UDC 004.457

Detecting the operation of keyloggers using the dendritic cell algorithm with multiple resolutions

Hennadii Shybaiev[1], Leonid Galchynskyi[1]

[1] *National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute", Institute of Physics and Technology*

Abstract

Throughout time, criminologists (or their colleagues in history) have tried to develop the most reliable methods of protecting information. Currently, the most common method of information processing is the computer, so today's information protection specialists face the task of protecting data in computers, in which the most common method of information input is data input from the keyboard by the user. Keystroke logging, also known as keylogging, consists in intercepting keystroke codes from the user. This data may contain passwords, personal correspondence, or other confidential information. Therefore, it is very important to pay attention to this method of user interaction with your "machine", because it is through this method that an attacker can steal information directly from the keyboard. Unlike traditional malware such as worms or viruses, some types of keyloggers cannot be detected by modern antivirus protection methods.

The paper presents the results of a study of the application of the dendritic cell algorithm with multiple resolutions for the task of determining the presence of a keylogger in the system. Based on the simulation, a new effective model for determining the presence of a keylogger is proposed

Keywords: keylogger, API functions, detection, dendritic cell algorithm, multiple resolution, wavelet transform, interrupt request, DCA, HIS

Introduction

The term "keylogger" itself is neutral and only describes the function of the program. Most publications describe a keylogger as software designed to covertly record and monitor all keystrokes. This definition is not entirely accurate, as a keylogger can be either hardware or software. Although keylogger hardware is much less common than keylogger software, it still needs to be considered when discussing information security. A keylogger is an application that is not a malicious program in itself, because it does not pose a direct threat to the resources of the computer system, but is only designed to record and transmit to the outside the information that the user generates using the keyboard. Such ambivalence allows keyloggers to be present in a variety of legitimate programs, ranging from parental control to monitoring the work of employees by the administration during the working day. Keyloggers are sometimes used to allow users to switch between keyboard layouts, or to invoke certain software tasks using "hotkeys" (for example, Keyboard Ninja). However, the practice of the past decades indicates that keyloggers are mainly used by cybercriminals as a spying tool to steal private user data, such as passwords, usernames, email account passwords, PIN codes and account numbers for electronic payment systems, etc. It is clear that the illegal use of keyloggers begins with the unauthorized installation of this program on the user's workstation. At the same time, attackers try to write their spyware so that its presence on the user's computer is as hidden as possible. The actual illegal installation and use of keyloggers, together with the covert nature of its use, creates a serious cyber threat. It is quite difficult to give an accurate quantitative assessment of this threat. Spy keyloggers account for 6 to 8 percent of all cyber threats globally by 2023. On the other hand, back in 2007, John Bambenek, an expert at the SANS Institute, estimated that approximately 10 million computers in the US alone were infected with keylogger malware, with direct losses estimated at approximately $25,000,000. Taking into account the trends of rapid growth of cyber

threats over the last decade, it can be said that hundreds of millions of computers in the world are currently infected with malicious keyloggers, with correspondingly increased losses.

It is natural that the importance of such a cyberthreat raised and still raises the question of combating it, primarily solving the problem of detecting an illegally installed keylogger. As mentioned above, keyloggers can be divided into two categories:

- keyboard log devices
- keylogger software

In this work, we will focus on the problem of category 2, as it is more significant for the problem of cyber threats derived from keyloggers. The way to detect the presence of an illegal keylogger is through the detection of anomalous behavior of one or more systems in the operating system, in particular through the analysis of network traffic. Another marker can be anomalous activity of certain, specific for the processing of key messages, calls of API functions. As a result, this requires knowledge of the features of the character generation mechanism on the computer keyboard, as well as the features of the system mechanism of the computer's interaction with the networks to which it is connected.

Low-level interaction with the keyboard via an I/O port.Interaction with the system keyboard controller takes place through the input/output port 64h. Reading a byte from this port makes it possible to determine the state of the keyboard controller; writing a byte makes it possible to send a command to the controller.

Interaction with the microcontroller in the keyboard itself occurs through input/output ports 60h and 64h[1]. Bits 0 and 1 in the status byte (port 64h in read mode) allow interaction control: before writing data to these ports, the first bit of port 64h must be 0. When data is available to read from port 60h, the first bit of port 64h is 1. Enable bits /disable keyboard in the command byte (port 64h in write mode) determines whether the keyboard is active or not, and whether the keyboard controller will cause a system interrupt when the user presses a key.

Bytes written to port 60h are sent to the keyboard controller, and bytes written to port 64h are sent to the system keyboard controller. This hardware framework enables the operating system installed on the computer to process this data, which in turn allows application programs to interpret the processed bytes, which ultimately turns keystrokes into readable text. After the data sent by the keyboard appears on port 60h, the keyboard hardware interrupt IRQ handler is activated. Then everything depends on the operating system. In the future, we will consider the most common Windows operating system among users. Interaction with external devices such as a keyboard, mouse, trackball, etc. for Windows is built on a unified mechanism of system drivers. Regardless of how the keyboard is physically connected, keyboard drivers use the keyboard class system drivers to handle data. This happens regardless of the equipment used. These drivers are called class drivers because they support system requirements regardless of the hardware requirements of a particular device class. The corresponding functional driver (port driver) supports the execution of I/O operations according to the device in use. On Windows x86, this is implemented in a single system keyboard driver (i8042) and mouse driver. The PS/2 keyboard driver stack is shown in Figure 1.

Driver stack (from top to bottom):

- Kbdclass - high-level filter driver, keyboard class
- application high-level filter driver
- keyboard class
- i8042prt – functional keyboard driver
- root bus driver

For a keyboard that connects via a USB port, the driver stack looks a bit different. Mice for modern personal computers are also mostly connected via a USB port.

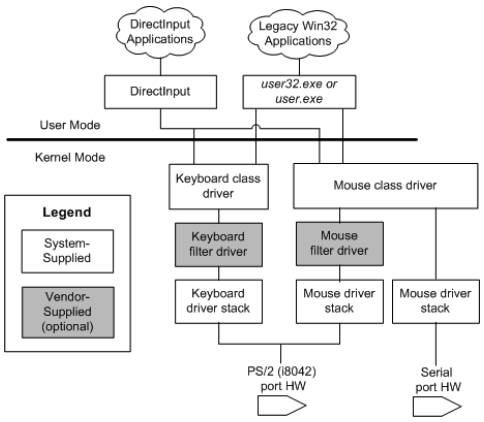

Figure 1. PS/2 keyboard driver stack[2]

However, the general scheme remains the same. In user mode, the Microsoft Win32/64 subsystem accesses the keyboard using the Raw Input Thread (RIT), part of the csrss.exe system

process. During boot, the system creates a RIT and a System Hardware Input Queue (SHIQ).

The RIT thread opens the keyboard class device driver for exclusive use and uses the ZwReadFile function to send it an input/output request (IRP) of type IRP_MJ_READ. When the request is received, the Kbdclass driver marks it as pending, queues it, and returns a STATUS_PENDING code. The RIT must wait for the IRP to complete, and to determine this, it uses an asynchronous procedure call, or ACP. RIT processes incoming data as follows. All input events from the keyboard are queued by the hardware input system. These, in turn, are converted to Windows messages (such as WM_KEY*, WM_BUTTON*, or WM_MOUSEMOVE) and then placed at the end of the virtualized input queue. The key scan codes in Windows messages are replaced by virtual key codes that correspond not to the location of the keys on the keyboard, but to the action that the key performs (the function it performs). The code conversion mechanism depends on the current (active) keyboard layout, simultaneous key presses (for example, SHIFT) and other factors.

Next, the thread processes the keyboard messages that entered the thread's message queue in the message processing loop.

$$
\begin{gathered}
while(GetMessage(\&msg, 0,0,0)) \\
\{ \\
TranslateMessage(\&msg); \\
DispatchMessage(\&msg); \\
\}
\end{gathered}
$$

The GetMessage function retrieves keyboard (mouse) events from the queue and redirects them using the DispatchMessage function to a window procedure that processes the message for the window that currently has input focus. The TranslateMessage function is called before the DispatchMessage function to create "token" messages such as WM_CHAR, WM_SYSCHAR, WM_DEADCHAR, and WM_SYSDEADCHAR using the WM_KEYDOWN, WM_KEYUP, WM_SYSKEYDOWN, WM_SYSKEYUP messages as a base. These "symbolic" messages are placed in the program's message queue; however, note that the original keyboard messages are not removed from this queue. Mouse events are ignored by the TranslateMessage function. The following notifications are supported for the keyboard: which window is currently in keyboard focus, which window is currently active, which keys are pressed, and the status of the input cursor.

Information about which keys are pressed is stored in a synchronous key status array. This array is connected to variables for each thread's local input state. An asynchronous key status array that contains similar information is used for all threads. Arrays display the state of all keys at a given time, and the GetAsyncKeyState function allows you to determine whether or not a particular key was pressed at a given time. GetAsyncKeyState always returns 0 (ie, not pressed) if called by another thread (ie, not the thread that created the window that is currently the focus of the input state).

The GetKeyState function differs from GetAsyncKeyState in that it returns the keyboard state at the time the last keyboard message is pulled from the thread queue. This function can be called at any time, regardless of which window is currently in focus.

In addition to this classic keystroke scheme, Windows developers offer an alternative input model called the raw input model, which simplifies the development of programs that use non-standard input devices.

When using it, the application receives device-independent input in the form of a message (for example, (WM_CHAR)) that is sent to the application's windows. In the raw input model, the application must register the device, specifically the keyboard, from which it wants to receive data. The application then receives the input from the user data via the WM_INPUT message Two data transfer methods are supported: the standard method and the buffering method In order to interpret the input data, the program must obtain information about the input device, which can be done using the GetRawInputDeviceInfo function.

Variants of software keyloggers

Knowing the keystroke processing mechanism, let's consider the main methods used by malware authors to implement keyloggers. Keyloggers can intercept keystroke data that is transmitted by one processing subsystem to another subsystem at any stage of the processing sequence. Globally, methods of creating software keyloggers are divided into user mode methods and kernel mode methods.

Kernel-mode keyloggers are much more difficult to develop than user-mode keyloggers. To write such keyloggers requires deep knowledge of the hardware and the operating system, sometimes using undocumented capabilities, but accordingly it is much more difficult to detect, because they are completely unnoticed by all user mode programs. Both methods documented in the Microsoft Driver Development Kit and undocumented methods can be used to intercept. There are relatively few kernel-level keyloggers, the bulk of keyloggers are user-level keyloggers, they are the ones who cause the majority of losses, accordingly, the relevance of the problem of detecting these keyloggers does not decrease.

User mode keyloggers. A number of models and methods have been proposed over the years to address the general problem of malware. However, when applied to the specific problem of keylogger detection, all existing solutions are unsatisfactory. Signature-based solutions have limited applicability because they are easily evaded and require isolation and extraction of an existing signature before a new threat can be detected. The irrelevance of the signature approach is also expressed in the fact that even middle-skilled programmers can easily develop new versions of existing keyloggers and thus make the previously valid signature ineffective [3].

Detection methods based on behavior also proved to be ineffective. They aim to distinguish between malicious and safe programs by profiling the behavior of legitimate programs or malicious programs[4]. Although there are a variety of techniques for analyzing predictable behavior, most of them rely on capturing system calls or library calls that are called at runtime. Unfortunately, it is extremely difficult to clearly identify a keylogger that hunts system calls, as there are many legitimate programs (e.g. shortcut managers, keyboard remapping utilities) that intercept keystrokes in the background and exhibit very similar behavior.

Methods that used the idea of detecting a keylogger by anomalous behavior of network traffic also turned out to be unreliable.

All these approaches took into account the essential, but incomplete functionality of the keylogger, in particular its main part - the keylogger.

There are two ways to capture keystrokes:

- local (by connecting the process's standard messaging interface)
- global (by connecting the internal elements of the window manager)

In local interception, the keylogger must run part of its code in the address space of the process being "listened to". Since both mouse and keyboard events in the standard keystroke engine are passed through a predefined set of data structures provided by the OS, the keylogger does not even need a priori knowledge of the process intended to register the keys.

With global interception, the keylogger uses global data structures and runs as a separate process. In addition, it is possible to embed the keylogger code completely into the code of an executable program, such as a browser or word processor, thus turning the infected application into a keylogger.

On this basis, several types of keyloggers can be defined[5]. Figure 2 schematically shows the flow of key messages depending on the type of keylogger.

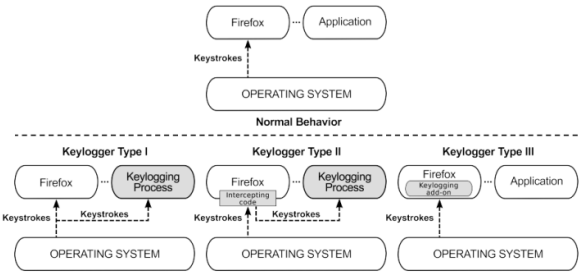

Figure 2. Key message processing schemes depending on the type of keylogger presence.

Unlike Type I and Type II recorders, which typically come as compiled code, Type III recorders are often delivered as interpreted scripts written in languages such as JavaScript. In this case, the source code can be easily obtained and can be used to perform more accurate damage tracking. Type III keyloggers are mostly installed on web browsers, due to their distribution and cross-platform nature. Both Type I and Type II keyloggers are relatively easy to implement on Windows, while Type I loggers can be installed on Unix-like X11 and GTK OSes. For both the first and the second type, two strategies can be used: interception and polling.

In addition to calls to API functions, the operation of the keylogger is also characterized by writing to a file, as in a keyboard log with corresponding calls to API functions for files: CreateFile, OpenFile, ReadFile and WriteFile, as well as access to a network port through a set of API functions: socket, send, recv , sendto and recvfrom.Since these three groups of functions

characterize the work of keyloggers in general, it is actually only on the basis of a joint assessment of all these three groups that an anomaly can be objectively established, which characterizes the work of an illegally installed keylogger.

As already mentioned, the use of exclusively evaluating the abnormal behavior of only one side of the keylogger's work gives disappointing results for keylogger detection. This paper presented an algorithm to detect a single keylogger to a system based on the correlation of various behavioral components by monitoring certain API function calls made by the keylogger to perform keylogging. Calling these functions during the specified period of time may pose a threat to the security of computer systems. For example, an application calling GetKeyboardState or GetAsyncKeyState and writing data to a file by calling the WriteFile function usually indicates keyboard log activity. Also, the keylogger is designed to send intercepted keystrokes to the attacker, so we may see a large amount of outbound traffic during this period. The ratio of the frequency of function calls generated by the keylogger over a period of time can indicate abnormal activity in our system. It is generally accepted that tracking and correlating keyboard events with other behavioral data, such as accessing files or sending packets, will improve the process of detecting keyloggers.

Instead of trying to detect keyloggers only by analyzing network traffic, it is more promising to detect a single keylogger in a system by monitoring and calculating the correlation of different actions in the system[6] represented by the execution of different API function calls that may indicate the presence of a keylogger in system. API function calls made by the keylogger are tracked in the user environment. The monitoring program intercepts calls to API functions in the user-level environment of the operating system. Kernel-level direct call API functions will not be considered in this study.

To detect a keylogger in the system, it is natural to expect a correlation between different groups of system calls, and if a high correlation value is calculated, for example, exceeding 0.5, then this indicates the presence of a keylogger, and based on the discrete nature of the data, the correlation is calculated using the rank correlation method [7]. The results calculated on the test data showed a significant improvement in the detection of keylogger installation, but at the same time a significant number of false positives. The authors [3] conducted detailed numerical experiments with various keyloggers. The results of the experiments showed ambiguous correlation values. There are many reasons for this. The first reason is that different events occur in different time windows. Therefore, our algorithm gives inaccurate results. The second reason is that there are many idle periods in the datasets. Idle periods increase the correlation value, which affects our detection scheme. To improve our detection scheme, we need to implement a more intelligent keylogger presence detection scheme.

Dendritic cell algorithm

The idea that the penetration of malicious software into computer networks has a certain analogy with the infection of the human body by viruses and bacteria has existed for a long time. However, a little later ideas appeared to apply models of the human immune system. Artificial immune systems have been developed by observing the behavior of the human immune system and modeled one or more immune cell-organ interactions. One of them: hazard theory is one of the well-known HIS models and inspired the development of an algorithm based on the behavior of dendritic cells (DC), known as DCA[9]. In recent years, this theory has evolved and further developed as deterministic DCA - dDCA[10]. Different algorithm specifications provide a similar framework; however, they differ in key aspects that determine its behavior. Since the focus of this study is the development of a model based on dDCA, any subsequent references to DCA refer to its deterministic version. DCA has three phases, namely feature selection, context detection and evaluation, and classification. DCA differs from other AIS algorithms in the following features[8]:

- multiple signals are combined to represent information about the environment or context
- signals are combined with the antigen in a temporal and distributed manner
- pattern matching is not used to perform detection, unlike negative selection
- cells of the innate immune system are used as inspiration rather than adaptive immune cells, and unlike clonal selection, no dynamic learning is performed

Artificial immune systems are based on one or more interactions between immune cells and

organs and have been created by studying how the human immune system behaves. HISseeks to offer defense mechanisms and protection against invasive agents, including bacteria, parasites and viruses. Immunological mechanisms, including negative selection, clonal selection, immunological networks, and risk theory inspired IDS. One of the key theories of HIS—danger theory—motivated the creation of the DCA algorithm, which is based on the activity of dendritic cells.

Three stages in the evolution of the DCA algorithm can be noted: the prototype DCA, a more complex version using stochastic components, and deterministic DCA (dDCA). Although many algorithm specifications offer a similar structure, they differ in important ways that affect the behavior of the algorithm. Any further references to DCA refer to its deterministic form, as the aim of this study is to build a model based on dDCA. Feature selection, context detection and evaluation, and classification are the three steps of DCA.

The interplay between signals released by cells and antigens is at the heart of the risk theory paradigm. These signals indicate whether a cell or tissue is acting normally or abnormally, such as when there is expected or unexpected cell death, stress, inflammation, or abnormal processes caused by malfunctioning cells. The signals perceived by the dendritic cell are divided into three classes: pathogen-associated molecular pattern (PAMP), safe signals (SS) and danger signals (DS). The input data must be represented as two signal categories, DS and SS, a categorical value that identifies each data instance, and a unique identifier for each data instance in order for deterministic DCA adaptation to work[11].

Figure 3 shows an analogy between a biological dendritic cell algorithm and an artificial immune algorithm based on dDCA[12].

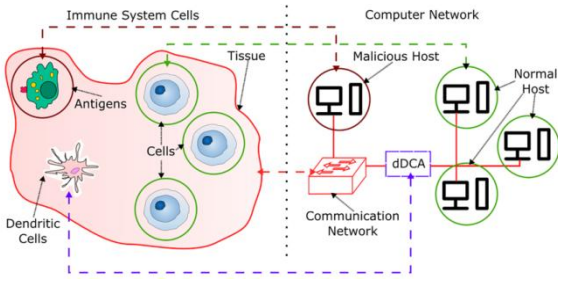

Figure 3. Analogy between a biological dendritic cell algorithm and an artificial immune algorithm based on dDCA[14].

Each of the required signal categories receives a feature or group of features from the original data set during a preprocessing and initialization step. The proposed method uses multiple feature class information to categorize signals. Before averaging the transformations, the features are ranked for each category according to the influence criterion. Each cell in the population has a set of instructions that it executes each time it updates. An important feature is that each DC in the population performs its own version of antigen sampling and signal collection. Diversity and feedback are maintained in the population by migrating DCs at different points in time, creating a variable "time window" effect that makes the system more robust. Thus, each DC in the population is an entity with its own set of behavioral instructions. DCs process input signals and then generate cumulatively updated output signals for application to antigen collection throughout the sampling period. Each DC can exist in immature, semi-mature, and fully mature states. The beginning of the state transformation from immature to semi-mature. - mature or not mature is controlled not by the collection of antigens, but by the influence of signals. Now the influence of signals on DC is again limited by the "migration threshold". Immature DCs perform three main functions each time they are updated.

Antigen sampling – The DC collects antigen from external sources (tissues) and stores them in an antigen storage data structure. Input signal processing - DCs collect the values of all input signals present in the storage area. Calculation of cumulatively updated output signals. Each DC receives input signals and produces intermediate output signals that are added to form the cumulatively updated output signals. The signal processing equation is presented in the form of a weighted sum equation[13]:

$$Output = (PW \sum_i P_i + DW \sum_i D_i + \\ + SW \sum_i S_i (1 + I))$$

Where PW is the weight associated with PAMP, DW is for danger signals, and SW is for safe signals where I stands for infection. Pi, Di, and Si are the input signal values belonging to

the PAMP signal, danger signal, and safe signal categories, assuming that there are multiple inputs per category. "I" is an infection signal that appears three times - once for each exit. The signal processing equation calculates the intermediate value of the output signal for the CSM, semi-mature and mature outputs. These values are then summed to obtain cumulative signal values. In the equation above, the PAMP weight values are used to create all other weights relative to the PAMP weight. The two main functions of the three output signals are (a) to limit the lifetime or sampling interval of the DC and (b) to determine the final state of the DC (semi-mature or mature) to determine whether the antigen is abnormal or not.

The above ideas for detecting keyloggers were implemented in [18] and showed better results in comparison with the methodology based on Spearman rank correlation calculations, both from the point of view of increasing the percentage of correct detection of keyloggers, and from the point of view of reducing the percentage of false positives. However, in this case too, some inaccuracies in the definition of keyloggers were noted. This raises questions about the search for improvement of the DCA methodology. Our working hypothesis is that better results of detecting keyloggers based on the DCA methodology can be achieved if we take into account significantly different call intensities of different groups of ARI functions. For this, a multiple resolution approach should be used. Multi-Resolution Analysis (MRA)[8] is the process of expressing a signal in terms of lower-resolution approximations and details to successively create higher-resolution versions until the original signal is reproduced.

Dendritic cell algorithm with multiple resolutions

The technique of describing a signal in terms of lower-resolution approximations and details to successfully create higher-resolution versions until the original signal is reproduced is known as multi-resolution analysis (MRA). The largest number of MRA levels that can be decomposed in wavelet analysis is 2J, where J = log2(N) and N is the finite length of the signal (or function). The decomposed signal is a truncated version of length 2J at each level j. A finite-time signal, such as DS or SS signal categories [DS,SS](t), where t denotes any instant of time, is specified as a recursive relation for each level of decomposition, with each level j for J, including lower approximations and details.

Discrete wavelet transform is used to decompose [DS,SS](t) signals while conserving energy and preventing signal downsampling[14]:

$$[DS, SS] = \sum_{n \epsilon Z} C_{j,n}\, \varphi_{j,n}(t) + \\ + \sum_{i=J}^{\infty} \sum_{n \epsilon Z} d_{j,n}\, \psi_{j,n}(t)$$

where n is the translation of the signal [DS,SS](t) in the integer domain Z, J denotes the maximum decomposition level, j represents the decomposition level such that j < J, and the wavelet ψ(t) and the scaling functions ϕ(t) are family of orthonormal bases.

For analysis with different resolutions, the signal [DS,SS(t)] is divided into details dj,n and approximations cj,n. At each level of decomposition, there is a recursive relationship between the coefficients.Due to the compression process, at each stage of the decomposition of the discrete wavelet transform, the sample size signal, a multiple of 2J. This limitation can cause timing ambiguities in decomposed signals. Due to the compact support of coupled quadrature filters, the coefficients of the discrete wavelet transform can create the effect of signal blurring. The maximum-overlap discrete wavelet transform, also known as non-decimated, stationary, translational, or time-invariant transform, is too redundant: the maximum-overlap wavelet transform requires O($N\log(2N)$) computational complexity compared to O(N) for the discrete wavelet transform, which leads to higher computational costs. Another important feature is that the maximum overlap wavelet transform prevents the signal from being downsampled, which is required for the analysis of the dendritic cell algorithm, and also allows the signal to be decomposed of any duration.

Using the algorithm

We need to understand how the operating system generates and manages keyboard events in order to understand how to reproduce the keystrokes that will be recorded by the keylogger. In addition, it is important to

understand how a keylogger records keystrokes during this procedure.

A keyboard interrupt is generated by the Windows operating system every time a key is pressed. The keyboard driver converts the interrupt into a system message and then adds it to the "system level message queue". The operating system forwards the message to the "application-level message queue" of the specified focused program, monitoring the focused program at the time the keyboard interrupt is generated. This program should now manage this key properly. The operating system will simply discard this key if it cannot find certain target programs. In this process, keyloggers use low-level operating system APIs such as GetKeyboardState or GetAsyncKeyState to intercept keystrokes or directly detect keyboard interrupts.

In order to force the keylogger to write the input stream to a file more often, we developed an application that simulates the input stream of keystrokes from the user. On the other hand, simulating keystrokes should not affect normal applications. The agent creates a secret window and sets it as active before starting the simulation. The hidden window receives the simulated keystrokes after they are created and simply ignores them. After simulation, the active window returns to its initial state. The above action is performed regularly by the keystroke agent. Consumers are practically unable to notice active window changes due to the short duration of this procedure.

To identify the keylogger among all other programs, an application was also developed that monitors API calls in the system from all programs. But in this work, we focused on 3 types of API calls, namely:

- keyboard status tracking
- recording in a file
- sending and receiving data from the network

We divided all events into the following types: dangerous event, safe event.

Dangerous event - an event that indicates an anomaly is called a danger signal. A low signal level may be abnormal. We included the following events:

- A small time interval between two consecutive calls to the WriteFile function. Because keylogger constantly saves keystrokes in log files, a slight difference in time will be observed. In contrast, a normal program will have a higher value of $\Delta t1$ between entries.
- Communication between different types of function calls. We consider this to be an event where file access or communication functions are called immediately after keystroke tracking operations, based on the behavior of keystroke spies. The value of the signal depends on the total number of file accesses and communication function calls during the given time interval

A safe signal is a signal that is a reliable indicator of steady-state or typical system performance:

- The time difference between two serially output functions of communication with the network (send, sendto and socket). This is necessary because keyloggers transmit data to attackers after intercepting user activity. We would expect a significant difference between the two consecutive functions in a typical scenario. On the contrary, we assume a short period of time during which the keylogger transmits data to the attacker.
- Small number of keyboard tracking function calls. Unlike keyloggers, legitimate programs like wordpad or notepad use far fewer keyboard monitoring features. Therefore, a minimum number of calls to a host is considered safe

The proposed model involves the use of a segmented dendritic cell algorithm as a detailed classification strategy, as well as deconstructed versions of the discrete wavelet transform of the maximum overlapping categories of SS and DS signals [15] for context detection and assessment. Feature aggregation uses a generalized method in which selected features are averaged (all features have the same weight) to create signal categories that are used as input to the dendritic cage algorithm. In addition, the processed data set contains a label for each data instance with concatenated labels such as source port, destination port, and protocol. The data set is partitioned into m consecutive segments, with each segment performing a maximum-overlap discrete wavelet transform using wavelet w on up to J transform levels, where $J > 0$, $J = \log 2M$, and M is the segment size. The multi-resolution technique attempts to incorporate a decomposed signal at J levels into the dendritic cell

algorithm's context estimation, with each dendritic cell collectively using a decomposition level as input for each signal category.

Our tests are designed to demonstrate that simulating keystrokes can increase the visibility of keylogger actions and, as a result, can increase the effectiveness of MDCA detection. To do this, we set up the same environment to run the keylogger instance (Spybot) and the benign instances.We collected the number of API requests caused by the Spyrix keylogger. The Y-axis shows the normalized API call frequency, and the X-axis shows the time in seconds. The numbers for the normalized API call rate are the sum of the values we get for 10 seconds divided by the maximum value over the entire period.

Figure 4 and Figure 5 show the number of calculation results of API calls generated by the Spyrix keylogger and the JetLogger keylogger, respectively[16].

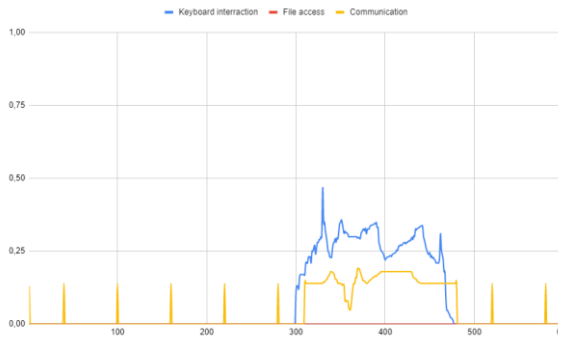

Figure 4. Number of API calls made by Spyrix keylogger

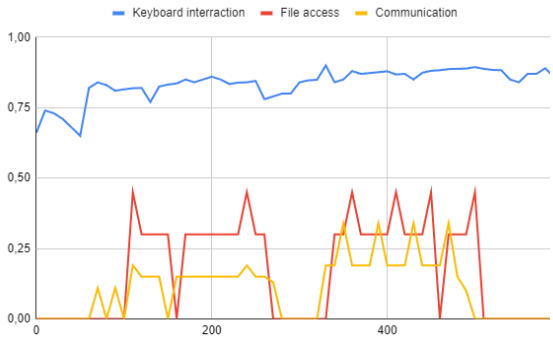

Figure 5. The number of API calls made by the JetLogger keylogger

Let's calculate the approximation and detail coefficients of the wavelet transformation for each of the types of API function calls and show the graphs for each of the groups of signals, respectively, in Figure 6, Figure 7, Figure 8:

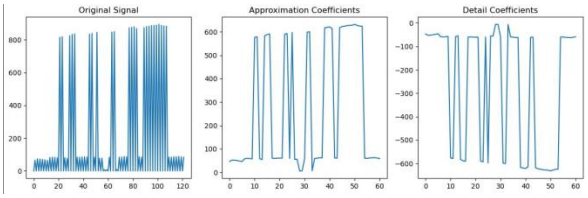

Figure 6. Graph of approximation and detailing for Keyboard interaction

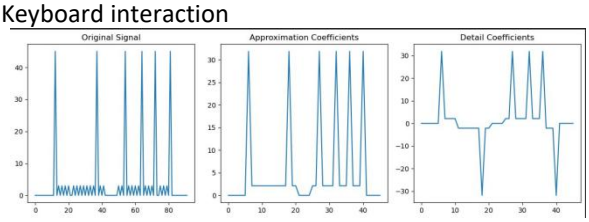

Figure 7. Graph of approximation and detailing for File access

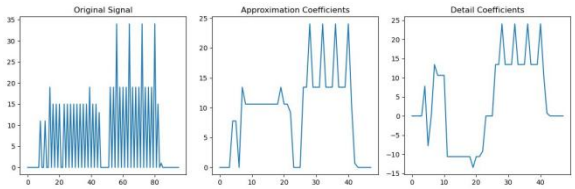

Figure 8. Approximation and detail graph for Communication

MRA processes the input signals, PAMP, DS and SS, of each data instance to obtain an intermediate result to further calculate the value of the so-called costimulatory molecular signalCSMp,which is a threat metric that can be used to identify the presence of a keylogger[18].

$$CSM_j(t+1) = \{CDM_j(t) + (SS_j(t+1) + DS_j(t+1))\}$$

If $CSM_p(t) \leq mt_j$ conversely,where $CSM_p(t), (SS(t+1), DS(t+1))$ signal values at the corresponding moments of time for this decomposition layer.

mt_j- the so-called maturation threshold, exceeding which indicates that the dendritic cell is responding to a threat. The value for is determined empirically and in our case, for simplification, it is assumed to be equal for all layers.

To compare the results of keylogger detection by different methods, it is necessary to introduce the concepts of positive P and negative N classes. A positive class refers to any attack present in the dataset (ie, anomalies). The negative class refers to normal behavior. To generate a comparison matrix, the classified entries are compared to the true classes of the data set (i.e. the ground truth), the elements of the matrix are detailed as follows:

- Correctly classified attacks are considered true positive TPs

- If TP records are misclassified, they are considered false negatives for FN
- In the case of normal behavior, correctly classified records are known as TN True Negatives
- Misclassified normal records are known as false positive FPs

For further analysis and comparison, we will use performance indicators. Accuracy refers to the ratio of correctly classified instances to the total number of verified instances:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precession represents the proportion of correctly classified attacks and is indicated:

$$Precision = \frac{TP}{TP + FP}$$

Similarly, recall (or true positive rate) recall (or true positive rate) represents the probability of detecting an attack:

$$Recall = \frac{TP}{TP + FN}$$

So, let's give a table with performance indicators of a dendritic cell model with multiple resolutions and a simple dendritic cell algorithm.

As can be seen from the comparative Table 1, for the multi-resolution dendritic cell model was able to achieve an accuracy (Acc.) of 95.50%, a precision of 98.42% (Prec.) and a recall (Rec.) of 96.89% compared to the simple dendritic cell algorithm, which was only able to achieve 91.17% accuracy, 96.73% accuracy, and 93.68% recall for the Spyrix keylogger. For the JetLogger keylogger, the multi-resolution dendritic cell model was able to achieve an accuracy (Acc.) of 94.35%, 97.29% precision (Prec.), and 96.77% recall (Rec.) against the dendritic cell algorithm, which performed 93.23 % accuracy, 97.56% accuracy and 95.23%. It should be noted that in [18] similar results were obtained for the case of DCA. Therefore, the proposed multi-resolution dendritic cell approach provided a significant improvement in accuracy for detecting the operation of a keylogger in the system. Therefore, we can see in the Table that the multi-resolution dendritic cell algorithm was able to better detect the presence of a keylogger in the system. Experimental results show that keystroke simulation can improve the visibility of keylogger behavior, and show how the keystroke agent we created can force a spy to take action on file access and communication.

Conclusions

A keylogger's ability to avoid detection determines its success. In this study, we considered strategies for detecting user-level software keyloggers, which are based on the idea of taking into account the interrelationship of several factors, which are generated by groups of specific ARI - functions. The study shows that the most promising approach is the use of the dendritic cell algorithm.

Scientific novelty consists in applying the dendritic cell algorithm with multiple resolutions. Experimental results showed that the architecture we developed can increase the detection rate of keyloggers while reducing the probability of effective evasion.

Practical significance research consists in increasing the effectiveness of anti-keyloger applications.

Further research in-plane deepening of the level of detail of the dendritic cell algorithm with multiple resolutions and refinement of parameters are planned.

Table 1.
Comparison of detection criteria for DCA and MRA

Keyloggers	Accuracy (DCA)	Accuracy (MRA)	Precision (DCA)	Precision (MRA)	Recall (DCA)	Recall (MRA)
Spyrix	91.17	95.50	96.73	98.42	93.68	96.89
JetLogger	93.23	94.35	97.56	97.29	95.23	96.77

References

[1] C. Herley and D. Florencio, “How to login from an Internet café without worrying about keyloggers,” in Symposium on UsablePrivacy and Security (SOUPS), vol. 6, 2006.

[2] Configuration of keyboard and mouse class drivers https://learn.microsoft.com/en-us/windows-hardware/drivers/hid/keyboard-and-mouse-class-drivers.

[3] Al-Hammadi and U. Aickelin, “Detecting Bots Based on Keylogging Activities,” in Proceedings of the 2008 Third International Conference on Availability, Reliability and Security, 2008, pp. 896–902.

[4] Ali et al. (2018) Ali MH, Mohammed BADA, Ismail A, Zolkipli MF. A new intrusion detection system based on fast learning network and particle swarm optimization. IEEE Access. 2018;6:20255–20261.

[5] Pratik Santoki, "Design and Implementation of Detection of Keylogger", International Journal of Engineering Development and Research (IJEDR), ISSN:2321-9939, Volume.2, Issue 2, pp.1999-2017, June 2014.

[6] Almasalmeh, Saidi & Trabelsi (2019) Almasalmeh N, Saidi F, Trabelsi Z. A dendritic cell algorithm based approach for malicious TCP port scanning detection. 2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC); 2019. 877–882.

[7] Rank Correlation Methods Public Program Analysis, 1981ISBN : 978-1-4684-6685-0Ron N. Forthofer, Robert G. Lehnen.

[8] Julie Greensmith, Uwe Aickelin, Steve Cayzer, “Introducing Dendritic Cells as a Novel Immune-Inspired Algorithm for Anomaly Detection”, International Conference on Artificial Immune Systems, 2005.

[9] Al-Hammadi, Y., Aickelin, U., and Greensmith, J. DCA for bot detection. In Proceedings of the IEEE World Congress on Computational Intelligence (WCCI) Hong Kong (2008), pp. 1807–1816.

[10] Chelly & Elouedi (2011) Chelly Z, Elouedi Z. Further exploration of the fuzzy dendritic cell method. Berlin: Springer; 2011. 419–432

[11] Greensmith, J., Aickelin, U. and Twycross, J., Detecting Danger: Applying a Novel Immunological Concept to Intrusion Detection Systems, International Journal of Communications, Network and System Sciences, 2010

[12] Greensmith, J., Aickelin, U., Twycross, J. (2006). Articulation and Clarification of the Dendritic Cell Algorithm. In: Bersini, H., Carneiro, J. (eds) Artificial Immune Systems. ICARIS 2006. Lecture Notes in Computer Science, vol 4163. Springer, Berlin, Heidelberg

[13] Limon-Cantu D, Alarcon-Aquino V. Multiresolution dendritic cell algorithm for network anomaly detection. PeerJ Comput Sci. 2021 Oct 19;7:e749. doi: 10.7717/peerj-cs.749. PMID: 34805504; PMCID: PMC8576553

[14] Daubechies (1992) Daubechies I. Ten lectures on wavelets. CBMS-NSF Regional Conference Series in Applied Mathematics; Society for Industrial and Applied Mathematics; 1992

[15] Alarcon-Aquino & Barria (2009) Alarcon-Aquino V, Barria J. Change detection in time series using the maximal overlap discrete wavelet transform. Latin American Applied Research. 2009;39:145–152

[16] Shibaev, G., & Galchynskyi, L. (2023). Detection of keyloggers using the dendritic cell algorithm with multiple resolutions. Grail of Science, (30), 173–176. https://doi.org/10.36074/grail-of-science.04.08.2023.027

[17] Elisa, Chao & Yang (2020) Elisa N, Chao F, Yang L. A study of the necessity of signal categorization in dendritic cell algorithm. In: Ju Z, Yang L, Yang C, Gegov A, Zhou D, editors. Advances in Computational Intelligence Systems. Cham: Springer International Publishing; 2020. 210–222

[18] Jun Fu, Huan Yang (2014), Enhancing Keylogger Detection Performance of the Dendritic Cell Algorithm by an Enticement Strategy, The 28th Research Institute of China Electronics Technology Group Corporation; 2014.