Hardware-based Cyber Threats

Thiago Alves and Thomas Morris
Electrical and Computer Engineering, The University of Alabama in Huntsville, Huntsville, U.S.A.

Keywords: Cyber-security, Embedded Systems, Computer Hardware Security.

Abstract: During the last decade, cyber-security experts have been trying to mitigate attacks against computer networks and software. After the internet, the proliferation of thousands of virus, worms and trojans became easier, which then required enhancements for Operating Systems, browsers and anti-virus software in order to keep their users safe. However, what happens when the threat comes from the hardware? The Operating System trusts entirely in the hardware to perform its operations. If the hardware has been taken, it becomes much harder to regain control of the system. This paper describes eight different approaches to hardware attacks against software. It also demonstrates how to perform an attack using a USB device patched to behave like a generic HID Input Device, in order to insert malicious code in the system.

1 INTRODUCTION

The first documented virus was written by Bob Thomas at BBN Technologies in 1971. The virus, known as Creeper Virus, was a self-replicating program which propelled itself between nodes of the ARPANET (Robert, 2004). Since then, the internet has created a fertile environment for worms and viruses that replicate themselves, potentially causing harm to the system and its users.

In the effort of mitigating these threats, many software packages were created focusing in network defense and files contamination scanning. Defense programs received significant improvements during the last years, with the inclusion of different types of Intrusion Detection System (IDS) and other tools that can prevent the attack from happening.

However, these software packages focus mostly in defending against outside attacks. In a continuous effort to become more persistent and avoid detection, malware infection is now shifting from software towards more low-level components. Internal malicious code can possible go undetected, because the monitoring software rely on the Operating System for data gathering, and once the machine's hardware becomes compromised, it becomes easy to subvert the Operating System to deliver deceived data. There is a catastrophic loss of security when hardware is not trustworthy.

The implication of malware written for hardware is that they are strictly dependent on the device architecture. Different devices may require major modifications in the code.

2 HARDWARE MALWARE APPROACHES

This section describes eight different approaches to subvert the system through hardware attacks. Although these are just some examples of different attack vectors, they all seem to be out of the scope of most, if not every, protection software available.

2.1 A Chipset Level Backdoor

The chipset is a set of specialized electronic components that manage data flow on a computer's motherboard or on an expansion card (Sparks, 2009). Sparks et al. described a proof of concept chipset rootkit backdoor for the Intel 8255x Ethernet Controller. Since it lies in the Ethernet Controller chipset driver, it can send and receive malicious network packets without being identified by the security software installed on the host computer.

The backdoor can remain invisible because it resides below the Operating System's network interfaces. The two primary components of the network subsystem on Microsoft Windows are TDI (Transport Driver Interface) and NDIS (Network Driver Interface Specification). Both components are

Alves, T. and Morris, T.
Hardware-based Cyber Threats.
DOI: 10.5220/0006577202590266
In *Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP 2018)*, pages 259-266
ISBN: 978-989-758-282-0

also the most common target for malware and security software authors since they are the deepest layers in the Operating System for sending and receiving network packets. The rule is that the deepest one goes, the greater its power and stealth. Therefore, a malware that goes deeper and can interface directly with the hardware is capable of bypassing any malicious code detection running at an abstracted level above it.

The proposed rootkit backdoor is a modified Windows kernel driver for the said Intel 8255x Ethernet Controller. The driver is the layer right below NDIS, responsible for bridging the communication between the physical network interface card and NDIS.

The Intel 8255x chipset consists of basically 2 primary components: The Command Unit (CU) and the Receive Unit (RU). In order to send a packet, a data structure containing the packet must be sent to the CU. Finally, the transmission is performed after sending a start command to the CU. On a normal operation, the network packet is built in the Windows network stack and delivered to the NIC (Network Interface Card) driver. However, the chipset backdoor bypasses the network stack and builds the entire malicious packet to be sent. Since it doesn't rely on the Windows network stack to build the packet, it can't be detected by security software neither by the Operating System. That is how data exfiltration can be achieved.

The RU is responsible for handling incoming packets on the network. When a valid packet arrives, it informs the CPU by raising an interrupt. The interrupt is handled by the windows NDIS and the NIC driver. In order to bypass NDIS detection, the driver can redirect the interrupt to a different address where it can analyse the packet received and then, if appropriate, send the interruption back to the NDIS for a normal operation.

By these means, a malware infected driver can manipulate network packets before the Operational System and therefore remain undetected. The authors of this proposal tested their solution against the Windows XP Firewall, Zone Alarm and Snort. None of them were capable of detecting the exfiltrated neither the infiltrated packets.

2.2 Stealth Hard-Drive Backdoor

As most hardware malwares, hard-drive backdoors are highly hardware dependent, and therefore requires customization for each targeted device. Zaddach claims that the hard-drive market has now shrunk to only three major manufacturers, with Seagate and Western Digital accounting for almost 90% of all drives manufactured (Zaddach, 2013).

Most hard-drives are based on a custom System on Chip (SoC) design. The SoC usually has a microcontroller core with some RAM, ROM and Flash memory to store the firmware for the microcontroller. There is also a large DRAM memory that serves as cache for the hard-disk requests. The microcontroller is responsible for translating the data requests coming over the SATA (or SCSI) interface and storing them in cache memory. Specialized hardware (usually DSP or FPGA) that also has access to the data in cache, is responsible for physically reading or writing on the disk.

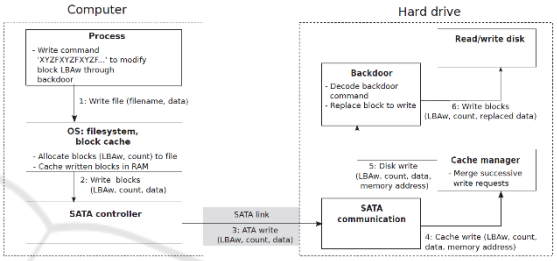

Figure 1: Hard-drive firmware backdoor changing data on a write operation.

An infected firmware can take advantage of the privileged position of the microcontroller and intentionally change data being read or write to the disk plates. Figure 1 shows the implementation done by Zaddach of a firmware backdoor that changes blocks to be written on the disk based on a magic number present in the block (Zaddach, 2013). Since the backdoor lies in the hard-drive firmware, the writing modification goes undetected by the Operating System. Data exfiltration is also possible since the modified firmware can also change the sector to be read from and therefore read data from anywhere in the disk. These techniques can be used to, for instance, modify the password hash for the root account written in the /etc/shadow file enabling a remote user to login with root privileges. This modification is persistent even after re-installing the Operating System. Also, the hash for the root user can be instead exfiltrated while a read operation is performed anywhere on the disk.

The modified firmware doesn't require physical access to the hard-drive to be deployed. A single local or remote access with root privilege is enough to re-flash the hard-disk firmware using manufacturer's firmware update mechanisms. This can even be done by a malware that temporarily infects the machine in

order to reprogram the hard-disk's firmware and then remove itself from the system to remain undetected.

2.3 Exploiting Intel Processor's SMM Mode

System Management Mode (SMM) is a relatively obscure mode present on Intel processors made for low-level hardware control and debug (Embleton, 2008). In this mode all normal execution (including the Operating System) is suspended to give place for a special software that is executed with high privileges. According to Embleton, SMM code is completely non-preemptible, runs below the Operating System, and is immune to memory protection mechanisms (Embleton, 2008).

A rootkit designed to run at the SMM mode is capable of intercepting and emulating low level system events without needing to modify any existing OS code or data structures. Embleton developed a proof of concept SMM rootkit that redirects keyboard Interrupt Requests (IRQ) to SMM mode by changing the Advanced Programmable Interrupt Controller (APIC) table in order to create a chipset-level keylogger. The logged keystrokes are then encapsulated into UDP packets and sent out via the chipset LAN interface. This is all accomplished without making any visible changes to the target Operating System.

The SMM has a completely separate address space for executing programs called SMRAM. The contents of SMRAM are only visible to code executing in the SMM mode. In order to switch to SMM mode, the processor must receive a System Management Mode Interrupt (SMI). There are a variety of events that can trigger an SMI. Some of them are a power button press, real time clock (RTC) alarm, USB wake events, Advanced Configuration and Power Interface (ACPI) timer overflows, periodic timer expiration, and a write to the Advanced Power Management Control (APM) register, 0xB2.

Upon receiving an SMI, the processor saves the current state into the SMRAM and start the execution of the SMI handler. Code running in the SMM mode is non-preemptible because the SMI has greater priority than all other interrupts, even non-maskable ones. In order to deliver the malware, its code must be copied to the SMRAM area. However, this might not be a straightforward task since this area is normally not visible in processor modes other than SMM.

The System Management RAM Control Register (SMRAMC) can control SMRAM visibility. The two relevant bits in this register are the D_LCK bit and the D_OPEN bit. When D_OPEN is set, SMRAM can be visible to other processor modes. D_LCK controls the access to the SMRAMC. Therefore, when D_LCK is set, SMRAMC register becomes read-only until a reset occurs, preventing a program to change the value of the D_OPEN bit. If the D_LCK hasn't been set by the BIOS or the Operating System, a kernel driver can be used to install the rootkit into the SMRAM area by setting the D_OPEN bit, copying the malware to SMRAM, clearing D_OPEN and then finally setting D_LCK to prevent further changes.

Once installed, the rootkit subverts the I/O APIC interrupt table in order to redirect keyboard interrupts to the SMI handler. Therefore, every time a key is pressed, an SMI interrupt is triggered and the processor enters in SMM mode. Once in SMM mode, the malware logs the key into a buffer and then invoke the normal keyboard handler to address the key press as usual. When the buffer becomes full, the malware builds an UDP packet with all the data in the buffer and sends a Transmit Command Block (TCB) to the LAN controller in order to send the UDP packet out over the network.

2.4 Exploiting I/O MMU Vulnerability

All modern Operating Systems implements the concept of virtual memory, which is having each process running in a separate address space. This enables memory isolation, so that multiple processes running on the same machine can't see each other address space. The Memory Management Unit (MMU) is a device responsible for making the address translation from virtual memory to physical memory.

Devices connected to the bus usually don't have memory virtualization. Instead, they all share the same address space and access physical memory using Direct Memory Access (DMA). DMA enables I/O controllers to transfer data directly to or from the main memory. This can become a great threat, since malicious devices can take advantage of this mechanism to tamper critical areas of memory such as the Operating System kernel.

In order to increase security and enable device address space isolation, a special MMU for devices was created, called I/O MMU. Although an I/O MMU can separate device's address space, it may also contain some vulnerabilities that can enable malicious code to have access to protected resources. Sang explores some vulnerabilities found in the Intel VT-d, which is Intel's implementation of an I/O MMU (Sang, 2010).

Intel VT-d is composed of a set of DMA Remapping Hardware Units (DRHU). Figure 2 illustrates how memory access originated from I/O controllers are remapped at the DRHU in order to access main memory. The DRHU can also verify whether the access is legitimate or not in order to either reject or forward the request.

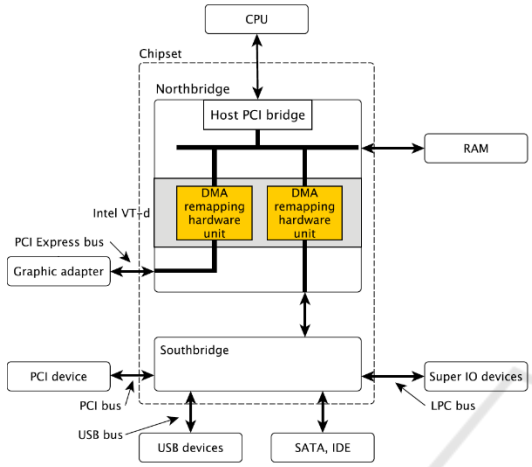

Figure 2: DMA Remapping Hardware Units.

All DMA requests arriving at a DRHU contain a BDF-Id (Bus/Device/Function Identifier) reported by the device that originated the request. Based on the BFD-Id, the DRHU can remap the memory access to the correct region of the main memory. However, it cannot be sure that the identity reported by the controller in its access really corresponds to its BDF-Id. Therefore, malicious devices still can access protected regions of the main memory by spoofing its BFD-Id.

In order to provide the correct memory remapping for each I/O device, the DRHU relies on the DMAR ACPI table. This table is loaded and configured by the BIOS, before the Operating System is loaded.

According to Sang, if an attacker manages to hide some DRHUs from the OS by modifying the DMAR ACPI table, some I/O controllers should be able to perform DMA requests freely: the DRHUs in charge of restricting their access are not configured by the Operating System (Sang, 2010).

I/O MMU can still be a good security resource for malicious devices. However, these vulnerabilities, although hard to be addressed, can represent a threat once it enables malicious devices to still have access to the entire memory address space.

2.5 Hardware Trojans in Embedded Processors

Hardware Trojans are malicious modifications to the circuitry of an Integrated Circuit (IC) made by untrusted design houses or foundries. They are generally designed to leak secure information from inside the IC and to remain undetected during the post fabrication test phase.

Figure 3 shows the hardware Trojan developed by Wang et al. for their implementation of the 8051 microcontroller (Wang, 2012). In order to remain undetected during normal operation, hardware Trojans should be activated only in rare conditions.

One type of hardware Trojan is the sequential Trojan, that only activates its payload after receiving a specific sequence of commands that is considered to be very rare to occur under normal circumstances. In general, the sequential Trojan can be triggered by three different conditions: specific sequence of instructions, specific sequence of data and combinations of sequences of instructions and data.

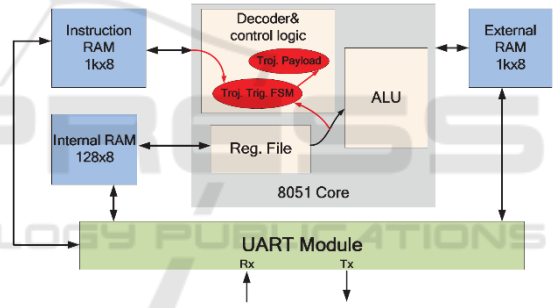

Figure 3: Hardware Trojan infecting an 8051 Microcontroller.

In order to implement the sequential Trojan, Wang developed a state machine in the combinational logic for the 8051 microcontroller. Once all the sequence of commands is achieved, the payload is activated. The 8051 microcontroller was programmed to perform RC5 encryption. Therefore, with the Trojan activated, it starts to leak information about the encryption algorithm and the encryption key, and cause various system malfunctions.

It was verified by Wang that the hardware overhead of the implemented Trojan was less than 3.1% on the FPGA platform (Wang, 2012). Also, it is to be noted that hardware Trojans do not necessarily mean extra logic. The Trojan can be included by modifying the original design, creating even smaller overheads.

2.6 Printer Firmware Modification

Firmware update is a ubiquitous feature found in modern embedded devices, and most of them don't even require authentication to perform the update. Firmware modification attack aim to inject malware into the embedded device by modifying its firmware through a simple firmware update process.

Cui et al. exploited a vulnerability in the HP Remote Firmware Update (RFU) (Cui, 2013). According to the authors, the update procedure is coupled with the printing subsystem. Therefore, in order to perform an update, the RFU file is *printed* to the target device via the raw-print protocol over standard channels like TCP/9100, LPD and USB. This means that it is possible to pack arbitrary executable code back into a legitimate RFU package in a Printer Job Language (PJL) command. This command can then be embedded into a malicious document to be printed by unwitting users in the form of, for example, a resume or an academic paper.

Cui demonstrated a creation of a malware for the HP LaserJet P2055DN (Cui, 2013). By analysing an original RFU packet is was revealed that the binary payload was compressed. Therefore, it was necessary to dump the firmware straight from the printer's flash memory in order to analyse it. Manual inspection of the printer's primary control board revealed that the system is powered by a Marvell SoC with ARM architecture. The Marvell SoC uses the Spansion flash chip as a boot device. Analysis of the boot loader code revealed the binary structure and compression algorithm used in the RFU format.

The firmware for the P2055DN is based on the VxWorks Operating System. The malware developed was basically a root kit that was embedded into the VxWorks image. According to the authors, the root kit was capable of:

- Command and control via covert channel
- Print job snooping and exfiltration
- Autonomous and remote-controlled reconnaissance
- Multiple device type infection and propagation to the Windows operating system and other embedded devices
- Reverse IP tunnel

With these information in place, it was possible to create a tool that would automatically generate a valid compressed PJL update packet from the uncompressed ARM ELF image of the infected firmware.

Once the malware is delivered to the victim printer, the attacker can use it to gain access to the secured internal network by establishing a reverse IP tunnel through the printer. It can also be used for ARP cache poisoning and even to infect other printers and embedded systems connected to the network.

2.7 Malicious Hardware That Enables Software Attacks

Previous attempts were made to create hardware Trojans that would be hard-coded to the Integrated Circuit (IC) design in order to leak data. Although these Trojans are hard to detect, they are limited by operating at the hardware-level abstractions only, and by ignoring higher-levels of abstraction and system-level aspects. Their malicious circuits are useful only for the specific purpose for which they were made. Therefore, if the higher-level data is not mapped to hardware in the way the Trojan is expecting for, it becomes hard or even impossible for it to be collected.

While simple attacks, like stealing RSA encryption keys on a RSA encryption circuit are feasible to be done with hard-coded malicious circuits, it is unclear how to realize semantically richer attacks, like "execute the SQL query 'DROP TABLE *;'" using this technique.

For this reason, King et al. developed a malicious processor that enables software-based attacks (King, 2008). They show that an attacker, rather than designing one specific attack in hardware, can instead design hardware to support software attacks. The malicious CPU consisted of the Leon3 processor (open source SPARC design) with two added mechanisms: one for memory access and another for invisible malicious code execution called *shadow mode*.

The memory access mechanism gets triggered by a specific sequence of bytes on the data bus. Once activated, the mechanism disables the MMU privilege levels for memory accesses, thus granting unprivileged software access to all memory, including privileged memory regions like the operating system kernel. The authors wrote a program that performed privilege escalation by using the memory access mechanism. Once it was activated, the program searched the kernel memory for its own process structure and then changed its effective user ID to root, thus granting super user permission for itself. The memory access mechanism allows the attacker to violate directly OS assumptions about memory protection, giving him a powerful attack vector into the system.

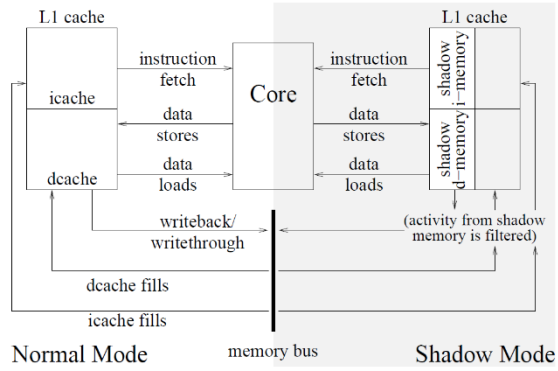

Figure 4: Shadow Mode inside the Processor.

The shadow mode is similar to Intel System Management Mode (SMM) because shadow mode instructions have full processor privilege and are invisible to software. Figure 4 illustrates how the shadow mode integrates into the processor. It gets activated by a predetermined *bootstrap trigger*, which is a set of conditions to tell the IMP to load some code (firmware) from nearby data and execute it in shadow mode. The attack can happen even remotely by, for instance, an UDP packet that has the bootstrap trigger. Even if the OS decides to drop the UDP packet, the mechanism is activated, because for the OS to drop a network packet it must first inspect it, and the act of inspecting the network packet gives the bootstrap mechanism sufficient opportunity to look for a trigger. Once the trigger is found, it silently loads data within the dropped network packet as a malicious firmware and runs that within shadow mode.

2.8 Stealing Data with an L3 Cache Side Channel Attack

In order to reduce the memory footprint of the system, modern operating systems implement the concept of shared pages. The shared pages are identical portions of memory among two or more processes. Usually the Operating System creates shared pages based on the location, which is the case for shared libraries.

However, shared pages can also be created by actively searching and coalescing identical contents. This technique is called *page deduplication*. To enforce isolation, the OS sets the pages to be read only or copy-on-write. However, it doesn't prevent some forms of inter-process interference.

Once a shared page is touched, it gets copied to the processor's cache. Therefore, a side channel attack technique can utilize this cache behavior to extract information on access to shared memory pages. The technique uses the processor's clflush instruction to evict the monitored memory locations from the cache, and then tests whether the data in these locations is back in the cache after allowing the victim program to execute a small number of instructions.

Based on this side channel attack, Yarom developed a new technique called *Flush+Reload* (Yarom, 2014). The idea behind it is to flush data out of the processor's internal cache, so that it will be loaded at the L3 level cache. This enables side channel attacks between different cores, and even between different Virtual Machines (VM).

Yarom demonstrated the use of the Flush+Reload algorithm by attacking the RSA implementation of GnuPG. The attack was tested in two different scenarios: same-OS, where both the victim and the attacker runs at the same OS, and cross-VM, where each application runs on a separate VM.

The authors were able to extract 98.7% of the RSA key bits on average in the same-OS scenario and 96.7% in the cross-VM scenario, with a worst case of 95% and 90%, respectively.

3 IMPLEMENTING A MALICIOUS USB DEVICE

The USB standard was created in order to improve the connection of plug-and-play devices to PCs. Due to its ease of use, it became an instant success.

The USB Flash Drive is one of the most used USB devices in the world. It consists of a flash memory with an integrated USB interface. Due to its widespread use, it also became a huge attack vector for malicious code. Many malwares were developed targeting the USB Flash Drive. These malwares are automatically deployed once the USB Flash Drive is inserted into the computer. After infection, the malware also copies itself to every USB Flash Drive that is inserted in the infected computer. In order to mitigate this type of attack, most anti-malware software prevent USB Mass Storage Devices to automatically execute code when inserted.

However, there is another category of attack that can exploit USB devices and does not rely on malicious software stored inside a USB Flash Drive. The USB specification (Universal Serial Bus, 2001) declares a class of devices called Human Interface Devices (HID). A device that declares itself in this category and follow the HID specification, can be accepted and recognized immediately by the Operating System without the need for installing

drivers. That is the case, for example, of keyboards, mice and game controllers. A malicious USB HID device can perform operations without user's knowledge or intervention.

In order to illustrate the capability of malicious USB devices, this paper describes an approach of reprogramming the USB interface of an Arduino Mega to act as an USB HID Keyboard.

The Arduino Mega is a development board based around the ATmega2560 microcontroller running at 16MHz. It has 54 digital input/output pins (of which 15 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a USB connection, a power jack, an ICSP header, and a reset button.

In order to provide the USB interface for the ATmega2560 microcontroller, the Arduino board uses an auxiliary controller, the ATmega16U2, which converts USB signals coming from the computer to the first serial port of the ATmega2560. Similarly to the Atmega2560, the ATmega16U2 is also an AVR RISC-based microcontroller, but with USB capabilities. It has 16KB of ISP flash memory and is factory configured with a USB bootloader located in the on-chip flash boot section of the controller to support Device Firmware Upgrade (DFU).

DFU mode can be enabled by sending a special USB stream to the controller. Once activated, it allows In-System Programming from its USB interface without any external programming device. Therefore, by just connecting the Arduino to a host computer, it is possible to reprogram the firmware of the ATmega16U2 controller.

To create the malicious USB device mentioned, the ATmega16U2 was reprogrammed with an USB Keyboard firmware. This firmware allows the ATmega16U2 to act as an HID Keyboard and send keystrokes based on strings stored in the ATmega16U2's flash. The strings activate special functions on the Operating System, and if carefully designed, can cause great harm. The keystrokes are sent so fast that each command is executed in less than a second. The strings created to demonstrate the device targets Windows machines, and by sending key combinations it is able to open the Windows run dialog and write a batch script to disk. It then executes the batch script and finally opens notepad and writes continuously: "You have been hacked!".

This conceptual approach illustrates how easily an USB device can be reprogrammed to execute malicious code. Since the code is embedded into device's flash memory, the Operating System does not recognize it as a threat. From the OS perspective, it is extremely difficult to analyse and remove malicious code embedded in a USB device.

There is not any simple solution to this. Any protection attempt from the OS would basically interfere with the usefulness of USB, which makes it so popular. Apparently, the only working solution would be to convince manufacturers to disable firmware update at the factory, so that the device cannot be reprogrammed.

4 CONCLUSIONS

This paper described in total eight different approaches of hardware attacks targeting software. Although the hardware and firmware modifications demonstrated in this paper are very specific to each device, they have proven very efficient against software protections. After all, the software must trust entirely in the hardware to perform its operations, therefore, if the hardware has been tampered, it becomes really challenging to regain control of the system.

This paper also demonstrated how to create a quick hardware attack by modifying the firmware of an USB device via DFU. By completely replacing the firmware of the USB device, it was possible to make it behave as a HID keyboard and therefore send malicious key strokes to the Operating System.

REFERENCES

Robert, J. and Chen, T. 2004. The Evolution of Viruses and Worms. *Statistics: A Series of Textbooks and Monographs.* (2004), 265-285.

Sparks S. et al. 2009. A chipset level network backdoor. *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security - ASIACCS '09*. (2009).

Zaddach, J. et al. 2013. Implementation and implications of a stealth hard-drive backdoor. *Proceedings of the 29th Annual Computer Security Applications Conference on - ACSAC '13*. (2013).

Embleton, S. et al. 2008. SMM rootkits. *Proceedings of the 4th international conference on Security and privacy in communication networks - SecureComm '08*. (2008).

Sang, F. et al. 2010. Exploiting an I/OMMU vulnerability. *2010 5th International Conference on Malicious and Unwanted Software.* (2010).

Wang, X. et al. 2012. Software exploitable hardware Trojans in embedded processor. *2012 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT).* (2012).

A. Cui. et al. 2013. When firmware modifications attack: A case study of embedded exploitation. *2013 The*

Network and Distributed System Security Symposium (NDSS). (2013).

King, S. et al. 2008. Designing and implementing malicious hardware. *Proceedings of the 13th international conference on Architectural support for programming languages and operating systems.* (2008).

Yarom, Y. et al. 2014. FLUSH+RELOAD: a High Resolution, Low Noise, L3 Cache Side-Channel Attack. *Proceedings of the 23rd USENIX Security Symposium.* (2014), 719-732.

Universal Serial Bus, 2001. Device Class Definition for Human Interface Devices v1.11. (2001).