IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 12, 2018, accepted November 27, 2018, date of publication December 7, 2018,
date of current version December 31, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2884964

Detection and Elimination of Spyware and Ransomware by Intercepting Kernel-Level System Routines

DANIAL JAVAHERI[1], MEHDI HOSSEINZADEH[2,3], AND AMIR MASOUD RAHMANI[1]

[1]Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran 1477893855, Iran
[2]Department of Computer Science, University of Human Development, Sulaimaniyah 0778-6, Iraq
[3]International Campus, Iran University of Medical Sciences, Tehran 1449614535, Iran

Corresponding author: Mehdi Hosseinzadeh (hosseinzadeh.m@iums.ac.ir)

ABSTRACT Spyware is the most complex, obfuscated, and targeted class of malware, which has grown dramatically in recent years. Spyware is designed for secret, long-term, and persistent missions. This paper provides a novel method for detection, tracking, and confronting the stealth and obfuscated spyware and ransomware, including keyloggers, screen recorders, and blockers. The proposed method of this paper is based on a dynamic behavioral analysis through deep and transparent hooking of kernel-level routines. We used linear regression, JRIP, and J48 decision tree algorithms as a classifier to recognize three classes of malware. This paper presents the main architectural plan of an anti-spyware application to track spyware footprints in order to detect and force terminate running processes, eliminate executable files, and restrict network communications. The efficiency of the proposed method was evaluated from the viewpoint of accuracy in detecting real-world samples of spyware by ROC curve analysis and from the viewpoint of success rate to confront effectively with active spyware. Our proposed method was able to recognize spyware with an accuracy of about 93% and an error rate near 7%. In addition, the proposed system can disinfect an operating system from infection by spyware with a hit rate of about 82%.

INDEX TERMS Malware analysis, spyware detection, stealth, obfuscation, data mining.

I. INTRODUCTION

Rapid development and prevalence of Internet systems are causes of growth in cybercrimes such as hacking, phishing, identity theft and malware propagation [1]. It is also noteworthy that criminals are moving faster than the development of the technologies that ensure security [2]. Malware is a software designed for malicious goals such as stealing vital data and executing destructive binaries. Therefore, malware compromises the availability, confidentiality, and integrity of a victim's system. Malware is software such as a Trojan, viruse, worm, backdoor, rootkit, botnet or spyware that intrudes a system without the consent of its owner [3].

The growth rate of malware has drastically increased in recent years. According to [4], 670 million malware samples were detected by McAfee Labs in 2017. A report by Kaspersky Lab in [5] indicates that the number of new malware files detected by its products in 2016 increased to 323,000 per day. This is while similar amount was only 13,000 in 2015. According to [6], the total number of malware discovered in 2016 showed a growth of 36% compared to the year 2014. The reason for the high rate of malware generation is obfuscation and polymorphic engines. Moreover, the newly produced malware is getting more complex, targeted and shrewd.

Malware is designed for different types of Operating Systems (OS) such as Windows, Android, Mac, and iOS. Hence, it can be categorized based on the design platform. In recent years, with the significant increase in the popularity of smartphones, Android OS has become one of the world's most popular smartphone platforms [7], [8]. Android malware attacks are therefore increasing rapidly. Android was the target of 97% of global mobile malware in 2013. In 2016, the number of Trojans contained in Android adware increased to nearly 700,000 [8]. Although malware on other platforms such as Android and Mac has been growing, malware designed specifically for the Microsoft Windows family of operating systems accounts for over 90% of all malware [9]. Figure 1 indicates the growth of malware for different operating systems between 2009 and 2018 [10]. It is evident that

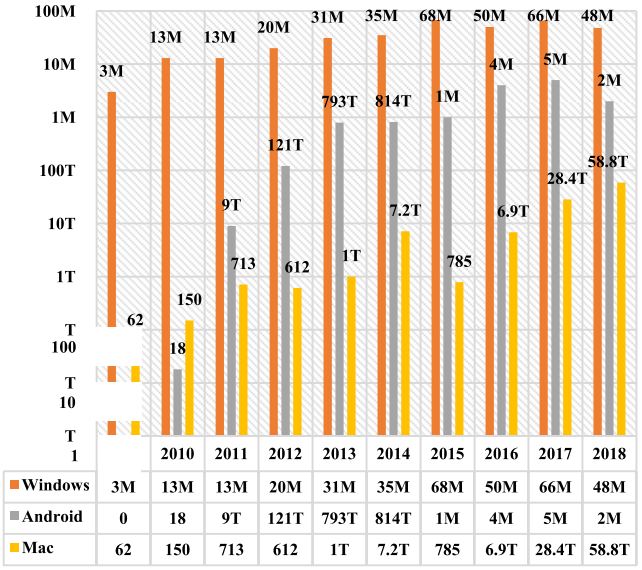

FIGURE 1. A comparison of malware growth for different operating systems during the last 10 years (T is thousand and M is million).

the amount of Windows malware was 24 times higher than that of Android malware in 2018.

Spyware is a latent malicious executable that collects important and valuable data such as passwords and financial information without the permission of the owners and sends them to the attacker. Spyware is one of the most dangerous and mysterious classes of malware designed for secrecy and durability. Attackers install spyware binary files on the compromised machine to steal valuable information in silence; then they can establish a long-term connection based on the covert channel to the victim's machine in order to launch an Advanced Persistent Threat (APT) attack [11].

Malicious programmers have introduced code obfuscation, dynamic code loading [12] and metamorphic engine –which is self-mutating and consequently, changes its fingerprint automatically on every execution in order to bypass traditional scanners that work based on string and fingerprint matching [13]. Therefore, traditional signature-based methods and generic classifiers such as the method in [14] become less potent in detecting metamorphic malware and novel patterns of malware attacks [15] specifically in the spyware class. Hence, the behavior analysis of malware is needed to detect malignancy. Traditionally, malware analyses have been a tedious and time-intensive manual process because they are needed for timely zero-day discoveries [16]. Analysis of very large volumes of malware samples is only possible via automated dynamic analysis. Therefore, malware analyzers search for automation processes of malware analysis.

It is noticeable that according to a Symantec report [17], the number of zero-day vulnerabilities discovered in 2015 has increased to 125% compared to the year before. These vulnerabilities are very valuable for spyware to design a pattern of attack while remaining anonymous. Hence, spyware is the most complex and obfuscated class of malware. In addition, it is almost impossible to recognize all novel stealth spyware with a generic malware classifier. Therefore, a particular classifier for detection of stealth and obfuscated spyware was intended in our study.

The rest of this paper is organized as follows. Spyware attack techniques are introduced in Section 2. The proposed method for detection and tracking is explained in Section 3. The architectural plan of our proposed anti-spyware application and the proposed method for disinfection of malware are described in Section 4. Finally, Section 5 indicates and analyzes the results of the evaluation of the proposed system.

II. SPYWARE AND RANSOMWARE

Spyware is divided based on the performance mechanism and the type of hijacked information. Keyloggers and screen recorders are the most dangerous and mostly used kinds of spyware. As mentioned before, stealth and durability are the two most important non-functional attributes (quality attributes) in the architecture of the mentioned malware.

In order to hijack information, spyware needs hooking OS facilities, especially system functions (APIs).[1] The aim of hooking is to act as a mediator between the OS resources and incoming requests from the user or installed applications on the victim system. Therefore, it is able to log and hijack incoming information. Most of the malware can send the hijacked information into pre-determined destinations and Command and Control (C&C) center through network connections.

Ransomware is another class of malware studied in this paper. Ransomware is applied as malware because of its malignant behavior, which forces victims to take undesired and compulsory actions like extortion of money. A common category of this class of malware is blockers. In this study, we concentrated on keyloggers, screen recorders, and blockers.

A. KEYLOGGERS

Keyloggers are smart malicious binaries that take an action to record keys that the user presses when the system is active or at sensitive times. For example, when the user wants to enter his/her email password in the internet browser, do online shopping, or write a document in a text editor. Spyware can distinguish valuable times by calling *FindWindow* and *GetWindow* system functions to get the name of running windows [18]. Another method is to use *Process32First* and *Process32Next* system functions for searching among active processes of the OS to find special process names such as 'iexplore' [19]. This tactic gives rise to a decrease in the size of hijacked information by filtering insignificant information. This tactic has another obvious advantage: when hijacked information is sent via the network connection to C&C, decreasing the size of the sent information can shorten the sending time, especially when spyware is using a low-bandwidth covert channel such as timing-base channels.

[1] Application Programing Interface

In addition, decreasing the size of the hijacked information and limiting activation time of the spyware extremely decrease the probability of anomaly patterns recognition at network traffic or malicious behavior of the OS by surveillance tools like Antivirus and SIEM.[2] Profound data mining on a large number of keyloggers shows that these malware take an action to hook *NTUserGetASyncKeyState* and *NTUserGetKeyState* form the System Service Descriptor Table (SSDT(shadow in order to meet their purpose that is discovering what keys are pressed. The above-mentioned functions deliver received states from the upper APIs at the user-level to drivers that manage the keyboard. The aim of a keylogger is to act as a mediator between these system functions and user requests they come to know through discovering the pressed keys. This is done by reading the values of the mentioned function parameters and recording them in a log file.

B. SCREEN RECORDERS

Screen recorders are another common subclass of spyware. They take screenshots from victim user activities. A screen recorder needs to stay hidden very well. By running different types of spyware in analyzer environments equipped with monitoring tools (e.g. cuckoo sandbox and APIMon) we could derive their system calls. Although some types of spyware try to hide their APIs through obfuscation techniques, especially dynamic programming, we keep the analyzer environment transparent by embedding monitoring facilities at the lowest possible level of the OS and altitude of the Input/Output (I/O) stack.

A screen recorder sends its requests in the form of a handle to *GetWindowDC* API from the User32.dll library in order to image a window screen or desktop screen. The mentioned function returns all contents of the desired window including the menus and labels in the form of a Device Context (DC) structure as return values. The return values of this function are passed, as input arguments, to another system function, i.e. *BitBlt,* which belongs to the GDI32.dll library. The output of the recent function is an image of the input DC. Other parameters of this function determine characteristics of the output image such as length and width [20]. The structure of system functions *GetWindowDC* and *BitBlt* are as follows.

```
HDC GetWindowDC
(
_In_  HWND hWnd
);
```

```
BOOL BitBlt(
_In_  HDC hdcDest,
_In_  int nXDest,
_In_  int nYDest,
_In_  int nWidth,
_In_  int nHeight,
_In_  HDC hdcSrc,
_In_  int nXSrc,
_In_  int nYSrc,
_In_  DWORD dwRop);
```

Furthermore, *GetWindow* and *FindWindow* system functions are necessary for this spyware. Screen recorders create a handle for imaging by searching the name of a window with the mentioned system functions. The aim of this action is to restrict the imaging to special windows such as an internet browser; for this reason, some security applications name their windows randomly in order that malware cannot search them.

C. BLOCKERS AND CRYPTORS

Static behavioral analysis of several samples of blockers shows that most of them are filter drivers placed in altitudes higher than those of the OS keyboard and mouse original drivers in the space of I/O stack. Then, using Windows Filtering Platform (WFP) facilities, this malware takes action to capture the sent IRPs[3] into hardware and drop them [18]. Therefore, the connection between the OS and the keyboard/mouse is interrupted.

Another mechanism used in blockers is to hook the *SendInput* system function from the User32.dll library at Ring (3) [18] or its equivalent function at Ring (0) from the OS *NTUserSendInput*. In this way, incoming requests to the mentioned functions receive no response and pressed keys by the user do not influence the OS. The results of our experiments showed that hooking of the *NTUserSendInput* function at the kernel-level was more common in blockers. This is because unhooking a blocked API at the kernel-level is tremendously harder than at the user-level and sometimes it is irrevocable.

Blockers show a message that binds the victim to pay expenses to open his/her system. In this respect, they are placed in the ransomware class. Another category of ransomware encrypts important low size files such as docx, pdf, pptx, xlsx and accdb documents with encryption algorithms. The victim is then obliged to pay in order to provide the key for decryption. This category of ransomware is known as Cryptors.

D. RELATED WORKS

Some methods have been introduced in recent years for recognition of malware based on the static or dynamic behavior analysis that they use such as API calls, opcodes, register values and other features for behavioral modeling.

Islam *et al.* [21] proposed a new method for classification of malware based on integrated static and dynamic features. The authors extracted some features through a static analysis by unpacking malware manually and some other features through a dynamic analysis by execution of the malware, which unpacks itself. They claimed that their method had an error rate of less than 2.7%.

Gupta and Kumar [22] proposed a novel method to detect the execution of the malicious application in a cloud environment. This method uses a sequence of system calls for making a signature to distinguish malicious behaviors during the execution of unknown applications. The accuracy of this

[2]Security Information and Event Management

[3]Input/Output Request Packet

method has been evaluated to be about 98% by using a dataset collected from email traffic of the University of New Mexico in private cloud resources.

Wang *et al.* [23] presented a new method to classify malware based on a social network analysis. Jang's method features derived from the graph structure of system calls. They used social network properties as applied to the call graph, such as the degree of centrality, the degree of distribution, density, component ratio and coefficient of clustering. A system call graph consists of system calls found in the execution of individual malware families to explore the distinguishing features of various malware families.

Wang and Wang [24] introduced a new classifier based on the Support Vector Machine (SVM) algorithm to solve the problem of classification accuracy regarding unspecified malware. Malware detection depends on correct extraction and completeness of training signatures. This model is able to classify 60 families of real malware automatically. The overall detection accuracy of this model was estimated to be more than 85% for unspecific malware.

Das *et al.* [25] proposed an approach for online malware detection using hardware-enhanced architecture by a combination of processor and FPGA.[4] The authors used a frequency-centric model for feature construction using system call patterns of unknown malware and benign samples. They claimed that their model could detect 46% of malware within the first 30% of its execution and 97% of malware during a complete execution.

Liu *et al.* [26] proposed a machine learning based malware detection method. The authors used n-gram of opcode as grayscale images for feature selection and Shared Nearest Neighbor (SNN) as a clustering algorithm to classify unknown malware. They used a dataset with 20,000 samples of malware and benign files to evaluate their model. The results of the evaluation showed a best accuracy of 98.9% for classification of the unknown malware and successfully detected 86.7% of novel malware.

Lin *et al.* [27] proposed a new mechanism to overcome challenges of time-consuming of dynamic analyses. They used a modified Xen hypervisor, which could generate a virtual clock source according to a predefined speed ratio. This ability helped accelerate the running of the sandbox on the modified hypervisor. An evaluation by the authors indicated that performance increased up to 42%.

There is an important issue with extracting the name of APIs in most of the behavioral analysis methods. Some methods focus on the static analysis of Portable Executable (PE) structure to extract API names from the Import Address Table (IAT), but these methods are very vulnerable in confronting with obfuscation and metamorphism techniques such as IAT encryption, IAT smashing and dynamic programming. Some others use hooking to extract APIs during the execution of malware, which could evade from the mentioned obfuscation techniques, but correct hooking is a serious challenge itself.

[4]Field-Programmable Gate Array

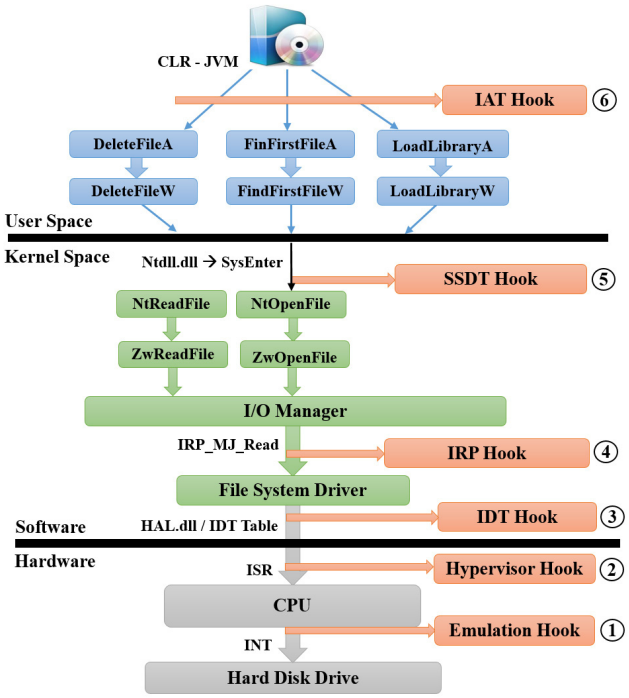

FIGURE 2. The process of interaction between the HW, OS and applications and proper location for hooking.

E. HOOKING

Hooking is a technique to capture application interactions with the OS. This process initiates with the formation of agents between the OS and an application. The purpose of the application request seizure is mainly to obtain information, theft or reject the application request [19]. This technique is used by malware to destroy the OS facilities or information theft of another application. It is also used by security applications to make self-defense system, analyze malware by sniffing API calls and protection of the OS [28].

It is possible to capture a request based on six different kinds of hooking in the user and kernel space of the OS. Figure 2 shows the way of interaction between the OS, hardware equipment and an application at a user/kernel-level and indicates all kinds of hooking [28].

The depth of hooking is very important because the analyzer always needs to control the analysis process, at least at a layer lower than the spyware. There is a direct relation between the depth of hooking and the accuracy of the analysis process. This issue is indicated in Equation 1 [28]. For instance, if an analyzer installs its hooks at the user-level, it will lose the possibility of the analysis of active spyware in the kernel space. Or if the analyzer installs its hooks by changing the SSDT table addresses, it never observes spyware which executes its malicious behavior by direct sending IRPs to the device manager.

$$Depth\ Level_{(Analyzer)} \leq Depth\ Level_{(Spyware)} \quad (1)$$

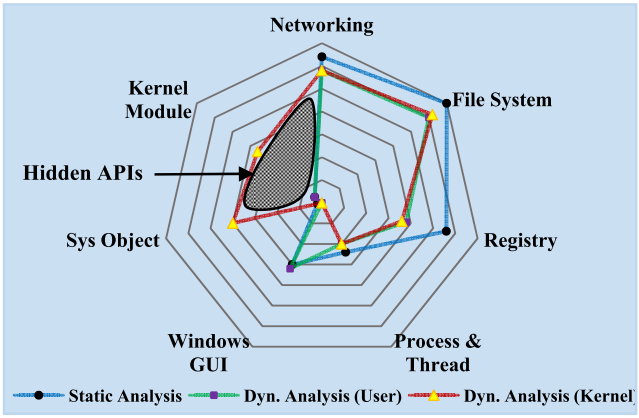

FIGURE 3. A comparison of the number of extracted APIs in different kinds of analysis.

In a deep analysis of some intelligent samples of spyware, we could understand that a static analysis is not enough to extract APIs because we found very critical APIs by a dynamic analysis when malware was under the supervision of a sandbox never seen in a static analysis. As mentioned before, the supervision depth in a dynamic analysis is a vital factor. Some of the sandboxes using user-level hooking techniques to extract APIs were not able to see the interactions of spyware in the kernel space, because spyware uses kernel-level APIs, in this case, the sandbox needs to use kernel-level hooks or hardware based emulation. For example, a processor emulation is able to see the interactions of active spyware in the kernel and can log the APIs. Figure 3 shows a comparison between the number of APIs extracted in a static analysis, user-level dynamic analysis and kernel-level dynamic analysis, which is categorized by seven sources of behavior in the OS. This result is very important because it indicates that the spyware tries to hide its critical APIs that it uses to call the kernel modules and to modify the system objects in them. By this, the spyware intends to mislead the analyzers.

The method used in the implementation of monitoring facilities for correct extraction of APIs is the most decisive factor in the detection of stealth and obfuscated spyware.

III. PROPOSED METHOD FOR DETECTION

The proposed method of this study is presented in two sections. The first section describes the way of spyware detections and the second section explains how to eliminate the detected pre-activated spyware. This class of malware has the maximum ability of stealth. Thus, it is more difficult to detect them rather than other classes of malware. In the proposed method of this study, the dynamic behavior of the spyware and blockers forms a basis for detection. Dynamic behavioral analysis methods are used because the new malware is equipped with various kinds of obfuscation techniques and metamorphic engines to deceive static behavioral analyses. Obfuscation techniques such as Random Hash Table, Control Flow Graph (CFG) smashing, IAT evacuation, and string encryption cause the signature-based and static behavioral analysis methods to have less potential in the analysis of obfuscated malware. Furthermore, malicious programmers use anti-reverse engineering techniques to prevent disassembling of their executable files like [29], which make serious problems for a static analysis. Hence, recently, more attention has been paid to analyze the dynamic behavior of malware. And most of the novel methods for malware detection, such as [21], [22], [25], and [27] are based on dynamic behavioral analyses. Although, some claim that it is possible to deceive a dynamic behavioral analysis, for example by shadow attacks in [30], dynamic behavioral analyses are still the most effective method.

Our proposed method considers the interaction of malware with the OS to model the behavior. It can recognize the nature of spyware based on such interactions. This is possible by using kernel drivers in the implementation of the proposed method. The reason for using the kernel drivers is confronting with active spyware, which needs to avoid limitations of the User Access Control (UAC) in a user space and nullify defensive equipment of the spyware.

In the proposed method, first a monitoring driver loads the SSDT and SSDT Shadow table directly from the owner modules (ntoskrnl.exe and win32k.exe). It is important to load the mentioned table directly rather than using preloaded tables in the main memory to detect the spyware. SSDT and SSDT shadow tables maintain the address of kernel-level win32 APIs. The OS uses the mentioned tables to map user-level APIs onto the equivalent APIs in the kernel [18], [19]. The original address for each row of the table, which indicates the location of the relevant API in main memory, is calculated. It is necessary to mention that both SSDT and SSDT Shadow tables are a subset of the Service Description Table (SDT). Data structures of the mentioned tables were figured out from [31] and presented below.

```
typedef struct
_KSYSTEM_SERVICE_
TABLE
{
    PULONG
ServiceTableBase;
    PULONG
ServiceCounterTableBase
;
    ULONG
NumberOfService;
    ULONG
ParamTableBase;
}
```

```
typedefstruct
_KSERVICE_TABLE_
DESCRIPTOR
{
KSYSTEM_SERVICE_TABLE
ntoskrnl;
KSYSTEM_SERVICE_TABLE
win32k;
KSYSTEM_SERVICE_TABLE
notUsed1;
KSYSTEM_SERVICE_TABLE
notUsed2;
}
```

A. DETECTION OF KEYLOGGERs

To detect keyloggers, the loaded driver in the memory kernel space calculates the original address for each API basis on the

version of the OS by using Equation 2.

$$OriginalAdd. = ServiceTableBase + SysEnterAdd. \times 4 \quad (2)$$

In the above equation, the original address of each API (each row of the SSDT) is obtained by the adding the address of the related service (API) toward the table basis and four times of the *SysEnter* command address. *SysEnter* command maps the user-level APIs onto kernel-level APIs in the NT[5] family of Windows operating systems [19]. After calculation of the original addresses, we should compare them with available addresses in the current SSDT table (the table available in the memory of the system). The reason for this comparison is to find discrepancies among addresses in order to determine APIs hooked by spyware.

Any contradiction between the main address and the current address for *NTUserGetKeyState* and *NTUserGetAsyncKeyState* functions of the SSDT shadow table is enough to confirm the presence of a keylogger in the system. The algorithm for detection of keyloggers is a follows.

1. *Load SSDT and the SSDT Shadow tables from the ntoskrnl.dll and win32k.exe modules directly.*
2. *Define the structure of the SDT table (parent of SSDT/ SSDT Shadow).*
3. *Define the structure of SSDT and SSDT Shadow table.*
4. *Calculate the original address for each API (rows of SSDT/ SSDT Shadow).*
5. *Read the current address for each API in SSDT and SSDT shadow previously loaded by the OS at startup time.*
6. *Compare the original and current address for NTUserGetKeyState and NTUserGetASyncKeyState APIs.*
7. *Any difference between addresses will indicate existence of a hook and thus infection by a keylogger.*

This malicious behavior was detected and classified by training a dynamic behavioral model based on the Linear Regression classification algorithm. The model can be formulated as Equation 3.

$$M = A \times X_A + B \times X_B + C \times X_C + \ldots \quad (3)$$

In the above formula, M is the index of keylogging malicious behavior. *A, B, C*are the indices of APIs that the spyware calls for hooking the mentioned rows of the SSDT shadow table sorted according to the sequence of call times and X_A, X_B, X_C are repetition frequencies of the APIs.

B. DETECTION OF SCREEN RECORDERS

In the proposed method to detect screen recorders, a driver in the kernel space hears all incoming requests by installing a hook on the *GetWindowsDC* and *BitBlt* system functions. Then, by considering five behavioral features, i.e. the frequency of repetition, uniqueness of applicant process, existence of sequence between the incoming requests, state of applicant process and value of parameters in the system calls, we could determine the behavior of screen recording and malicious spying intent. By performing data mining on more than 5000 real-world samples of spyware collected from [32], [33] during 2013-2018, we could derive several rules to detect and classify screen recording behaviors. These rules were produced in Weka 2.7.9 using the JRIP algorithm in a 7-Folds cross-validation training model and represented as a decision tree via the J48 algorithm. Figure 4 shows the proposed method for detection of screen recorders.

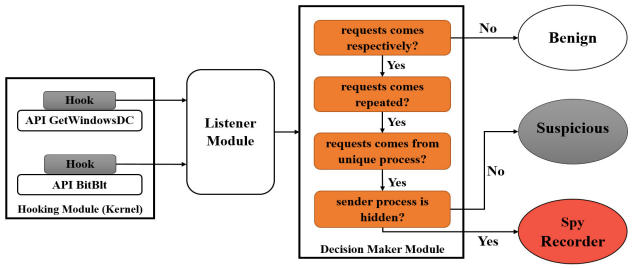

FIGURE 4. The proposed algorithm to detect screen recorders.

It is noteworthy that our proposed method faced a major challenge in hooking the SSDT shadow tables. The problem was to find the address of that table in the main memory. The SSDT shadow table loads in different addresses for each version of Windows OS and it is limited to read-only access [18]. We used the Windbg debugger to find the address of SSDT/SSDT Shadow table in each version of Windows OS.

C. DETECTION OF BLOCKERS

The proposed algorithm for detection of blockers is similar to the one proposed for keyloggers (Linear Regression was used). However, the difference is in the presence of a hook on the *NTUserSendInput* system function. If the mentioned function is hooked, the existence of a blocker in the system will be definite. Hence, the trained model classified the blockers by recognizing the malicious behavior of hooking-related row of the SSDT shadow table. The manufacturer has not documented the mentioned function because of its high sensitiveness.

IV. PROPOSED METHOD FOR DISINFECTION

The second section of the proposed method engages in tracking and confronting the detected spyware. Confronting process is performed in two stages. In the first step, tracker modules try to determine the footprints of the active spyware on the system. In the next step, the confronting modules try to stop the spyware activities and eliminate its executable. The architectural plan of the proposed method is composed of tracking and disinfection subsystems, which is illustrated in Figure 5.

It is the duty of the tracking subsystem to determine the process ID (PID) of the spyware and its executable paths on the hard disk as soon as the spyware is detected, and disinfection subsystem confronts the tracked spyware in order to

[5]New Technology

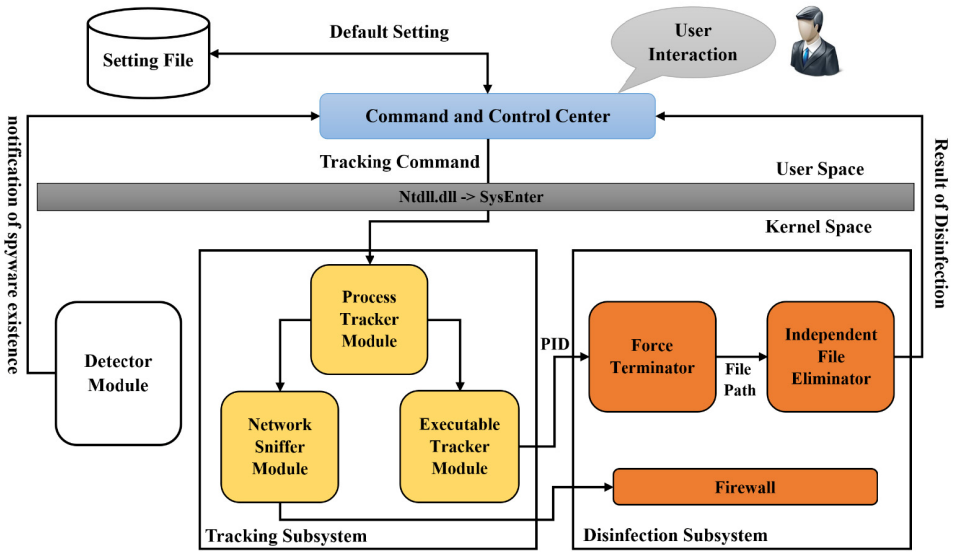

FIGURE 5. The architectural plan of the proposed anti-spyware.

terminate the process and eliminate the executable files. The tracking subsystem is composed of three modules: Process Tracker Module (PTM), Executable Tracker Module (ETM) and Network Sniffer Module (NSM). They track processes, executable paths and network interactions, respectively. The confronting subsystem uses the Force Terminator Module (FTM) to force terminate the process. An Independent File Eliminator (IFE) is used for the independent and decisive elimination of executable files. It is also equipped with an embedded firewall to block the network communications. All of the mentioned components are managed by the commander module, which is an application at the user-level and able to interact with the administrator of the system.

A. TRACKING SUBSYSTEM

After malware is detected, its existence is declared to the tracker modules of the tracking subsystem. In this subsystem, first a tracking command is delivered to the PTM. The task of this module is to find the spyware's PID by calling the *PSGetProcessID* function at the OS kernel-level. The obtained PID is delivered to ETM, which is the second tracker module in the tracking subsystem. This module is a filter driver in the OS kernel. Using the *PSGetProcessImageFileName* system function, the module gets access to enter the memory allocated to the spyware process and then it can derive the executable path among data structures in the main module of memory. At the same time that the second module of the tracking subsystem is tracking the malware executable file, the third module (NSM) receives the spyware's PID and begins to sniff the network interactions.

Most of the spyware need to define a socket by a combination of the IP/Port address and type of protocol to send stolen data to the related C&Cs. This is possible by using *WSASocket* and *Socket* API calls to create the socket. Then it sends and receives data using the *Send, SendTo, WSASend*APIs [20]. All of the mentioned APIs are placed in the WSock32.dll library. The structures of the mentioned APIs are given below.

```
SOCKET WSAAPI
socke(
_In_ int af,
_In_ int type,
_In_ int
protocol
);
```

```
int connec(
_In_ SOCKET s,
_In_ const struct
sockaddr * name,
_In_ int namelen
);
```

```
Int send(
_In_ SOCKET s,
_In_ const char
*buf,
_In_ int len,
_In_ int flags
);
```

B. DISTRACTING SPYWARE

In the proposed system, we can access the stolen data by capturing the second parameter of the *Send* system function (∗*buf*). Furthermore, it is possible to change the stored data in a buffer in order to deliver fake data to the spyware and change the IP or domain address of the first parameter (*SOCKET s*) in order to lure the spyware to an analyzer sinkhole or honeypot. Moreover, the tracker subsystem logs all network interactions and sorts them using their timestamps to complete network forensic processes and detection of an intruder.

C. DISINFECTION SUBSYSTEM

When the tracking process is completed, the second module of the tracking subsystem (ETM) delivers a distinctive executable file path with a PID to the confronting subsystem in order to perform the confront process. The proposed system can detect, track and confront with several pieces of malware at the same time.

1) FORCING TERMINATION

In the confronting subsystem, first the FTM receives the target's PID and terminates it by force. This module is a kernel driver, which uses no limitation advantages of the OS for the kernel space. This module calls the *ZwTerminatorProcess*

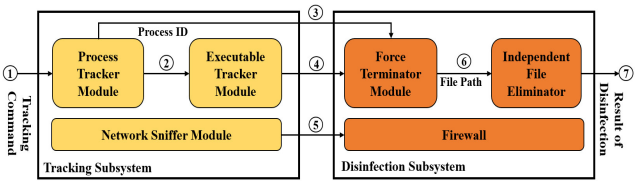

FIGURE 6. Collaboration between the tracking and disinfection subsystems.

kernel routine to terminate a malware process regardless of limitations of UAC and Kernel Path Protection (KPP) [19]. This routine is also able to pass the defensive facilities of malware. It is very important to mention that the termination process does not perform in real-time because the spyware process should be in running state in order to track and locate the executable path file. Hence, the termination process begins with a suitable delay after detection of spyware.

2) INDEPENDENT ELIMINATION

After the forced termination of the malware process, the result of the operation is passed to the second module in the disinfection subsystem (IFE). Then it eliminates target executable spyware file independent of system functions at the user-level. The independence of the user-level system functions gives rise to release from the system limitation in order to eliminate the protected files because of all malware, especially spyware, which needs high durability. It bans the elimination of their executable files. Figure 6 shows the architecture and data flow between the tracker and disinfector subsystems.

The only remained challenge is the existence of an active handle to the spyware files that was detected and terminated by the owner process of that handle in the previous step of the proposed method. IFE is a kernel filter driver, and if it fails to remove executable files from hard disk, it will try to send the IRP to hard disk directly in order to remove the malware files independent of the OS. Hence, the IFE module has the ability to tolerate one failure.

The noticeable issue is the altitude and Group Order (GO) of the filter driver. The method proposed in this paper loads a filter driver in at the lowest possible altitude higher than the original file system driver with the highest possible group order in the I/O stack space, which can give the best results in detecting and confronting the stealth spyware. Reference [28, Fig. 7 and eq. 4] shows best location for the ETM and IFE in the I/O stack.

$$\begin{aligned} FilterDriverAlt._{(ETM/IFE)} &< FilterDriverAlt._{(spyware)} \\ FilterDriverGO_{(ETM/IFE)} &\geq FilterDriverGO_{(spyware)} \end{aligned} \quad (4)$$

3) NETWORK LIMITATION

The network limiter module is a small firewall embedded in the disinfection subsystem. We defined necessary rules for configuring network interface card and possibility to block the suspicious IP, URL addresses and ports. NSM can send an order to active proper rules to block the spyware interactions by reading and interpreting the logged data of spyware activities. Correct, opportune and on-time configuration of the firewall is very important, because the user information is not stolen from the starting time of confronting the spyware process to its completion. Hence, if confronting process fails in the forced termination or in executable file elimination, the user information cannot be stolen and the occurred fault in the confronting subsystem will be tolerated. Another advantage is in confronting smart spyware with melt-ability, which can receive commands from the C&C center and commit suicide or change their behavior. Thus, the presence of the firewall is essential to prevent the spyware from suicide.

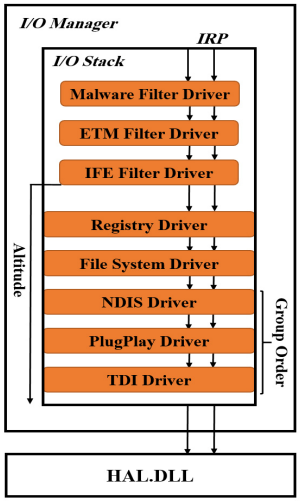

FIGURE 7. Proper altitude and group order for the ETM and IFE modules.

D. DRIVERS CONNECTION

One of the main challenges in the implementation of the proposed system was the connection among drivers in the kernel space and their security, especially confidentiality and integrity of the messages exchanged between kernel drivers and control center at the user-level because we loaded drivers with top-most privileges in the OS kernel.

The main methods of connection between a kernel driver and user application are shared memory, file mapping, pipelines and message queue. The pipelines are the most suitable method for the proposed system because of their security and speed of communication. Pipelines are divided into two main classes, i.e. named pipelines and unnamed pipelines. Unnamed pipelines are used to connect locally on a system and only at the user-level, but named pipelines are used to connect applications on several machines within a network or used to connect applications at the user-level and kernel-level [34], [35]. Therefore, we need to use a named pipeline. A connection via a pipeline is possible by defining symbolic links in source and destination drivers. These symbolic links are made by the system functions *IoCreateUnprotectedSymbolicLink* and *IoCreateSymbolicLink* at the

kernel-level. The difference between these two functions is that the symbolic links made by *IoCraeteSymbolicLink* are usable only at kernel-level drivers and they are hidden for applications at the user-level. Symbolic links are made by *IoCreateUnprotectedSymbolicLink* and used by applications at the user-level. Since the design of the proposed method for detecting, tracking and confronting specific malware requires drivers in the kernel space with the highest access level to OS, it should eliminate any misusing of these drivers. Two security layers are designed in the architectural plan of the proposed system. The sender driver uses the RSA[6] algorithm to make a digital signature and the other side receiver driver validates the originality of signature of coming packets. This originality means that the request belongs to the commander module at the architecture of the proposed system. Moreover, all packets are encrypted by the mentioned algorithm and key distribution is considered to avoid any Man-in-the-Middle (MITM) or Cross-Site Request Forgery (CSRF) attack.

Advantages and contributions of our proposed method are listed below:

- ✓ Using a specific and accurate classifier to recognize the classes of spyware and ransomware rather than a generic classifier.
- ✓ A deep and transparent hooking system able to detect novel obfuscated and stealth spyware.
- ✓ Disinfection of the malware by an effective method (not mere detection).
- ✓ Explaining the details and way of implementation for the proposed method and presenting the main architecture of a reliable anti-spyware to detect, disable and disinfect the malware.
- ✓ Competent evaluation using real-world samples of the spyware.

V. EVALUATION AND CONCLUSION

In this section, we evaluate the accuracy rate of the proposed method in detection and elimination of spyware based on confronting real-world samples. Then we will analyze and interpret the evaluation results. In order to do the tests, the proposed method was implemented in an antispyware application. C++ was employed to implement the kernel drivers and C#.net was used for the Graphical User Interface (GUI).

A. CLASSIFICATION OF SPYWARE

In order to evaluate the classification quality of the proposed method, a test dataset consisting of 4951 real-world samples of spyware collected from [32], [33] during the five recent years (between 2013-2018) and 3025 benign executable files was performed. The test dataset was completely separated from the training data and consisted of three classes of malware i.e. keyloggers, screen recorders, and blockers as well as one class of benign files. The benign class was collected from OS files, trusted application binaries, and some video game binaries. Figure 8 shows the percentage of the participants for each class of the dataset.

[6]Rivest-Shamir-Adleman public-key cryptosystems

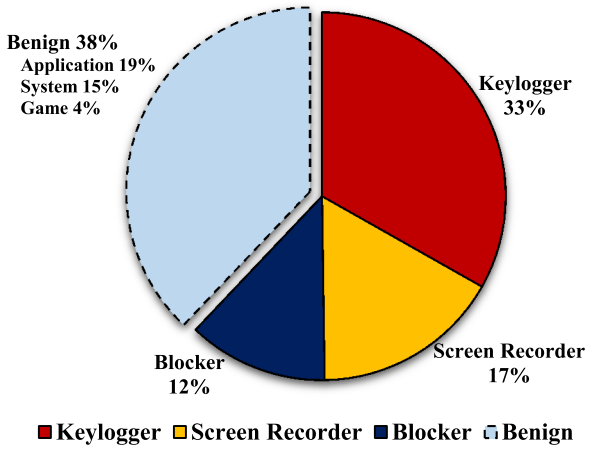

FIGURE 8. Dataset used in testing the model.

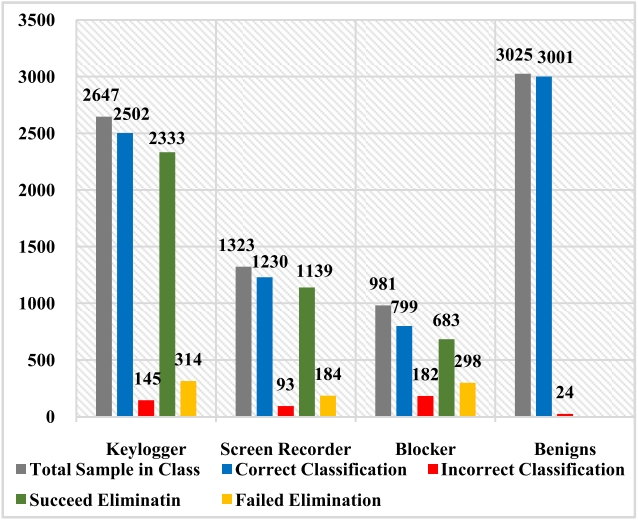

FIGURE 9. The results of the classification.

The samples in the test dataset were classified by the proposed method. The results for each class are presented in Figure 9.

It is necessary to mention that some malware can have several malicious behaviors at the same time, specifically in spyware. In the proposed method, the behavior detected earlier was a basis for classification. The cause of running malware was a detection free from any use of signatures and just based on a dynamic behavioral analysis and confronting it.

The figure indicates the number of samples classified correctly or incorrectly per class. Moreover, the figure shows the number of activated malware pieces successfully eliminated from the OS as well the number of samples that the proposed method was not able to detect.

B. ACCURACY OF DETECTION AND HIT RATE OF DISINFECTION

In order to calculate of detection rate (accuracy) and miss-classification (error) rate of the proposed method, Equation

5 was used.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (5)$$

In the above equation, TP (True Positive) is the number of spyware samples, which are correctly detected as spyware and TN (True Negative) is the number of benign files, which are correctly detected as benign. FP (False Positive) indicates the number of benign files mistakenly detected as malware while FN (False Negative) indicates the number of malware samples mistakenly detected as benign. Accuracy is defined as the ratio of sum of TP and TN to the sum of all samples while the error rate is the ratio of sum of FP and FN to all samples. These metrics were calculated for all four classes in the test data.

Hit/Miss rate of disinfection is defined as the number of successful/failed reactions of the proposed system in the forced termination of spyware process, elimination/quarantine of executable files and restriction of network connections ratio to all of the active spyware. The hit rate of disinfection was calculated through Equation 6.

$$Hit\ Rate = \frac{Terminated \cap Eliminated \cap Net.Restricted}{Total\ Detected\ Spyware} \quad (6)$$

In the above equation, the hit rate of the proposed method in malware disinfection is defined as the ratio of number of malware samples whose processes are terminated, their executable files are successfully deleted from the hard disk, and their network connections (if any) are disconnected to the total number of detected samples.

All the mentioned metrics including the accuracy and error rates of detection, and hit/miss rates of elimination for each class were calculated separately using the mentioned equations and the results are shown in Figure 10. It is noteworthy that the metrics related to malware elimination i.e. hit/miss rate were calculated only for three malware classes with the benign class excluded.

In Figure 10, the averages of the existing metrics calculated based on Equation 7 are shown on the Average column group.

$$Average(Accuracy) = \frac{\sum_1^N (Accuracy)}{\sum (N \times W)} \quad (7)$$

Where, N is the number of evaluated classes and W is the weight of each class, which is the ratio of the number of records in a class to the total number of records in the dataset. The above equation was used for calculation of the accuracy, error, hit and miss rates. As indicated in Figure 10, the proposed method was able to detect the spyware with an average accuracy of 92.32% and eliminate them with a hit rate of 81.28%.

Then, in order to compare the accuracy rate of the proposed method in detection of the classes in the dataset, a Receiver Operating Characteristic (ROC) curve was used. In this curve, the abscissa indicates the True Positive Rate (TPR) and the ordinate shows is the False Positive Rate (FPR). TPR is the ratio of TP to the sum of TP and FN and FPR is the ratio of FP to the sum of FP and TN. The ROC curve for each class is shown in Figure 11.

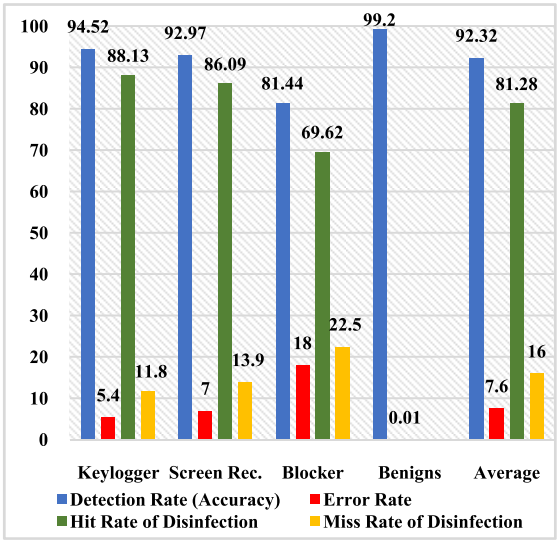

FIGURE 10. The results of the evaluation of the proposed system in detection and elimination of spyware.

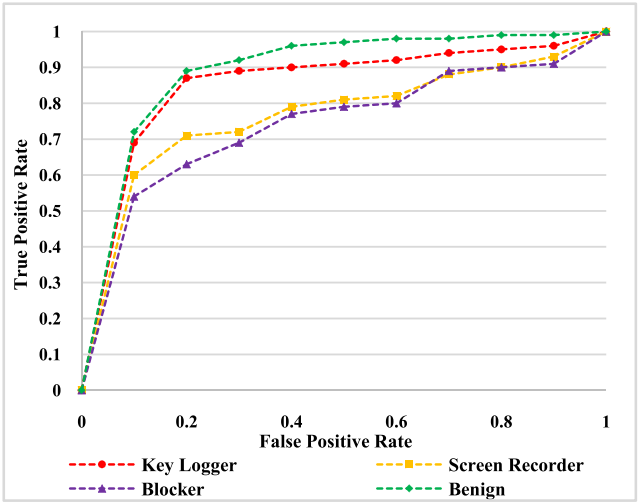

FIGURE 11. The ROC curve of the proposed method.

As Figure 11 shows, the ROC analysis demonstrates the accuracy of the proposed method in classifying keyloggers is better than other classes of spyware.

C. UTILIZATION OF RESOURCES

This evaluation was conducted to determine the utilization and optimization levels of the proposed anti-spyware application in the OS resources usage. A test was performed according to the following scenario.

All samples of spyware were run one by one on a Virtual Machine (VM) with two cores of CPU of a maximum speed of 2.4 GHz, 2 GB of memory and the x86 version of Windows 8 as the OS. The VM was equipped with the proposed anti-spyware application. CPU and memory usage of the anti-spyware during 60 minutes of the scan and confront process are shown in Figure 12.

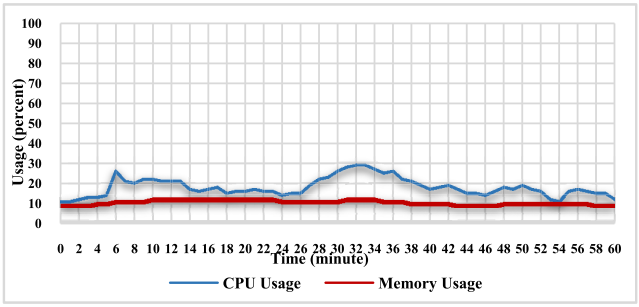

FIGURE 12. CPU and memory usage of the anti-spyware.

As it is evident from Figure 12, the CPU usage by the proposed anti-spyware application for detecting and confronting active malware was always less than 30% with an average percentage of 18.09. The average memory usage by the application was 10.62%.

It is necessary to mention that a dynamic behavioral analysis requires execution of every sample of malware separately. Hence, a secure environment like a VM or sandbox is essential for the risk-free running of the malware [19]. This use of virtualization tools could ensure the security and isolation of the target system while maintaining the spyware on the target system's visibility and transparency. We performed our experiments on an ideal scale so that generalization to greater scales was possible.

D. CONCLUSION

As mentioned in this paper, malware growth has been rapidly increasing in recent years, specifically for the Windows platforms. Spyware was one of the most dangerous, mysterious and shrewd class of malware designed for stealth and long-term missions in order to locate and steal vital information. Keyloggers, screen recorders, and blockers are great common subclasses of spyware and ransomware, which were investigated in this paper. In this paper, we proposed a novel and efficient method based on the dynamic behavior analysis to recognize the process and executable files of the spyware and perform a real-time action to confront them through deep and transparent hooking of kernel-level system calls. The proposed method was trained based on three machine learning algorithms. Furthermore, we presented details of the implementation of the proposed method including the main architecture of the anti-spyware system. Finally, real-world samples of the spyware collected from credible references were used to evaluate the accuracy of the proposed method in classifying spyware and its effectiveness in disinfection of the malware. The results indicated that the accuracy of the proposed method in detection and elimination of keyloggers was better than other classes of malware.

REFERENCES

[1] T. Huang and Y. Zhao, "Revolution of securities law in the Internet age: A review on equity crowd-funding," *Comput. Law Secur. Rev.*, vol. 33, no. 6, pp. 802–810, 2017.

[2] G. L. White, "Education and prevention relationships on security incidents for home computers," *J. Comput. Inf. Syst.*, vol. 55, no. 3, pp. 29–37, 2015.

[3] L. Boero, M. Cello, M. Marchese, E. Mariconti, T. Naqash, and S. Zappatore, "Statistical fingerprint-based intrusion detection system (SF-IDS)," *Int. J. Commun. Syst.*, vol. 30, no. 10, p. e3225, 2017.

[4] J. Yan, Y. Qi, and Q. Rao, "Detecting malware with an ensemble method based on deep neural network," *Secur. Commun. Netw.*, vol. 2018, Mar. 2018, Art. no. 7247095, doi: 10.1155/2018/7247095.

[5] *Kaspersky Lab Report*. Accessed: 2017. [Online]. Available: https://usa.kaspersky.com/about/press-releases/2016_kaspersky-lab-number-of-the-year-2016-323000-pieces-of-malware-detected-daily

[6] N. Khan, J. Abdullah, and A. S. Khan, "Defending malicious script attacks using machine learning classifiers," *Wireless Commun. Mobile Comput.*, vol. 2017, Feb. 2017, Art. no. 5360472, doi: 10.1155/2017/5360472.

[7] S. Arshad, M. A. Shah, A. Wahid, A. Mehmood, H. Song, and H. Yu, "SAMADroid: A novel 3-level hybrid malware detection model for Android operating system," *IEEE Access*, vol. 6, pp. 4321–4339, 2018.

[8] Y. Du, J. Wang, and Q. Li, "An Android malware detection approach using community structures of weighted function call graphs," *IEEE Access*, vol. 5, pp. 17478–17486, 2017.

[9] E. M. Rudd, A. Rozsa, M. Günther, and T. E. Boult, "A survey of stealth malware attacks, mitigation measures, and steps toward autonomous open world solutions," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 2, pp. 1145–1172, 2nd Quart., 2017.

[10] *AV-Test IT Security Insstitute*. Accessed: 2018. [Online]. Available: https://www.av-test.org/en/statistics/malware/

[11] G. Zhao, K. Xu, L. Xu, and B. Wu, "Detecting APT malware infections based on malicious DNS and traffic analysis," *IEEE Access*, vol. 3, pp. 1132–1142, 2015.

[12] S. Sun, X. Fu, H. Ruan, X. Du, B. Luo, and M. Guizani, "Real-time behavior analysis and identification for Android application," *IEEE Access*, vol. 6, pp. 38041–38051, 2018.

[13] I. Ismail, M. N. Marsono, B. M. Khammas, and S. M. Nor, "Incorporating known malware signatures to classify new malware variants in network traffic," *Int. J. Netw. Manage.*, vol. 25, no. 6, pp. 471–489, 2016.

[14] S. S. M. Chow, L. C. K. Hui, S. M. Yiu, K. P. Chow, and R. W. C. Lui, "A generic anti-spyware solution by access control list at kernel level," *J. Syst. Softw.*, vol. 75, nos. 1–2, pp. 227–234, 2005.

[15] D. Carlin, A. Cowan, P. O'Kane, and S. Sezer, "The effects of traditional anti-virus labels on malware detection using dynamic runtime opcodes," *IEEE Access*, vol. 5, pp. 17742–17752, 2017.

[16] S. Y. Yerima, S. Sezer, and I. Muttik, "High accuracy Android malware detection using ensemble learning," *IET Inf. Secur.*, vol. 9, no. 6, pp. 313–320, 2015.

[17] Symantec. *Internet Security Threat Report 2016*. Accessed: 2016. [Online]. Available: https://www.symantec.com/security-center/threat-report

[18] A. G. Ourimi, "Propose an optimal and transparent framework for automatic malware analysis," M.S. thesis, Dept. Comput. Eng., Iran Univ. Sci. Technol., Tehran, Iran, 2014.

[19] A. M. Lajevardi, "Design and implementation of a behavior-based method for malware detection," M.S. thesis, Dept. Comput. Eng., Iran Univ. Sci. Technol., Tehran, Iran, 2013.

[20] C. Petzold, *Programming Windows*, 6th ed. New York, NY, USA: Microsoft Press, 2013, pp. 489–493 and 1083–1093.

[21] R. Islam, R. Tian, L. M. Batten, and S. Versteeg, "Classification of malware based on integrated static and dynamic features," *J. Netw. Comput. Appl.*, vol. 36, no. 2, pp. 646–656, 2013.

[22] S. Gupta and P. Kumar, "An immediate system call sequence based approach for detecting malicious program executions in cloud environment," *Wireless Pers. Commun.*, vol. 81, no. 1, pp. 405–425, 2014.

[23] J.-W. Jang, J. Woo, A. Mohaisen, J. Yun, and H. K. Kim, "Malnetminer: Malware classification approach based on social network analysis of system call graph," *Math. Problems Eng.*, vol. 2015, Aug. 2015, Art. no. 769624, doi: 10.1155/2015/769624.

[24] P. Wang and Y.-S. Wang, "Malware behavioural detection and vaccine development by using a support vector model classifier," *J. Comput. Syst. Sci.*, vol. 81, no. 6, pp. 1012–1026, 2015.

[25] S. Das, Y. Liu, W. Zhang, and M. Chandramohan, "Semantics-based online malware detection: Towards efficient real-time protection against malware," *IEEE Trans. Inf. Forensics Secur.*, vol. 11, no. 2, pp. 289–302, Feb. 2016.

[26] L. Liu, B.-S. Wang, B. Yu, and Q.-X. Zhong, "Automatic malware classification and new malware detection using machine learning," *Frontiers Inf. Technol. Electron. Eng.*, vol. 18, no. 9, pp. 1336–1347, 2017.
[27] C.-H. Lin, H.-K. Pao, and J.-W. Liao, "Efficient dynamic malware analysis using virtual time control mechanics," *Comput. Secur.*, vol. 73, pp. 359–373, Mar. 2017, doi: 10.1016/j.cose.2017.11.010.
[28] D. Javaheri and M. Hosseinzadeh, "A framework for recognition and confronting of obfuscated malwares based on memory dumping and filter drivers," *Wireless Pers. Commun.*, vol. 98, no. 1, pp. 119–137, 2018.
[29] M. Madou, B. Anckaert, P. Moseley, S. Debray, B. De Sutter, and K. De Bosschere, "Software protection through dynamic code mutation," in *Proc. 6th Int. Workshop Inf. Secur. Appl.*, Jeju Island, South Korea, 2005, pp. 194–206.
[30] W. Ma, P. Duan, S. Liu, G. Gu, and J.-C. Liu, "Shadow attacks: Automatically evading system-call-behavior based malware detection," *J. Comput. Virology*, vol. 8, pp. 1–13, May 2012.
[31] B. Schreiber, *Undocumented Windows 2000 Secrets: A Programmer's Cookbook*. Boston, MA, USA: Addison-Wesley, 2001, pp. 99–100.
[32] *Virus Sign Malware Data Base*. Accessed: 2016. [Online]. Available: http://www.virussign.com
[33] *Adminus Malware Database*. Accessed 2018. [Online]. Available: http://www.adminus.net
[34] P. Silberschatz, B. Galvin, and G. Gagne, *Operating System Concepts*, 9th ed. Hoboken, NJ, USA: Wiley, 2013, pp. 587–588 and 613–614.
[35] A. S. Tanenbaum and H. Bos, *Modern Operating Systems*, 4th ed. Upper Saddle River, NJ, USA: Pearson Hall, 2015, pp. 119–120.

DANIAL JAVAHERI received the B.Sc. and M.Sc. (Hons.) degrees in computer engineering from Islamic Azad University (IAU), Borujerd, in 2012 and 2014, respectively, and the Ph.D. degree with a total average of 19.88 in computer engineering-software systems from the Science and Research Branch, Islamic Azad University, Tehran, Iran. For several years, he has researched software security, network forensic, and malware analysis. He teaches computer engineering courses at IAU as a lecturer. In 2015, he joined the Cyber Security Research Center, IAU, as a Software Security Specialist, and he succeeded to release several publications including journal and conference papers and a book in the field of network security and malware analysis. He has received two prizes from the Iran's National Elites Foundation 2018.

MEHDI HOSSEINZADEH received the B.Sc. degree in computer hardware engineering from the Islamic Azad University, Dezful, Iran, in 2003, the M.Sc. and Ph.D. degrees in computer system architecture from the Science and Research Branch, Islamic Azad University, Tehran, Iran, in 2005 and 2008, respectively. He is currently an Associate Professor with the Iran University of Medical Sciences, Tehran, Iran. His research interests are information technology, data mining, big data analytics, e-commerce, e-marketing, and social networks.

AMIR MASOUD RAHMANI received the B.Sc. degree in computer engineering from Amir Kabir University, Tehran, in 1996, the M.Sc. degree in computer engineering from the Sharif University of Technology, Tehran, in 1998, and the Ph.D. degree in computer engineering from Islamic Azad University (IAU), Tehran, in 2005. He is currently a Professor with the Department of Computer Engineering, IAU. He is the author/co-author of more than 190 publications in technical journals and conferences. His research interests are in the areas of distributed systems, ad hoc, wireless networks, and evolutionary computing.