RESEARCH ARTICLE OPEN ACCESS

Unprivileged Black-Box Detection of User-Space Key loggers

[1]Mr.S.Jagadeesan,M.Sc, MCA., M.Phil., ME[CSE]., [2]S.Rubiya,
[1]Assistant professor, [2]Final year,
Department of Computer Applications,
Nandha Engineering College/Anna University,
Erode.

----------**********----------

Abstract:

Software keyloggers are a fast growing class of invasive software often used to harvest confidential information. One of the main reasons for this rapid growth is the possibility for unprivileged programs running in user space to eavesdrop and record all thekeystrokes typed by the users of a system. The ability to run in unprivileged mode facilitates their implementation and distribution, but,at the same time, allows one to understand and model their behavior in detail. Leveraging this characteristic, we propose a new detection technique that simulates carefully 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. We have successfully evaluated the underlying technique against the most common free keyloggers. This confirms the viability of our approach in practical scenarios. We have also devised potential evasion techniques that may be adopted to circumvent our approach and proposed a heuristic to strengthen the effectiveness of our solution against more elaborated attacks. Extensive experimental results confirm that our technique is robust to both false positives and false negatives in realistic settings.

Keywords — **Invasive software, keylogger, security, black-box, PCC**

----------**********----------

I. INTRODUCTION

KEYLOGGERS are entrenched on a machine to deliberately monitor the user action by logging keystrokes and finally delivering them to a third party [1]. While they are rarely used for genuine drives (e.g., surveillance/parental monitoring infrastructures), key loggers are often unkindly exploited by assailants to steal intimate information. Many credit card numbers then passwords have been occupied using key loggers [2], [3], which makes them one of the most unsafe types of spyware recognised to date.

Key loggers can be applied as tiny hardware plans or more suitably in software. Software-based key logger image be additional classified based on the privileges they require to perform. Key loggers applied by a kernel unit run with full freedoms in kernel space. Equally, a fully poor key logger can be applied by a simple user-space procedure. It is significant to notice that auser-space key logger can easily rely on recognised sets of poor APIs commonly available on modern working systems (OSs). This is not the case for a key logger applied as a seed module. In seed space, the computer operator must rely on kernel-level amenities to interrupt all the messages posted by the console driver, certainly needful a substantial effort then information for an real and bug-free implementation. Also, a key logger applied as a user-space procedure is much calmer to deploy since no special consent is required. A

user can mistakenly regard the key logger as a inoffensive part of software and being cuckolded in performing it. On the conflicting, kernel-space key loggers need a user with wonderful user freedoms to deliberately install and execute unsigned code within the kernel, a repetition often prohibited by

modern working systems such Gaps Vista or Windows 7. In light of these comments, it is no surprise that 95 out of a hundred of the current key loggers run in user space [4]. Notwithstanding the rapid development of key logger-based deceptions (i.e., identity theft, password leakage, etc.), not many effective and efficient solutions have been proposed to address this problem. Old-style defense mechanisms use Finger production plans similar to those used to detect worms and worms. Inappropriately, this plan is hardly real against the vast amount of new key logger alternatives developing every day in the wild. In this paper, we propose a new method to detect key loggers running as poor user-space procedures. To match the same placement model, our method is completely applied in an poor process. As a consequence, our answer is portable, uninstructive, easy to install, then yet very real. In the final part of this paper, we further authenticate our approach with a home full-grown key logger that efforts to thwart our discovery method.

2 INTERNALS OF MODERN KEYLOGGERS

Breaking the privacy of an separate by classification his keystrokes can be committed at many dissimilar levels. For example, an assailant with bodily access to the mechanism might bug the hardware of the keyboard. A lying audio releases produced by the user typing [6], or convenient to purchase a software solution, install it on all implemented in many different ways. Aimed at instance, outside keyloggers rely on approximately physical property, either the mechanism. Contingent on the location, a keylogger can be owner of an Internet cafe´, in turn, may find it more the electromagnetic releases of a wireless keyboard [7]. the stations, and have the logs released on his own Hardware keyloggers are still outside plans, but are applied as dongles placed in amid keyboard and motherboard. All these plans, though, need bodily access to the board machine.

To overwhelmed this kerb, software approaches are additional commonly used. Hypervisor-based keyloggers (e.g., BluePill [8]) are the frank software development of hardware based keyloggers, factually execution a man-in-the-middle bout amid the hardware then the working system. Kernel keyloggers originate instant in the chain then are often applied as part of more multifaceted rootkits. In difference to hypervisor-based methods, hooks are straight used to interrupt buffer-processing proceedings or additional seed messages. Albeit effective, all these methods need advantaged access to the machine. Furthermore, script a kernel driver hypervisor-based methods pose even more challenges requires a considerable effort and knowledge for an effective and bug-free implementation (smooth a single bug may lead to a kernel fright). User-space keyloggers, on the additional hand, do not need any special honour to be deployed. They can be connected and performed irrespective of the freedoms granted.

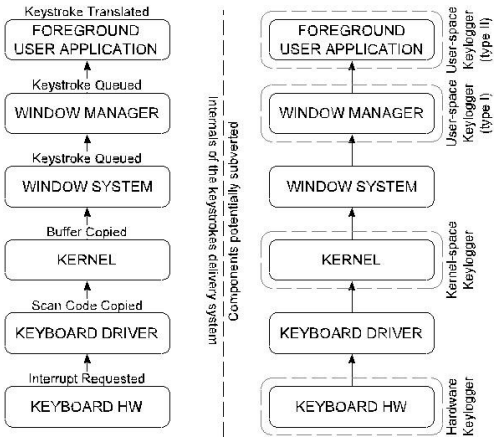


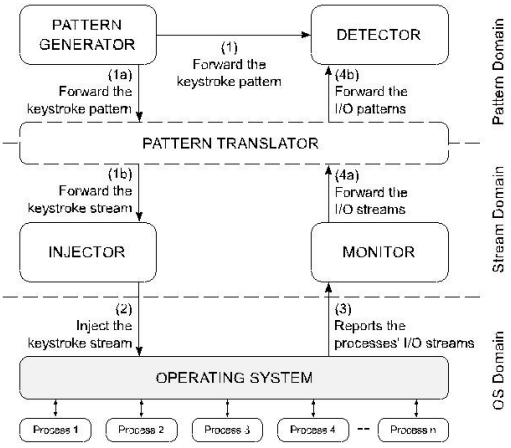
Fig. 1. The distribution stages of a keystroke, then the mechanisms possibly undermined (we omit hypervisor-based methods for the sake of clarity)

This is a feat unbearable for kernel keyloggers, since they need either super user freedoms or a susceptibility that allows random kernel code implementation. Also, user space keylogger authors can securely rely on well-documented sets of APIs usually obtainable on modern working systems, with no singular programming skills obligatory.

3.APPROACH

Our method is openly focused on scheming a detection method for poor user-space keyloggers. Unlike other classes of keyloggers, a user-space keylogger is a contextual process which lists operating-system-supported hooks to furtively

snoop (and log) every keystroke delivered by the user into the current forefront application. Our goal is to stop user-space keyloggers after theft intimate data originally envisioned for a (trusted) genuine foreground request. Hateful foreground requests furtively classification user-issued keystrokes (e.g., a keylogger deceiving a right-hand word computer application) and request-specific keyloggers (e.g., browser plugins furtively execution keylogging activities) are outdoor our danger model then cannot be recognised using our discovery technique. Too note that a contextual keylogger cannot brood a forefront request and steal the current request focus on request deprived of the user directly noticing. Our perfect is founded on these comments and travels the option of isolating the keylogger in a skilful setting, where its behaviour is straight exposed to the discovery system. Our method includes controlling the keystroke proceedings that the keylogger obtains in input, and continually nursing the I/O action made by the keylogger in output. To declare discovery, we leverage the instinct that the association amid the input and output of the skilful setting can be modelled for most keyloggers with very decent estimate. Regardless of the alterations the keylogger performs, a typical pattern observed in the keystroke proceedings in input shall somehow be reproduced in the I/O action in output. Once the input then the output are skilful, we can classify common I/O designs and flag discovery.



Our method completely disregards the gratified of the then the output data, and emphases wholly on their distribution. Warning the method to a measureable analysis allows the aptitude to tool the detection method with only poor devices, as we will better exemplify later. The fundamental model adopted, however, gifts extra tests. First, we must prudently deal with possible data alterations that may introduce measureable differences amid the input and the production patterns.

4. ARCHITECTURE

Our project is based on five dissimilar mechanisms as portrayed in injector, screen, pattern translator, sensor, design generator. The working system at the lowest contracts with the details of I/O then occasion treatment. The OS Area does not expose all the details to the higher levels deprived of using advantaged API calls. As a consequence, the injector and the screen operate at additional level of concept, the Stream Domain. At this level, keystroke events and the bytes production by a process seem as a stream produced at a specific rate. The task of the injector is to inject a keystroke stream to fake the behaviour of a user keying at the keyboard. Likewise, the monitor annals a stream of bytes to continually imprisonment the output behaviour of a specific process. A stream representation is only worried with the delivery of keystrokes or bytes produced over a given gap of observation, deprived of entailing any extra qualitative info. The injector obtains the input watercourse from the design translator, which acts as bond between the Watercourse Area and the Pattern Domain. Likewise, the screen delivers the output stream logged to the pattern interpreter for further analysis. In the Pattern Area, the input stream and the production stream are both represented in a additional abstract form, called Abstract Keystroke Pattern (AKP).

4.1 Injector

The role of the injector is to inject the input stream into the system, faking the behaviour of a operator at the console. By project, the injector must content several supplies. First, it must only rely on poor API calls. Second, it must be accomplished of injecting keystrokes at mutable

rates to competition the delivery of the input watercourse. Lastly, the subsequent sequence of keystroke events shaped should be no dissimilar than those made by a real user. In additional words, no user-space keylogger should be someway able to differentiate the two types of proceedings. To speech all these subjects, we influence the same method working in automatic testing. On Windows-based working systems this functionality is if by the API call keybd_ occasion. In all Unix-like OSes secondary X11 the same functionality is obtainable via the API call XTestFake- KeyEvent.

4.2 Monitor

The monitor is accountable to best the output stream of all the consecutively processes. As done for the injector, we allow only poor API calls. In addition, we favour strategies to perform realtime monitoring with negligible above and the best level of resolve possible. Lastly, we are interested in application-level figures of I/O doings, to avoid dealing with filesystem-level hiding or other possible nuisances. In specific, the presentation pawns of each procedure are made obtainable via the class Win32_Process, which supports an effectual query-based border. The pawn Write Transmission Count covers the total number of bytes printed by the procedure meanwhile its creation.

4.3 Pattern Translator

The role of the design interpreter is to alter an AKP into a stream and vice versa, assumed a set of shape parameters. A design in the AKP procedure can be modelled as a order of samples created from a stream tested with a unchanging time intermission. A example Pi of a design P is an nonconcrete picture of the amount of keystrokes produced throughout the period interval i. Apiece example is stored in a regularised procedure in the intermission ½0; 1_, anywhere 0 and 1 reproduce the predefined least and all-out number of keystrokes in a assumed time intermission. To alter an input project into a keystroke stream, the design interpreter reflects the next shape limits: N, the number of examples in the pattern; T, the continuous time intermission between any two consecutive samples; Kmin, the least amount of keystrokes per example allowed; then Kmax, the all-out number of keystrokes per example allowed.

4.4 Detector

The attainment of our discovery algorithm lies in the aptitude to infer a cause-effect relationship amid the keystroke watercourse vaccinated in the system and the I/O behavior of a keylogger procedure, or, more exactly, amid the own patterns in AKP form. Though one must inspect every applicant process in the scheme, the discovery algorithm functions on a single procedure at a time, classifying whether there is a robust resemblance between the input design and the output pattern got from the analysis of the I/O behavior of the board process. Exactly, given a predefined input design and an output design of a specific process, the goal of the discovery algorithm is to control whether there is a match in the designs and the target procedure can be recognised as a key logger with good likelihood. In difference to other association metrics, the PCC events the forte of a linear association between two sequence of samples, disregarding any nonlinear connotation. In our location, a linear need well approaches the association amid the input pattern then an output pattern shaped by a keylogger. The instinct is that a keylogger can only brand local choices on a perkeystroke basis with no information around the global delivery. Thus, in code, the subsequent behavior will linearly estimated the unique input stream injected into the system. In detail, the PCC is resilient to any change in location and scale, namely no difference can be observed in the correlation coefficient if every sample Pi of any of the two patterns is transformed into a _ Pi þ b, where a and b are arbitrary constants. This is important for a number of reasons.

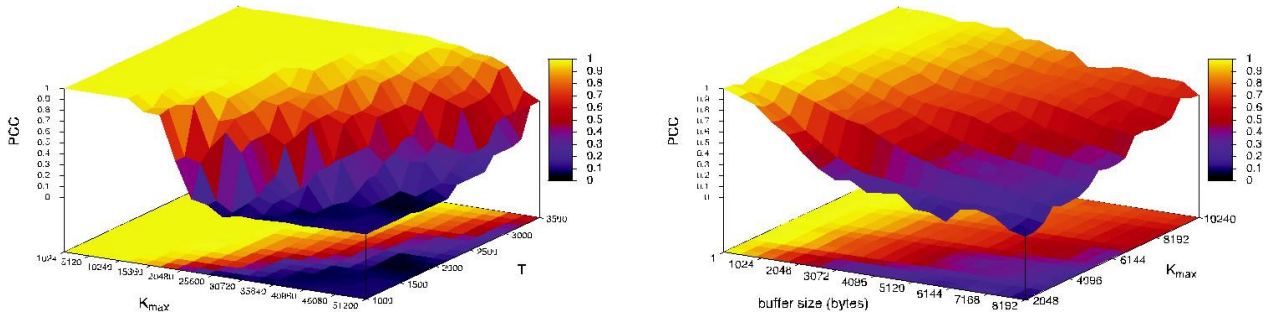
5. KEYLOGGER DETECTION

To evaluate the ability to detect real-world keyloggers, we experimented with all the keyloggers from the top monitoring free software list [5], an online repository continuously updated with reviews and latest developments in the area. In addition, some of the keyloggers examined included support for encryption and most of them used variable-length encoding to store manually installed each keylogger, launched our detection system for N _ T ms, and recorded the results; we asserted successful.

Another potential issue rises from keyloggers removal a fixed-format shot on the disk every time a change of focus is noticed. The header typically covers the date and the name of the target application. However, as we intended our discovery system to change focus at every sample, the header is red-faced out to disk at each time intermission along by all the keystrokes injected. As a consequence, the output pattern checked is just a location alteration of the sole, with the shift assumed by size of the header itself. Thanks to the location invariance stuff, our detection algorithm is naturally resilient to this alteration.

5.1 False Negatives

In our approach our method, false positives output design of a keylogger wounds an unpredictably low PCC value. To test the robustness of our method against untrue rejections, we made numerous trials with our own artificial keylogger. Our evaluation starts by analyzing the impact of the amount of examples N and the time interval T on the final PCC value. For each design generation algorithm, we plot the PCC slow with our prototype keylogger which we arranged so that no cushioning or data alteration was taking place. Figs. 3a and 3b portray our answers with Kmin ¼ 1 and Kmax ¼ 1;000. We detect that once the keylogger logs each keystroke without presenting delay or extra noise, the number of examples N does not affect the PCC value. This behaviour should not propose that N has no effect on the production of false rejections. Once noise in the output watercourse is to be predictable, advanced values of N are indeed wanted to produce more stable PCC values and evade false rejections. In difference, Fig. 3b shows that the PCC is sensitive to low values of the time interval T. The effect observed is due to the incapability of the system to absorb all the injected keystrokes for time intermissions smaller than 450 ms. Fig. 3c, in turn, shows the influence of Kmin on the PCC (with Kmax still constant). The consequences settle our observations in Section 4.4, i.e., that patterns branded by a low variance hinder the PCC, and thus a high variability in the inoculation design is desirable. We now analyze the impact of the all-out number of keystrokes per time intermission Kmax. High Kmax values are predictable to increase the level of erraticism, reduce the quantity of noise, and induce a more distinct delivery in the output watercourse of the keylogger. The keystroke degree, though, is clearly bound by the length of the time interval T. Fig. 4 portrays the PCC slow with our example keylogger for N ¼ 30, Kmin ¼ 1, and RND pattern cohort algorithm. The number reports very high PCC values for Kmax < 20,480 and T ¼ 1;000 ms. This behaviour reflects the incapability



of the system to engross more than Kmax _ 20,480 in the given time intermission. Increasing T is, however, adequate to allow advanced Kmax values without significantly impacting the PCC. For example, with T ¼ 3,500 ms we can dual Kmax without level-headedly degrading the final PCC value.

Transformations. First, we tested with a keylogger using a nontrivial fixed-length indoctrination for keystrokes. Fig. 5a portrays the consequences for dissimilar values of padding p with N ¼ 30, Kmin ¼ 1, and Kmax ¼ 1,024. A value of p ¼ 1,024 simulates a keylogger writing 1,024 bytes on the disk for each eavesdropped keystroke. As discussed in Section 4.4, the PCC should be unaffected in this case and presumably exhibit a constant behavior. The figure confirms this intuition, but displays the

PCC lessening linearly after p _ 10;000 bytes. This behavior is due to the limited I/O throughput that can be achieved within a single time interval.

The behavior observed is very evaluation, our technique can still handle this class of keyloggers correctly for reasonable buffer sizes. Fig. 6 depicts our discovery consequences in contradiction of a keylogger buffering its output through a fixed-size buffer. The amount shows the impact of numerous likely choices of the bumper size on the final PCC value. We can notice the pivotal role of Kmax in definitely stating discovery. For instance, cumulative Kmax to 10,240 is necessary to achieve sufficiently high PCC values for the largest plentiful size future. This trial demonstrates once again that the key to discovery is persuading the pattern to definitely emerge in the output distribution, a feat that can be easily obtained by choosing a highly mutable immunisation design with low standards for Kmin and high values for Kmax. We believe these consequences are hopeful to admit the robustness of our detection technique in contradiction of false negatives, even in presence of multifaceted data transformations.

5.3 False Positives

In our method, false positives may occur when the output pattern of some kind procedure unintentionally scores a important PCC value. If the value happens to be better than the designated threshold, a false discovery is highlighted.

6 .EVASION AND COUNTERMEASURES

In this section, we speculate on the possible evasion techniques a keylogger may employment once our detection plan is prearranged on real systems.

6.1 Aggressive Buffering

A keylogger may rely on around forms of violent buffering, for example flushing a very large buffer every time intermission t, with t being possibly hours. While our model can possibly address this scenario, the very large gap of observation required to collect a drivable amount of examples would make the following detection technique irrational. It is important to point out that such a kerb stems from the appeal of the method and not from a project flaw in our detection model. For instance, our model could be practical to memory access designs instead of I/O designs to make the resulting discovery technique resistant to aggressive cushioning. This plan, however, would require a hardwearing substructure (e.g., virtualized environment) to monitor thememory accesses, thus hindering the welfares of a fully poor solution.

6.2 Trigger-Based Behavior

A keylogger may activate the keylogging activity only in face of specific events, for example when the user launches a specific request. Inappropriately, this trigger-based behavior may successfully evade our detection method. This is not, however, a shortcoming exact to our method, but rather a more fundamental kerb common to all the existing detection techniques based on lively analysis [17]. While we trust that the problematic of activating a specific behaviour is orthogonal to our work and already focus of much ongoing investigation, we point out that the user can still mitigate this threat by occasionally re-releasing detection runs when essential (e.g., every time a new particularly subtle context is accessed). Since our technique can vet all the procedures in a single detection run, we believe this plan can be realistically used in real-world situations.

6.3 Discrimination Attacks

Imitating the user's behaviour may depiction our method to keyloggers talented to tell artificial then real keystrokes apart. A keylogger may, for example, ignore any contribution failing to show known arithmetical properties— e.g., not akin to the English language. Though, since we switch the input design, we can prudently make keystroke scancode sequences displaying the same statistical properties (e.g., English text) predictable by the keylogger, and therewith do a separate detection run thwarting this evasion method. Around the case of a keylogger disregarding keystrokes when detecting a high (nonhuman) inoculation rate. This plan, though, would make the keylogger disposed to to denial of service: a system obstinately generating and exfiltrating bogus keystrokes would persuade this type of keylogger to enduringly disable the keylogging activity. Recent work proves that building such a system is feasible in repetition (with

reasonable overhead) using normal two facilities [18].

6.4 Decorrelation Attacks

Decor relation attacks effort at breaking the correlation metric our method relies on. Meanwhile of all the attacks this is exactly custom-made to thwarting our method, we hereby suggest a experiential envisioned to vet the system in case of negative discovery results. This is the case, for example, of a keylogger trying to generate I/O noise in the contextual and lowering the association that is bound to exist between the pattern of keystrokes injected I and its own output pattern O. In the attacker's ideal case, this interprets to PCCðI;OÞ _ 0. To approximate this result in the general case, however, the attacker must adapt its disbursement strategy to the pattern cohort algorithm in use, i.e., once switching to a new inoculation I0 6¼ I, the output design should reflect a new delivery O0 6¼ O. The assailant could, for instance, enforce this property by adapting the noise generation to some input distributionspecific variable (e.g., the present keystroke rate). Failure to do so will result in random noise uncorrelated with the injection, a scenario which is already touched by our PCCbased detection technique.

This method, often used to liken time series, warps sequences in the time measurement to determine a measure of similarity independent of nonlinear variations in the time dimensions. To evaluate our heuristic, we applied two different keyloggers trying to evade our detection technique. The first one, K-EXP, uses a similar thread to write a random amount of bytes which increases exponentially with the number of keystrokes already logged to the disk. Since the alteration is nonlinear, we expect heavily disconcerted PCC values.

7. RELATED WORK

While ours is the first method to solely rely on unprivileged mechanisms, several methods have been recently future to detect privacy-breaching malware, counting keyloggers. Behaviour-based spyware detection has been first presented by Kirda et al. Their approach is custom-made to malicious Internet Explorer loadable modules. In particular, modules nursing the user's activity and disclosing private data to third parties are highlighted as malware. Their examination models hateful behavior in footings of API calls invoked in response to browser events. Those used by keyloggers, though, are also usually used by legitimate programs. Their approach is therefore prone to false positives, which can only be mitigated with continuously efficient whitelists. Other keylogger-specific methods have optional detecting the use of well-known keystroke interception APIs. Aslam suggest binary static analysis to locate the envisioned API calls. Unfortunately, all these calls are also used by legitimate requests (e.g., shortcutmanagers) and this approach is again prone to false positives. push this method further, specifically targeting Windows-based working systems.

Earlier to our method is the answer proposed by Al-Hammadi. Their plan is to model the keylogging behavior in footings of the number of API calls issued in the gap of observation. To be more exact, they observe the frequency of API calls invoked to 1) intercept keystrokes, 2) script to a file, and 3) delivery bytes over the network. A keylogger is detected once two of these incidences are found to be highly correlated. Since no bogus proceedings are issued to the system (no injection of crafted input), the correlation may not be as strong as predictable. The resulting value would be even more reduced in case of any delay presented by the keylogger. Furthermore, since their inspection is eavesdropped. This evaluates the impact of numerous conditions. First, the experiment fakes a keylogger randomly dropping keystrokes with a certain likelihood.

8. CONCLUSIONS

In this paper, we obtainable an poor black-box approach for precise discovery of the most common keyloggers, i.e., user-space keyloggers. We modeled the behavior of a keylogger by surgically correlating the input (i.e., the keystrokes) with the output (i.e., the I/O patterns produced by the keylogger). Impact of N on the DTW. keystroke patterns. We then deliberated the problem of choosing the best input pattern to recover our detection rate. Then, we presented an implementation of our detection technique on

Windows, arguably the most susceptible OS to the danger of keyloggers. To found an OS-independent architecture, we also gave application details for other operating systems. We positively evaluated our prototype scheme against the most communal free keyloggers [5], with no false positives and no untrue negatives reported. Other new results with a homegrown keylogger demonstrated the effectiveness of our technique in the general case. While attacks to our detection method are possible and have been discussed at length in Section 6, we believe our approach considerably raises the bar for protecting the user in contradiction of the danger of keyloggers.

REFERENCES

[1] T. Holz, M. Engelberth, and F. Freiling, "Learning More About the Underground Economy: A Case-Study of Keyloggers and Dropzones," Proc. 14th European Symp. Research in Computer Security, pp. 1-18, 2009.

[2] San Jose Mercury News, "Kinkois Spyware Case Highlights Risk of Public Internet Terminals," http://www.siliconvalley.com/ mld/siliconvalley/news/6359407.htm, 2012.

[3] N. Strahija, "Student Charged After College Computers Hacked," http://www.xatrix.org/article2641.html, 2012.

[4] N. Grebennikov, "Keyloggers: How They Work and How to Detect Them," http://www.viruslist.com/en/analysis? pubid=204791931, 2012.

[5] Security Technology Ltd., "Testing and Reviews of Keyloggers, Monitoring Products and Spyware," http://www.keylogger. org, 2012.

[6] L. Zhuang, F. Zhou, and J.D. Tygar, "Keyboard Acoustic Emanations Revisited," ACM Trans. Information and System Security, vol. 13, no. 1, pp. 1-26, 2009.

[7] M. Vuagnoux and S. Pasini, "Compromising Electromagnetic Emanations of Wired and Wireless Keyboards," Proc. 18th USENIX Security Symp., pp. 1-16, 2009.

[8] J. Rutkowska, "Subverting Vista Kernel for Fun and Profit," Black Hat Briefings, vol. 5, 2007.

[9] J.L. Rodgers and W.A. Nicewander, "Thirteen Ways to Look at the Correlation Coefficient," The Am. Statistician, vol. 42, no. 1, pp. 59-66, Feb. 1988.

[10] J. Benesty, J. Chen, and Y. Huang, "On the Importance of the Pearson Correlation Coefficient in Noise Reduction," IEEE Trans. Audio, Speech, and Language Processing, vol. 16, no. 4, pp. 757-765, May 2008.

[11] L. Goodwin and N. Leech, "Understanding Correlation: Factors that Affect the Size of r," Experimental Education, vol. 74, no. 3, pp. 249-266, 2006.

[12] J. Aldrich, "Correlations Genuine and Spurious in Pearson and Yule," Statistical Science, vol. 10, no. 4, pp. 364-376, 1995.

[13] W. Hsu and A. Smith, "Characteristics of I/O Traffic in Personal Computer and Server Workloads," IBM System J., vol. 42, no. 2, pp. 347-372, 2003.

[14] H.W. Kuhn, "The Hungarian Method for the Assignment Problem," Naval Research Logistics Quarterly, vol. 2, pp. 83-9, 1955.

[15] G. Kochenberger, F. Glover, and B. Alidaee, "An Effective Approach for Solving the Binary Assignment Problem with Side Constraints," Information Technology and Decision Making, vol. 1, pp. 121-129, May 2002.

[16] BAPCO, "SYSmark 2004 SE," http://www.bapco.com, 2012.

[17] A. Moser, C. Kruegel, and E. Kirda, "Exploring Multiple Execution Paths for Malware Analysis," Proc. IEEE 28th Symp. Security and Privacy, pp. 231-245, May 2007.

[18] S. Ortolani and B. Crispo, "Noisykey: Tolerating Keyloggers via Keystrokes Hiding," Proc. Seventh USENIX Workshop Hot Topics in Security, 2012.