterns of attack. Gathering data about some of those patterns in the form of coevolving sets can give in effect detector sets containing knowledge about attack subspaces. This mechanism can be compared to vaccine, that makes natural immune system more effective against certain illnesses.

Variability parameter $\boldsymbol{v}$ has been proven to be important factor in the detector development process by influencing on *Self* volume. This parameter is responsible for tolerance and false alarms levels. Therefore, an algorithm of calculating $\boldsymbol{v}$ from learning is very important in the attempt to improve the detection ability of a system.

Further research may involve different parameter types and greater number of them. Analysis of the detection process data shows, that the system performs very effective in the case of Satan attack in a relatively short duration time, and has more problems with attacks like Portsweep or Neptune, although their duration last several times longer (see Table 1). Dependencies between parameters and attack types are also promising field of research.

References

[1] D. Beasley, D. R. Bull, and R. R. Martin. A sequential niche technique for multimodal function optimization. *Evolutionary Computation*, 2(1):101–125, 1993.

[2] D. Dasgupta and F. González. An immunity-based technique to characterize intrusions in computer networks. *IEEE Transactions On Evolutionary Computation*, 6(3):281–291, 2002.

[3] S. Forrest, A. S. Perelson, L. Allen, and R. Cherukuri. Self-nonself discrimination in a computer. In *Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy*, pages 202–212, 1994.

[4] P. K. Harmer, P. D. Williams, G. H. Gunsch, and G. B. Lamont. An artificial immune system architecture for computer security applications. *IEEE Transactions On Evolutionary Computation*, 6(3):252–280, 2002.

[5] http://www.ll.mit.edu/IST/ideval/index.html.

[6] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer, 1992.

[7] M. Ostaszewski. Anomaly detection in computer networks based on artificial immune systems (in polish). Master's thesis, University of Podlasie, Siedlce, Poland, 2005.

[8] J. Paredis. Constraint satisfaction with oevolution. In *New Ideas in Optimization*, pages 361–366. 1999.

[9] M. Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th Systems Administration Conference*, pages 361–366, 1999.

[10] S. T. Wierzchon. *Artificial immune systems. Theory and application (in polish).* Exit, Warsaw, Poland, 2001.