RESEARCH PAPER | Engineering | Volume - 5 | Issue - 1 | Jan Special Issue - 2015 | ISSN - 2249-555X

Analysis and Implementation of Decipherments of KeyLogger

KEYWORDS Keylogger Detection, AntiKeylogger, Security, Detection of Keylogger

Parth Mananbhai Patel
ME (CSE), S.P.B.Patel Institute of Technology, Mehsana, Gujarat

Prof. Vivek K. Shah
S.P.B.Patel Institute of Technology, Mehsana Gujarat

ABSTRACT Software keyloggers are very famous tool which are often used to harvest confidential information. One of the main reasons for this rapid growth of keyloggers is the possibility for unprivileged programs running in user space to eavesdrop and monitor all the keystrokes typed by the users of a system. Implementation and Distribution of these type of keyloggers are very easy because of the ability to run in unprivileged mode. But, at the same time, allows one to understand and model their behavior in detail. Taking benefit of this characteristic, we propose a new detection technique that simulates crafted keystroke sequences in input and observes the behavior of the keylogger in output to unambiguously identify it among all the running processes. We have prototyped our technique as an unprivileged application, hence matching the same ease of deployment of a keylogger executing in unprivileged mode.

I. INTRODUCTION

Key loggers are implanted on a machine to intentionally monitor the user activity by logging keystrokes and eventually delivering them to a third party. While they are seldom used for legitimate purposes (e.g., surveillance/parental monitoring infrastructures), key loggers are often maliciously exploited by attackers to steal confidential information.

II.LITERATURE SURVEY

Different works deal with the detection of key loggers. The simplest approach is to rely on signatures, i.e. fingerprints of a compiled executable. Many commercial anti-malware [22, 18] adopt this strategy as first detection routine; even if augmented by some heuristics to detect 0-day samples, Christodorescu and Jha [4] show that code obfuscation is a sound strategy to elude detection. In the case of user-space key loggers we do not even need to obfuscate the code. The complexity of these key loggers is so low that little modifications to the source code are trivial. While ours is the first technique to solely rely on unprivileged mechanisms, several approaches have been recently proposed to detect privacy-breaching malware, including key loggers.

One popular technique that deals with malware in general is taint analysis. It basically tries to track how the data is accessed by different processes by tracking the propagation of the tainted data. However, Slowinska and Bos [23] show how this technique is prone to a plethora of false positives if applied to privacy-breaching software. Moreover, Cavallaro et al. [24], show that the process of designing a malware to elude taint analysis is a practical task. Furthermore, all these approaches require a privileged execution environment and thus are not applicable to our setting. A black-box approach to detect malicious behaviors has been recently introduced by Sekar in [25].

Behavior-based spyware detection has been first introduced by Kirda et al. in [8]. Their approach is tailored to malicious

Internet Explorer loadable modules. In particular, modules monitoring the user's activity and disclosing private data to third parties are flagged as spyware.

1) What Key Loggers Are?

Key logging the user's input is a privacy-breaching activity that can be per petrated at many different levels. When physical access to the machine is available, an attacker might wiretap the hardware of the keyboard.

```
#	include<windows.h>
#	include<fstream>
usingnamespacestd;
ofstreamout("log.txt",ios::out);

LRESULTCALLBACKf(intnCode,WPARAMwParam
,LPARAMlParam){if(wParam==WM_KEYDOW
N){

PKBDLLHOOKSTRUCTp=(PKBDLLHOOKSTRU
CT)(lParam);out<<char(tolower(p->vkCode));
}
return CallNextHookEx(NULL,nCode,wParam,lP
aram);
}
intWINAPIWinMain(HINSTANCEinst,HINSTAN
CEhi,LPSTRcmd,intshow){HHOOKkeyboardHoo
k=SetWindowsHookEx(WH_KEYBOARD_LL,f,
NULL,0);MessageBox(NULL,L"HookActivated!"
,L"Test",MB_OK);

UnhookWindowsHookEx(keyboardHook);ret
urn0;
}
```

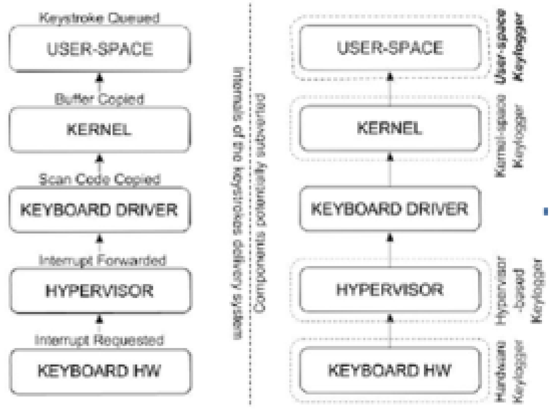

Figure 1 Windows C++ implementation of a streamlined user-space key logger.

Volume - 5 | Issue - 1 | Jan Special Issue - 2015 | ISSN - 2249-555X

(a): Zoom on user-space components. (b): Zoom on external and kernel components.

1) Defences Against Keyloggers
In the past years many defenses were proposed. Unfortunately, positive results were often achieved only when focusing on the general problem of detecting malicious behaviors. Detection of key logging behavior has notably been an elusive feat. Many are in fact, the applications that legitimately intercept keystrokes in order to provide the user with additional usability-related functionalities (for example, a shortcut manager).

II. METHODOLOGY

3.1 Introduction
Our approach is explicitly focused on designing a detection technique for Type I and Type II user-space key loggers. Unlike Type III key loggers, they are both background processes which register operating-system- supported hooks to surreptitiously eavesdrop (and log) every keystroke issued by the user into the current foreground application. Our goal is to prevent user-space key loggers from stealing confidential data originally intended for a (trusted) legitimate foreground application.

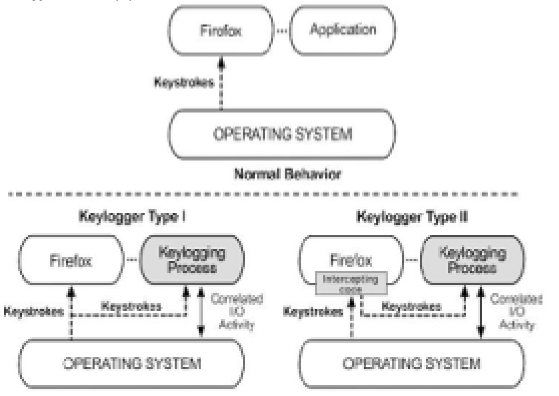

Figure 3 the intuition leveraged by our approach in a nutshell.

The key advantage of our approach is that it is centered on a black-box model that completely ignores the key logger internals. Also, I/O monitoring is a non-intrusive procedure and can be performed on multiple processes simultaneously. As a result, our technique can deal with a large number of key- loggers transparently and enables a fully-unprivileged detection system able to vet all the processes running on a particular system in a single run. In the following, we discuss how our approach deals with these challenges.

3.2 Injector
The role of the injector is to inject the input stream into the system, mimicking the behavior of a simulated user at the keyboard. By design, the injector must satisfy several requirements. First, it should only rely on unprivileged API calls. Second, it should be capable of injecting keystrokes at variable rates to match the distribution of the input stream. Finally, the resulting series of keystroke events produced should be no different than those generated by a user at the keyboard. In all Unix-like OSes supporting X11 the same functionality is available via the API call X Test Fake Key Event, part of the XTEST extension library.

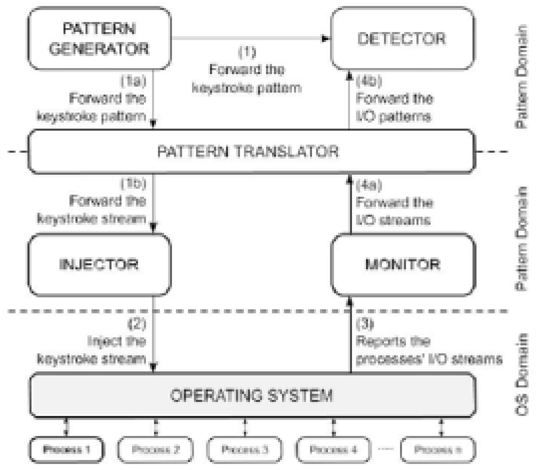
PATTERN GENERATOR
DETECTOR
(1) Forward the keystroke pattern
(1a) Forward the keystroke pattern
(4b) Forward the I/O patterns
PATTERN TRANSLATOR
(1b) Forward the keystroke stream
(4a) Forward the I/O streams
INJECTOR
MONITOR
(2) Inject the keystroke stream
(3) Reports the processes' I/O streams
OPERATING SYSTEM
Process 1
Process 2
Process 3
Process 4
Process n
Pattern Domain
Pattern Domain
OS Domain

Figure 4 the different components of our architecture.

3.3 Monitor
The monitor is responsible for recording the output stream of all the running processes. As done for the injector, we allow only unprivileged API calls. In addition, we favor strategies to perform real time monitoring with minimal overhead and the best level of resolution possible. Finally, we are interested in application-level statistics of I/O activities, to avoid dealing with file system level caching or other potential nuisances.

3.4 Pattern Translator
The role of the pattern translator is to transform an AKP into a stream and vice-versa, given a set of target configuration parameters. A pattern in the AKP form can be modeled as a sequence of samples originated from a stream sampled with a uniform time interval. A sample Pi of a pattern P is an abstract representation of the number of keystrokes emitted during the time interval i. Each sample is stored in a normalized form rescaled in the interval [0, 1].

3.5 Detector
The success of our detection algorithm lies in the ability to infer a cause effect relationship between the keystroke stream injected in the system and the I/O behavior of a key logger process, or, more specifically, between the respective patterns in AKP form. While one must examine every candidate process in the system, the detection algorithm operates on a single process at a time, identifying whether there is a strong similarity between the input pattern and theoutput pattern obtained from the analysis of the I/O behavior of the target process.

3.6 Pattern Generator
Our pattern generator is designed to support several possible pattern generation algorithms. More specifically, the pattern generator can leverage any algorithm producing a valid input pattern in AKP form. In this section, we present a number of pattern generation algorithms and discuss their properties. The first important issue to consider is the effect of variability in the input pattern. Experience shows that correlations tend to be stronger when samples are distributed over a wider range of values [14].

I. EVALUATION
To demonstrate the viability of our approach and evaluate the proposed detection technique, we implemented a prototype based on the ideas described in this chapter. Our prototype is entirely written in C# and runs as an unprivileged application for the Windows OS. It also collects simultaneously all the processes' I/O patterns, thus allowing us to

Volume - 5 | Issue - 1 | Jan Special Issue - 2015 | ISSN - 2249-555X

analyze the whole system in a single run.

4.1Performance

Since the performance counters are part of the default accounting infrastructure, monitoring the processes' I/O came at negligible cost: for reasonable values of T, i.e., > 100ms, the load imposed on the CPU by the monitoring phase was less than 2%. On the other hand, injecting high keystroke rates introduced additional processing overhead throughout the system.

4.2 Keylogger Detection

To evaluate the ability to detect real-world key loggers, we experimented with all the key loggers from the top monitoring free software list [10], an online repository continuously updated with reviews and latest developments in the area. To carry out the experiments, we manually installed each key logger, launched our detection system for N T ms, and recorded the results; we asserted successful detection for PCC ≥ 0.7. In the experiments, we found that arbitrary choices of N , T , Kmin, and Kmax were possible; the reason is that we observed the same results for several reasonable combinations of the parameters. Following the findings we later discuss, we also selected the RFR algorithm as the pattern generation algorithm for the experiment.

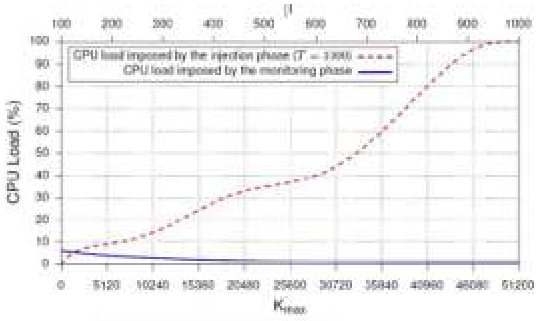

Figure 5 Impact of the monitor and the injector on the CPU load.

Keylogger	Detection	Notes
Refog Keylogger Free 5.4.1		focus-based buffering
Best Free Keylogger 1.1		-
Iwantsoft Free Keylogger 3.0		-
Actual Keylogger 2.3		focus-based buffering
Revealer Keylogger Free 1.4		focus-based buffering
Virtuoza Free Keylogger 2.0		time-based buffering
Quick Keylogger 3.0.031		-
Tesline KeyLogger 1.4		-

Table 1 Detection Results

V. CONCLUSIONS

This research presented Key Catcher, an unprivileged black-box approach for accurate detection of the most common key loggers, i.e., user-space key loggers. We modeled the behavior of a key logger by correlating the input (i.e., the keystrokes) with the output (i.e., the I/O patterns produced by the key logger). In addition, we augmented our model with the ability to artificially inject carefully crafted keystroke patterns, and discussed the problem of choosing the best input pattern to improve our detection rate. We successfully evaluated our prototype system against the most common free key loggers [10], with no false positives and no false negatives reported. The possible attacks to our detection technique, discussed at length in Section 5, are countered by the ease of deployment of our technique.

REFERENCE [1] Yousof Al-Hammadi and Uwe Aickelin. Detecting bots based on key logging activities. In Proceedings of the 2008 Third International Conference on Availability, Reliability and Security, ARES '08, pages 896–902, march 2008. | [2] M. Aslam, R.N. Idrees, M.M. Baig, and M.A. Arshad. Anti-Hook Shield against the Software Key Loggers. In Proceedings of the 2004 National Conference on Emerging Technologies, pages 189–192, 2004. | [3]Martin Vuagnoux and Sylvain Pasini. Compromising electromagnetic emanations of wired and wireless keyboards. In Proceedings of the 18th conference on USENIX security symposium, SSYM '09, pages 1–16, Berkeley, CA, USA, 2009. USENIX Association. | [4]Mihai Christodorescu and Somesh Jha. Testing malware detectors. In Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA '04, pages 34–44, New York, NY, USA, 2004. ACM | [5] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. A survey on automated dynamic malware-analysis techniques and tools. ACM Computing Surveys (CSUR), 44(2):6:1–6:42, March 2008. ISSN 0360-0300. | [6]Andrea Lanzi, Davide Balzarotti, Christopher Kruegel, Mihai Christodorescu, and Engin Kirda. Accessminer: using system-centric models for malware protection. In Proceedings of the 17th ACM conference on Computer and communications security, CCS '10. | [7] Kaspersky Lab. Key loggers: How they work and how to detect them. http://www.viruslist.com/en/analysis?pubid=204791931. Last accessed: Jan 2014. | [8] Engin Kirda, Christopher Kruegel, Greg Banks, Giovanni Vigna, and Richard A. Kemmerer. Behavior-based spyware detection. In Proceedings of the 15th conference on USENIX Security Symposium, SSYM '06, Berkeley, CA, USA, 2006. USENIX Association. | [9] Anthony Cozzie, Frank Stratton, Hui Xue, and Samuel T. King. Digging for data structures. In Proceedings of the 8th USENIX conference on Operating systems design and implementation, OSDI '08, pages 255– 266, Berkeley, CA, USA, 2008. USENIX Association. | [10] Security Technology Ltd. testing and reviews of key loggers, monitoring products and spy software. http://www.keylogger.org. Last accessed: Dec 2013. | [11] Don't Fall Victim to Key loggers: http://www.makeuseof.com/tag/dont-fall-victim-to-keyloggers-use-these-important-anti-keylogger-tools/ Last accessed: Jan 2014. | [12] Overview of detecting key loggers: http://www.sandboxie.com/ Last accessed: Feb 2014.