EAI Endorsed Transactions
on Security and Safety

Research Article EAI.EU

Mouse Underlaying: Global Key and Mouse Listener Based on an Almost Invisible Window with Local Listeners and Sophisticated Focus

Tim Niklas Witte

Institute of Computer Science, Osnabrück University, Germany

Abstract

Keyloggers are serious threats for computer users both private and commercial. If an attacker is capable of installing this malware on the victim's machine then he or she is able to monitor keystrokes of a user. This keylog contains login information. As a consequence, protection and detection techniques against keyloggers become increasingly better. This article presents the method of Mouse Underlaying for creating a new kind of software based keyloggers. This method is implemented in Java for testing countermeasures concerning keylogger protection, virtual keyboard, signatures and behavior detection by anti-virus programs. Products of various manufacturers are used for demonstration purposes. All of them failed without an exception. In addition, the reasons why these products failed are analyzed, and moreover, measures against Mouse Underlaying are developed based on the demonstration results.

Received on 02 July 2018; accepted on 09 October 2018; published on 15 October 2018

Keywords: Computer security, Information security, Keylogger, Malware, Security, Spyware

doi:10.4108/eai.15-10-2018.155740

*Corresponding author. Email: wittet@uni-osnabrueck.de

1. Introduction

Keyloggers are software and hardware components (devices between keyboard and I/O port mostly plugged into the end of the keyboard cable) for monitoring keystrokes [16]. They are powerful spying tools because an attacker is able to reconstruct user activities on the compromised system. For this reason, special police forces, intelligence services and similar organizations use these tools frequently [8]. In many cases it is easier to reconstruct an encrypted email via keylogging than decrypt this email. In addition, keyloggers are simple to install. The installation process is comparable to a virus installation: the target needs to open an infected file. In most cases, an attacker applies social engineering such as opening an image [4]. However, criminals (black hat hackers) often employ keyloggers for obtaining login information about bank accounts of users. Protection techniques exist for protecting users against keyloggers. However, Mouse Underlaying is capable of bypassing any of them easily.

This article is structured as follows: Both known methods for keylogging and protection techniques against them are introduced. Hereafter, the total implementation of Mouse Underlaying is explained in detail. In the following, countermeasures are tested with the keylogger applying Mouse Underlaying. Subsequently, the result of this evaluation based on keylogger protection techniques and differences between Mouse Underlaying and other keylogging methods is analyzed. In the end, the results are summarized to develop effective countermeasures against Mouse Underlaying.

2. Software Based Keyloggers

Software keyloggers are applications for monitoring user keystrokes, without the awareness of the user, in order to retrieve information such as login details [24]. Frequently, keyloggers have capabilities that extend beyond this keylogging function: screen scrapers take periodic screenshots, advanced keyloggers are similar to backdoor viruses because they allow remote control. For this reason, the attacker tracks anything on the victim's computer, for instance, file operations (executing, printouts, copy and paste operations etc.) [26]. In most cases, this information is stored in a log file hidden in the file system. Hence, this log file

is difficult to distinguish from regular operating system files [5]. Typically, these files are encrypted [14]. For this reason, it is impossible for the user to notice this attack. A keylogger application sends the collected information back to the spying person using an email or uploads it directly to a server (e.g. FTP) [23]. Keyloggers are different from other types of malware because they do not damage the system or propagate to other systems. Keylogger applications are designed for running in stealth mode. Therefore, they have both a low CPU and memory usage. In addition, they do not show in the process list (e.g. Task Manager) [26]. As a consequence, it is a challenge to distinguish them from legitimate programs.

2.1. General processing of keystrokes

Methods for software keyloggers must be specified for each operating system: this article is focused on Windows™. As depicted in Figure 1 Windows employs an event mechanism: if a key is pressed, then the keyboard driver translates this keystroke into a `WM_KEYDOWN` Windows Message. Subsequently, this message is forwarded to the Windows System Message Queue (WSMQ). Ensuing, Windows conveys this message to the Thread Message Queue (TMQ) of the focused window. In the end, a thread of this window polls this queue (Thread Message Loop) and processes this Windows Message [21, 26].

There is a minimum time between two keystrokes. If the time span between these two keystrokes is shorter than this minimum time, then the second keystroke will be ignored. This should prevent the WSMQ from overloading [4].

2.2. Kernel-based Keyboard Filter Driver Method

Figure 1 depicts how the the keylogger is installing a Keyboard Filter Driver before the system's keyboard device driver takes effect [12]. In order to install this filter driver administrator privileges are required [17]. In the following, this filter driver captures keystrokes of the user and relays them by cloning. However, keystrokes are monitored at the kernel level. In other words, keystrokes will be monitored even before the operating system takes effect [8]. As a consequence, this keylogger is invisible for detection techniques.

2.3. Windows Keyboard Hook Method

A hook is an interface for allowing the program to execute extension code. Often these functions are provided by the operating system. Normally, keyboard hooks are applied to recognize shortcuts and react to them [13]. However, a hook-based keylogger exploits this functionality to monitor keystrokes: The

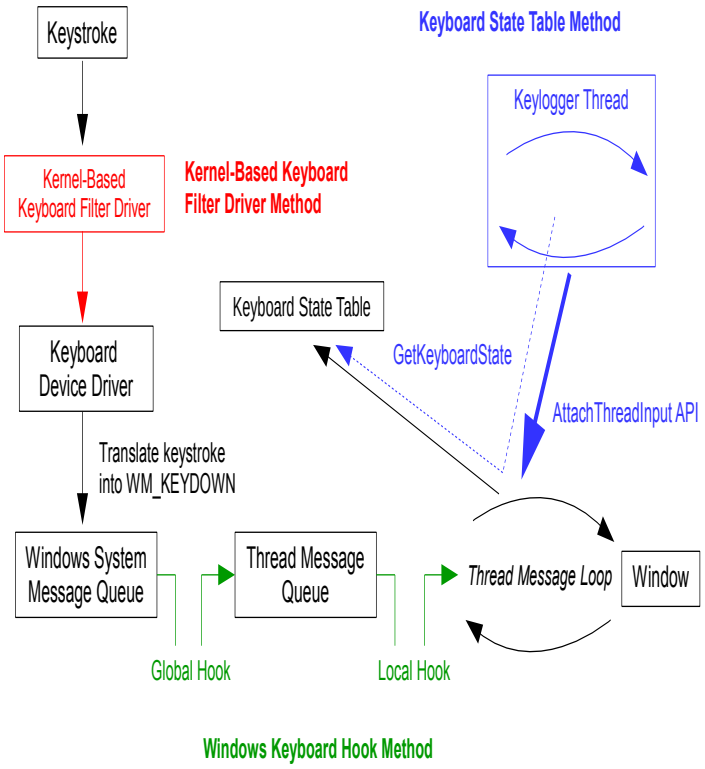

Figure 1. Processing keystrokes in Windows with keylogging methods.

keylogging process registers itself into this keyboard hook. For this reason, every `WM_KEYDOWN` Windows Message reaches the keylogger process before entering the window that receives this message [20, 21].

Keyboard hooks can be classified into two distinct types:

1. Global (system-wide) keyboard hook: This hook is linked between the WSMQ and TMQ. Therefore, this hook is able to capture global key events of a system [26].

2. Local (thread-specific) keyboard hook: This kind of hook is linked between the TMQ and the Thread Message Loop of a specific window. For this reason, a local hook is only capable of monitoring keystrokes for this specific application [26].

2.4. Keyboard State Table Method

Every application using a window refers to a Keyboard State Table. This table contains the status of 256 virtual keys. Typically, this table is used by programs for detecting more then one key state at the same time [18]. This is in most cases implementing shortcuts in a program (e.g. `[CTRL] + [C]` for copying text).

As presented in Figure 1, the keylogger adds its own thread into the Thread Message Loop by applying the

AttachThreadInput API. Then the keylogger employs the GetKeyboardState API function to determine information about the Keyboard State Table [4]. Generally this thread contains an infinite loop with another loop inside, which requests every key state by applying this function: if a key is pressed, then it is logged [32].

3. Countermeasures

3.1. Key Event Encryption

An encryption unit is linked between the Keyboard Device Driver and the WSMQ. All incoming WM_KEYDOWN Windows Messages by the Keyboard Device Driver will be encrypted prior to entering the WSMQ. The encryption of a WM_KEYDOWN Windows Message does not change the type of this message. This message is still a WM_KEYDOWN Windows Message but the information which key is pressed is changed. However, a decryption unit is linked between TMQ and the window. For this reason, all incoming encrypted WM_KEYDOWN Windows Messages will be decrypted before entering the window for processing [2, 6].

As a consequence, keyloggers which apply the Windows Keyboard Hook Method or Keyboard State Table Method monitor encrypted keystrokes. In other words, these keyloggers record wrong keystrokes. The Key Event Encryption can not prevent keylogging by the Kernel-based Filter Driver Method because the keystrokes are monitored even before these reach the encryption unit.

3.2. Anti-Hook Technique

This technique is based on the fact that some keyloggers apply API functions for hooking. The detection mechanism works as follows: firstly, the total system will be scanned for enumerating each process. In the following, every process will be enumerated by all DLLs (Dynamic Link Libraries). For installing a hook the command SetWindowHookEx is necessary [20]. This command resides under the banner of USER32.LIB. Consequently, if a process is using this API, then this process will be marked. At the end, all marked processes will be listed, and the user has to decide whether this process has rights for hooking keystrokes [7, 28].

As a result, keyloggers which apply the Windows Keyboard Hook Method will be detected. Although the Anti-Hook Technique can not prevent keylogging by the Keyboard State Table Method and Kernel-based Keyboard Filter Driver Method, these methods do not apply an API for hooking keystrokes.

3.3. Fool a keylogger: conceal sensitive data

This simple scheme is based on the fact that a keylogger is capable of monitoring every keystroke but it is not able to understand them. In other words, a keylogger fails to recognize which application is employing keystrokes and which are used [26].

Typically, an operating element (e.g. the password field in the window) reacts to keystrokes if this operating element has focus. If no operating element is in focus, then only the layout is able to react to keystrokes by a local key listener [10].

The user types a part of his or her password then he or she clicks elsewhere, e.g. in a other text field. Now the password field is out of focus. Yet, the user types random characters. In the next step, the user clicks again on the password field, and so on [28].

As a consequence, the attacker is unable to reconstruct the login information from the keylog (recorded keystrokes). The attacker does not know which characters of login information belong together in a row. Between the characters of the real login information are the random characters typed by the user. Regardless, the attacker could use brute-force to determine the real login information by trying each possible combination of monitored keystrokes in a row. However, this process takes a long time [10].

Generally, this scheme is capable of confusing all keylogging methods previously mentioned. This scheme fails if the keylogger monitors the mouse's actions too because the attacker recognizes in the keylog the changed focus.

3.4. HoneyID

Based on generating keystrokes this technique baits keyloggers. This mechanism consists of three modules: trap manager, bogus event generator and spyware detector. The trap manager gathers the CPU usage of each process. While the bogus event generator is imitating keystrokes, the spyware detector compares the CPU usage of each process before and while bogus event generator evokes the keystroke imitation. If the CPU usage of a process is increasing, then this process reacts upon the imitated keystrokes. In other words, this process monitors keystrokes [28, 33].

For this reason, the keyloggers which apply the Windows Keyboard Hook Method or the Keyboard State Table Method will be detected. However, a detection of keyloggers based on Kernel-based Keyboard Filter Driver Method fails, because the applied Kernel-Based Keyboard Filter Driver is not a process. Therefore, its CPU usage will not be listed.

3.5. Random Multiple Layouts

Every key has a unique value, called a scan code. If a key is pressed then the keyboard controller generates a corresponding scan code for this keystroke. In the following, the keyboard driver translates this scan code based on the current keyboard layout into a virtual-key code. Afterwards, the keyboard driver generates a `WM_KEYDOWN` Windows Message which contains this virtual-key code information. This Windows Message is pushed into the WSMQ [22].

This keylogger protection technique applies multiple keyboard layouts to mislead keyloggers: For each keystroke the current keyboard layout is randomly replaced with another layout. Therefore, the generated `WM_KEYDOWN` Windows Message contains translated virtual-key code information based on this randomly replaced layout keyboard. In other words, this Windows Message contains an incorrect virtual-key code. This virtual-key code information of the Windows Message will be corrected by converting back into the original keyboard layout before entering the window [6].

Keyloggers applying the Windows Keyboard Hook Method or Keyboard State Table Method record keystrokes based on the Windows Message, which contains incorrect virtual-key code information. In other words, these keylogging methods monitor incorrect keystrokes. However, Random Multiple Layouts can not prevent keylogging by the Kernel-based Keyboard Filter Driver Method because the keystroke (scan code) is recorded even before it is translated based on the keyboard layout.

3.6. Detection by Support Vector Machine

A Support Vector Machine (SVM) is a machine learning algorithm. A SVM determines the best separating line into a dataset by maximizing the distance between each data point and this line. In other words, a SVM classifies the given datasets into two classes. Every data point before the line belongs to the same class. As a consequence, every data point behind the line belongs to the other class [30].

The applied data points of the dataset for keylogger detection have the following form:

$$\begin{pmatrix} DD \\ DU \\ UD \\ UU \\ RT \end{pmatrix}$$

- Down-Down (DD): time difference between when a key is pressed and when the next key is pressed [22].
- Down-Up (DU): time difference between when the same key is pressed and when it is released [22].
- Up-Down (UD): time difference between when a key is released and when the next key is pressed [22].
- Up-Up (UU): time difference between when a key is released and when the next key is released [22].
- Reaction Time (RT): time difference between when a key is pressed and when it is received by the process [22].

This training dataset is generated by typing keystrokes while:

1. a keylogger is active.
2. no keylogger is active.

Meanwhile, a global keyboard hook is applied to measure the time of a key press and its release to calculate DD, DU, UD, and UU values. To calculate the corresponding RT value, the time when the target window receives this keystroke is logged. This target window displays the reception of this keystroke in a text box. Afterwards, the SVM is applied to determine the best separating line into this training dataset. Now, the data points can be classified into *keylogger active* or *no keylogger active* based on this separating line as presented in Figure 2. This figure is a simplified data representation, because only two dimensions are represented instead of five. To check the system for the existence of a keylogger, a keystroke is typed and the corresponding DD, DU, UD, UU, and RT values will be determined. If this data point lies in the *keylogger active* section, then a keylogger is monitoring keystrokes [22].

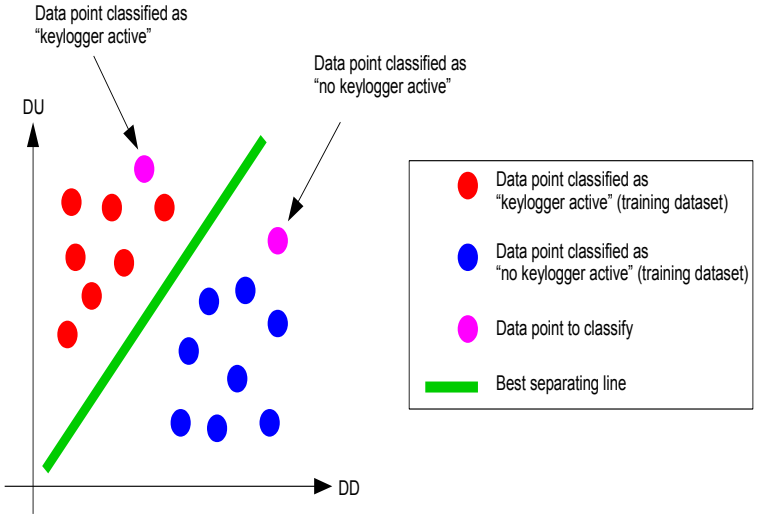

Figure 2. Keylogger detection by Support Vector Machine.

Generally, this technique is based on the fact that a keylogger delays the arrival of the Windows Message to the target window by monitoring keystrokes. The

keystroke information of this message will be copied and stored such as in a log file. This technique is able to detect keyloggers which apply the Windows Keyboard Hook Method. Recorded time differences (DD, DU, UD, UU, and RT value) are longer than normal because the Windows Message must enter this hook (additional element) before entering the window (target). Detection of keyloggers based on the Keyboard State Table Method is difficult, because there is no entering of the Windows Message into its additional thread. This thread is only monitoring the Keyboard State Table concurrent (multithreading) with the keystroke processing. For this reason, monitored time differences are minimally higher than normal. Besides, this technique is unable to detect keyloggers applying the Kernel-based Keyboard Filter Driver Method due to keystrokes that are logged even before the time differences are determined.

4. Mouse Underlaying

4.1. General Approach

The keylogger generates a small transparent window which is always under the mouse pointer requesting focus. Hence, local listeners are capable of capturing the entire user input (keystrokes and mouse clicks). In the following, this keylogger window closes while the captured user input is imitated. The intra-system focus handing for windows secures that the actual window receives the imitated user input.

The overall implementation of Mouse Underlaying is presented in Figure 3. The keylogger window has the following properties: undecorated, visible, same position as mouse pointer, size of 1x1 pixel, opacity of 1 %, focusable and always on top (1). If the opacity value is lower then 1 %, then there is an incompatibility with the used key and mouse listener.

The moveWindow_thread contains an infinite loop setting the keylogger window position equal to the mouse pointer position (2). In addition, the requestFocus_thread is always checking which window is in focus: if the keylogger window is not in focus, then this window receives focus (3). Overall, it is impossible for the keylogger window to lose focus, e.g. if another window is opening. Moreover, the closedWindow_thread is constant checking if the keylogger window is closed then there is a shortcut for closing a window imitated after the keylogger window opens again (4). This mechanism closes another weakness of this technique (see #2 in the listing below).

If a key is pressed (5), then all applied threads will be paused (6). Afterwards, the keystroke will be recorded (7). Subsequently, the keylogger window is closing (8). After a brief waiting time, this keystroke will be imitated: there is a key pressure imitation after a waiting time, followed by a key release

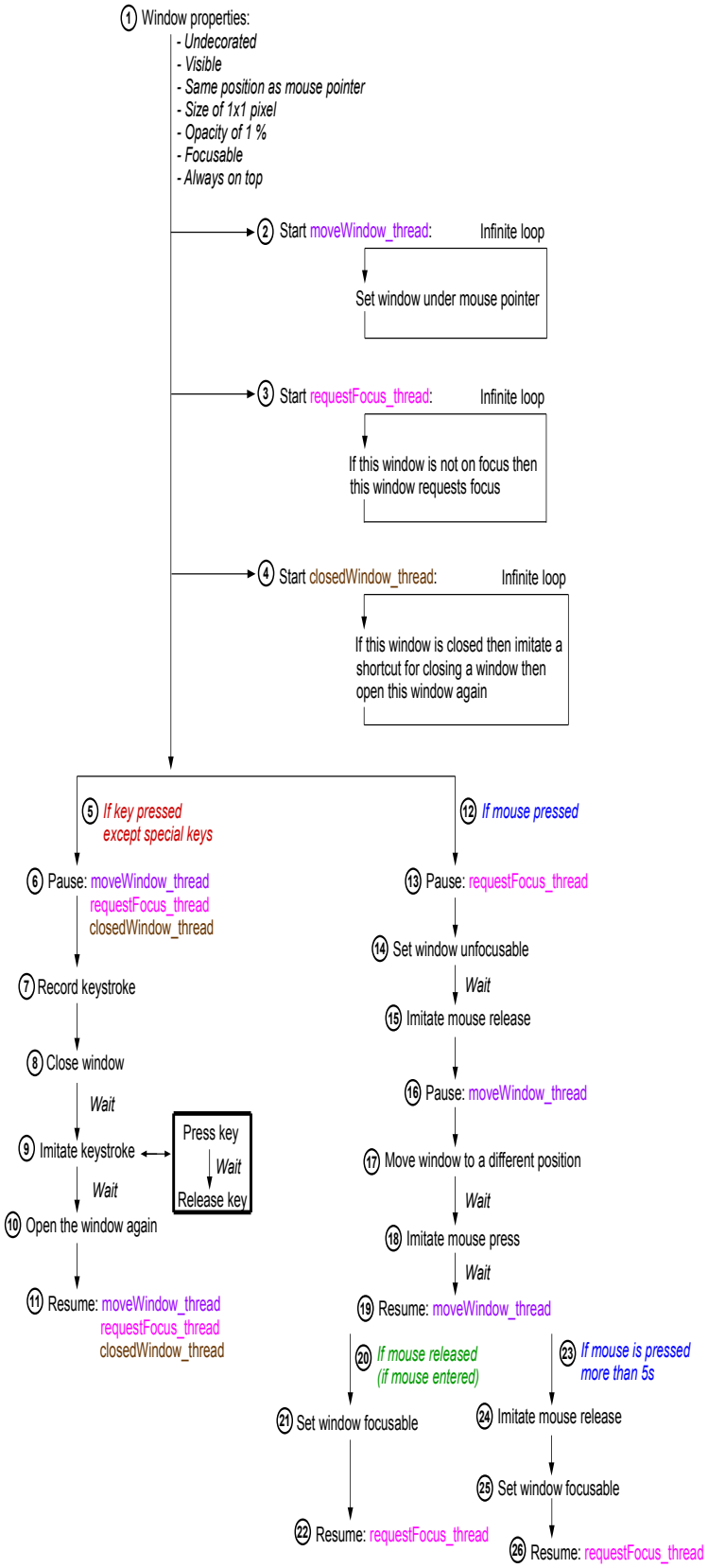

Figure 3. Implementation of Mouse Underlaying.

imitation (9). However, an exception exists for the following keys: [CTRL], [SHIFT], [CAPS LOCK], [ALT] and [ALT GRAPH]. This is a requirement for imitating capitals and special characters, moreover applying key combinations. Without this exception there is an imitation of key release directly after the imitation of key pressure. Hence, the operating system does not longer recognize this key pressed, although the user still holds the key down. After a brief waiting time, the keylogger window opens again (10). Finally, all applied threads will be resumed (11).

If a mouse button is pressed (12), then the requestFocus_thread will be paused (13). Subsequently, the keylogger window changes to an unfocusable state (14). After a short waiting time, the released mouse button will be imitated (15). Afterwards, the moveWindow_thread will be paused (16). Now the keylogger window moves to a different position in the vicinity of the mouse pointer (17). After a short waiting time, the pressed mouse button state will be imitated (18). According to an again waiting time, the moveWindow_thread will be resumed (19). Due to this resumed thread, the keylogger window is always set directly under the mouse pointer. If the mouse is released (perceived with a MouseEntered-Event) (20), then the keylogger window changes to focusable (21). Finally, the requestFocus_thread will be resumed (22).

This mechanism enables the user to move windows, tabs, etc. with the mouse while recording the mouse events. The mouse press by the user is intercepted by the focusable keylogger window. The imitated mouse release while the keylogger window is under the mouse pointer is only for resetting the mouse state. Though, this MouseReleased-Event will be relayed by the operating system to the actual window because the keylogger window is unfocusable. The move of the keylogger window to a different position followed by an imitated mouse press leads to a MouseEntered-Event for the actual window and leads to a MouseExit-Event for the keylogger window. In addition, there is a MousePressed-Event for the actual window. The moveWindow_thread sets always this keylogger window under the mouse pointer. However, this movements do not lead to a MouseEntered-Event for keylogger window and do not lead to a MouseExit-Event for the actual window due to the unfocusable state of the keylogger window. Overall, the keylogger window is under the mouse pointer while the mouse is pressed in the actual window. However, if the mouse is released by the user then a MouseEntered-Event for keylogger window and MouseExit-Event for the actual window will be created. Moreover, this MouseReleased-Event will be also relayed to the actual window due to the unfocusable keylogger window. If there is only a mouse click (mouse button pressed after fast releasing) then (21) will be immediately triggered. Hence, mouse clicks will be relayed to the actual window. In total, a MousePressed-Event and a MouseReleased-Event reach the actual window.

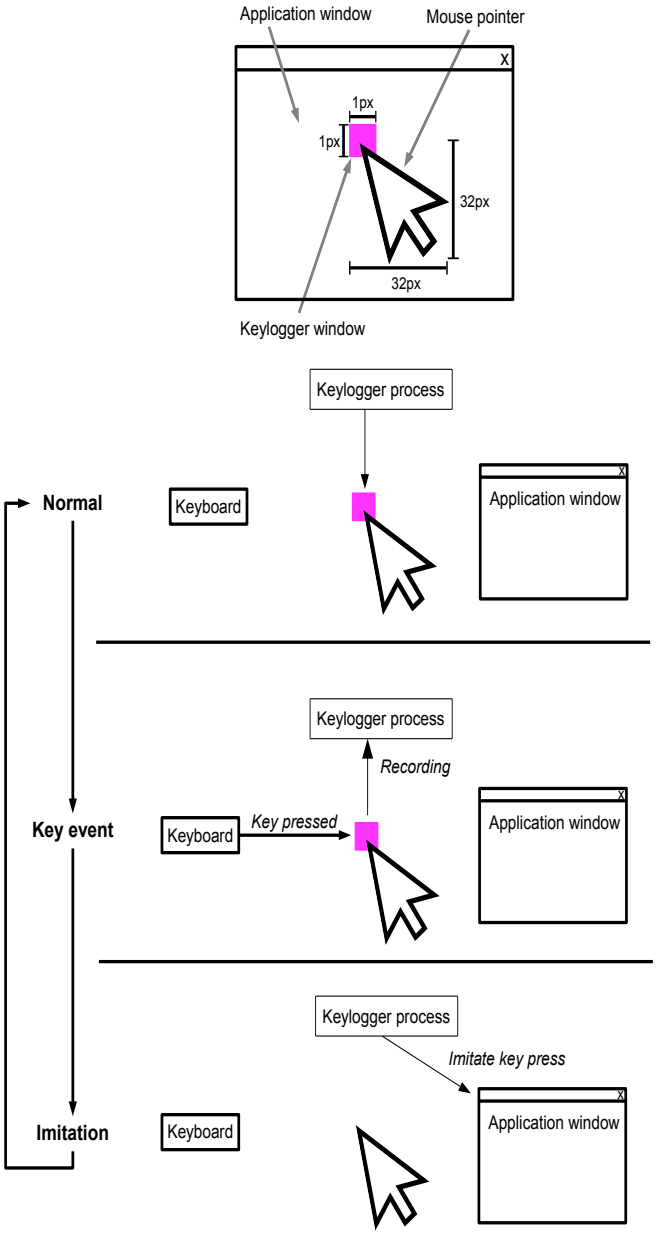

Figure 4. Overview of Mouse Underlaying.

As presented in Figure 4, by closing the keylogger window, the actual window in the background regains focus. Hence, the imitation of key and mouse events can be transmitted to the actual window. The imitation is required because the focusable keylogger window intercepts all key and mouse events by the user. Without this imitation, all of these events would only be recorded, but there would be no relaying of them to the actual window. As a consequence, the user would not be able to interact with the actual window.

However, Mouse Underlaying has two vulnerabilities:

1. While the user is pressing his or her mouse button, the keylogger window is unfocusable. Therefore the local key listener is disabled. Hence, keystrokes are not recorded. However, this is untypical user behavior. Although, there is a mechanism: if the mouse is pressed more than five seconds (23), then the released mouse state will be imitated for resetting the mouse state (24). Subsequently, the keylogger window changes to focusable (25). Finally, the requestFocus_thread will be resumed (26).

2. Various key combinations are able to close the focused window, such as [ALT] + [F4] in Windows. If the user applies this shortcut, then he or she will close the keylogger window and no keylogging will be possible. Therefore, the closedWindow_thread employs an infinite loop checking whether the keylogger window is closed (4). In this case, the shortcut for closing a window will be imitated. Afterwards, the closed keylogger window opens again. An exception exists for this mechanism while the keylogger process is relaying the keystroke to the actual window. This exception is realized by pausing (6) and resuming (11) the closedWindow_thread. With this mechanism the Mouse Underlaying is able to relay the effect of this shortcut to the actual window.

Windows pushes in regular intervals (system update time) the Windows Message of the WSMQ to the TMQ of a current focused window [26]. As a consequence, the waiting time must be higher than the duration of these regular intervals for securing a registration of:

- A changed window focus.
- Imitated key and mouse events.
- A switch between of a focusable and unfocusable state of a window.
- An opening and closing of a window.

4.2. Protocol And Relay Keystrokes

Figure 5 shows the manipulation of keystroke processing in Windows by Mouse Underlaying: Ultimately, Windows employs a WSMQ for managing which TMQ of a window receives information about keystrokes [4]. The keylogger window is always on focus. For this reason, the TMQ of this window obtains WSMQ information about keystrokes. If a key is pressed, then the keylogger window is closing and subsequently removed from the WSMQ. Simultaneously, the actual window is on focus, and its TMQ receives WSMQ information about keystrokes. While imitating the keystroke a WM_KEYDOWN Windows Message is created by the keylogger process and automatically pushed to the WSMQ. In the following, the TMQ of the actual window is receiving information about this keystroke. In other words, with this function the keylogger window is capable of relaying logged keystrokes to the actual window. There is no difference in generating keystrokes between a keyboard device driver and a process. Hence, the actual window is able to react upon any kind of generated keystroke. The same applies to mouse events: instead of a WM_KEYDOWN there is a WM_MOUSEDOWN, WM_MOUSEUP etc. Windows Message.

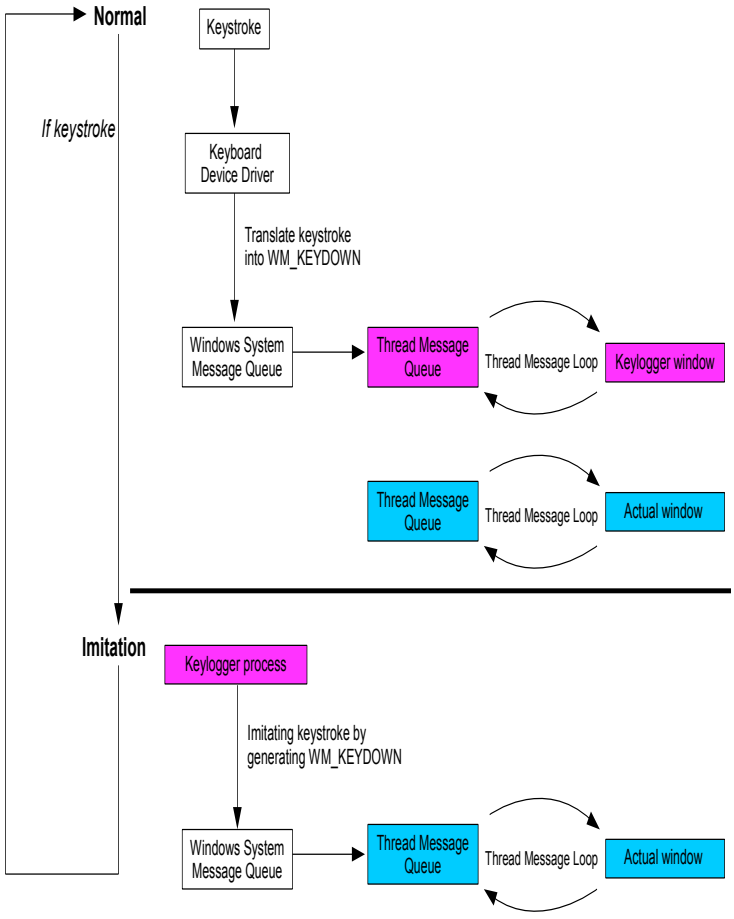

Figure 5. Manipulation of keystroke processing in Windows by Mouse Underlaying.

4.3. Change Window Focus

Figure 6 presents the changing of window focus while Mouse Underlaying is active: The keylogger window is in focus, and its TMQ comes first in the WSMQ. If there is a mouse click, then the keylogger window intercepts it. In addition, the keylogger window is closing, and its

TMQ is removed from the WSMQ. The actual window (window no.1 in Figure 6) has focus. Therefore its TMQ is now first listed in the WSMQ. The imitated mouse click (same position as the intercepted mouse click) in a window not focused (window no.3 in Figure 6) sets this window in focus and puts its TMQ on the first place in the WSMQ. If this mouse click resides on an operating element, then this element is focus in the window. However, if this window is in focus again, then this element is automatically in focus too. The reopening of the keylogger window sets this window in focus and assigns its TMQ to first listed in the WSMQ.

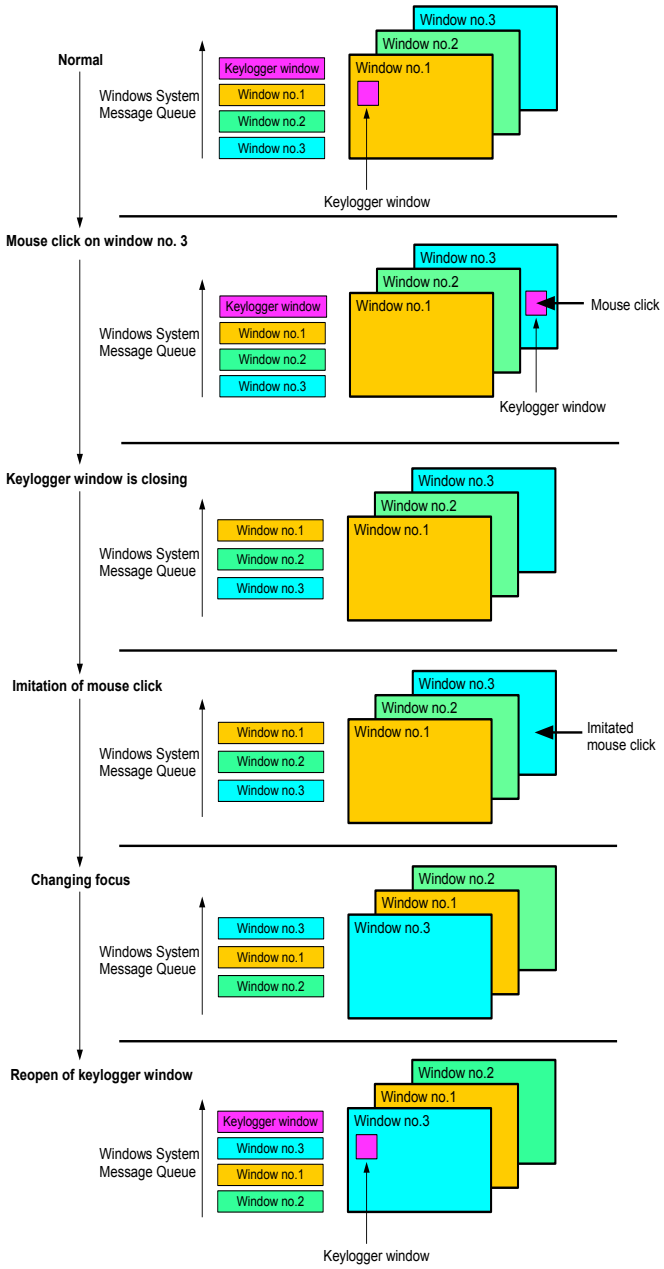

Figure 6. Changing window focus while Mouse Underlaying is active.

5. Evaluation

A *sample.jar* file (programming language: Java) is generated applying Mouse Underlaying. This window is a JFrame. A key and mouse listener is employed for monitoring key and mouse events of the user. Mouse and key events are imitated by a robot. The program contains a kill switch: if [ESC] is pressed, then the program will stop. However, the program *sample.jar* displays captured keystrokes in a console. Furthermore, no admin rights, only standard user privileges are applied to the following operating systems: Windows (Version: XP - 10), OS X (Version: 10.8 - 10.13), Linux (Ubuntu and Debian), Solaris, Android and Qubes OS.

5.1. Operating System Compatibility

Mouse Underlaying works perfectly under Windows, Linux and Solaris. However, there is a minor issue with OS X as described in the following paragraph.

Table 1. Compatibility of Mouse Underlaying with operating systems.

Name	Version	Vulnerable
Windows	XP	✓
	Vista	✓
	7	✓
	8	✓
	8.1	✓
	10	✓
OS X	10.8 (Mountain Lion)	✓
	10.9 (Mavericks)	✓
	10.10 (Yosemite)	✓
	10.11 (El Capitan)	✓
	10.12 (Sierra)	✓
	10.13 (High Sierra)	✓
Linux	Ubuntu 16.04.3	✓
	Debian 9.3	✓
Solaris	11.3	✓
Android	8.1.0	✗
Qubes OS	3.2	✗

Before *sample.jar* is being launched on OS X, a window appears indicating that this program is using accessibility features (imitation of keystrokes and mouse clicks) as shown in Figure 7.

As usual in OS X, the keylogger process generates Window Messages (in Windows: Windows Messages) for imitating key and mouse events and tries to push these Messages into the Event Queue (in Windows: WSMQ). However, OS X prevents pushing Window Messages into the Event Queue by processes. Normally, only the corresponding device driver to this event has privileges to push Window Messages into this Event Queue. However, the user is able to define

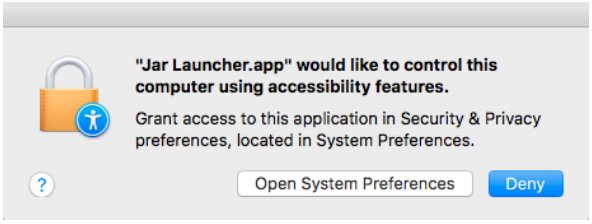

Figure 7. Message of OS X before launching sample.jar.

exceptions [15]. After the user confirms this message by defining an exception for this keylogger process, Mouse Underlaying will work perfectly. Social Engineering is an effective method to convince the user to confirm this message.

Android is not vulnerable for Mouse Underlaying because it does not provide software-induced focus request. Moreover, the graphical user interface of Android is not a typical desktop. Only one window can be opened exclusively. The other windows are stored in a queue [29]. The following sequence of activities can be observed:

1. if the current window closes;
2. then the next window in this queue will open immediately;
3. if the keylogger window is open;
4. then the user can see the background image, which is suspisious;
5. (again) mouse or keystroke by the user;
6. keylogger window closes;
7. evoking next window in the queue;
8. keylogger imitates mouse or keystroke;
9. keylogger window opens again;
10. the user can see the background image again which is suspisious, and so on.

However, Mouse Underlaying does not work on Qubes OS. For testing purposes, the keylogger was started in a VM while trying to monitor keystrokes in another VM. Qubes OS employs a compartmentalization. For this reason, there is an instantiation of each VM [25]. On this basis, the keylogger window receives only focus in one VM but not in other VMs. Hence, the keylogger window is not able to get the overall focus on the desktop. Therefore, local key and mouse listener are not effective. Basically, Mouse Underlaying works on Qubes OS only if one VM is used. In order to achieve that the the attacker has two effective methods to apply the keylogger:

1. sending an HTTP link file to the user opening a website with a bound keylogger. From this moment on the attacker is capable of monitoring all keystrokes in the browser because this browser and the keylogger run in the same VM.
2. a modification of Mouse Underlaying can partially bypass the compartmentalization of Qubes OS: the keylogger window is open, then it closes briefly in order to open again. As a result, the keylogger window has obtained the total focus on the desktop including all VMs. Local key and mouse listener are now able to monitor user activities in one VM but no imitation of user activities in other VMs is possible because of the compartmentalization.

Some operating systems such as Linux and Solairs employ multiple desktops. For testing purposes the keylogger was started in the first desktop instance while trying to log keystrokes in the second desktop instance. Mouse Underlaying is not able to record keystrokes in the second desktop instance because the keylogger window exists only in first desktop instance. However, the modification of Mouse Underlaying applied in Qubes OS transfers the keylogger window to the second desktop instance (currently used). In addition, this window is on focus. As a consequence, keystrokes will be recorded.

For ensuring a platform independence of Mouse Underlaying there are different waiting times for each operating system based on the system update time for window focus. For example, in Windows the update time is determined the WSMQ. However, this does not apply to operating systems based on compartmentalization. The system update time is influenced by the hardware performance and current CPU usage. Moreover, Mouse Underlaying requires a desktop allowing more than one window opened concurrently.

5.2. Minor Limitations

Normally, every window in Windows 8 (or higher) which is not on focus has a gray layout. The focused window has its normal layout as presented in Figure 8 (a).

While Mouse Underlaying is running on a Windows-type operating system, every window appears as not focused. In other words, each window has a gray layout as depicted in Figure 8 (b). The focused window is the keylogger window (size of 1x1 pixel therefore almost invisible for the user).

In addition, during the relaying process of user activities the focus changes twice. This relaying process is extremely fast. For this reason, the window layout changes similarely fast: the active layout of the actual

(a) inactive (b) active

Figure 8. Changed window focus behaviour due to Mouse Underlaying.

window displays only a few milliseconds before it turns into gray again. For example in the case of word processing, the user does not notice this change because he or she focuses on the text and not on the window frame while writing. However, there is no text cursor in text fields because this operating element is not on focus. While relaying user activities this text cursor is appearing for milliseconds. Furthermore, if the user is typing faster than the waiting time (the time elapsed between to keystrokes is shorter than the waiting time), then this keystroke will not be recorded and imitated because the user is writing while the keylogger window is closed. This usually happens when the user constantly presses a key. The same applies to mouse events but this is an unusual user behavior.

Besides, there is a minimal delay (barely noticeable) between typing keystroke and see result on the screen. Moreover, while Mouse Underlaying is active, automatic completion such as in browsers is disabled due to the browser not being in focus.

5.3. Countermeasures

Antivirus software. This *sample.jar* file was uploaded to the website `www.virustotal.com` which uses signatures of a lot of anti-virus software for detecting malware [3, 31]. The results of this analysis is demonstrated in Figure 9: none of 60 used anti-virus programs were able to recognize *sample.jar* as malware (keylogger).

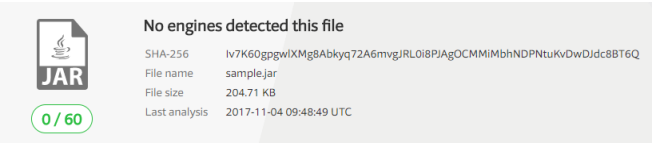

Figure 9. Analysis by virustotal.com.

In addition, for testing purposes the following behavior controls of anti-virus software are applied: G DATA, Symantac, Kaspersky, Avira and Trend Micro. None of them was able to recognize *sample.jar* as malware (keylogger).

Keylogger protection software. A whole variety of keylogger protection software by different manufacturers has been tested. None of them was able to prevent monitoring keystrokes by *sample.jar* as illustrated in Table 2.

Table 2. Keylogger protection software attempting to prevent keylogging.

Name	Attack Prevented
Elite Anti Keylogger	✗
G DATA	✗
Ghostpress	✗
GuardedID	✗
Kaspersky	✗
KeyScrambler	✗
SpyShelter	✗
Zemana	✗

Virtual keyboards. Virtual keyboards are applied for testing their capability to prevent capturing keystrokes by *sample.jar*. As displayed in Table 3 none of them was able to prevent keylogging by Mouse Underlaying.

Table 3. Virtual keyboards attempting to prevent keylogging by Mouse Underlaying.

Name	Attack Prevented
Free Virtual Keyboard	✗
Kaspersky Virtual Keyboard	✗
Windows On-Screen Keyboard	✗

6. Analysis

6.1. Countermeasures

A detection of Mouse Underlaying by signatures and behavior controls of anti-virus programs failed. Mouse Underlaying is based on actions which are often applied by usual programs such as moving, opening and closing of its window, and requesting focus. These actions are benign because there is no system damage or serious system change caused by them. Moreover, Mouse Underlaying was virtually unknown at this time.

Keylogger protection techniques presented in Table II employ the Key Event Encryption [2]: `WM_KEYDOWN` Windows Messages will be encrypted by the encryption unit of the Key Event Encryption and pushed to the WSMQ. At the end of the keystroke handling process there is a decryption of this Windows Message by the decryption unit before entering the keylogger window. However, each keystroke by the user will be recorded in a decrypted state. As usual, there is an imitation of these keystrokes. The generated `WM_KEYDOWN` Windows Message of the keylogger process will be encrypted

by the encryption unit and pushed to the WSMQ. The same applies for this generated Windows Message by entering the actual window. In other words, this generated Windows Message will reach the actual window in a decrypted state. As a consequence, the Key Event Encryption does not prevent Mouse Underlaying: The keystroke (of the user) will be recorded in a decrypted state and the imitated keystroke reaches the actual window in a decrypted state.

Moreover, keylogger protection techniques and behavior controls of anti-virus software often additionally apply the Anti-Hook Technique [9]: Mouse Underlaying does not use an API for hooking keys because it applies a sophisticated window focus in combination with a local key listener. For this reason, Mouse Underlaying is undetectable for the Anti-Hook Technique.

Basically, Mouse Underlaying can be overreached such as other keyloggers with changing focus while typing. By modifying the local mouse listener into recording mouse clicks in the keylog, Mouse Underlaying is able to recognize a changing focus.

Generated keystrokes by HoneyID are kernel based keystrokes because these do not originate from a keyboard driver. By imitation of a keystroke a `WM_KEYDOWN` Windows Message is created and pushed into the WSMQ by a process instead of a keyboard driver. However, there is no difference between a `WM_KEYDOWN` Windows Message generated by a keyboard driver or a process [28]. For this reason, a local key listener reacts to generated keystrokes by HoneyID. As a consequence, the CPU usage of the keylogger increases while HoneyID is imitating keystrokes. However, this increasing is limited by a low CPU usage value based on a maximum count of imitation due to using waiting times. Therefore, it is a challenge to distinguish a keylogger applying Mouse Underlaying from other processes based on the fact that other processes do not have a constant CPU usage.

However, Random Multiple Layouts can not prevent keylogging by Mouse Underlaying because this incorrect virtual-key code information of the Windows Message will be converted back into the original keyboard layout before entering the keylogger window. For this reason, the keylogger window receives the Windows Message containing the correct virtual-key code information.

Moreover, the detection technique based on the SVM is able to detect Mouse Underlaying. While Mouse Underlaying is active, a Windows Message has double the normal distance to reach the target (actual window): Reaching the keylogger window, monitoring and imitating it, then reaching the target. Therefore, the measured time differences are approximately twice as high as normal.

The window of a virtual keyboard application is unfocusable: if this window is focusable, then it blocks the generated key events independently. As mentioned above, there is no difference in generating keystrokes between a keyboard device driver and a process. Hence, the local key listener is able to react to them. However, virtual keyboards only prevent keylogging by the Kernel-based Keyboard Filter Driver Method because the generated Windows Messages are pushed into the WSMQ by the process of a virtual keyboard. Hence, these messages do not enter the Kernel-Based Keyboard Filter Driver.

6.2. Differences between Mouse Underlaying and conventional keylogging methods

Mouse Underlaying does not install an additional element (thread, hook, or driver) to the keystroke processing of the operating system like conventionally applied keylogger methods [4]. These methods get the keystroke information illegally because there is no focused window. Mouse Underlaying records keystrokes in accordance with the operating system policy (legal) by using a sophisticated window focus. The operating system sets its window, residing at the end of the keystroke handling process due to its focused state. For each key press the keylogger window closes and relays this keystroke to the actual window by imitation.

For this reason, Mouse Underlaying substantially changes the flow of the Windows Messages: The keylogger window intercepts all incoming Windows Messages by logging. Afterwards, the keylogger window closes. Subsequently, a clone of these messages will be generated by imitating the same keystroke. Now this generated Windows Message will be pushed into the WSMQ. Afterwards, this message reaches the actual window (target). In other words, while Mouse Underlaying is active, a Windows Message has double the normal distance to reach the target: Reaching the keylogger window, monitoring and imitating it, then reaching the target. Unlike Kernel-based Keyboard Filter Driver Method and Windows Keyboard Hook Method, there is no interception of all incoming Windows Messages by logging them [26]. Hence, there is no recreation of this message by imitation required. As a consequence, the original Windows Message reaches the window (target) in one way. There is a further entering of the additional element. Unlike the Keyboard State Table Method, there is no entering of the Windows Message into its additional thread [4]. This thread is only monitoring the Keyboard State Table concurrent (multithreading) with the keystroke processing. The table indirectly represents a `WM_KEYDOWN` Windows Message: If this message is generated then its information about the keystroke exists in this table.

Mouse Underlaying applies a 1x1 pixel-sized and very transparent window. Hence, the user does not

notice this window as focused. The user is mistakenly convinced that the actual window (in the background) has focus. Conventional keylogging methods operate directly on the operating system or kernel level. Hence, these methods are not applied on the graphical user interface. As a consequence, the user is not visually fooled.

For installing an additional element applied by conventional keylogging methods, kernel modification (driver installation) or changes in the configuration file of the operation system (hook registration and thread attaching) are required [17]. As a consequence, anti-virus programs will report this action to the user. These actions also require administrator privileges.

Mouse Underlaying operates directly on the graphical user interface (desktop) by applying:

- Imitation of key and mouse events.
- Information about the mouse pointer position.
- Local key and mouse listener.
- Moving, opening and closing of its window.
- Switching between a focusable and unfocusable state.
- Requesting focus.

Thus, only standard user privileges are required. Moreover, these actions are often applied by usual programs. As mentioned above, these actions are also benign. As a result, Mouse Underlaying will not be identified as suspicious (malware) by anti-virus programs.

Moreover, if the attacker applies one of the conventional keylogging methods, he or she often tries to cover their tracks by uninstalling the additional element after the attack. However, operating systems and other tools log when an additional driver (Kernel-based Keyboard Filter Driver Method) is installed, hooks are applied (Windows Keyboard Hook Method) etc. [27]. Therefore, conventional keylogging methods are traceable after the attack. An attacker who applies Mouse Underlaying needs only to terminate this process. For this reason, it is a challenge to prove Mouse Underlaying after the attack.

7. Outlook and further research

7.1. Disabling keyboard and mouse

If there is no local key listener, then Mouse Underlaying is capable of disabling the keyboard instead of keylogging. The same applies to the mouse by removing the used mouse listener. However, other techniques require administrator privileges for disabling both keyboard and mouse due to kernel modifications [27]. For this reason, there is a detection by behavior controls of an anti-virus program. In addition, Mouse Underlaying with cumulative modifications (see above) can be applied in Qubes OS to disable the entirety of keyboard and mouse functions (also in other VMs).

7.2. Optimization

Appearance of focus layout. While Mouse Underlaying is active in Windows 8 (or higher) each window has a grey layout. Generally, regular users are not able to recognize this layout modification. Though there are computer-experts and highly attentive users noticing this modification enabling them to react appropriately. On that account, Mouse Underlaying should disable this layout modification. A registry key modification is a way to do that [11]. However, if the keylogger implements this directly via command, then the keylogger will be detected by behavior control of an anti-virus program.

Instead of manipulating a registry key there is another way to solve this layout problem by modifying Mouse Underlaying: the program takes a screenshot. The window size is identical with the screen size and requesting focus. This window contains the screenshot taken. If there is a key or mouse event (noticed by local listeners), then the window will close. After imitating this event there is another screenshot activity. At this point, the window opens with the updated screenshot. However, this modification causes a still image on the desktop. In general, the user does not notice this attack while, for instance, watching a video or playing a game. Moreover, a number of operating systems such as Linux noticed a white contour for a few seconds during the opening process.

Imitation of keystrokes. For imitating keystrokes Mouse Underlaying forges pressure and release of a particular key. Between pressure and release there is a waiting time based on the system update time. This waiting time is identical for each imitation. For this reason, it is possible to develop specific detection techniques. These techniques analyze the elapsed time of a pressed key. If this time is always the same or too short for human activites, then Mouse Underlaying is active. As a consequence, the length of the waiting time must be within particular interval boundaries. The length of the waiting time is determined by a random value within these boundaries. The interval range is determined by an analysis of human keystrokes.

8. Conclusions

This paper has described a new software-based keylogger method: Mouse Underlaying does not install an additional element (thread, hook, or driver) to the keystroke processing of the operating system like

conventional applied keylogger methods. For each key press the keylogger window closes and relays this keystroke to the actual window by imitation. This keylogger window is almost invisible. For this reason, the user is mistakenly convinced that the actual window is on focus and there exists no focused keylogger window. In other words, Mouse Underlaying visually spoofs the user.

Besides, Mouse Underlaying has the following advantages compared with conventional applied keylogging methods:

- No modification of kernel or configuration file for installing an additional element to the keystroke processing of the operating system.
- No administrator privileges are required, only standard user privileges.
- Obtains keystroke information legally by employing a focused window.
- Indirect manipulation of Windows Messages flow (keystroke processing): keylogger protection techniques based on encryption can not prevent this attack.
- No identification as suspicious by anti-virus programs based on benign operations such as moving, open and closing of its window (often applied by usual programs).
- Difficult to prove after the attack.

Although, Mouse Underlaying has the following disadvantages:

- Requires a desktop which allows several windows open at the same time.
- Operating systems chart the focus by layout of the window and the operating element. There is no focused window and no focused operating element represented. Besides, while typing keystrokes there are two focus changes (also represented in the layout). Overall, the user could become suspicious.
- The employed waiting time is different for each operating system.

Conventional keylogger protection techniques are designed to prevent keylogging by methods which install an additional element in the keystroke processing of the operating system. However, these protection techniques are not designed to prevent an almost invisible window to receive keystroke information and relay it to the actual window (manipulation of Windows Messages flow). Overall, Mouse Underlaying is a serious threat. However, there are various countermeasures and detection techniques for Mouse Underlaying:

- The behavior control of an anti-virus program must analyze the WSMQ on irregularities: if a window appears frequently on the first place (focused window) even directly after its intermediate disappearance, then this application should be identified as suspicious. The MessageQueue-Class in C# and C++ is capable of analysing the WSMQ [19]. This method is able to detect the main part of Mouse Underlaying, a window frequently opening and closing and always on focus.
- Besides, the behavior control of an anti-virus program shall identify an application as suspicious if its position changes often.
- Moreover, there should be a limited protected area within the mouse pointer: operating systems and anti-virus programs shall prevent the movement of applications in this area.
- However, a complete disabling or ignoring of imitating both key and mouse events is not useful because physically incapacitated people use tools such as voice control. In this way it may be seen as a discrimination. Over and above, remote maintenance tools use this functionality too.
- Furthermore, if a program is imitating mouse and key events, then the operating system and the related anti-virus program must report this action to the user in form of a message box. However, OS X is the only operating system capable of this type reporting.
- In addition, an 1x1 pixel-sized window is suspecious because this is untypical for a useful program. Hence, the window manager must recognize the suitable window. Therefore, it must provide a minimum window size. If a window is smaller than the minimum size, then the window manager is terminating the process of the undersized window.
- Moreover, based on the fact that the keylogger window is always under the mouse pointer, Mouse Underlaying can be detected with a modification of HoneyID: there is a determent of CPU usage of each process. Instead of imitating keystrokes there is an imitation of mouse movement. Meanwhile the CPU usage of each process is determined again and compared during this imitation. If there is an increase of CPU usage of a process, then this process will use Mouse Underlaying.
- Another method to detect Mouse Underlaying generates a window with both a local key and mouse listener. The process of this window

imitates both mouse and key events at a high frequency. If events are recorded with a delay (elapsed time between imitation and recording) by the local listeners, then Mouse Underlaying is active.

- In addition, an analysis of the keylog is able to detect Mouse Underlaying: if there are always the same two characters in a row such as *tteesstt* then Mouse Underlaying is active. Of the two same characters the first is user induced and the second is the result of a keylogger imitation.

- An effective measure against Mouse Underlaying is compartmentalization such as in Qubes OS. There is a instantiation of each process. In other words, every process is isolated from the others. Microsoft operating systems can be protected against Mouse Underlaying with Bromium Secure Platform. This is a security software which runs every application in a micro VM [1].

Annotation

This paper was accepted to *The 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications* (IEEE TrustCom-18) in New York. Unfortunately, there was not financial support to pay conference fee etc. therefore I must withdraw this paper.

Acknowledgements. I would like to thank the Distributed Systems Group from the Osnabrück University for valuable feedback, discussions, and work. I would also like to thank Jörg Krywkow who did an extended proofreading of my first paper. Moreover, I would like to thank the Cyber Analysis and Defense Group from the Fraunhofer FKIE for value feedback and good suggestions for the evaluation.

References

[1] (2016) *Bromium- Redefining Endpoint Security*. Tech. rep., Bromium.

[2] (2017) *GuardedID*. Tech. rep., StrikeForce Technologies.

[3] Abrams, R. (2017) Virustotal tips, tricks and myths. In *VB2017*.

[4] Abukar, Y., Maarof, M., Hassan, F. and Muse Abshir, M. (2014) Survey of keylogger technologies **5**: 25–31.

[5] Akhil S, Neeraja M Nair, A.P.A.R. (2014) Detection and prevention of keylogger spyware attacks. *International Journal of Computer Engineering & Technology (IJET)* .

[6] Ali, T.O.M., Awadelseed, O.S.A. and Eldewahi, A.E.W. (2016) Random multiple layouts: Keylogger prevention technique. In *2016 Conference of Basic Sciences and Engineering Studies (SGCAC)*: 1–5. doi:10.1109/SGCAC.2016.7457997.

[7] Aslam, M., Muzammil Baig, M. and Asif Arshad, M. (2004) Anti-hook shield against the software key loggers. *CiteSeerX* .

[8] Creutzburg, R. (2017) The strange world of keyloggers - an overview, part I **2017**: 139–148.

[9] Dadkhah, M., Davarpanah Jazi, M., Ciobotaru, A.M. and Barati, E. (2014) An introduction to undetectable keyloggers with experimental testing **4**: 1–5.

[10] Florencio, D. and Herley, C. (2006) How to login from an internet cafe without worrying about keyloggers .

[11] Honeycutt, J. (2005) *Microsoft Windows Registry Guide, Second Edition* (Redmond, WA, USA: Microsoft Press).

[12] Howard, A. and Hu, Y. (2012) An approach for detecting malicious keyloggers. In *Proceedings of the 2012 Information Security Curriculum Development Conference*, InfoSecCD '12 (New York, NY, USA: ACM): 53–56. doi:10.1145/2390317.2390326, URL http://doi.acm.org/10.1145/2390317.2390326.

[13] Kishore Subramanyam, Charles E. Frank, D.F.G. (2007) *Keyloggers: The Overlooked Threat to Computer Security*.

[14] Lee, R. (2004) Keystroke logging investigation. *SANS Security Essentials (GSEC)* .

[15] Miller, B.P., Cooksey, G. and Moore, F. (2006) An empirical study of the robustness of MacOS applications using random testing. In *Proceedings of the 1st International Workshop on Random Testing*, RT '06 (New York, NY, USA: ACM): 46–54. doi:10.1145/1145735.1145743, URL http://doi.acm.org/10.1145/1145735.1145743.

[16] NameHemita Pathak, Apurva Pawar, B.P. (2015) A survey on keylogger: A malicious attack. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)* **4**.

[17] Navarro, J., Naudon, E. and Oliveira, D. (2012) Bridging the semantic gap to mitigate kernel-level keyloggers. In *2012 IEEE Symposium on Security and Privacy Workshops*: 97–103. doi:10.1109/SPW.2012.22.

[18] Olzak, T. (2008) Keystroke logging (keylogging) .

[19] Oney, W. (2002) *Programming the Microsoft Windows Driver Model, Second Edition* (Redmond, WA, USA: Microsoft Press), 2nd ed.

[20] Ortolani, S., Giuffrida, C. and Crispo, B. (2013) Unprivileged black-box detection of user-space keyloggers. *IEEE Transactions on Dependable and Secure Computing* **10**(1): 40–52. doi:10.1109/TDSC.2012.76.

[21] Peter Schartner, M.F. (2011) System-manipulation using windows-messaging-hooks. *Semantic Scholar* .

[22] Pillai, D. and Siddavatam, I. (2018) A modified framework to detect keyloggers using machine learning algorithm : 1–6.

[23] Preeti Tuli, P.S. (2013) System monitoring and security using keylogger. *International Journal of Computer Science and Mobile Computing* **2**.

[24] Rahim, R., Nurdiyanto, H., Ahmar, A., Abdullah, D., Hartama, D. and Napitupulu, D. (2018) Keylogger application to monitoring users activity with exact string matching algorithm **954**: 012008.

[25] Rutkowska, J. and Wojtczuk, R. (2010) *Qubes OS Architecture*. Tech. rep., Invisible Things Lab.

[26] Sagiroglu, S. and Canbek, G. (2009) Keyloggers: Increasing threats to computer security and privacy. *IEEE Technology and Society Magazine* **28**(3): 10–17. doi:10.1109/MTS.2009.934159.

[27] Silberschatz, A., Galvin, P.B. and Gagne, G. (2008) *Operating System Concepts* (Wiley Publishing), 8th ed.

[28] Solairaj, A., Prabanand, S.C., Mathalairaj, J., Prathap, C. and Vignesh, L.S. (2016) Keyloggers software detection techniques. In *2016 10th International Conference on Intelligent Systems and Control (ISCO)*: 1–6. doi:10.1109/ISCO.2016.7726880.

[29] Song, M., Song, H. and Fu, X. (2011) Methodology of user interfaces design based on android. In *2011 International Conference on Multimedia Technology*: 408–411. doi:10.1109/ICMT.2011.6002076.

[30] Tharwat, A., Hassanien, A.E. and Elnaghi, B.E. (2017) A ba-based algorithm for parameter optimization of support vector machine. *Pattern Recognition Letters* **93**: 13–22. doi:10.1016/j.patrec.2016.10.007, URL https://doi.org/10.1016/j.patrec.2016.10.007.

[31] Verma, A., Rao, M., Gupta, A., Jeberson, W. and Singh, V. (2013) A literature review on malware and its analysis **5**.

[32] Wazid, M., Katal, A., Goudar, R.H., Singh, D.P., Tyagi, A., Sharma, R. and Bhakuni, P. (2013) A framework for detection and prevention of novel keylogger spyware attacks. In *2013 7th International Conference on Intelligent Systems and Control (ISCO)*: 433–438. doi:10.1109/ISCO.2013.6481194.

[33] Ysterud, S.A. (2014) *Keylogging of user interaction in physical and virtual environments and its implications for honeypot analysis*. Master's thesis, University Of Oslo.