Mimicking User Keystrokes to Detect Keyloggers with Dendritic Cell Algorithm

Jun Fu[a], Huan Yang[a], Yiwen Liang[b], Chengyu Tan[b]
[a] The 28th Research Institute of China Electronics Technology Group Corporation, Nanjing 210007, China
Email: {doctorfj, happyfairy106}@163.com
[b] Computer School, Wuhan University, Wuhan 430079, China
Email: ywliang@whu.edu.cn, nadinetan@163.com

***Abstract*—Evasive software keyloggers hide their malicious behaviors to defeat run-time detection. In this paper, based on the analysis of the evasion mechanisms used by common software keyloggers, we established a framework for their detection. Mimicking user keystrokes, the framework we built could induce keyloggers showed more obvious malicious activities. These 'amplified' activities are then correlated by the dendritic cell algorithm (an immune-inspired algorithm) to final determine the existence of a keylogger in a host. Preliminary experimental results showed that the framework could improve the performance of keylogger detection and hard to evade.**

Keywords-keylogger; keystroke simulation; dendritic cell algorithm (DCA); correlation

I. INTRODUCTION

With the development of e-commerce and online games, software keyloggers which steal confidential information by monitoring a user's keyboard actions are becoming a new trend for malware [1]. They intercept and log all keystrokes, and transmit this information to profit-driven attackers. Unlike other types of malicious programs, keyloggers are designed to capture what is done on a PC without attracting the attention of users and present no threat to the system [2]. This makes them largely undetectable by most anti-virus and anti-keylogger applications [3], [4].

To overcome the problems above, security experts are trying to use behavior-based detection techniques that analyze API calls of a process to classify it as keylogger or not [4], [5]. However, these methods all have some shortcomings to some extent. Detection [4] relies on single behavior (setting Windows hooks) has a high rate of false positives (FP) [5]. Though correlation of multiple behaviors (keystroke tracking, file access and network communication) reduces the FP rate, it seems that the detection is prone to be evaded when specified time window and simple correlation algorithm are used [5].

For the purpose of improving the detection performance, Fu [6] uses an immune-inspired algorithm - dendritic cell algorithm (DCA) to correlate the behaviors mentioned by [5]. As an algorithm, the DCA performs multi-sensor data fusion on a set of input signals, and these signals are correlated with potential 'suspects', leading to information which will state not only if an anomaly is detected, but in addition the culprit responsible for it. By using variable time windows and time sequence of the different behaviors, the DCA improves the detection performance to a certain degree. But every coin has two sides. The correlating feature of the DCA can be exploited by crafty attackers to evade detection by reducing the frequency of the malicious behaviors [7]. The experimental results of [6] support the above claim.

Man-to-machine interfaces cannot be ignored when fight against keyloggers [2]. In this paper, we analyzed the evasion mechanisms used by common software keyloggers. We discovered that the keystroke (especially keystroke of the special key, such as '*Enter*' key) frequency is an important trigger for keyloggers to log and send captured information. As a result, we built an induction-correlation framework for keylogger detection. In this framework, we synthesized man-to-machine interactions by implementing a keystrokes simulation program. The program can induce keyloggers to exhibit more malicious activities without disturbing normal applications. Then, the 'amplified' behaviors are correlated by the DCA in order to identify the keylogger as early as possible. Experiments were conducted to test capabilities of our framework to improve the detection rate and reduce the possibility of successful evasion.

II. RELATED WORK

Since signature-based detection has nothing to do against novel keyloggers [2], security experts are now focusing their attentions to behavior-based detection techniques that analyze API calls of a process to classify it as benign or malicious. As keyloggers always use Windows hooks, Aslam [4] disassembles all running processes searching for *SetWindowsHookEx* function to find keylogger processes. This method has a high rate of false positives as legitimate applications also use this function to set hooks [5].

Rather than relying on single behavior (setting hooks), Al-Hammadi [5] proposes a method to detect keylogging activities with correlations between different behaviors (keystroke tracking, file access and network communication). Although the technique has a relatively low false positive rate, the detection rate is not high because specified time windows and simple correlation algorithm (an algorithm using Spearman's Rank Correlation) are used [6].

Based on the work above, Fu [6] uses the immune-inspired dendritic cell algorithm (DCA) to correlate multiple behaviors described in [5]. The DCA is based on an abstract model of the behaviors of dendritic cells which are natural intrusion detection agents of the human body. These cells collect antigens and signals (environmental conditions of the

antigens), and combine the evidence of damage (signals) with the collected suspect antigen to provide information about how 'dangerous' a particular antigen is. The DCA performs multi-sensor data fusion on a set of input signals and antigens, leading to information which states not only if an anomaly is detected, but in addition the culprit responsible for it. More information about the DC and the DCA please refer to [8].

The input signals defined in [6] are derived from the frequency of invocations of keystroke tracking functions, the time difference between two consecutive *WriteFile* calls, the relation between different categories of function calls and the time difference between two outgoing consecutive communication functions. The process (identified by PID) which causes the calls is defined as antigens [6]. The DCA correlates antigens with input signals, resulting in a pairing between signal evidences and antigen suspects, and the identification of the keylogger process in the end. However, as the DCA distinguishes between normal and potentially malicious antigens on the basis of neighboring antigens, the crafty attackers can exploit this correlating feature to evade detection by reducing the 'concentration' of antigens in DCs [7].

The experimental results of [6] confirmed the above conclusion. The keylogger they used in experiments hid its behaviors by logging and sending keys only when enough keystrokes were intercepted or special keys (such as '*Enter*' key) were pressed. In experiments that long sentences were entered, the frequency of the malicious activities generated by the keylogger decreased significantly compared to the one observed in short sentence scenarios. The same trends were also found in detection performances of the DCA because of the reducing of the 'concentration' of the malicious antigens. In the real world, we believe users keystroke patterns are similar with the long sentence scenarios described in [5] and [6]. This challenges the DCA to detect keyloggers in the real environment.

III. KEYLOGGER ANALYSIS

In this paper, we analyzed the source code of some typical open source keyloggers running on *Windows NT* operating systems, such as *Keymail V0.7*, *Spybot V1.2* and *Morsa-Keylogger V1.8*. Then we compiled and executed these source codes to find their run-time features. Based on the static and dynamic analysis, we discovered the relationships between different behaviors generated by these keyloggers, and revealed the evasion mechanisms often used by them.

Through static analysis, we found that all keyloggers worked in a similar manner. They all firstly tracked keystrokes and then wrote them to a file or/and send them to a destination across the Internet (via Email, FTP and etc.). The most important difference between these keyloggers was the timing that triggered file access and communication activities. These activities were performed when:

1) *intercepted every keystroke;*
2) *the keystrokes intercepted reached a certain amount;*
3) *special keys (such as 'Enter' key) were pressed.*

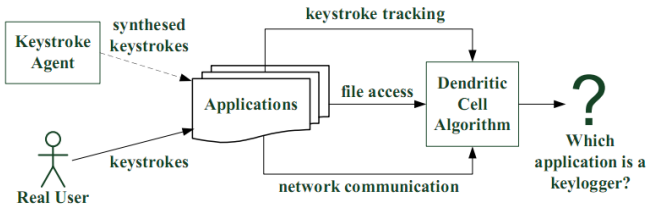


Figure 1. The induction-correlation framework for software keylogger detection by the DCA.

After running the compiled source codes, we found the keyloggers (such as *Keymail*) using the *1)* trigger condition generated more file access and communication behaviors. However, these behaviors were relatively rarely observed when the keyloggers (such as *Spybot* and *Morsa-Keylogger*) with the *2)* and the *3)* trigger conditions were executed. So we could make a conclusion that it is the *2)* and the *3)* trigger conditions that gave keyloggers capabilities to evade the correlation-based detection (such as the DCA).

Fortunately, the evasion mechanism above is a double-edged sword. Besides hiding keylogger behaviors, it greatly exposes the existence of the keylogger when high frequency of keystrokes (especially special keystrokes) is encountered. That is why we use the keystroke simulation to enhance the detection performance of the DCA.

IV. INDUCTION –CORRELATION FRAMEWORK

We assume that the host to be monitored is infected with a keylogger without a user's awareness. The installed keylogger logs and sends the captured information when a user types his/her privacy via keyboard. In this paper, we propose an enhanced approach to detect software keyloggers on a host. The approach consists of two steps: 1) the induction of the keyloggers, 2) the correlation of the behaviors exhibited by them. We emphasize on the first step, describing how it improves the performance of the second step. The framework of our approach is shown in Fig. 1.

Because the frequency of keystrokes in real environment is not high, the behaviors of the keylogger are not evident enough [6]. Therefore, we design a keystroke agent application to frequently generate random keystrokes, hoping that these keystrokes will be seen by the keylogger, but will not affect normal applications. As a result, the behavior of the keylogger will be more obvious in the stimulation of a large number of random keystrokes in a short time. Meanwhile, the keystroke agent holds the simulated keystrokes within a hidden application it creates to avoid them passing to the other applications. Thus the normal applications will not be affected by the simulated keystrokes since they only focus on the keystrokes passed to them.

We also focus on keystroke tracking, file access and network communication behaviors exhibited by applications. By correlating API calls generated by these behaviors, the DCA can classify the application running in a host as a keylogger or not. For example, file access shortly after the keystroke tracking strongly indicates that there exist keylogging activities. The more active the application is in that period, the more likely it is a keylogger.

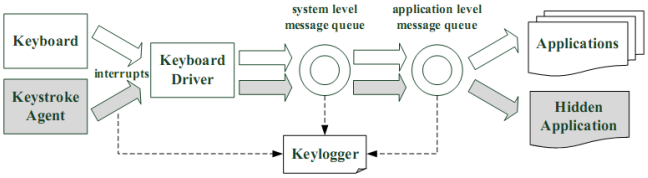


Figure 2. How keystrokes are handled by a Windows NT operating system and intercepted by a keylogger.

A. *The induction phase*

In the induction phase, we synthesize random keystrokes to induce keyloggers. In order to find a way to simulate keystrokes which will be seen by a keylogger, we must understand how an operating system generates and handles keyboard events. We also must be clear that how a keylogger intercepts keystrokes in this process.

As in Fig. 2, a *Windows NT* operating system generates a keyboard interrupt when a key was pressed. Then the keyboard driver transforms the interrupt to a system-defined message and puts it into the '*system level message queue*'. Tracking the focused application at the time when the keyboard interrupt was generated, the operating system passes the message to the '*application level message queue*' of that specific focused application. Now it's the responsibility of that application to handle this key accordingly. If the operating system does not find any specific focused application, it simply discards that key. In this process, keyloggers employ very low level operating system calls [9], such as *GetKeyboardState* or *GetAsyncKeyState*, to intercept keystroke messages or detect keyboard interrupts directly. So the keyloggers see everything whenever a key is pressed.

In this paper, we design a keystroke agent according to the mechanisms described above. By invoking system kernel (*keybd_event*), the agent simulates keyboard event completely. Because a keylogger tracks keystrokes from all applications (including keystroke agent application) in order to log sensitive data entered in them, it could see the simulated keystrokes since they are the same with the real keystrokes. But the keylogger doesn't understand what it sees and it can't tell the keystrokes generated by real users via keyboard from the ones generated by phantom users via our keystroke agent. When we simulate keystrokes frequently, in order to log and send these plentiful keys, the keylogger has to perform more file access and communication behaviors.

On the other hand, the simulated keystrokes must not affect normal applications. Before starting to simulation, the agent creates a hidden window and sets the current active window to it. Then the simulated keystrokes are generated and passed to the hidden window which simply discards the keys received. After the simulation, the active window is set back to the active window before simulation. The keystroke agent regularly performs the procedure above. Since the execution time of this process is very short, the active window switches are almost imperceptible to users.

B. *The correlation phase*

In this paper, we use the dendritic cell algorithm (DCA) to correlate API calls generated by all running applications to identify keylogger applications. In order to obtain the API calls, we implement a hook program to monitor three types of function calls:

1) ***Keyboard Tracking APIs****: GetKeyboardState, GetAsyncKeyState and GetKeyNameText.*

2) ***File Access APIs****: CreateFile, OpenFile, ReadFile and WriteFile.*

3) ***Communication APIs****: socket, send, recv, sendto and recvfrom.*

These API functions are often employed by keyloggers to implement their keylogging and other features, but also may form part of legitimate usage. Therefore, an intelligent correlation method such as the DCA is required to determine if the invocations of such functions are indeed anomalous. Signals and antigens are vital input to the DCA. To facilitate comparison, we use the same definitions of the signal and antigen described in [6].

V. EXPERIMENTS

The aim of our experiments is to verify that the keystroke simulation we proposed can enhance the visibility of a keylogger's behaviors, and thus can improve the detection performance of the DCA. To achieve this goal, we chose the same keylogger instance (*spybot*) and benign instances (*notepad* and *mirc* [10]) used in the experiments in [6], and set up the same network environment for their running.

The experiments were divided into two groups which showed the detection performance differences between the DCA with the keystroke agent (E2) and the DCA without (E1) the keystroke agent. The agent simulates keystrokes once every 5 seconds, synthesizing some random (numbers and letters) and special keys ('*Enter*' key). Each experiment was repeated for 10 times and lasted 600 seconds. Without any operations in the first 60 seconds, we used *notepad* and *mirc* to input sentences for 180 seconds respectively with an interval of 60 seconds. We had no operations in the final 120 seconds.

In both groups of experiments, we monitored two scenarios of typing. We typed short sentences in one scenario (E1.1 and E2.1) and long sentences in the other (E1.2 and E2.2). The sentence ended with '*Enter*' key. By monitoring two typing scenarios, we were able to show the effects of different keystroke patterns on our detection scheme and the effects of keystroke simulation in different input mode.

In [6], the contents of the sentences were random, and the lengths of the sentences were not mentioned. To get closer to real user keystroke patterns, we collected 200 commonly used English sentences: 100 long (19-48 characters) and 100 short (9-23 characters) sentences, and typed them one by one in corresponding scenarios. Because of the changes in typing mode, we adjusted the signal threshold values used in [6]: N_{ph} *= 1301(times/s),* N_{pl} *= 1105(times/s),* N_{d1l} *= 7000(ms),* N_{d1h} *= 20000(ms),* N_{d2h} *= 2(times/s),* N_{s1l} *= 5000(ms),* N_{s1h} *= 10000(ms),* N_{s2l} *= 55(times/s).* They were set based on the statistical results of the frequency of API calls. Take

keyboard tracking API calls for example, the average frequency generated by spybot was 1203(times/s), the standard deviation was 88(times/s). And the max frequency concerning notepad or mirc was 55(times/s). Therefore, we set $N_{ph} = 1203 + 88 = 1301(times/s)$, $N_{pl} = 1203 - 88 = 1105(times/s)$, $N_{s2l} = 55(times/s)$. The maliciousness of keyboard tracking API calls was depend on whether they were in a high frequency (greater than N_{ph}) or a low frequency (less than N_{s2l}). The population of DCs was set to 100, the DCA chose 10 DCs every time an antigen or a signal was arrived for their storage. The fuzzy migration threshold value of DCs was between 1500 and 2000. The weight matrix was the same with [6].

A. *Results*

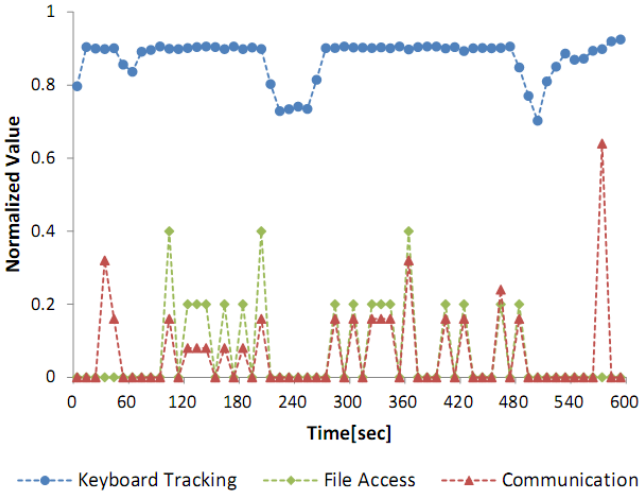
We first give a look at the frequency of API calls generated by *spybot* in E1 and E2. The x-axis represents time in seconds while the y-axis represents the normalized value of API call frequencies. The normalized API call frequency values represent the total value we get during 10 seconds divided by the maximum value of the whole period (600 seconds).

For *spybot* instance, the results from E1 and E2 show that there are significant differences between API call frequency without keystroke simulation and the one with keystroke simulation in both two typing scenarios, as depicted in Fig. 3.

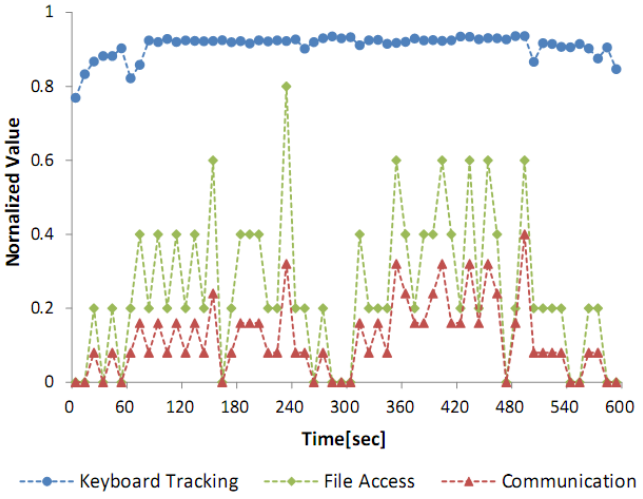
From Fig. 3(a) and Fig. 3(b), we can see that although the keyboard tracking API calls generated by *spybot* maintain a high frequency in both E1.2 and E2.2, the frequencies of file access and communication API calls in E2.2 are much higher than the ones in E1.2, especially when long sentences are typed for text editing (using *notepad* from 61 to 240 second) or online chatting (using *mirc* from 301 to 480 second). The same differences are also shown in E1.1 and E2.1 when short sentences are entered. Therefore, we can conclude that the keystroke agent does has the ability to amplify the malicious behavior exhibited by *spybot*.

The intercepted API calls invoked by *spybot*, *notepad* and *mirc* are used to generate corresponding signals using the method described in section IV-B. The DCA then processes and analyzes these signals to determine which process has keylogger behaviors. **错误！未找到引用源。** gives the results from the DCA. The values in the last two columns are the mean values and standard deviation values (in parentheses) in 10 repeated experiments.

A threshold (T) is applied to *MAC* to make the final classification decision. The process whose *MAC* is higher than T is termed malicious, and vice-versa. Because the dataset contains one malicious instance and two benign instances, we can define $T = 1/(1+2)$. So the process with $MAC > 1/3$ is considered to be a keylogger process. From the table, we observe that the DCA detects the keylogger process in all scenarios except in E1.2. This means that the keylogger is more difficult to be detected when user inputs long sentences. However, with the help of the keystroke agent, the DCA detects keylogger process with no false negatives no matter which input mode is used. It is also noteworthy that no false positives are generated in all scenarios.



(a) API functions invoked by *spybot* in E1.2



(b) API functions invoked by *spybot* in E2.2

Figure 3. API functions invoked by *spybot* in long sentences scenarios (E1.2 and E2.2). We use *notepad* program for text editing during 61-240 seconds and *mirc* program for online chatting during 301-480 seconds.

TABLE I. RESULTS FROM THE DCA

Scenario	*Process Name*	*The Number of Antigens*	*MAC*
E1.1	spybot	6767(23.0)	0.540(0.0049)
	notepad	1470(0)	0.087(0.0022)
	mirc	1804(3.8)	0.087(0.0027)
E1.2	spybot	5164(35.5)	0.271(0.0111)
	notepad	1140(0)	0.024(0.0020)
	mirc	1795(5.9)	0.037(0.0030)
E2.1	spybot	8045(23.2)	0.599(0.0065)
	notepad	1520(0)	0.064(0.0030)
	mirc	1718(1.4)	0.109(0.0024)
E2.2	spybot	6819(43.4)	0.427(0.0060)
	notepad	1640(0)	0.056(0.0033)
	mirc	1804(5.4)	0.071(0.0029)

B. Discussions

From the results of the experiments, we can find that although the *MAC* values in TABLE Ⅰ are relatively low compared to the results from the experiments in [6], the basic experimental conclusions do not change:

- Keystroke simulation can enhance the visibility of keylogger behaviors, and thus can improve the detection performance of the DCA.
- Keystroke patterns have an obvious impact on the detection performance of the DCA. And the effects of keystroke simulation vary in different input mode.

Fig. 3 demonstrates that the keystroke agent we implemented can induce *spybot* to perform more file access and communication behaviors. And the simulation method improves the *spybot* detection performance of the DCA to some extent in the same environment, as depicted in TABLE Ⅰ. The *MAC* value increases by about 58% and 11% in average when we type long sentences (E1.2 and E2.2) and short sentences (E1.1 and E2.1) respectively. In 20 experiments without simulation (E1.1 and E1.2), only 10 experiments detects *spybot*, detection rate is 50%. In contrast, all 20 experiments report *spybot* detection when keystroke agent is used.

The improvement of detection efficiency in long sentences scenarios is much higher than short sentences scenarios. That may because the *spybot* behaviors are already obvious enough since the '*Enter*' key is typed frequently when the user type short sentences. So there is not much room for improvement in short sentences scenarios.

The keystroke agent passes synthesized random keys to a hidden window created by it. Some keyloggers not only can intercept these keys, but also can know the destination of them (We find *spybot* has this ability in experiments). As a result, keyloggers can defeat the keystroke agent by the way that not to handle the keys send to the hidden window created by the agent. The future work will find a possible way to solve it.

VI. CONCLUSION

The success of any keylogger is determined by its ability to evade detection. In this paper, we analyzed the evasion mechanisms used by common software keyloggers and proposed an induction-correlation framework for keylogger detection. In this framework, keystrokes simulation raises the frequency of the keystrokes, and thus induces keyloggers produce more malicious behaviors to deal with these synthesized keystrokes. Then the 'amplified' behaviors are correlated by the DCA in order to find the keylogger process as early as possible to reduce the loss of privacies. Experimental results showed that the framework we built can improve the keylogger detection rate and reduce the possibility of successful evasion..

ACKNOWLEDGMENT

This work was supported by the Defense Industrial Technology Development Program of PR China (*GrantNo. A1420080183*).

REFERENCES

[1] "A considerable increase has been seen in the number of malicious programs with keylogging functionality," http://www.securelist.com/en/analysis/204791931/.

[2] S. Sagiroglu and G. Canbek, "Keyloggers," Technology and Society Magazine IEEE Fall 2009, vol. 28, pp. 10–17.

[3] M. Baig and W. Mahmood, "A Robust Technique of Anti Key-Logging using Key-Logging Mechanism," in Inaugural IEEE-IES Digital EcoSystems and Technologies Conference (DEST'07), 2007, pp. 314–318.

[4] M. Aslam, R. Idrees, M. Baig, and M. Arshad, "Anti-Hook Shield against the Software Key Loggers," in Proceedings of the National Conference on Emerging Technologies, 2004, pp. 189–191.

[5] Y. 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.

[6] J. Fu, Y. Liang, C. Tan, and X. Xiong, "Detecting Software Keyloggers with Dendritic Cell Algorithm," in Proceedings of the 2010 International Conference on Communications and Mobile Computing (CMC), 2010, pp. 111–115.

[7] M. SalmanManzoor, S. Tabish, and M. Farooq, "A Sense of 'Danger'for Windows Processes," in Proceedings of the 8th International Conference of Artificial Immune System (ICARIS 2009), 2009, pp. 220–233.

[8] J. Greensmith, U. Aickelin, and G. Tedesco, "Information fusion for anomaly detection with the dendritic cell algorithm," Information Fusion, vol. 11, no. 1, pp. 21–34, 2010.

[9] C. Herley and D. Florencio, "How to login from an Internet café without worrying about keyloggers," in Symposium on Usable Privacy and Security (SOUPS), vol. 6, 2006.

[10] "mIRC client application," http://www.mirc.com.