Keyloggers in Cybersecurity Education

Christopher A. Wood[1] **and Rajendra K. Raj**[2]
[1]Department of Software Engineering, Rochester Institute of Technology, Rochester, New York, USA
[2]Department of Computer Science, Rochester Institute of Technology, Rochester, New York, USA

Abstract— *Keylogger programs attempt to retrieve confidential information by covertly capturing user input via keystroke monitoring and then relaying this information to others, often for malicious purposes. Keyloggers thus pose a major threat to business and personal activities such as Internet transactions, online banking, email, or chat. To deal with such threats, not only must users be made aware about this type of malware, but software practitioners and students must also be educated in the design, implementation, and monitoring of effective defenses against different keylogger attacks.*

This paper presents a case for incorporating keylogging in cybersecurity education. First, the paper provides an overview of keylogger programs, discusses keylogger design, implementation, and usage, and presents effective approaches to detect and prevent keylogging attacks. Second, the paper outlines several keylogging projects that can be incorporated into an undergraduate computing program to educate the next generation of cybersecurity practitioners in this important topic.

Keywords: Computer security, keylogging, rootkits, secure coding, cybersecurity education.

1. Introduction

Keylogging programs, commonly known as keyloggers, are a type of malware that maliciously track user input from the keyboard in an attempt to retrieve personal and private information. Increasing computer use for common business and personal activities using the Internet has made effective handling of keylogging urgent [1]. Additionally, the Internet has not only become a major conduit for placing and distributing malicious programs, but also an aid in their infection and execution. The enormous potential of the Internet has therefore led to an increase in keylogging attempts with a linear annual increase in unique keyloggers [2].

A study of keylogging programs, along with anti-keylogging techniques, thus should be included in cybersecurity education for several reasons. First, keyloggers incorporate a wide array of cybersecurity issues and provide a practical approach to understanding topics such as attacker goals, varieties of malware and their implementation, the role of malware in infecting and controlling a system, and how stealth is achieved in an infected system. Second, students will understand tools and mechanisms that aid in the detection and prevention of keyloggers. Commercial anti-malware programs handle common keylogging malware fairly well as they tend to be static in nature and form, but are not as effective in detecting state-of-the-art malware that employ novel stealth and behavior mechanisms without easily recognized static signatures or patterns [3]. Whether the detection is via active system monitoring for malware memory footprints or for keylogger-like behavior, a more dynamic approach to detecting keyloggers is needed. In fact, the degree of dynamism separates mediocre anti-malware programs from effective ones. Ensuring that a security practitioner learns about handling keylogging malware is thus important in cybersecurity education.

The rest of the paper is laid out as follows. Section 2 presents the the current state of keylogging malware and Section 3 outlines software design and implementation techniques used in keylogging. Tools and techniques used to detect and prevent keylogging are presented in Section 4. Based on the authors' experiences, Section 5 outlines appropriate programming projects in keylogging that can be used in university education in cybersecurity. Section 6 summarizes the current status of this work.

2. An Overview of Keylogging

The keyboard is the primary target for keyloggers to retrieve user input from because it is the most common user interface with a computer. Although both hardware and software keyloggers exist, software keyloggers are the dominant form and thus are the focal point in this paper. For completeness however, this paragraph mentions hardware keyloggers as they do pose a significant security threat. A common example of a hardware keylogger is a "ghost" device that may be physically attached to a target machine to extract and store keystrokes on persistent storage within the same device. For example, inexpensive hardware keylogging devices such as the Spy Keylogger [4] act as a medium between the physical keyboard USB adapter and computer's motherboard USB port; as the "man in the middle," the device stealthily captures and stores all user keystrokes on its memory. Similarly, wireless hardware keyloggers translate and store encrypted keystoke bits sent from a wireless keyboard to its computer. Olzak [5] provides a reasonable starting point for additional information about hardware keyloggers.

This paper focuses on software keyloggers because they are the dominant form of keylogging. Both inexpensive

consumer-oriented and specially-developed keylogging programs are readily available on the Internet [1]. These keyloggers need to be adapted to each target operating system to ensure I/O is handled appropriately. System differences thus inevitably lead to operating system specific mechanisms implemented in software keyloggers: use of the keyboard state table, system routine hooks, and kernel-mode layered drivers. Additional detail about techniques used in the development, distribution, execution and detection of user- and kernel-mode keyloggers, particularly on Microsoft Windows operating systems, are presented later; in this paper, note that a reference to Windows means a Windows NT variant.

A fundamental concept behind keyloggers and similar malware is their pattern of attack. Most malware infections follow a fairly standard attack pattern that involves the sequential order of development, distribution and infection, and execution stages. The initial phase is vital to the process as any malware that is not yet implemented cannot be used by an attacker. What is unique about the development stage is that it places emphasis on how the latter stages will be accomplished. Distribution and execution can both be implemented as a component of the malware and therefore are a contributing factor in its design and development.

Remote keylogger distribution is a vital step for remote infection. Currently there are many ways to distribute keyloggers using the Internet. A study by Provos et al. [6] shows that there are four distinct approaches to malware placement on the Internet for distribution:

1) Advertisements. These provide a common hosting place for malware. As advertisements often tend to be redirections chained together, it is possible for third-parties to inject the location of malicious content into one of the nodes in the chain.
2) Third-party widgets. As with advertisements, widgets are fundamentally embedded links, often to an external Javascript function or similar entities, that can be redirected to dangerous locations.
3) User contributed content. Here a typical web user physically uploads content to a public location. If the web master does an inadequate job of checking content legality and validity via appropriate sanitization techniques, malicious content placement may occur.
4) Web server security mechanisms. These mechanisms also play an important role as they can impede malware placement on web sites by controlling server content such as HTML, Javascript, PHP (or other scripting languages and applications), and database contents. Therefore, an attacker who gains control of these security mechanisms has the ability to completely control the content on the web server and use it to her advantage.

Malware distribution is often followed by infection, which can be accomplished through both web application exploits and social engineering techniques. "Drive-by-downloads", as they are called, are forms of exploitation that involve the automatic download and execution of malicious binaries when a user visits a dangerous remote location [6]. These are accomplished by exploiting insecure browser vulnerabilities using malicious code that will invoke system routines or shell commands on the victim's computer to initiate the retrieval of the malware.

The other option for the attacker trying to infect a machine that has no identifiable security vulnerabilities is to trick the user into self-infection. In other words, the attacker will employ what is referred to as "social engineering" [7] to create interest in the user to perform an action that will result in the remote retrieval of malware.

The final stage in the attack pattern is for the keylogging malware to begin executing, and can occur in several different ways depending on the implementation and context of the keylogger. However, most realistic keyloggers share two common operations: (a) hooking into user input flow to receive keystrokes and (b) transporting the data to a remote location. The implementation of these operations is discussed in the next section.

3. Design and Implementation

Keylogger design and implementation strategies are based upon several factors: the infecting medium, the type of target machine, the lifetime of the keylogger, and the level of stealth and footprint left on the machine while active. Infection mechanisms depend on the form of the keylogger. For instance, a software keylogger targeting the user-mode of an operating system is often injected remotely and a hardware keylogger via physical device placement. Software keyloggers require a well-crafted infection mechanism to ensure proper installation, for example, a web browser exploit. Depending on the browser being used, the attacker can identify and exploit existing security vulnerabilities. A typical browser exploit will utilize a client side language like Javascript to craft and execute an attack [8]. These local attacks generally aim to create a buffer overflow in the browser or a related component such as a plugin to redirect the control and data flow of the target to allow malicious code to be executed.

Once the infection mechanism has been implemented, the keylogger designer will focus on its execution. Most keyloggers share a common execution technique known as hooking, though each keylogger will implement it in a different way depending on the context for which the keylogger is needed. The basic goal of hooking is to intercept the normal control flow and alter information returned by a target system routine [9]. Hooks can be implemented in any level of the operating system for most functions, which makes them a general technique to be utilized by keylogger developers. This paper uses the term hooking to describe any technique that intercepts data in an existing control flow for malicious use.

High-level keyloggers executing in the user-mode of an operating system are implemented using a variation of user-mode hooks. In a Windows operating system, keystroke events from the user are flagged through a message mechanism that transfers data from the keyboard device to the window procedure that is to respond to the the keystroke. This message mechanism can be hooked to provide an attacker with access to these keystroke events even before they reach the target application.

Depending on context, the keylogger being developed can implement a global or local hook to retrieve keystroke events. Global hooks monitor system-wide messages while local hooks monitor application-specific messages [1]. Using these hooked messages, the attacker can read the keystrokes entered, modify keystroke data, and even interrupt the message flow entirely. Typically however, keyloggers implemented in this manner will only read the keystroke data and forward the message to the next member in the chain.

Low-level kernel-mode keyloggers are typically implemented as *rootware* [10], a combination of both rootkits and spyware, that employ another variation of hooking. A rootkit is a small set of programs or tools that covertly run on an infected machine in order to provide long-term, undetected access to the root of a system for the attacker. Stealth is typically a high priority for a rootkit as it is intended to be a ”permanent” modification to the operating system kernel. Spyware is software that collects user data without the consent of the victim [11]. Using these two terms, rootware keyloggers are hidden software that hook into vital system routines to collect and transport user keystrokes without victim awareness or consent.

The kernel remains an ideal target for a rootkit to achieve its desired level of stealth and life time. This is because once a rootkit attains unrestricted access as a kernel component it can modify kernel memory, objects, and modules to mask its presence. For instance, a rootkit targeting a Windows operating system can be implemented as a device driver that can be dynamically loaded onto the system. From there, it can modify entries in the linked list of EPROCESS structures to hide running processes. Such a technique is an example of a Direct Kernel Object Modification (DKOM) [10].

Implementing keyloggers as device drivers is another common approach to gain privileged access to the kernel to intercept I/O data. Most modern operating systems permit device drivers to be chained or layered in a stack formation to dynamically provide additional functionality to a device or set of devices in a system. Using this approach, rootware developers can layer their drivers on top of the device driver stack and intercept I/O requests that pass between the keyboard device and kernel in order to extract keystroke data that maps to specific ASCII characters. This layered driver approach as implemented on a Windows operating system is depicted in Figure 1.

Once the extraction mechanism has been designed and

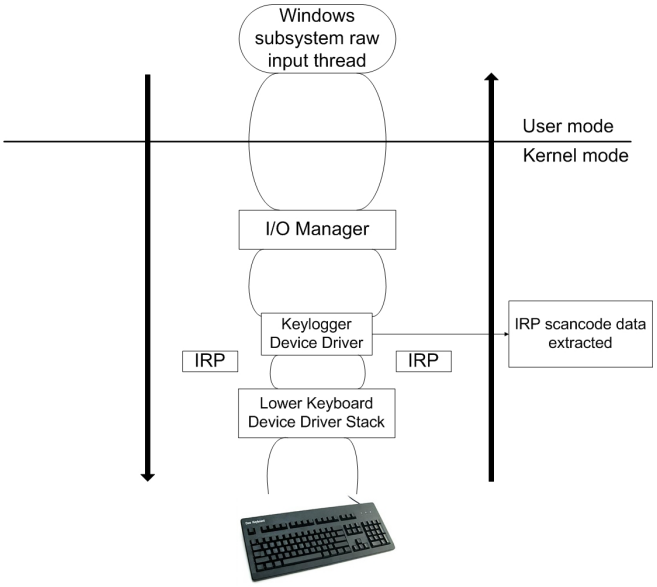

Figure 1: Layered device driver interception of I/O data.

implemented, the attacker needs to focus on data transportation. Rootware software will typically utilize covert channels, or concealed communication pathways, to break through firewalls undetected in order to send data across a network [9]. When designing a covert channel, the attacker must aim for a minimal footprint on the system and a unique structure [9]. A minimal footprint will allow communication to take place while effecting as few system components as possible, thus effectively making the channel harder to detect as its operation is less noticeable. A unique structure is also important when designing a covert channel because the greater its novelty, the less likely it is to be identified by common malware detection programs.

A kernel-level keylogger will need to use kernel sockets and networking routines to accomplish its networking goal. On the other hand, a user-level keylogger can avoid the complexity of kernel-level networking by implementing a covert channel using language supported networking capabilities or operating system routines. However, in terms of effectiveness and visibility, kernel-level channels are less susceptible to detection if implemented correctly and thus merit further study. Hoglund and Butler [9] provide additional information pertaining to the development of kernel-level covert channels.

4. Detection and Prevention

So far this paper explored keyloggers from a *black hat* viewpoint, that is, the design, implementation, and use of keyloggers. This section addresses a major goal of cybersecurity education, which is to train students in becoming *white hat* hackers, i.e., practitioners who can identify a security

weakness and help software system developers fix breaches before malware is able to take advantage of the system. A study of keylogger detection and prevention is thus critical for white hat hackers: detection focuses on identifying a keylogger that has already infected a system for it to be removed appropriately while prevention focuses on denying keyloggers any access to a system.

Malware detection is often viewed as being static or dynamic. Static detection involves signature-based pattern recognition while dynamic detection involves behavioral- and operational-based monitoring. Static detection requires malware detection software to monitor a system for recognizable malicious signatures or checksums. These signatures are essentially sequences of machine instructions that correspond to suspicious activity performed by a program on the host machine [12]. There are two significant but related problems with this technique: (a) the malware detection program needs to be constantly updated with new malware definitions and (b) no protection is provided against malware whose signature is not present in the repository. This is highly relevant to keylogging malware because they typically do not have a unique signature. Therefore, dynamic detection techniques must be employed to detect keylogging malware.

Behavioral-based detection techniques monitor the system for suspicious behavior that may be implemented by a keylogger, such as system file modifications or I/O data tampering. However, due to the differences in keylogger behavior and implementation techniques, existing solutions for dynamic detection have had mixed success. For example, Aslam et al. [13] describe an anti-hook shield that operates by flagging programs that hook system routines often targeted by keyloggers; this approach, however, also flags programs that hook the same system routines legitimately.

An interesting example of dynamic malware detection is the layered-architecture, behavior-based, malware detector proposed and evaluated by Martignoni et al. [14]. This model addresses the semantic gap between high-level behavior descriptions and their low-level computer representations, and succeeds largely due to the unique layered architectural approach to modeling the semantic gap via a hierarchical structure to translate high-level behaviors to low-level machine instructions. This modeling scheme allows this detector to input suspicious behavior mechanisms for use with a system-wide process execution monitor to flag suspicious activity if a process's activity closely matches the behavior specification.

Tainted data analysis is another useful detection mechanism that is specifically targeted towards kernel-level keyloggers. It has been observed that a majority of kernel-level keyloggers modify the normal flow of data of a keyboard driver or driver stack in order to extract and transmit keystroke data. Therefore, the extraction of user keystroke data occurs while data is being moved along the chain of keyboard device drivers in the kernel. The detection mechanism implemented by Le et al. [15] uses this observation by monitoring user keystroke data only as it moves along the chain of keyboard drivers. This is done by first tainting or marking user keystroke data when it comes to the base keyboard driver and monitoring it as it moves along the chain. If at any point a driver in that chain attempts to modify the data and pass it along, it will be marked. This allows any suspicious modification behavior to be flagged by the operating system and can help accurately identify the driver that performed the modification.

Many of the dynamic detection mechanisms being researched, implemented, and tested are still in the prototype stage, and therefore not included in commercial malware detection programs. The typical computer user has to rely on existing tools and anti-malware programs to help detect keyloggers on their machines. For instance, RootkitRevealer [16] is an advanced rootkit detection program that helps with detection, but finds it difficult to identify rootkits that hide their presence in the system by modifying privileged operating system data or memory. Other anti-malware programs such as Norton from Symantec and McAfee provide malware detection services that differ based on the level of support paid by the consumer. Most of these tools rely on signature-based detection, and struggle with detecting unique keyloggers. Therefore, a proactive approach is needed to stop keyloggers before they infect a system.

Efforts to prevent keyloggers or any type of malware lie in threat mitigation tools that detect and stop malware before the malware can impact its targets. Such tools include antivirus software, intrusion prevention systems, firewalls and routers, and even application settings [3]. Figure 2 depicts the layering of such tools in an attempt to protect the host machines from malware infection.

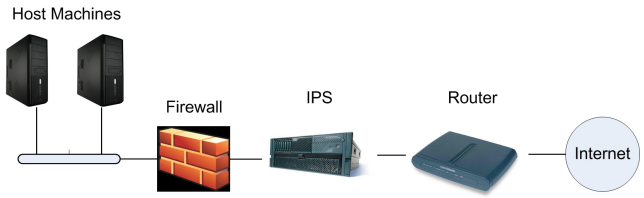

Figure 2: Layering of threat mitigation tools to prevent malware infection.

Antivirus software is perhaps the most commonly used form of malware prevention as it performs a wide array of mitigation tasks including, but not limited to, critical system component scanning, real-time activity monitoring for suspicious behavior, file scanning, and network filtering.

Intrusion prevention systems (IPS) come in network- and host-based forms depending on the medium used by the malware to be stopped. Network-based IPS perform packet sniffing and analyze network traffic to identify and stop suspicious activity at its root [3]. Typical network-based IPS

tend to be inline, thus effectively making them behave like a network firewall. The prevention techniques utilized by IPS products typically consist of a combination of attack signatures and analysis of network and application protocols, which essentially means that they scan network activity for previously observed malicious behavior to identify potentially dangerous malware. Similarly, host-based IPS products monitor host activities such as network traffic, system logs, running processes, and system and application configuration changes to prevent keylogger infection [3].

The outermost level of prevention involves the use of network firewalls and routers to permit or deny network traffic to a local machine based on a defined rule set. Routers typically offer less robust prevention capabilities than firewalls because they restrict access based on a broad set of rules. Tasks such as ingress and egress filtering are often performed by routers to help lighten the workload of firewalls that exist underneath the router [3]. Much like IPS products, firewalls come in both network- and host-based forms, depending on the usage context. Network-based firewalls are devices employed around the perimeter of a network to impede external threats, whereas a host-based firewall is software set up on a single host to monitor the network traffic of that same machine [3]. Network-based firewalls are designed using several different prevention mechanisms, such as deny by default rulesets, ingress and egress filtering, and network address translation. Host-based firewalls use similar rulesets when determining the validity of network traffic but also incorporate application-specific settings, antivirus software, and intrusion prevention mechanisms to help decrease infection.

Applications that operate using incoming or outgoing network traffic are potential gateways for infection, and need to be monitored to ensure that they prevent unauthorized machine access. If applications are configured to emphasize security over functionality, the likelihood of infection decreases substantially, but there are obvious limits on how much functionality can be eliminated. For instance, having an e-mail client block attachments that are of a certain extension (such as .bat) or filter spam messages greatly decreases the chance that an infection is spread through these media. Web browsing applications can also help impede malware infection by filtering web site content to legitimate data.

Finally, when keylogger detection and prevention techniques are inadequate in terms of capability or performance, user applications can simply avoid keyboard input by using alternatives for user input. A simple alternative is the use of automatic form fillers for web browsers; not needing to type in sensitive information each time a specific web page is visited reduces the leaking of private data. Another alternative is to use a different input mechanism; for example, audio input and appropriate speech recognition software could effectively foil a running keylogger.

5. Sample Keylogging Projects

This section proposes a set of projects to permit the hands-on design and development of both keylogging programs and anti-keylogging programs. These projects were originally developed as exercises in an independent study course taken by the first author, and are currently being adapted for use in courses such as Secure Coding taught by the second author. The outlined projects may also be useful in courses in Computer Security or Network Security.

A high-level outline of these projects is presented here, with additional details available in an online appendix at an authors-maintained website [17].

Project 0: Legal and Ethical Issues

Although issues of laws and ethics in the development, usage, detection and prevention of keyloggers are beyond the scope of this paper, these issues are a critical aspect of a programming course in keylogging [18]. So before students commence work on programming projects in keylogging, ensuring that students are unambiguously aware of applicable legal and ethical issues.

This non-programming project, therefore, provided students with the opportunity to explore legal and ethical issues and discuss them in class. Case studies from recent newspaper or online headlines will be used to drive this project. For example, keylogger programs and keylogger hardware can be legally developed in many jurisdictions across the world, and even sold legally; the usage of these programs, however, raises issues both of legality and ethics. For example, is it acceptable for a parent to monitor their children's activities online? On the other hand, Is it acceptable for an employer to monitor an employee's activities when at work or at home using employer equipment?

In this project, students will explore a recent example of keylogging exploit, write an essay on the legality and ethics involved in the exploit, and engage in a class discussion of these issues.

Project 1: Virtual Machines and Device Drivers

This programming project introduces students to virtual machines and device drivers to prepare them for rootkits and keyloggers. To utilize project time effectively, students will be assigned prior readings on virtual machines, appropriate debuggers and driver loaders, and the Microsoft Windows driver kits and build environments. Ideally the instructor in a formal course offering can substitute some activities via class demonstrations and lectures.

Student activities in this project include:

- Creation of a VMWare virtual machine.
- Inspection of the code and available documentation to understand the design and implementation details of the device drivers.
- Use keyboard input and a kernel-mode debugging tool.

Project 2: Introduction to Rootkits

This project focuses on helping students understand the basic operation of a keylogging rootkit via tracing and analyzing kernel-mode debug print statements. Based on our experience, the Klog rootkit by Clandestiny [19] seems appropriate for first time malware developers because it shows both how to implement a rootkit as a Windows device driver via detailed documentation and how to layer drivers on top of device driver stacks [9].

Student activities in this project include:

- Creation of a VMWare virtual machine.
- Inspection of code and available documentation to understand the rootkit design and implementation details.
- Building and loading the Klog rootkit onto a virtual machine.
- Testing the rootkit using keyboard input and a kernel-mode debugging tool.

Students will also appreciate the issues and complexity in keylogger implementation, and will begin to visualize potential keylogger detection and prevention mechanisms, and why these may potentially be hard to implement.

Project 3: Linux Keylogger Implementation

This project reinforces basic understanding of keylogger implementation by utilizing a partially implemented Linux keylogger and allowing students gain experience working with Linux I/O ports, process and memory management routines, and file I/O using the C programming language. More specifically, students explore the implementation of a modified but functional version of LKL GNU/Linux keylogger [20], understand Linux system routine calls necessary to perform keylogger-related actions, and load and test the keylogger.

Student activities in this project include:

- Exploring the LKL keylogger via a guided step-by-step walkthrough of the program.
- Filling in new code segments into the provided scaffolding structure as directed.
- Reflecting on their achieved understanding of this Linux keylogger.

In summary, this project introduces students to keylogger development on Linux. Although the Klog and LKL keyloggers share fundamental concepts, students learn that the implementation details are substantially different depending on the target machine and execution mode.

Project 4: Networked Keylogger

Keyloggers typically use a client-server model for network communication for some needed functionality. An attacker runs a remote server to accept incoming connections with the keylogger client that then takes the responsibility to transmit data to the server for persistent storage. In this project, students implement a server program that accepts remote data from a keylogger and writes this data locally to a file.

Student activities in this project include:

- Diagramming the client-server model needed in this project, depicting the relationships and operations of each module in the model.
- Building a server program by first understanding the basics of UNIX networking using C.
- Addition of code needed to the existing keylogger to establish a connection with the server and send data across this connection for storage.

In summary, this project introduces students to keylogger network communication, understand covert channel implementation, and help them gain invaluable insights into keylogger detection and prevention.

Project 5: Kernel-Level Covert Channel

A realistic keylogger will utilize a form of covert channel to send data across a network, whether it is through a new protocol or applying clever stealth techniques such as steganography. This project involves kernel development on Microsoft Windows operating systems to implement a basic covert channel by extending the Klog rootkit functionality to send data to a remote location for storage instead of the local host machine.

The goal is to get students thinking about kernel-level networking complexity and the abundance of communication possibilities. The approach that is used throughout the following exercises makes use of the Transport Driver Interface (TDI) in order to add networking functionality to the rootkit.

Student activities in this project include:

- Study of kernel-level networking techniques via assigned readings.
- Filling in new code segments to complete the partially implemented networking scaffold provided.
- Testing of the networked Klog rootkit using multiple clients and a server machine to gauge the effectiveness of the implemented covert channel, and document their findings.

In summary, this project makes students understand the networking complexity inherent in a real-world networked keylogger. Lightweight and stealthy networking techniques are difficult to implement depending on the target system. By allowing students to work with both user- and kernel-mode network communication they will gain a better understanding of the challenges faced by both the attackers and defenders when dealing with networked keyloggers.

Project 6: Detecting Keyloggers

This final project will focus on dynamic detection techniques that are aimed at detecting keyloggers installed on a system. A wide variety of detection mechanisms and techniques that are being researched, tested, and used in

actual anti-malware programs. Students will therefore be required to take a step back from programming projects and approach the problem of detecting keyloggers and similar malware from a higher-level using the experiences gained from the previous projects.

In this non-programming project, students will have the opportunity to explore the different detection mechanisms that are documented for keyloggers through a series of directed research-based literature surveys.

Student activities in this project include:

- Exploring a multitude of keylogger detection mechanisms.
- Critically analyzing a single detection mechanism and reporting on the findings.
- Presenting their findings in class.

In summary, this project permits students to build upon their malware development experiences and apply their knowledge to conduct a critical assessment of malware detection and prevention techniques.

The above collection of projects will provide students with the background needed to understand and deal with state-of-the-art keyloggers as they cover a wide range of related topics in keylogger design and implementation including legal and ethical issues, actual coding, and current practice in this area. These projects are also motivational because they provide a hands-on introduction to programs vital for software security. In short, a study of keyloggers is indeed a useful component of a modern cybersecurity education.

6. Final Remarks

This paper examined the current state of keyloggers and how they can play an invaluable role in cybersecurity education. We explored hardware and software keyloggers and examined techniques to defend against keyloggers. Finally, a set of programming projects for incorporating keyloggers in cybersecurity education was presented.

These projects are currently being re-worked for use in a course on Secure Coding to be offered at RIT in the fall of 2010. The final projects used in this course, and their efficacy will be reported in a future paper.

Acknowledgment

The first author was supported by internal grants from the Honors Program at the Rochester Institute of Technology. Projects described in this paper utilized the physical security laboratory maintained by the Department of Computer Science, as well as the virtual machine laboratory developed by the Department of Networking, Security, and System Administration, and operated by RIT's Information & Technology Services division. Thanks are also due to Minseok Kwon for providing feedback on earlier drafts of this paper.

References

[1] S. Sagiroglu and G. Canbek, "Keyloggers," *IEEE Technology and Society Magazine*, vol. 28, no. 3, pp. 10 –17, fall 2009.

[2] A. Emigh, "The crimeware landscape: Malware, phishing, identity theft and beyond," A Joint Report of the US Department of Homeland Security — SRI International Identity Theft Technology Council, the Anti-Phishing Working Group, October 2006.

[3] P. Mell, K. Kent, and J. Nusbaum, "Guide to malware incident prevention and handling," National Institute of Standards and Technology, Gaithersburg, MD, Tech. Rep. 800-83, November 2005.

[4] ThinkGeek.com, "Spy keylogger," 2010 (accessed May 8, 2010), http://www.thinkgeek.com/gadgets/security/c49f/.

[5] T. Olzak, "Keystroke logging (keylogging)," Adventures in Security, April 2008 (accessed May 8, 2010), http://adventuresinsecurity.com/images/Keystroke_Logging.pdf.

[6] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu, "The ghost in the browser analysis of web-based malware," in *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*. Berkeley, CA, USA: USENIX Association, 2007, pp. 4–4.

[7] T. Thornburgh, "Social engineering: the 'dark art'," in *InfoSecCD '04: Proceedings of the 1st annual conference on Information security curriculum development*. Kennesaw, Georgia: ACM, 2004.

[8] S. Shah, "Browser exploits - attacks and defense," London, 2008 (accessed May 8, 2010), http://eusecwest.com/esw08/esw08-shah.pdf.

[9] G. Hoglund and J. Butler, *Rootkits: Subverting the Windows Kernel*. Addison-Wesley Professional, 2005.

[10] J. Butler, B. Arbaugh, and N. Petroni, "R^2: The exponential growth of rootkit techniques," in *BlackHat USA 2006*, 2006 (accessed May 8, 2010), http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Butler.pdf.

[11] Symantec Corporation, "Viruses and risks," April 2010, http://www.symantec.com/norton/security_response/index.jsp.

[12] M. Baig and W. Mahmood, "A robust technique of anti key-logging using key-logging mechanism," in *IEEE-IES Digital EcoSystems and Technologies Conference, 2007*, February 2007, pp. 314 –318.

[13] M. Aslam, R. N. Idrees, M. M. Baig, and M. A. Arshad, "Anti-hook shield against the software key loggers," in *Proceedings of the National Conference of Emerging Technologies*, 2004.

[14] L. Martignoni, E. Stinson, M. Fredrikson, S. Jha, and J. C. Mitchell, "A layered architecture for detecting malicious behaviors," in *RAID '08: Proceedings of the 11th international symposium on Recent Advances in Intrusion Detection*. Heidelberg: Springer-Verlag, 2008.

[15] D. Le, C. Yue, T. Smart, and H. Wang, "Detecting kernel level keyloggers through dynamic taint analysis," College of William & Mary, Department of Computer Science, Williamsburg, VA, Tech. Rep. WM-CS-2008-05, May 2008.

[16] B. Cogswell and M. Russinovich, "Rootkitrevealer v1.71," 2006 (accessed May 8, 2010), http://technet.microsoft.com/en-us/sysinternals/bb897445.aspx.

[17] C. Wood and R. K. Raj, "Sample keylogging programming projects," 2010 (accessed May 8, 2010), http://www.cs.rit.edu/~rkr/keylogger2010.

[18] B. Whitty, "The ethics of key loggers," Article on Technibble.com, June 2007 (accessed May 8, 2010), http://www.technibble.com/the-ethics-of-key-loggers/.

[19] J. Todd, "Clandestine file system driver," 2005 (accessed May 8, 2010), http://www.rootkit.com/newsread.php?newsid=386.

[20] LKL, "Linux keylogger," 2010 (accessed May 8, 2010), http://sourceforge.net/projects/lkl/.