SMMDecoy: Detecting GPU Keyloggers using Security by Deception Techniques

Ijlal Loutfi
Informatics Department, University of Oslo, Gaustadalleen 23B, Oslo, Norway
ijlall@uio.no

Keywords: Keylogger, System Management Mode, Deception, Detection.

Abstract: Human computer interaction is a fundamental part of the modern computing experience. Everyday, millions of users rely on keyboards as their primary input interface, and use them to enter security sensitive information such authentication credentials. These can be passwords, but also multi-authentication factors received from other devices, such as One Time Passwords and SMS's. Therefore, the security of the keyboard interface is critical. Unfortunately, both PS/2 and USB keyboards have open buffers which are vulnerable to sniffing by keyloggers. This paper focuses on the detection of the stealthiest variance of keyloggers, which is deployed within IO devices firmware, such as GPUs. We propose to use principles of security by deception: We inject decoy credentials into the open keyboard buffers, and give GPU keyloggers the opportunity to sniff them. These decoy credentials are then sent to a remote server that can raise an alarm anytime an attacker uses them. We assume a strong adversary that can infect both the GPU and the kernel. Therefore, we propose to deploy the solution within System Management Mode, and leverage Intel Software Guard Extensions for network communication. Both SMM and SGX are hardware protected against the OS and DMA, and provide thus strong security guarantees to our solution, which we name SMMDecoy.

1 INTRODUCTION

Despite recent advances in user authentication, the most commonly deployed mechanism in the internet today is still passwords. Everyday, millions of users rely on their username/password credentials to gain access to security-sensitive digital services. However, password-based authentication is routinely compromised. The main attack vectors can be classified in three categories: 1) the server side where password files are stored, 2) users' bad cognitive habits of choosing weak and redundant passwords or falling prey to social engineering 3) and finally, vulnerabilities within end-user devices where passwords are stored and operated on (Herley, 2009) (Florencio and Herley, 2007).

As a matter of fact, over the past decade, we have witnessed many password disclosures of high profile companies such as Sony, Yahoo, and LinkedIn. Analysis of these breaches revealed that many of these online service providers implement bad security practices, such as using unsalted hashes, and weak deprecated hashing algorithms. The silver lining of this slew of high profile password compromises, is the increased awareness about the issues, which has prompted many companies to review and harden their password servers' security. Consequently, this has been increasing the cost of attack of database servers, and thus, turning the attention of attackers to endpoint devices as a more attractive attack vector (Thomas et al., 2017) (Holz et al., 2009).

Furthermore, the adoption of multi-factor authentication by many online service providers has further motivated attackers to focus on endpoints. For example, many banks distribute One Time Password generators (OTP) to their users, while others rely on out-of-band methods such as SMS to send second factor authenticators. However, users still need to input these second factors into the same endpoint device where the password is entered. This mechanism consolidates the whole authentication process into one attack vector, aka the endpoint device, where they can be exposed to a host of attacks. One such critical endpoint attack vector is the keyboard device. Since users almost exclusively use it as an input channel, it is of uttermost importance that it preserves the confidentiality and integrity of users' data (Jacob, 1996).

1.1 Problem Statement

Unfortunately, keyboards suffer from a big sniffing problem. This is due to their hardware interface, which is open for direct reading by privileged software as well as DMA capable devices. This is true for both PS/2 and USB keyboards (Ortolani and Crispo, 2012).

- for PS/2 keyboard, the data output buffer which resides on the chipset keyboard controller and is used to transfer scan codes upon a key press/release, is exposed to the platform host (Zhang et al., 2015b).
- For USB keyboards, the data transaction buffers which are memory mapped are also exposed to the host(Ladakis et al., 2013).

One critical malware category that takes advantage of this insecure keyboard interface is keyboard sniffers, more commonly referred to as keyloggers. Once they infect a platform, keyloggers log keyboard activity, and leak data to remote third parties. As documented in numerous disclosed attacks, keyloggers pose a serious threat against personal and financial data, despite continuous efforts to guard against them. For instance, keyloggers have been used to steal 10775 unique bank account credentials for customers who have shopping at Barnes and Noble stores over a period of 7 months (Schmidt, 2012) . Even governmental agencies have relied for years on hard disk based keylogges, as documents by recent leaks (Thomson, 2013) .While keyloggers can be implemented on either hardware or software, we focus in this paper on the latter, as it is much more widespread and assumes a stronger attacker. Software keyloggers can be classified under three categories: (Ortolani and Crispo, 2012):

- User-level Keyloggers: They often hook into application-level API, but can also reliably detected, through more privileged system level hook-based techniques.
- Kernel-level Keyloggers: they hook into kernel-level API, and by doing so, inadvertently modify the kernel's code base, and thus its signature. Therefore, they can often be detected based on integrity verification and code attestation techniques.
- Firmware Level Keylogger: They can exploit firmware level vulnerabilities within the BIOS or any IO device, especially ones which are open to general computations such as modern GPUs. Therefore, they live outside of the CPU execution environment, and can evade its detection mechanisms. They can rely on the IO device DMA capabilities to directly read and sniff the keyboard's data registers.

In this paper we focus on firmware level keyloggers, and formulate our research questions as follows:

How can we reliably detect the presence of stealthy GPU keyloggers on endpoint devices? This research question is important the following reasons:

- The increased complexity of devices functionality, which is correlated with an increased complexity of its corresponding firmware, has made the firmware attack vector significantly larger.
- Many firmware malware categories, such keyloggers, do not need to hook into any kernel API or structures. Therefore, system level detection mechanisms such as code integrity and control flow integrity are inefficient against it.
- CPU-based monitoring solutions, e.g. antivirus systems, cannot monitor code residing on other execution environments within other devices.
- The increasing ease of deploying firmware Malware. This is especially true for devices such as GPUs, which have become open to general-purpose computations. In fact, GPUs have been traditionally used to process graphics rendering code, relieving the CPU this way from these heavy computations. However, the popularity of the gaming and AI industries, and their increasing demand for more GPU computational power and functionality, has made GPU general purpose computing much more extensive. Attacker are naturally interested in exploiting this large attack vector. Furthermore, 99 percent of worldwide GPUs support GPGPU computations, which greatly increases the infection ratio of GPU malware. This is different from previous firmware attacks, which had to be more targeted, and thus limited to smaller infection ratios (Ladakis et al., 2013).

*For the remainder of this paper, we will use GPUs as an example for devices' firmware level software.

The solution proposed in this paper is inspired by deception-based techniques, which are traditionally used within the server side in order to harden the detection of password database files breaches. An example of such solutions are honeywords, where each user is associated with one legitimate password, and n fake ones. This directly increases the effort required by attackers to brute force passwords. Furthermore, in case a fake one is used, it automatically triggers an alarm signalling a potential breach (Wang et al., 2018).

Similarly, this paper's main intuition is that we need to deploy a transparent mechanism which can

inject intentionally crafted noise, which mimics authentication credentials, to the keyboard's buffer, and allow any potential GPU keyloggers to monitor and sniff it. SMMDecoy would subsequently send a list of injected decoy credentials to a remote third party. We propose to deploy such a solution within System management mode, SMM, in order to take advantage of its integrity and transparency guarantees. We also propose to use Intel Software Guard Extension remote attestation capabilities to send decoy credentials over the internet. We call our solution, SMMDecoy. The rest of the paper is organized as follows: section 2 presents the necessary background. We then introduce the threat model, the solution design, and its message flow. An overview of previous related work is presented next. The paper closes with a set of conclusions as well as an overview of open questions and suggested future work.

2 BACKGROUND

2.1 Keyboards Hardware Interface

Keyboards are such a prolific part of everybody's computing experience. They are also a critical security component, since they are used as the main input channel on many types of endpoint devices and are relied upon to communicate security sensitive information to the system software and eventually to our trusted online service providers. Examples of such information are authentication credentials. There are several ways of connecting a keyboard to an x86 Intel platform: they can be either wireless or wire-base. For the latter, we can further classify the connections as either PS/2 or USB based. While the latter rely on a serialized protocol, their interface to the host is different:

- The PS/2 Keyboard interface is composed of keyboard processor which resides inside the keyboard itself, and a keyboard controller which is part of the host chipset. The interface of the keyboard controller has 2 pairs of output buffers in the direction of the CPU, and 1 pair in the direction of the keyboard. The output buffer is used to transfer the scan code if a key is pressed. It is readable by any software and causes keyboard sniffing problem (Brouwer, 2009).
- USB based keyboard don't have a chipset-resident keyboard controller. They have instead a host controller and a root hub. The host controller is represented by a set of buffers and structures, which are mapped to main memory, and are accessible through a set of system registers, also creating a snigging vulnerability (Brouwer, 2009).

2.2 Scan Codes

"A Scan Code is a data packet that represents the state of a key. If a key is pressed, released, or held down, a scan code is sent to the computers onboard keyboard controller. There are two types of scan codes: Make Codes and Break Codes", which are used for the events of key press or release. For every keyboard key, there exists a unique make code and break code. When a user presses a key on the keyboard, a scan key is sent to the keyboard chipset-resident controller, which then buffers it on the output data buffer. It then raises the Interrupt request line, which will cause the IRQ 1 to be fired if it is not masked. When the interrupt is scheduled, the corresponding keyboard handler reads the output buffer and converts the scan codes into their corresponding key value. It is important to note that the scan codes within the output buffer can be read by software, even outside the interrupt handler procedure, and this is the crux of the keyboard sniffing challenge (Mike, 2009) (Brouwer, 2009) .

2.3 System Management Mode

System Management Mode, SMM, is a highly privileged x86 CPU mode. SMM code is part of the BIOS code that resides on the SPI flash memory. During the system boot up and before the operating system is loaded, the BIOS loads SMM into a hardware protected memory area referred to as SMRAM, and which is not addressable from any other CPU mode, including kernel and VMX modes (Zhang et al., 2015a). SMM implements a number of SMI handlers, which traditionally handle system control functions, such as power management. In order to execute SMI handlers, an SMM pin should be asserted, which will then trigger an SMM interrupt. Before the system switches to SMM mode, the CPU state is securely saved into SMRAM, so that it can return to it upon exiting SMM. This makes SMM highly transparent to all privileged system level software. This feature has been recently motivating many novel ways of using SMM for non-traditional purposes, e.g. debugging and system introspection (Delgado and Karavanic, 2018).

2.4 Software Guard Extensions

Intel's SGX are security extensions which are come as part newer Intel X86 CPUs. Its main aim is to instantiate an isolated trusted execution environment within

the user space, called an enclave. Enclave code and data reside in specialized protected memory called enclave page cache (EPC), which encrypted, and hardware protected. No privileged mode code can access the enclave, including the OS and hypervisor. SGX enclaves also rely on the intel management engine EPID group identity to establish a remote attestation protocol with Intel attestation servers, and through it to third party service providers (Van Bulck et al., 2017).

3 SOLUTION OVERVIEW

3.1 Threat Model

SMMDecoy assumes an active attacker who has unlimited computing resources and can exploit zero-day vulnerabilities of the host OS and user level applications. It also assumes that the GPU is compromised, and so are all other I/O devices, with the exception of the keyboard. Therefore, the only trusted components of the system, are the BIOS and the keyboard. SMMDecoy requires BIOS to be trusted only upon boot up, and not during runtime. We also assume that the attacker does not have physical access to the machine. We do not consider Denial-of-Service (DoS). The BIOS is trusted because newer X86 platforms are equipped with a Static Root of Trust of Measurement, SRTM, with a corresponding secure implementation of a Core Root of Trust of Measurement, which can ensure the code integrity of the BIOS upon boot. Some solutions such as HP SureStart also ensure that BIOS recovery as well, in case a code integrity compromise is detected.

3.2 High Level Architecture

The intuition behind SMMDecoy is to inject specially crafted noise which mimics genuine authentication credentials into the keyboard output buffers. We assume that potential firmware keyloggers will be monitoring the buffers. We will then communicate these decoy credentials to relevant remote third parties. If we detect any authentication attempt using any of the reported decoy credentials, an alarm should be raised. Such a detection mechanism would subsequently provide a wealth of information about how the malware attack space, and the platforms from which it spreads. Thus, the requirements of such a solution are as follows:

1. SMMDecoy should be implemented within a system component that would allow it to be transparent to both the OS kernel and to the GPU malware.
2. The decoy authentication generation algorithm should mimic real world passwords as much as possible.
3. SMMDecoy should provide a mechanism for communicating with third party remote servers securely.

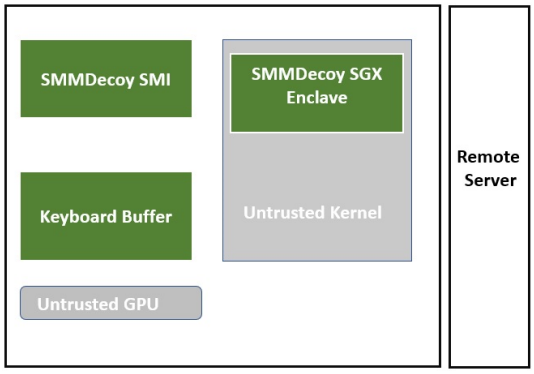

Figure 1: SMMDecoy Architecture: Trusted Components in Green.

As figure 1 illustrates, SMMDecoy is made up of two parts:

- SMMDecoy SMI handler :it is the trusted part, and is used to generate decoy credentials, inject them into the buffer interface, and then report them to a remote server.
- SMMDecoy SGX enclave; It used to establish an end-to-end secure channel between SMM and a remote server, which will be responsible for raising alarms when a decoy credential is used by attackers.

3.3 SMMDecoy Message Flow

At a conceptual level, SMMDecoy adopts the same architecture for both PS/2 and USB keyboards. However, their interfaces are different and so are their implementation details. We present SMMDecoy for each keyboard separately.

Step 0: This is a common step for both implementations, and it takes place before the deployment of the solution. We need to customize the BIOS firmware, and add to it the SMMDecoy SMI interrupt. If this solution is deployed within an enterprise environment, this can be done as part of the platform provisioning by the IT department. Upon system boot up, SMMDecoy SMI will be loaded securely into SMRAM. From this point on, SMMDecoy message flow will diverge depending on the keyboard connection, PS/2 or USB, which can be detected upon boot up.

3.3.1 PS/2 Message Flow

- In step 1, SMMDecoy SMI generates fake user authentication credentials, which mimic legitimate credentials (more details in section 4.5), and converts them into scan codes.
- In step 2, SMMDecoy SMI is triggered based on A timer. The system then enters SMM modes, saves the CPU system state, as well the keyboard buffer content into SMRAM. o SMIDecoy then sends a 0xD2 command into the keyboard control register, whose address is 0X64. This command allows anything that is subsequently written into the output buffer to appear as if it was generated by the keyboard.
- step 3, SMMDecoy decoy injects the scan code corresponding to the generated Decoy credentials into the keyboard data output buffer, by writing into address 0x60.
- In step 4, and after enough time elapsed to allow any potential firmware keylogger to sniff the keyboard output buffer, SMMDecoy SMI restores the state of the keyboard buffer, and exits SMM mode. This restores the CPU state, and gives back control to the OS so that it can resume its normal execution.
- In Step 5, SMMDecoy periodically communicates the list of decoy credential used to a remote server. We differ the details of this step to section 3.7.

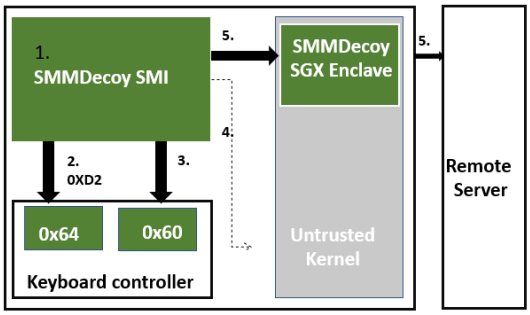

Figure 2: PS/2 SMMDecoy Message Flow.

3.4 USB Keyboard

- In step 1, SMMDecoy searches the system memory in order to find the memory address for the system keyboard buffer. In Linux, an attached USB device is represented by a USB Request Block (URB) structure, defined in the linux/usb.h header file of the Linux source tree. The keyboard buffer is part of this URB structure. The SMI then saves the physical address so as it can properly access it (Ladakis et al., 2013).
- In step 2, SMMDecoy SMI generates decoy credentials, which mimic legitimate credentials (more details in section 3.6), and converts them into scan codes. SMMDecoy SMI injects the scan codes corresponding to the generated Decoy credentials into kernel keyboard bugger, which it addresses using its physical address.
- In step 3, and after enough time elapsed to allow any potential firmware keylogger to sniff the keyboard output buffer, SMMDecoy SMI restores the state of the keyboard buffer, and exits SMM mode. This restores the CPU state, and gives back control to the OS so that it can resume its normal execution.
- In Step 4, SMMDecoy periodically communicates the list of decoy credential used to a remote server. We differ the details of this step to section 3.7.

3.5 Writing into the Buffer

For both the PS/2 and USB keyboards, we mentioned that the SMMDecoy SMI injects decoy credentials into either the output buffer of the system buffer. However, this is an abstraction, since the buffer size is limited and would require multiple coordinated writes. For instance, the Linux keyboard buffer has 16 bytes, and each scan code is 3 bytes long. Therefore, SMIDecoy is expected to perform multiple consecutive writes into the buffer (Mike, 2009).

3.6 Generating Fake Credentials

The injected decoy credentials need to be statistically indistinguishable from genuine credentials. This can be achieved by encoding password generation policies into the SMMDecoy SMI handler, such as using a combination of characters and numbers, and having a minimum password length. Furthermore, we can dynamically update this algorithm with contextual user specific data, that the SMMDecoy handler would collect from the system. Such as other passwords used by the user, his name/ID. . . etc

3.7 Establishing a Secure Channel between SGX Enclave and SMM

Deception techniques are useful only if we are able to detect the decoy credentials being used or leaked by potential malware at some point in time. For SMMDecoy, this can happen at two points:

- Local detection: SMM can choose well-crafted patterns for the decoy credentials it injects. Therefore, it could also intercept all outgoing networks

packets and look for the same pattern. Such solutions have been previously explored in the literature and are not the focus on this paper (Ortolani et al., 2010)

- Remote detection: SMMDecoy can send the injected decoy credentials to a remote server, which might be the service provider whose credentials we have been injecting into the keyboard buffer. While the actions the remote server takes upon detection of a used decoy credential are outside of the scope of the paper, we discuss a number of options here for the same of completeness. In fact, decoy credentials can be used to augment an already existing honeyword implementation. In this case, SMMDecoy will increase the probability of the attacker choosing a decoy honeyword to authenticate to the server provider. Furthermore, unlike a traditional honeyword which would only signal the existence of a breach, SMMDecoy reveals a wealth of information about the malware attack vector and the platforms from which it is spreads. Decoy credentials could also be used as a standalone honeyword where decoy accounts are provisioned in order to allow the attacker to log into them, and leave traces of their attack details, such as the amount of money they transfer.

We have considered two approaches to achieving the remote detection:

- Porting trusted network drivers into SMM.
- Relying on Intel SGX remote attestation.

While SMMDecoy proposes to use intel SGX, we discuss both approaches subsequently for completeness.

3.7.1 SMM Trusted Network Drivers

If SMMDecoy wants to send data over the network, it needs to make use of the network drivers which are part of OS. However, the latter is assumed to be malicious within our threat model. This question has been a common challenge for many SMM based solution. One way it has been solved is by porting commodity drivers into SMRAM. This has been possible because SMM mode is similar to kernel mode where privileged CPU instructions are available. Aurora authors also argue that the mechanism of interrupt rerouting helps SMM driver design concentrate on the interrupt handling rather than device initialization or resource management, making it thus faster (Liang et al., 2018).

3.7.2 Intel SGX Remote Attestation

In this approach, we propose to keep the SMMDecoy SMI simple, and rely on the remote attestation capabilities of Intel SGX to communicate with a remote server. This approach also respects the threat model, since Intel SGX enclaves are hardware protected from the operating system.

Provisioning. Key provisioning happens once, and it involves the following steps:

1. Authenticating SMMDecoy to Intel Remote Server.
   - During start-up, the BIOS uses the Intel remote server PKI to establish a secure channel with it.
   - The BIOS computes a token on the SMMDecoy to be loaded into SMRAM and sends its hash signature to verify its integrity and prove its identity.
2. Authenticating SGX enclave to Intel Remote Server, using intel SGX remote attestation
3. The enclave generates a symmetric secret key K which it securely forwards to the IAS, which securely forwards the key to the SMMDecoy SMI on the same platform as the SMMEnclave.

At this point, a unique session has been successfully established.

Communication. Once a shared secret key is established between SMMDecoy enclave and SMI, the interrupt is ready to secretly send decoy credentials to the enclave. The enclave then engages in a standard remote attestation protocol and establishes a secure channel with the remote server.

3.7.3 Proposed Implementation Details

We propose to use Coreboot as a BIOS distribution to implement SMMDecoy on. Coreboot is "an extended firmware platform that delivers a lightning fast and secure boot experience on modern computers and embedded systems. As an Open Source project, it provides auditability and maximum control over technology"(Zhang, 2013). This is important for us to be able to implement and deploy the custom SMMDecoy Interrupt handler into the platform.

4 RELATED WORK

In this section we discuss three lines of related work: GPU malware detection, SMM based systems and security by deception.

In order to detect stealthy GPU malware, prior work suggested monitoring the side effects the malware generates, as CPU solutions are unable to access

and scan the GPU. However, these measurements are only reliable in the case of malware which performs bulk DMA transfers, which is not the case for GPU keyloggers. Other work suggests using the cuda-gdb real time debugging capabilities in order to monitor the GPU's access patterns. However, GPU malware could remote debug points from its code base (Embleton et al., 2008).

SMM has been traditionally used to secure the execution of platform management functions such as power and hear control. However, it has been increasingly used to deploy security sensitive solutions, which require strong hardware access control guarantees. Such systems are HyperCheck that is used for hypervisor integrity verification and IOCheck and SMMDumper that scans system memory and dumps it for forensic analysis (Reina et al., 2012) (Zhang, 2013). Aurora leverages SMM to provide intel SGX enclaves enclaves with trusted network and time services, by porting their corresponding drivers into SMM (Liang et al., 2018). Researchers have also been long aware of the keyboard sniffing problem. TrustLogin proposes a solution to prevent credentials leakage while they are from the keyboard to the Network Interface Card, NICl. It uses SMM to encrypt the credentials, and forward them securely to the NIC. (Zhang et al., 2015b).

Finally, Deception and decoy has always been part of the defence arsenal of cybersecurity. The most widely discussed deception-based solution is arguably honeypots. The intuition behind honeypots is to "provide fake information which is attractive to attackers. The attacker, in searching for the honey of interest comes across the honeypot, and starts to taste of its wares. If they are appealing enough, the attacker spends considerable time and effort getting at the honey provided. If the attacker has finite resources, the time spent going after the honeypot is time not spent going after other things the honeypot is intended to protect. If the attacker uses tools and techniques in attacking the honeypot, some aspects of those tools and techniques are revealed to the defender in the attack on the honeypot" (Cohen, 2004). The earliest and most notable use of honeypots was in 1991 from ATT researcher in a paper called "jail", which aims to lure attacks in order to monitor their behaviour. Since that time, deception has increasingly been explored as a key technology area for innovation in information protection (Wang et al., 2018).The idea of honeypots has been further explored more recently, and applied to detect authentication into financial institutions, by creating an account which mimics a real account through all its attributes, minus the fact that it isn't backed with any money which can actually be stolen. When an attacker gains access to such accounts, it will be indistinguishable to them from any other real account, as they will have access to the same services, except the fact that the bank will not be validating any money transfers linked to the faked account. This constitutes a very efficient solution not only for the detection of account breaches, but also a great opportunity to learn about the behaviour, strategy and intention of attackers (Juels and Rivest, 2013).

4.1 Conclusions and Future Work

In this paper, we presented SMMDecoy, a deception-based technique to detect GPU keyloggers, which sniff the open keyboard interface. We protect against a strong adversary which can take control over the platform's user applications, kernel, and GPU. SMMDecoy generates and injects decoy credentials, which should indistinguishable from legitimate ones. They are then sniffed by the GPU malware. SMMDecoy relies on strong hardware enabled access control mechanisms. It uses SMM to protect the integrity and transparency of the decoy credentials' injection. It also uses SGX to establish a secure channel to a remote server, over which the injected decoy credentials would be forwarded. If a decoy credential is detected to be used by a malware, an alarm should be raised. Unlike traditional honeywords, an SMMDecoy alarm reveals a wealth of information about how the malware spreads. The paper also discusses the implementation feasibility of SMMDecoy. Future work is to naturally implement the solution and evaluate its performance. We also plan to use SMMDecoy as part of a longitude study in which we aim at detecting GPU, and other firmware, malware which is not possible to detect using traditional CPU based mechanisms.

ACKNOWLEDGEMENTS

The author would like to thank all of the reviewers. This work is supported by the department of informatics of the university of Oslo, and COINS Research School of Computer and Information Security.

REFERENCES

Brouwer, A. (2009). The at keyboard controller.

Cohen, F. (2004). The use of deception techniques : Honeypots and decoys.

Delgado, B. and Karavanic, K. L. (2018). EPA-RIMM: A framework for dynamic smm-based runtime integrity measurement. *CoRR*, abs/1805.03755.

Embleton, S., Sparks, S., and Zou, C. (2008). Smm rootkits: A new breed of os independent malware. In *Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks*, SecureComm '08, pages 11:1–11:12, New York, NY, USA. ACM.

Florencio, D. and Herley, C. (2007). A large-scale study of web password habits. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 657–666, New York, NY, USA. ACM.

Herley, C. (2009). So long, and no thanks for the externalities: The rational rejection of security advice by users. In *Proceedings of the 2009 Workshop on New Security Paradigms Workshop*, NSPW '09, pages 133–144, New York, NY, USA. ACM.

Holz, T., Engelberth, M., and Freiling, F. (2009). Learning more about the underground economy: A case-study of keyloggers and dropzones. In *Proceedings of the 14th European Conference on Research in Computer Security*, ESORICS'09, pages 1–18, Berlin, Heidelberg. Springer-Verlag.

Jacob, R. J. K. (1996). Human-computer interaction: Input devices. *ACM Comput. Surv.*, 28(1):177–179.

Juels, A. and Rivest, R. L. (2013). Honeywords: Making password-cracking detectable. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer ; Communications Security*, CCS '13, pages 145–160, New York, NY, USA. ACM.

Ladakis, E., Koromilas, L., Vasiliadis, G., Polychronakis, M., and Ioannidis, S. (2013). You can type , but you can ' t hide : A stealthy gpu-based keylogger.

Liang, H., Li, M., Zhang, Q., Yu, Y., Jiang, L., and Chen, Y. (2018). Aurora: Providing trusted system services for enclaves on an untrusted system. *CoRR*, abs/1802.03530.

Mike (2009). Operating systems development - keyboard.

Ortolani, S. and Crispo, B. (2012). Noisykey: Tolerating keyloggers via keystrokes hiding. In *Proceedings of the 7th USENIX Conference on Hot Topics in Security*, HotSec'12, pages 2–2, Berkeley, CA, USA. USENIX Association.

Ortolani, S., Giuffrida, C., and Crispo, B. (2010). Bait your hook: A novel detection technique for keyloggers. In Jha, S., Sommer, R., and Kreibich, C., editors, *Recent Advances in Intrusion Detection*, pages 198–217, Berlin, Heidelberg. Springer Berlin Heidelberg.

Reina, A., Fattori, A., Pagani, F., Cavallaro, L., and Bruschi, D. (2012). When hardware meets software: A bulletproof solution to forensic memory acquisition. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC '12, pages 79–88, New York, NY, USA. ACM.

Schmidt, M. S. (2012). Credit card data breach at barnes noble stores.

Thomas, K., Li, F., Zand, A., Barrett, J., Ranieri, J., Invernizzi, L., Markov, Y., Comanescu, O., Eranti, V., Moscicki, A., Margolis, D., Paxson, V., and Bursztein, E., editors (2017). *Data breaches, phishing, or malware? Understanding the risks of stolen credentials*.

Thomson, I. (2013). How the nsa hacks pcs, phones, routers, hard disks 'at speed of light': Spy tech catalog leaks.

Van Bulck, J., Piessens, F., and Strackx, R. (2017). Sgx-step: A practical attack framework for precise enclave execution control. In *Proceedings of the 2Nd Workshop on System Software for Trusted Execution*, SysTEX'17, pages 4:1–4:6, New York, NY, USA. ACM.

Wang, D., Cheng, H., Wang, P., Yan, J., and Huang, X. (2018). A security analysis of honeywords. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*.

Zhang, F. (2013). Iocheck: A framework to enhance the security of i/o devices at runtime. In *2013 43rd Annual IEEE/IFIP Conference on Dependable Systems and Networks Workshop (DSN-W)*, volume 00, pages 1–4.

Zhang, F., Leach, K., Stavrou, A., Wang, H., and Sun, K. (2015a). Using hardware features for increased debugging transparency. In *2015 IEEE Symposium on Security and Privacy (SP)*, volume 00, pages 55–69.

Zhang, F., Leach, K., Wang, H., and Stavrou, A. (2015b). Trustlogin: Securing password-login on commodity operating systems. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, ASIA CCS '15, pages 333–344, New York, NY, USA. ACM.