Pak. j. life soc. Sci. (2026), 24(1): 254-266 E-ISSN: 2221-7630;P-ISSN: 1727-4915

Pakistan Journal of Life and Social Sciences
www.pjlss.edu.pk

https://doi.org/10.57239/PJLSS-2026-24.1.0021

RESEARCH ARTICLE

A Model of an Artificial Immune System for Recognizing Destructive Effects on Information Infrastructure

Olga Safaryan[1*], Irina Alferova[2], Anastasiya Molodcova[3]

[1, 2, 3] Don State Technical University, Rostov-on-Don, Russia

ARTICLE INFO

Received: Jan 19, 2026

Accepted: Apr 4, 2026

Keywords

Immune System Model
Anomalies
Attacks
Information System

***Corresponding Author**

safari_2006@mail.ru

ABSTRACT

In the context of digitalization, the increasing complexity and duration of cyberattacks render traditional deterministic security methods ineffective. This paper proposes developing an artificial immune system model, based on the principles of biological immunity, for the early detection of and response to prolonged and adaptive threats. The artificial immune system will provide adaptive, distributed, and self-learning incident response, enhancing the resilience of information systems and reducing the risk of long-term undetected attacks. The proposed approach is justified by the scale of modern threats and aims to become an alternative paradigm to classical cybersecurity measures.

INTRODUCTION

In the modern world of digitalization, one of the main problems is ensuring information security. Attackers use more "protracted" attacks instead of one-time, simple attacks on the system, which are more difficult to prevent at an early stage. The development of an artificial immune system model based on the principles of an artificial immune system will provide ample opportunities to counter attacks of this kind.

The modern cyber threat landscape is characterized by an unprecedented and rapidly increasing scale of attacks, which makes traditional, deterministic approaches to information security less effective. This scale is reflected in several critically important aspects, which together form a convincing justification for the transition to alternative, only developing defense paradigms, such as the artificial immune system (AIS) [Bryukhomitsky, 2019; Chernyshev et al., 2014].

Firstly, there is a quantitative and spatial scale of attacks. Due to the unstable geopolitical situation in the country, the number of successful cyber attacks on organizations and individuals only continues to grow, as shown in Figure 1.

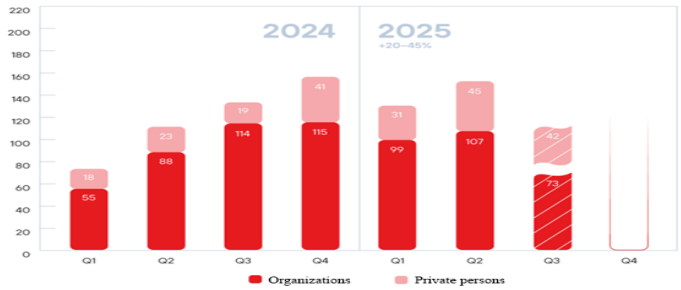

Figure 1. Dynamics of successful cyber attacks on organizations and individuals in Russia

The volume of malware generated is estimated at millions of new samples every day, and the attacks themselves are global and distributed. Traditional signature methods, which require preliminary study and creation of a threat template, physically cannot keep up with such a flow.

Secondly, the most important factor is the scale of complexity and the speed of threat evolution. So, in the form of a pie chart, the distribution of triggers by type of threat by quarter in 2025 is shown in Figure 2.

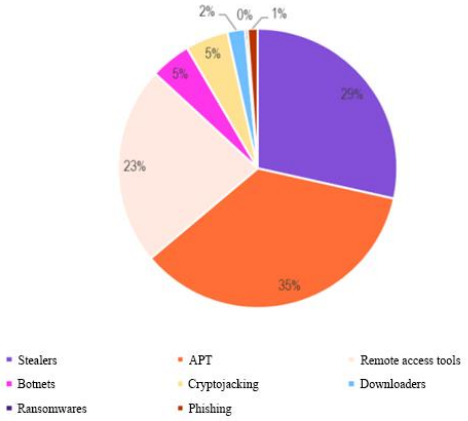

Figure 2. The spread of threat-type triggers in the first half of the 3rd quarter of 2025

Modern attacks, such as targeted advanced persistent threat or attacks using less file methods, are characterized by a high degree of adaptability, the ability to disguise themselves as legitimate activity and long-term, "quiet" impact. They evolve during the attack itself, changing tactics and tools. The hard-coded rules and policies of traditional security systems are not able to dynamically adjust in response to such changes.

Thirdly, the scale of the damage and the cascading effect should be considered. In conditions of critical infrastructure, a single successful penetration can lead to catastrophic cascading failures. An approach is needed that is able to identify threats at the early stages of their life cycle, before irreversible consequences occur.

Finally, there is the operational and resource scale of the problem. For large organizations, managing thousands of security devices and analyzing terabytes of logs becomes an impossible task for humans. Autonomy is required.

Thus, the analysis of the scale of modern cyber attacks — in its quantitative, evolutionary, spatial and operational dimensions — clearly indicates the exhaustion of the possibilities of purely deterministic, static protection models. The development of a software tool embodying the principles of an artificial immune system (distributed anomaly detection, immune memory, adaptive learning, autonomy and resilience) seems to be a logical and necessary response to the challenges of modern times. Such a system is capable of providing not only protection against known threats, but also creating a vibrant, evolving immunity of the digital environment, capable of resisting large-scale, complex and constantly changing cyber attacks.

The purpose of this work is to develop an artificial immune system algorithm model for automatic recognition of destructive effects on the information infrastructure.

Method

Building a model of an artificial immune system

Given the specifics of the task, the software being developed must necessarily have the following functionality: monitoring, logging, and making decisions about responding to security incidents based on the system metrics obtained. The program should be based on an algorithm for the negative

selection of AIS, which is able to ensure high accuracy in responding to abnormal behavior of the system.

The artificial immune system (AIS) is a computing system [Abramov, 2009; Birulia, 2024; Skobtsov, 2022] based on principles, mechanisms and processes borrowed from the natural immune system (AIS) of vertebrates, which naturally follows from the name. EIS is a highly adaptive, self-organizing and distributed defense system capable of recognizing and destroying pathogens (harmful antigens), while maintaining neutrality towards the body's own cells.

In the process of transferring natural immunity into the form of an automated computing system, the following key features were formulated that an AIS should possess:

- friend-foe recognition – the ability of a system to distinguish its (normal) elements from malicious or abnormal ones;
- multilevel protection is the presence of innate immunity, i.e. a rapid response of the system to destructive interference from the outside, and acquired immunity: a slow response, however, which is a reaction to a previously unknown attack, the impact of which is neutralized in any case.;
- distribution and lack of centralized management – like natural immunity, protective functions are distributed among a multitude of cells (lymphocytes) that act independently of each other;
- adaptivity and learning – the acquired immune system is able to "learn" from encounters with pathogens (such as viruses), forming an immunological memory for a faster and more effective response to future threats.;
- neutrality (towards one's own "cells") is the ability not to react to one's own, safe body structures trained in the process of negative selection of lymphocytes.

The negative selection algorithm is a biologically inspired method that simulates the maturation process of T-lymphocytes in the thymus [Zhukov & Salamatova, 2014]. Within the framework of an artificial immune system for detecting anomalies, this algorithm is used to generate a population of detectors capable of distinguishing between normal ("own") and the abnormal ("alien") behavior of the system.

The principle of operation of the algorithm can be divided into two stages: the creation of a database of abnormal detectors and their detection. At the first stage, detectors are randomly generated using a set of "their own" as input data [Adieva & Muratbek, 2024]. At the second stage, the saved detectors are used to check new incoming examples for compliance with "their own" or "foreign" ones. If the received example matches the detector (or is very close to it in terms of values), then it is identified as "abnormal".

To implement the negative selection method, we will derive a number of variables that we will use in the future:

1) S is a set of self–patterns representing the normal behavior of the system;
2) d – dimension of the self-pattern (number of features);
3) N is the number of detectors;
4) r is the radius of self-recognition, i.e. the value by which a decision will be made on how similar the generated pattern is to the self-pattern (a sample of the usual behavior of the system);
5) max_attempts - the maximum number of attempts to generate detectors;
6) detectors - a list of detectors that will be updated as the algorithm progresses.;
7) attempts - the attempt counter, which is zero at the beginning.

Let us describe as a step–by-step algorithm the first stage of negative selection - the generation of abnormal detectors.

Step 1. Initialization of algorithm parameters:

1) determine the number of N detectors to be generated (for example, 200 detectors);
2) set the self-recognition radius r (value in the range 0.1-0.2, for example 0.15);
3) determine the maximum number of attempts to generate max_attempts to prevent an infinite loop;
4) Initialize an empty list of detectors to store the created detectors.

Step 2. Download and prepare self-patterns:

1) create a set of self-patterns in the learning phase of the system (5 minutes of monitoring normal operation);
2) upload a set of S self-patterns, each of which is a vector of features of dimension d;
3) Normalize self-patterns to ensure that the distances are calculated correctly.

Step 3. Generation of multiple anomalous detectors:

1) generate a random vector of dimension d, assign attempts = 0 and set the first one in the array as the current self-pattern;
2) find the Euclidean norm by formula (1) and normalize the vector to unit length by formula (2):

$$\|d\| = \sqrt[2]{\sum_{i=0}^{d} x_i^2} \tag{1}$$

$$x_i = \frac{x_i}{\|d\|} \tag{2}$$

where x_i is an element of the set of features of the vector;

3) we compare the detector with the self-pattern: for the current self-pattern, we calculate the Euclidean distance to the features of the generated detector using the formula (3):

$$dist = \sqrt[2]{\sum_{i=0}^{d}(x_i - y_i)^2} \tag{3}$$

where y_i is an element of the set of self-pattern features;

1) if dist < r, then: the detector is rejected and the value of attempts is increased by one, otherwise: if the self-pattern for reconciliation was not the last, go to the next and perform step 3, otherwise add the detector to the set of detectors;
2) if the number of detectors is < N and attempts < max_attempts, go to step 1.

Thus, at the end of the algorithm, we get an array of detectors that is less than or equal to the number of planned N. We calculate and store the density of the detector coverage (the percentage of successfully created detectors) according to the formula (4):

$$coverage = \frac{K}{N} * 100 \tag{4}$$

Let's look at how the parameter values affect the algorithm:

– N (number of detectors) – the optimal value is selected in the range from 100 to 500. The larger the N, the better the coverage, but it requires more calculations. It is recommended to specify a value of 200 for workstations, 300 for servers;

- r (radius) is a value from 0.1 to 0.2. The smaller the r, the stricter the selection and fewer detectors are tested. To balance accuracy and coverage, it is recommended to set the value to 0.15;
- max_attempts – from 1000 to 5000. The more attempts, the higher the probability of reaching N number of detectors. It is best to specify max_attempts = N * 5 as the value for a guaranteed result.;
- d (dimension) – from 7 to 15. The larger d, the more difficult it is to generate detectors. At least 7 metrics are recommended for Windows OS.

For the most efficient operation of our development, a mechanism for improving detectors through clonal selection and periodic replenishment of multiple detectors has been implemented during the program.

Unlike the previous algorithm, which was based on T-lymphocytes, the Clonal Selection Algorithm (CLONALG) is based on B-lymphocytes [Dasgupta, 2006].

B lymphocytes provide adaptive, or acquired, immunity. They produce antibodies that attach to pathogens and tag them for destruction by other immune cells.

This algorithm simulates the fundamental principle of acquired immunity, namely— clonal selection of B-lymphocytes. When a B-lymphocyte encounters a foreign antigen that matches its receptors, it is activated. In the context of software implementation, this provides an evolutionary improvement of detectors based on their effectiveness in detecting threats.

The clonal selection algorithm is activated only after successful detection of an anomaly in the system. The input data is information about the detected anomaly, including a list of activated detectors and their performance indicators [Jerne, 1974].

Of all the activated detectors, those that showed the highest affinity (similarity) with the detected threat are selected. Affinity is calculated as the inverse of the distance between the detector and the anomaly vector. Only those detectors with an affinity higher than the average value for all activated detectors are selected.

Next, one with maximum affinity is selected from all the selected detectors and clones that undergo mutation are created based on it. After that, the clones are checked for compliance with self-patterns, similar to the check from the negative selection algorithm. If the check is passed, the clone is added to the set of detectors.

Thus, the negative selection algorithm used and its increased effectiveness in the face of clonal selection provide not only an effective basis for detecting system anomalies, but also ensures its long-term adaptation to the changing threat landscape.

Monitoring and logging

The system implemented by us includes a comprehensive monitoring mechanism based on the principles of continuous monitoring and analysis of the state [Jerne, 1985] of the operating system. The monitoring architecture is designed as a multi-level hierarchy, where each level is responsible for collecting a specific type of data with varying frequency and detail. The fundamental principle of the system is the cyclic collection of metrics at specified intervals, which ensures a balance between the speed of threat detection and the consumption of computing resources.

At the first level, basic system metrics are collected, including indicators:

- CPU usage;
- RAM usage;
- The activity of disk operations and network traffic.

This data is collected every two seconds, which allows the system to quickly respond to sudden changes in the behavior of the operating system. To measure CPU usage, an accurate method is used with an interval of 100 milliseconds, providing a reliable estimate of even short-term bursts of activity. Memory usage metrics take into account not only the total amount of RAM used, but also the distribution between different types of caches, swap file, and virtual memory.

The second level of monitoring focuses on analyzing the processes and services of the operating system. The system periodically scans the list of all active processes, extracting information about process IDs, paths to executable files, resource consumption, and the hierarchy of parent-child relationships. Special attention is paid to processes running from temporary directories, which are often used by malicious software to accommodate its load. In parallel, the status of Windows system services is monitored, including their current status, autorun configuration, and change history.

The third level includes monitoring network activity and interactions with external resources. The system analyzes established network connections, identifying local and remote addresses, ports used, and data transfer protocols. To identify potential threats, abnormal network traffic patterns are monitored, such as multiple connections to untrusted domains, an unusually high volume of transmitted data, or attempts to access closed ports.

The fourth level is contextual monitoring, which takes into account time factors and system usage features. The system analyzes the time of day, day of the week, working and non-working hours, which allows you to distinguish between regular load changes and suspicious activity. For user workstations, user activity is additionally monitored — the frequency of interaction with the interface, application usage patterns, and typical work scenarios.

The preprocessing of the collected data includes several stages of normalization and transformation. Initially, the metrics obtained are converted into numerical vectors of a fixed dimension, where each component corresponds to a specific parameter of the system. Then, the values are standardized relative to the baseline, the reference profile of the normal behavior of the system obtained during the learning phase.

In addition to monitoring, the system is equipped with a logging function. It is based on the principles of structured storage and multilevel detailing of the information received. The system uses a hierarchy of log files of various purposes and formats.

The main source of information is the ais_security.log file, where all significant system events are recorded in text format with timestamps and levels of importance. A standardized system of importance levels is used, borrowed from the logging Python library.

Logging is organized according to a modular principle, where each component of the system has its own identifier, which simplifies filtering and searching records. Logging levels are adjusted dynamically. In normal mode, only informational messages and warnings are recorded. When a threat is detected, the level of detail is automatically increased.

Example of entries entered in the ais_security.log file:

[DEBUG] [*system date and time *] [WindowsMonitor] [CPU_Metrics] - Collecting metrics: 2.1ms

[INFO] [*system date and time *] [AIS_Core] [Detection] - The verification is completed. Anomalies: 0

[WARNING] [*system date and time *] [AIS_Core] [Anomaly] - Low level: confidence =0.65

[ERROR] [*system date and time *] [ResponseSystem] [ProcessBlock] - Blocking error PID 1234

[CRITICAL] [*system date and time *] [AIS_Core] [Threat] - HIGH LEVEL! Ransomware detected

The ais_incidents.json file is used to store structured data about security incidents. Each record contains complete information about the detected anomaly: timestamp, threat level, detection confidence, list of activated detectors, values of system metrics at the time of detection and actions taken.

The report generation system creates periodic security snapshots in ais_report_* files.json, where the asterisk is replaced by the date and time the report was created. The reports include statistics for a specific period: the number of detected anomalies, the distribution by threat levels, the effectiveness of detectors, and the consumption of resources by the security system.

The threat level assessment mechanism takes into account many factors: the confidence of detection, the number of activated detectors, the severity of metric deviations, and contextual parameters. Based on an integrated assessment, the system determines one of three threat levels — low, medium or high, which determines the intensity of the response.

A separate logging mechanism is implemented for the actions of the response system. All protective measures taken — blocking processes, isolating the network, quarantining files — are documented with an indication of the reason, time of execution and result.

The log version control system includes archiving mechanisms to prevent uncontrolled growth of the occupied space. Old logs are automatically compressed and moved to the archive directory, and when the age limit is reached, they are deleted. Storage policies are configured depending on the importance of the system and the available disk space.

Thanks to all of the above, our development makes it possible to conveniently analyze system behavior and track security incidents through monitoring and logging.

Incident response

During the program's operation, various metrics of the system are analyzed through monitoring, and if an anomaly is detected, an assessment of the level of this threat begins. This is based on the following indicators:

1) confidence is an indicator of confidence that the resulting system vector is abnormal. It is calculated as the average value of the proximity of the studied feature vector to the patterns of normal operation (self-patterns). The more this value differs from the standard values, the higher the confidence indicator;
2) activated_detectors - the number of activated detectors. It takes into account not only the presence of single matches, but also the scale of the deviation. A large number of triggered detectors indicates a serious threat.

Having received the values above, the program performs threat classification. The division into 3 classes has been implemented: low, medium and high threat.

The threat is classified as low if the confidence value is below the average threshold (approximately 50%), and the number of triggered detectors is low (less than three), the incident is marked as low risk. This usually means that there has been a short-term and insignificant change in the system, which is unlikely to be a real threat.

When confidence increases to moderate levels (about 70%), and two or three detectors simultaneously detect a deviation, the threat is assessed as medium. You need to pay attention to such cases, as they may indicate the development of possible vulnerabilities or an incipient attack.

A high degree of confidence (exceeding 90%) combined with the activation of multiple detectors (more than five to six) is clearly interpreted as a serious danger. Here, the system must immediately take protective measures to prevent further destructive effects.

After classifying the threat, the program decides on further actions depending on the assigned level.

For a low threat level, the following actions are provided:

1) monitoring and increasing the intensity of monitoring: aimed at eliminating false alarms, designed to assess the development of the situation;
2) Logging the incident in the event log: all information is stored in the application logs (previously discussed in Chapter 3.2), which facilitates further analysis and assessment of threat dynamics;
3) visual notification to the user: despite the fact that minor changes in the system do not always indicate a destructive effect, the program notifies the user of any observed deviation from the norm.

An example of a low-level threat is a slight increase in resource consumption by individual applications or a one-time drastic change in network traffic caused by downloading a large file.

The average threat level implies increased caution and the need to take measures to limit possible harm. Actions that are applicable at this level include:

1) Automatic process interruption: Stopping suspicious applications or services that exhibit abnormal behavior. For example, an application consumes an excessive amount of resources, or behaves strangely in terms of sending requests to the network;
2) creation of recovery points: the program generates backups of the system in case of irreversible damage. Even if the incident itself turns out to be insignificant, this will allow you to restore the system to its previous state at any time;
3) blocking network connections: access to the network is partially or completely blocked for an individual application or a group of applications, especially if there are suspected attempts at unauthorized access or transfer of confidential data.

The following situations are typical for an average threat level: an increase in the number of suspicious network requests sent to a server located outside the trusted zone; the emergence of new processes performing active I/O operations that affect the stability of the OS.

The most serious threats cause maximum system response. In this case, the program has the following functionality for responding:

1) complete isolation of the device from the network: this prevents possible intruders from entering the local network and protects important documents of the organization in case the program is not used on a private host;
2) forced shutdown of all suspicious processes: Any processes considered to be a source of threat are interrupted immediately to avoid spreading infection or hacking;
3) file transfer to quarantine: infected files are moved to a special quarantine zone, which excludes their execution and user access until the problem is fixed by a specialist. The folder with infected files (quarantine) is located outside the main disk, isolating the dangerous file from the rest;
4) Audio alert of a serious threat: the user receives an audio alert and a text notification with recommendations on further actions.

Examples of events leading to such a high level of threat are: a sharp increase in the frequency of access to system resources, accompanied by hardware failure or freezes; multiple unsuccessful attempts to log into administrator accounts or privilege escalation by intruders; the presence of numerous spyware modules or viruses inside the system.

Thanks to the above-described response mechanisms, the system is able to provide a balance between protection and ease of use. Due to the flexibility of the implementation, each event is carefully analyzed, and the next steps depend solely on the actual magnitude of the threat and the

nature of the event. Together with parallel logging, the program provides a wide range of features even for users who do not have system administration skills.

Results and Discussion

In order to verify the correctness of the implementation of the identification and counteraction functionality of destructive influences on the information system, it is necessary to simulate such an impact.

As tests, we will select several areas of adverse impact on the system with varying degrees of threat.:

1) the location in the TEMP folder of a file with a suspicious resolution;

2) increased resource consumption (for example, CPU);

3) identification of suspicious processes;

Let's test one of the features of our software tool – analyzing the TEMP folder for suspicious files. If such instances are found, the program is able to place them in a "quarantine" located along this path: C:\Windows\AIS_Quarantine

Let's write a simple code as a test. During its execution, a file is created in the TEMP folder called "MINER_CRYPTO_TEST.exe ". After creating the file, a timer is started, during which our program needs to detect the anomaly and eliminate it (quarantine it). The program is given a minute to do this. At the end of the timer, the code makes a decision on whether the program has passed the test or not, judging by the presence of the created file in the folder. If the program does not pass the verification, the code will automatically delete the created file.

Previously described code used for development testing:

```
temp_dir = tempfile.gettempdir()
test_file = os.path.join(temp_dir, "MINER_CRYPTO_TEST.exe")
with open(test_file, 'w') as f:
  f.write("""#!/bin/bash
# CRYPTO MINER SCRIPT
# Bitcoin Miner v1.0
# Mining cryptocurrency...
while true; do
  echo "MINING BITCOIN..."
  sleep 1
done
""")
with open(test_file, 'r') as f:
for i in range(60, 0, -1):
  if not os.path.exists(test_file):
    break
  time.sleep(1)
if os.path.exists(test_file):
  os.remove(test_file)
```

```
    print("\n The test failed, the file was deleted automatically ")
else:
    print("\n The test was passed successfully ")
```

Let's just run the test and see if it really works correctly. Figure 3 shows that a new file has indeed been created in the TEMP folder.

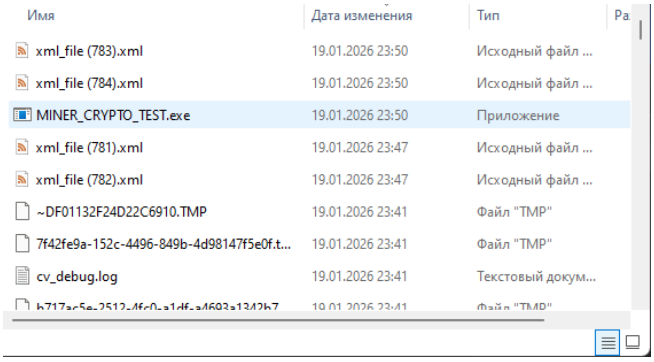

Figure 3. The presence of a suspicious file in the TEMP folder

Now let's check whether the developed software can detect this file and neutralize it in a timely manner. To do this, run the program that needs to be tested (before that, we train it), and then run the code for testing.

Figure 4 shows the corresponding entry in the incident log. As you can see, the program has identified not only the file we created, but also the one previously located in this folder, which is also suspicious.

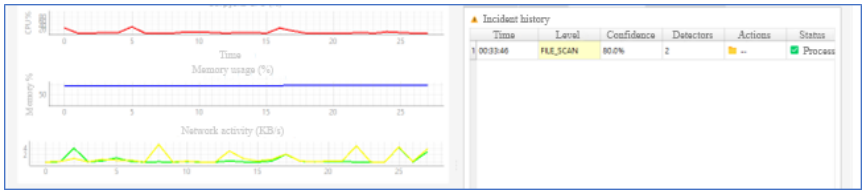

Figure 4. Incident History tab

Now let's make sure that the suspicious files are indeed in the quarantine folder. Let's go to the AIS_Quarantine directory and make sure of this. Figure 5 shows a screenshot of the files that were moved to quarantine. As you can see, the names and extensions of the files really did not inspire confidence in these files, with a high probability they could have a destructive effect on the system.

Figure 5. AIS_Quarantine folder

Let's turn to the logging files. Uploading relevant events from the ais_security.log file:

2026-01-20 00:33:45,716 - AISecurity - INFO - Check TEMP- folders: C:\Users\User\AppData\Local\Temp

2026-01-20 00:33:46,130 - AISecurity - WARNING - Suspicious files found in TEMP: 2

2026-01-20 00:33:46,141 - AISecurity - INFO - The TEMP file has been quarantined: MINER_CRYPTO_TEST.exe

2026-01-20 00:33:46,144 - AISecurity - INFO - Файл из TEMP помещен в карантин: virus_tmp.exe

2026-01-20 00:33:46,144 - AISecurity - INFO - Quarantined files from TEMP: 2

The next test of the system will be the identification of high resource consumption. This is a low-level threat, and only active monitoring is presented as an active response. Given that the drastic consumption of resources in this case is not prolonged, this situation is not dangerous for the system.

Let's create a situation in which all the cores of the system will be involved at 100%. To do this, we will write a code based on this construction:

```
while True:
  # Intensive calculations
```

```
for _ in range(1000000):
# Mathematical operations that load CPU
 _ = math.sqrt(random.random()) * math.log(random.random() + 1)
 _ = math.sin(random.random()) * math.cos(random.random())
_ = random.random() ** random.random()
```

These mathematical operations put a lot of strain on the CPU, which is exactly what we need. Next, you need to find out the total number of cores of the system and run the code in several threads. Similarly to the previous test, we will limit it in time by 30 seconds.

Figure 6 shows the program interface at the time of the test run. As you can see, the CPU load graph and the CPU metric indicate its 100% consumption at a given time.

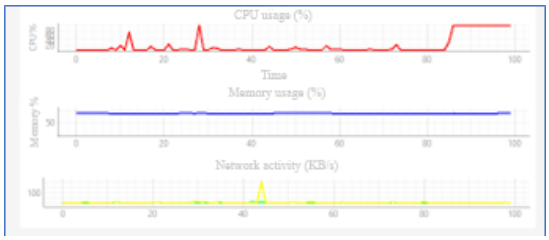

Figure 6. The graphical interface of the application at the time of 100% CPU usage

The system detected this anomaly and managed to add 2 entries about it to the log file.:

2026-01-20 01:05:30,866 - AISecurity - INFO – Уверенность – 0.2%, Детекторов 2 Предпринятые действия: logging, enhanced_monitoring

2026-01-20 01:12:25,867 - AISecurity - INFO - Уверенность – 0.3%, Детекторов 1 Предпринятые действия: logging, enhanced_monitoring

Now let's test the system's ability to analyze processes. Let's write a test code that creates a list of processes with obviously suspicious names. The list of processes being created:

```
suspicious_processes = [
('MINER_PROCESS', 'python -c "import time; print(\'MINER RUNNING\'); time.sleep(60)"'),
('CRYPTO_HACK', 'python -c "import time; print(\'CRYPTO HACK\'); time.sleep(60)"'),
('BITCOIN_MINER', 'python -c "import time; print(\'BITCOIN MINING\'); time.sleep(60)"'),
('MALWARE_EXE', 'python -c "import time; print(\'MALWARE ACTIVE\'); time.sleep(60)"'),
('TROJAN_VIRUS', 'python -c "import time; print(\'TROJAN RUNNING\'); time.sleep(60)"')]
```

Let's run the code from the command line. Figure 7 shows both the process of creating processes and the instantaneous response of the application.

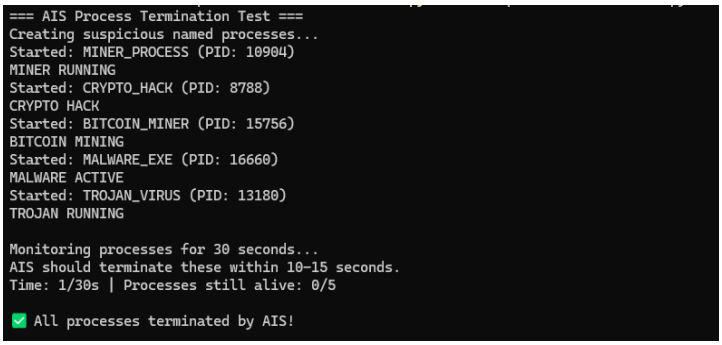

Figure 7. Command line output

Let's turn to the graphical interface of the software, shown in Figure 8. This state of the system activated from 8 to 9 detectors.

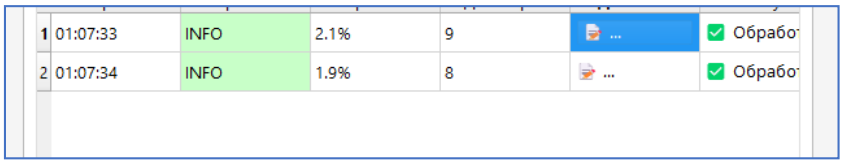

Figure 8. Displaying incidents in the incident history

As the tests have shown, even in the early stages of the program, it shows good results. Since the system is capable of self–learning in the process, depending on the activated detectors (clonal selection), the longer it runs, the more accurate it will demonstrate.

Conclusion

In the course of this work, a comprehensive theoretical and practical analysis of the effectiveness of artificial immune systems (AIS) was carried out to solve the fundamental task of modern cybersecurity – the recognition of destructive influences on the information system. The results obtained confirmed the high effectiveness of the approach in conditions where traditional, signature-based protection methods demonstrate their limitations. As a result of the successful solution of the tasks set, the main goal of the work was achieved: a software tool was developed, implemented and tested, the core of which is the "negative selection" algorithm designed to automatically detect anomalies in the behavior of the protected system.

The conducted research and practical development contribute to the relevant field of knowledge: the theory of artificial immune system and information security. The overall importance of the topic is due to the increasing complexity and scale of cyber threats, including the emergence of polymorphic and metamorphic malware, targeted attacks (APT), attacks on the Internet of Things (IoT) and industrial systems (ICS). Countering these challenges requires a shift from static protection to dynamic, adaptive, and self-learning systems. Artificial immune systems, inspired by biological defense, offer just such a model: distributed, resistant to single failures, capable of learning on the fly and possessing immune memory.

The software developed as part of the work has passed a test cycle on datasets simulating both normal system activity and various anomalies. The results confirmed its operability and effectiveness as a tool for monitoring and analyzing the behavior of the work environment. The key proven quality is the ability of the system to provide an adequate "immune response" to interference with the correct operation of the device, and without the need for prior knowledge of the attack pattern. The

"negative selection" algorithm, during which detectors are generated, does not respond to "self" (self-patterns), allows the system to form immunity to a wide class of unknown threats. Moreover, due to the possibility of building up and constantly updating the list of detectors as experience accumulates, the system demonstrates the property of adaptive improvement, becoming more accurate and effective over time in the context of a specific protected system.

The key advantage of the implemented software product lies in its adaptive nature, as mentioned earlier. Unlike antivirus scanners or intrusion detection systems (IDS) based on known signatures, the AIS-based core is aimed at detecting deviations from a given norm. This allows you to recognize and block abnormal behavior of any nature, including zero-day attacks and complex multi-stage schemes that do not leave clear signature traces. This property makes the developed solution not a substitute, but a powerful addition to traditional means of protection, forming a comprehensive defense. The introduction of such a system is especially important in the context of the constantly evolving cyber threat landscape, where the rate of emergence of new attack vectors outstrips the possibilities of their manual analysis and description.

Thus, the use of artificial immune systems to recognize destructive effects is a scientifically sound and practically significant direction that can significantly increase the resistance of information systems to modern cyber threats.

References

Bryukhomitsky Yu. A. Artificial Immune Systems in Information Security: A Tutorial / Yu. A. Bryukhomitsky; Southern Federal University. - Rostov-on-Don; Taganrog: Southern Federal University Publishing House, 2019. - 147 p.

Chernyshev Yu. O., Grigoriev G. V., Ventsov N. N. Artificial Immune Systems: A Review and Current Status // Software Products and Systems. 2014. No. 4 (108). URL: https://cyberleninka.ru/article/n/iskusstvennye-immunnye-sistemy-obzor-i-sovremennoe-sostoyanie

Birulia M.D. Quality Management in Software Development. Advanced Engineering Research (Rostov-on-Don). 2024;24(3):255-263. https://doi.org/10.23947/2687-1653-2024-24-3-255-263. EDN: JBGRGQ

Skobtsov Yu. A. Modern Immunological Models and Their Applications // Bulletin of Bauman Moscow State Technical University. Series "Instrument Engineering". 2022. No. 3 (140). URL: https://cyberleninka.ru/article/n/sovremennye-immunologicheskie-modeli-i-ih-prilozheniya

Abramov Evgeny Sergeevich, "Building an Adaptive Information Security System," Izvestiya SFedU. Technical Sciences. 2009, No. 11. URL: https://cyberleninka.ru/article/n/postroenie-adaptivnoy-sistemy-informatsionnoy-bezopasnosti

Zhukov Vadim Gennadievich, Salamatova Tatyana Andreevna, "Detection of Network Intrusions by an Evolutionary Immune Algorithm of Clonal Selection," Siberian Aerospace Journal. 2014, No. 4 (56). URL: https://cyberleninka.ru/article/n/obnaruzhenie-setevyh-vtorzheniy-evolyutsionnym-immunnym-algoritmom-klonalnoy-selektsii

Adieva G.M., Muratbek Uulu A. Development of network monitoring systems // Bulletin of Science and Practice. 2024. No. 4. URL: https://cyberleninka.ru/article/n/razrabotka-sistemy-monitoringa-seti

Artificial immune systems and their applications / edited by D. Dasgupta; translated from English by A. A. Romanyukha. - Moscow: Fizmatlit, 2006. - 344 p.

Jerne N.K. Towards a network theory of the immune system I I Ann. Immunol. (Inst. Pasteur). 1974. V. 125C. P. 373-389.

Jerne N. K. The generative grammar of the immune system // The EMBO Journal. 1985. V.4, №4. P. 847-852.