# Hardware backdooring is practical

Jonathan Brossard - jonathan.brossard@toucan-system.com

Security Research Engineer & CEO, Toucan System, France and Australia

Blackhat Briefings and Defcon conferences, Las Vegas, 2012

*"To create is to resist, to resist is to create"* – National Council of the Resistance

**Abstract.** This article will demonstrate that permanent backdooring of hardware is practical. We have built a generic proof of concept malware for the Intel architecture, Rakshasa[1], capable of infecting more than a hundred different motherboards. The first net effect of Rakshasa is to disable NX permanently and remove SMM[1] related fixes from the BIOS, an Unified Extensible Firmware Interface (UEFI)[2][3] firmware, or from a PCI[4] firmware, resulting in permanent lowering of the security of the backdoored computer, even after complete erasing of hard disks and re-installation of a new operating system. We shall also demonstrate that preexisting work on MBR subversions such as bootkiting and preboot authentication software brute-force or faking can be embedded in Rakshasa with little effort. More over, Rakshasa is built on top of free software, including the Coreboot[5], Seabios[6], and iPXE[7] projects, meaning that most of its source code is both already public and non malicious, therefore extremely hard to detect as such. We shall finally demonstrate that backdooring of the BIOS or PCI firmwares to allow the silent booting a remote payload via an http(s) connection is equally practical and ruins all hope to detect the infection using existing tools such as antivirii or existing forensics tools. It is hoped to raise awareness of the industry regarding the dangers associated with the PCI standard, especially question the use of non open source firmwares shipped with any computer and question their integrity or actual intend. This shall also result in upgrading the best practices in companies regarding forensics and post intrusion analysis by including the afore mentioned firmwares as part of their scope of work.

Keywords: Hardware backdooring, PCI firmware, BIOS, EFI, romkitting, remote boot, Botnet.

[1] Rakshasa is the Hindi word for deamon.

## Table of Contents

1 Introduction . . . . . . . . . . 3

2 Related work . . . . . . . . . . 3

3 Overview of the IBM PC and its legacy problems . . . . . . . . . . 5

4 Designing the perfect backdoor : scope of work . . . . . . . . . . 6

5 Implementation details : Rakshasa . . . . . . . . . . 7

6 Inner working of Rakshasa . . . . . . . . . . 8

7 Embedded features of Rakshasa . . . . . . . . . . 10

8 How to properly build a botnet from the BIOS : BIOSBonets . . . . . . . . . . 11

9 Why (possibly hardware assisted) encryption won't solve the problem . . . . . . . . . . 12

10 Conclusion . . . . . . . . . . 12

11 Acknowledgements . . . . . . . . . . 12

## 1 Introduction

A recent[8] report from the US-China Economic and security review commission by Northrop Grumman Corp called "Occupying the Information High Ground: Chinese Capabilities for Computer Network Operations and Cyber Espionage" concluded that: "This close relationship between some of China's -and the world's- largest telecommunications hardware manufacturers creates a potential vector for state sponsored or state directed penetrations of the supply chains for microelectronics supporting U.S. military, civilian government, and high value civilian industry such as defense and telecommunications, though no evidence for such a connection is publicly available." In other words : since China has become the de facto manufacturer of most IT equipment in the world, China can backdoor any computer at will. Anybody part of the supply chain can. We believe this is an euphemism : we shall here demonstrate the practicality of such a backdooring using existing open source software, lowering the bar of such an attack from state level or otherwise very large corporations to any 16bits assembly expert, as well as demonstrate that installing such a backdoor remotely is equally practical.

## 2 Related work

The first known virus, brain, was allegedly built in Pakistan in the early 80's. It was targeting the Master Boot Record (MBR) of the first bootable hard drive in order to gain early execution and used floppy disks to propagate. This attack vector has been replicated by literally thousands of viruses during the 80s and 90s, until the appearance of the internet, when viruses switched to userland in order to benefit from internet access as a propagation vector.

Gaining early execution has long been believed the best way to gain maximum privileges on IBM PCs. In 2009, at Cansecwest, Anibal Saco and Alfredo Ortega[9] demonstrated how they managed to patch a Phoenix-Award BIOS to embed malicious features (modifying the shadow file on Unix-like systems, or patch Microsoft Windows binaries). In 2007, John Heasman[10] demonstrated that infecting The Extensible Firmware Interface (EFI) bootloader would lead to the same results. If the former targeted one specific BIOS, the latter would be mitigated by reinstalling a sane bootloader.

Operating modifications on the file system isn't stealth and leaves clear forensics evidence : as a matter of fact, a simple one way checksum of all the existing files on the filesystem performed before and after infection from a sane operating system would detect the modification. Therefore, security researchers have conceived ways to subvert a running kernel on the fly without even touching the filesystem. Notable research include BootRoot[11]from Derek Soeder and Ryan Permeh, vbootkit from Kumar and Kumar[12], capable of bootkitting a Windows 7 kernel, the Stoned bootkit[13], and the Kon-boot commercial bootkit from Piotr Bania[14], capable of subverting all the existing NT kernels, from Windows XP to Windows 2008 R2 and Windows 7, in both 32 and 64 bits. Those attacks work by booting from an alternate media such as a floppy or usb stick, or by replacing the existing MBR (and restore its first sector in memory, emulating an interruption 0x19). Running a bootkit from an alternate medium leaves no forensic evidence. It is also worth noticing that the Stoned Bootkit managed to bootkit the Windows kernel in spite of possible encryption such as Truecrypt[15].

It is worth mentioning that the inner working of any bootkit is the same : hooking the interruption 0x13 (disk access) by patching the Interrupt Vector Table (IVT), set a rogue interruption handler, and emulate an interruption 0x19 by loading the first sector of the first bootable disk at 0x0000:0x7c00 before transferring execution to this location. This first sector will in turn load the operating system normally, but the rogue 0x13 interrupt handler will hook any read of a sector from disk and once the kernel of the main operating system is fully unpacked in memory, patch a few carefully chosen locations in order to modify it on the fly. Public payloads include patching the NT kernel to accept any password for any account, or load an unsigned kernel module, which can effectively execute any operation in ring 0 once the operating system is fully loaded (eg: allow local privilege escalation, remote control, etc).

Our main contributions are:

- Romkit not a single, but hundreds of different motherboards. this is better by two orders of magnitude over existing research.
- Embed any existing bootkits as part of a romkit without any modification.
- Use of routable ip packets to upgrade the romkit, execute remote payloads and optionally attack the LAN, either using an ethernet or Wifi stack.
- A mechanism to allow botnet resilience against law enforcement DNS take overs, over HTTPS using asymmetric cryptography and a replication mechanism making the rom-botnet close to impossible to shutdown.
- Permanent lowering of the security level of any future (unknown) operating system installed on the computer, when existing bootkits assume either a GNU/Linux or NT kernel.
- Persistence of the infection even if the main BIOS rom is ever flashed by infecting multiple PCI firmwares (such as a cdrom firmware) with malicious network firmwares, capable of upgrading any other PCI rom remotely (eg: infect the BIOS or main network PCI rom back), while preserving functionality.
- Disabling Address Space Layout Randomization (ASLR)[16] and NX[2] from NT kernels, making any future Windows operating systems vulnerability trivial to exploit on an infected machine.
- An infection mechanism offering both plausible deniability and non attribution, hence compatible with state level global hardware backdooring.

[2] The NX bit is the 63th (leftmost) bit of the Page Table Entry on amd64 architectures.

## 3 Overview of the IBM PC and its legacy problems

The IBM PC was originally designed around 1981. It has evolved since, in particular with the replacement of older Industry Standard Architecture (ISA)[17] peripherals in favor of much faster Peripheral Component Interconnect (PCI)[4] devices in 1996, and even faster devices with PCI Express (pushed by Intel in 2004, commonly referred to as PCI-E or PCIe)[18]. But the core design remains the same, the cpu still booting in 8086 compatibility mode (id est: 16b real mode) even on the most recent motherboards. It is worth mentioning that when first launched, Windows 95 had open network shares (netbios), which is a clear sign that around those dates, Microsoft had not anticipated the raise of the Internet and had therefore designed an operating system to be used primarily in LAN environments. Needless to say IBM couldn't have anticipated the internet either back in 1981. The basic design of the IBM PC is presented in figure 1 : from the bottom to the top, we find the Super I/O, to which are connected legacy ISA devices (hence slow devices) such as keyboards, mouse, and floppy drives. The Super I/O is connected to the South bridge, via an LPC bus. The Southbridge is responsible for handling faster peripherals (up to 2133 Mb / second for PCI-X 2.0), typically compliant with the PCI standard. Such devices can be for instance network (ethernet), sound or older graphic cards. The South bridge is himself connected to the North bridge via the internal bus, which is himself responsible for handling much faster peripherals typically compliant with the PCIe standard, with a debit up to 16 Gb/s for the version 3.0 of the standard. Such devices can be for instance newer 3D graphic cards, gigabit ethernet cards or enterprise storage (SAS). The North bridge can contain the cpu and has in any case a high speed connection with it through the Front Side Bus (FSB). Each layer of the architecture contains Direct Memory Access (DMA) chips, which are controlled by input/output memory management unit (IOMMU)[19] for performance and security reasons.

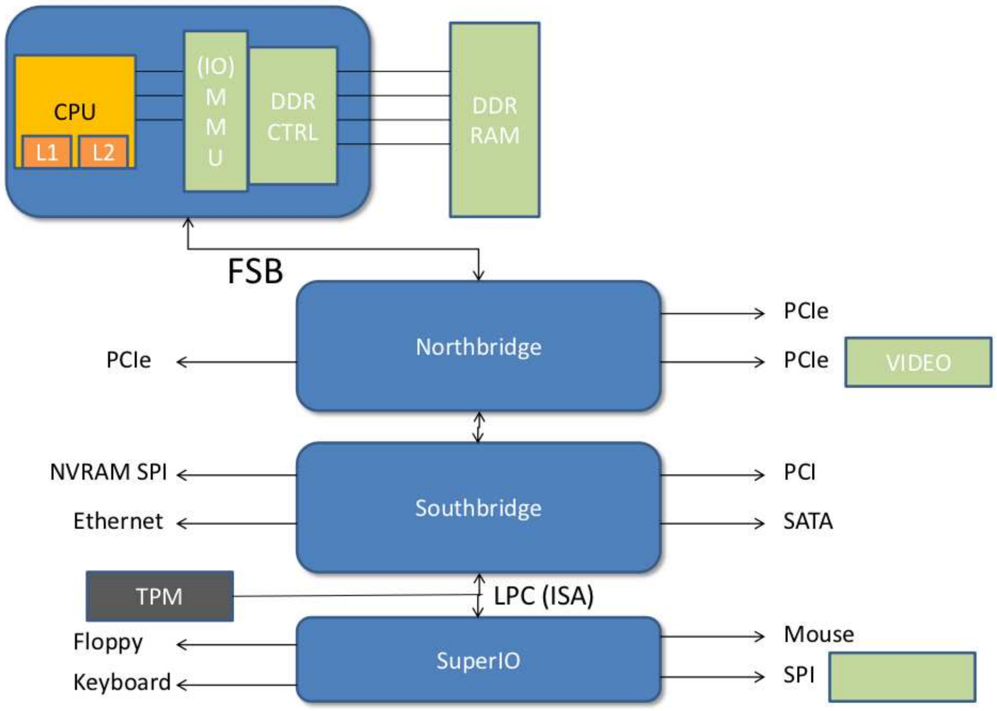

**Fig. 1.** Overview of the IBM PC architecture.

Because peripherals sometimes need upgrading, they are controlled by via embedded firmwares : ISA peripherals had embedded ISA roms, and PCI have PCI expansion roms. Even though end users are hardly aware of their very existence, those firmwares can be upgraded (flashed), usually using proprietary and vendor specific tools. To avoid trivial backdooring, some cards offer a physical switch which needs to be manually actioned to allow the flashing of the firmware.

An other critical firmware is the BIOS firmware, which takes place in the motherboard. It is responsible for detecting hardware such as RAM and peripherals at boot time, initialize an Interrupt Vector Table to allow interaction with those peripherals from the RAM, and load the Master Boot Record from the first bootable hard drive using interruption 0x19. When booting in 8086 compatibility mode, the boot loader can only rely on the IVT to load a kernel in memory, who will typically quickly switch to protected mode and never use the IVT ever again, favoring much faster device drivers for hardware interface from protected mode. The BIOS firmware can also be upgraded, and it is in fact pretty common to correct hardware bugs such as cpu bugs by pushing a signed microcode update to the cpu from the BIOS. Finally, the BIOS typically activates a bit in control registers to prevent the cpu to be switched to System Management Mode (SSM) - which is really the cpu patch mode - in order to prevent a class of attacks discovered by Loic Duflot[1] and released publicly by BSDaemon[20]. Some vendors, notably HP have started digitally signing their BIOSes to avoid rogue upgrading. In any case, an attacker with physical access to the BIOS or peripherals can replace the firmwares without relying on the operating system by writing to the chip using for instance self sufficient hardware tools based on FPGA chips.

One of the problem with the design of this architecture is rooted in the trust peripherals of each layer have in each others. For instance, the firmware of a cdrom PCI device can absolutely control a PCI network card. And in the same way, an ISA firmware can control an other ISA device. This behavior cannot be changed : this is how IBM PCs work.

It is also worth noting that the Trusted Platform Module[21] (TPM) is typically connected to the south bridge, very far from the cpu. Moreover, it is a passive component, meaning that software can chose to use it or not, but that the cpu cannot enforce its use. This is a serious weakness of the whole architecture as we will see later in this paper.

As one might expect, the firmwares embedded in BIOSes or PCI devices (PCI expansion ROMs) are vendor specific and totally not standard. The whole purpose of this paper is to explain how to generically modify such firmwares to create a backdoor that can not be detected from user land once a kernel has been loaded in RAM and a switch to protected mode has been performed. Because the BIOS firmware is strictly speaking the first piece of software to be executed on the computer, and because it gives early control to each PCI expansion ROM during early boot (before switching to protected mode), any malicious action routine executed at this stage enjoys full access to hardware resources. In particular, it is worth reminding that real mode is not capable of multitasking, and that such routines therefore have access to 100 per cent of the resources of the machine.

## 4 Designing the perfect backdoor : scope of work

The author of this white-paper doesn't usually spend time writing malwares. To the opposite, he spent quite a significant part of his life studying and reversing them. But to prove our point, let's pretend we'd really like to design a proper backdoor to be used in the wild. We'd also like to underline the fact that we believe most of the communication involving malware, if not all of it, coming from major antivirus vendors and happily relayed by servile media is blatantly tainted with FUD[3]. Instead of arguing on whether Flame and Stuxnet could have been written by amateurs instead of nation states, let's see how an attacker can write a nation state quality backdoor on a budget. This shall also serve as a good example of how a vendor manufacturer could design a proper backdoor with similar intend. First of all, we'd like our backdoor to be persistent. Not simply persistent between reboots, but also in case the user of the computer was to replace the entire operating system, possibly even portions of

[3] Fear, Uncertainty, Doubt.

the hardware (replacing the hard drive or the network card for instance), flash the BIOS or any other firmware on the motherboard and peripherals. To achieve this goal, we will avoid having a single point of failure and will hence need some degree of redundancy.

Of course, the backdoor should be as stealth as possible. Needless to say it shall not be detected by any antivirus on the market. It shall also be portable : ideally, we'd like it to be totally operating system independent. Because a significant portion of the IT budget of companies goes into expensive (and quite inefficient[4]) detection gear such as Intrusion Detection Systems (IDS), Intrusion Prevention systems (IPS) and firewalls, we'll need our backdoor to be capable to break the network perimeter of large companies in some way or an other.

In terms of functionality, we'd like the backdoor to allow remote updates and provide remote access. This implies some degree of network awareness.

Finally, if we want to match nation state quality backdoors, we need our creation to obey two golden rules always employed by secret services around the world : plausible deniability and non attribution. the first one is a mean to offer an alternative explanation in case the backdoor was to be discovered in spite of our best efforts to make it as stealth as possible. By having a second credible alternative to explain the presence of the backdoor in the system, nation states can deny any wrong doing and qualify their detractors of sheer believers of conspirationist theories. Non attribution is equally important and is the feature of not allowing the backdoor to be linked to any individual or state in particular. It is very much in the air for nation states to claim "it wasn't me ! ... it was China".

Needless to say the backdoor shall also be cheap : it shall therefore be blatant that a skilled individual can indeed create backdoors so far believed to be only possibly crafted by states in the story telling of the media, and eventually put in perspective the FUD that is served daily to both citizens and decision makers in the corporate as well as political worlds.

## 5 Implementation details : Rakshasa

The quadruple constraint : cheap development, vast features (such as network stack), hard detection and non attribution dictate one direction for our implementation : free and open source software. As a matter of fact, the direction taken by virtually any malware so far to rely on custom code entirely is a bad idea as it offers a large attack surface to antivirii in terms of detection, is often attributable if code is reused accross multiple malware, and is nowhere near cheap. To the opposite, using non malicious free and open source software as the core of the backdoor provides little angle of detection to antivirii, is non attributable (the source code is available to anyone on the internet), is as cheap as it gets. Plus it offers free maintenance from the community to the malware author.

In order to achieve both persistence and stealthiness, it was chosen to target primarily the BIOS. But to offer redundancy in case the BIOS was ever flashed, it has been decided to also provide an infection mechanism through PCI expansion ROMs, by targeting the firmware embedded in ethernet network cards[5].

Rakshasa is comprised of a custom version of Coreboot for the BIOS backend, of a custom SeaBIOS BIOS-payload to create and IVT, of a set of PCI expansion ROMS (SVGA driver and a custom iPXE

[4] From his humble experience with vulnerabilities and 0days, the author strongly believes looking for bugs and getting them fixed is the only reasonable protection against 0days. Unlike the aforementioned network gadgets (including antivirii/IDS/IPS), static analysis and fuzzing have proved to work. Symbolic execution is also a promising field of research even though exponential path explosion seems unavoidable in the current state of the art. We strongly recommend companies to invest in those techniques instead of silver-bullet-anti-0day-detection tools, which work neither in theory nor in practice.

[5] Like mentioned previously, the firmware controlling a network card could actually be placed in any other PCI device. Also flashing, say, the cdrom firmware with the very same infected firmware would offer even greater redundancy.

ROM), plus a custom active bootkit which is retrieved from the network.

Coreboot by itself isn't a full BIOS : it is only responsible for detecting the hardware present on the machine, perform a BIOS POST and transfer control to a "BIOS payload". This BIOS payload is in turn responsible for setting up and Interrupt Vector Table that will allow an operating system to interact with the hardware previously detected. In our setup, Coreboot and Seabios have been trivially patched not to display anything, even though Coreboot is in theory capable of displaying a custom user defined bootsplash at boot time, which would allow faking the image normally displayed by the original BIOS (typically containing the vendor logo etc). SeaBIOS was chosen for its simplicity, but alternative open source BIOS payloads are available which can also display menus and fool more advanced users that could want to modify their BIOS settings. Other BIOS payload can also contain EFI/UEFI extensions, which makes our technique applicable entirely to UEFI environments[6].

Coreboot having a very modular design, it is possible to embed about any PCI ROM along with it, meaning that we can stuff arbitrary code inside the BIOS chip itself. We have chosen to stick to the bare minimum, adding a video driver and a rogue iPXE ethernet firmware (equally patched not to display anything when operating). This later PCI firmware implements a super set of the original PXE[22] standard : instead of relying only on DHCP to acquire an IP address and the address of a TFTP server to download an operating system from, it can use a wide range of protocols, based on a user defined configuration file embedded in the firmware itself. iPXE contains working stacks for ethernet, wifi and even wimax data link layers protocols, a full featured IPv(4/6)/icmp stack, and implements all a malware writer could dream of in terms of upper layer network stacks : from UDP and TCP to DHCP, DNS, HTTP, HTTPS and (T)FTP among others.

Technically, Coreboot could also embed a full featured backdoor in the BIOS ROM itself. But since we'd like to offer upgrading facilities to Rakshasa and avoid leaving any trace of hostile code on the machine (to deceive forensics analyst in case of detection or at least suspicion), we'll refrain from using this technique : we'll boot our malicious payload from the network at each boot.

Instead of booting a normal operating system from the wire, we'll use iPXE to boot a bootkit remotely, which will in turn transparently load the bootloader from the first bootable disk by emulating interrupt 0x19, patch the kernel on the fly, and eventually load the operating system kernel as the user expects it, silently.

## 6 Inner working of Rakshasa

Rakshasa can typically be installed in one of two ways. The first one is, given physical access to the hardware, to flash the BIOS firmware. This can be achieved either by using a dedicated physical flasher (usually made of a FPGA)[7] or by relying on a generic firmware flasher (from usb or a cdrom for instance. PXE could also be used). This operation takes less than a minute in any case.
The second installation technique is a post intrusion one and doesn't require physical access to the hardware at all : once an attacker has achieved remote root on a computer, he can use the same generic flasher to install Rakshasa in place of the original BIOS. In case the operating system is not a Linux, one can simply pivot over the MBR upon next reboot. To achieve redundancy and avoid single points of failure, the network card firmware is also flashed with a rogue iPXE firmware.

Optionally, a second PCI firmware can be replaced by a modified version of iPXE (typically the cdrom). It is unfortunately not possible to maintain the functionality of this second device when doing so for the time being (though this is an implementation issue and not a theoretical one). If the BIOS ever attempted to boot from this second device, the net effect would be the same : the computer would

[6] In other words ,the root cause of the problems, namely a writable BIOS and a passive TPM are not solved with UEFI.

[7] This method has the main advantage of bypassing anti-flashing counter measures such as firmware signing on certain motherboards.

really boot from iPXE, that is, from the network.

iPXE is capable of attempting actions, and perform other ones upon failure. To maximize our chances of working in any given environment, we have configured iPXE to first attempt to connect to the internet via wifi. A number of common SSIDs and even wifi cyphers can be specified at compile time. Typically, we'll try to boot from an attacker defined SSID/WEP access point. In case of failure, we'll try to connect to a (short) list of common public SSIDs usually associated with open wireless access points. The main advantage of relying on wifi is to bypass any network filtering or detection mechanism in corporate environments. As a matter of fact, even if Rakshasa was to be used in a large organization whose access to the internet is normally done through an authenticating proxy, even if I(D|P)S were to be found in this network, even if proper DNS segregation was in place, the simple fact of using encrypted wifi as a link layer would totally breach the network perimeter and render all those costly devices entirely useless.

If Rakshasa still didn't manage to get access to the internet, it would attempt to acquire an IP address from DHCP over ethernet, and default to a chosen reasonable static LAN IP in case everything else failed. Speed is indeed critical to remain unnoticed, and some of those settings can be skipped by modifying a simple configuration file at compile time.

If at this stage Rakshasa still didn't get access at least to the LAN, it would simply boot the default operating system to avoid being detected.

At this stage, Rakshasa can perform a number of things. Assuming it didn't managed to connect to the internet over wifi but simply ethernet access, it could attempt to attack hosts reachable on the LAN using virtually any protocol based on top of IPv4/IPv6 or icmp. This can include, based on a simple iPXE configuration file : icmp host enumeration, tcp/dns port scanning, router farming (over http/https), exploiting network daemons or network stacks (ping of death, ipv6 attacks, sending broadcast traffic, smurf/DDoS attacks, exploiting remote overflows in anything routable, etc.). One could for instance attempt to modify the settings of an ADSL router to open ports. This is not believed to be specially smart or stealth, so this feature is only mentioned for the sake of completeness. As a matter of fact, acting as such would be contrary to our agreed principle of "plausible deniability" since hostile code would be embedded on the backdoored machine...

Rakshasa will then typically try to connect to a given host on the internet over https (say google.com because it is unlikely to trigger alarms, even if the connection is operated over an ethernet link). In case it succeeds, Rakshasa will assume it has full access to the internet. In the opposite case (which could happen if the infected computer had no wifi card and the network was segregated by an authenticating firewall), Rakshasa could attempt to reach the internet via TCP over DNS (which would suffice if the network didn't had proper DNS segregation between the LAN and the outside world) or TCP over icmp (why not !).

As the astute reader shall have understood by now, the sky is the limit : enhancing the capabilities of Rakshasa is really a matter of adding a few lines of text to its iPXE configuration file, and in the worst case scenario to slightly patch the existing network stacks to attempt new exotic protocols[8]. Even if access to the internet often fails, it isn't really a problem : sporadic access to an open wifi network in a airport or from a domestic network would be enough to upgrade Rakshasa from time to time.

Once full access to the internet is achieved (even every so often), Rakshasa will normally download a bootkit from a given location on the internet, such as a file called "foobar.pdf" on a given blog over https, "data.dat" from a ftp one etc. It is worth mentioning that there again, multiple tries are allowed, which offer greater resilience against a shutdown of the main command and control (a simple blog ?) from law enforcement. Having a number of hostnames or IP addresses and possibly services (even though https is really probably the safest option, and it will be the only one considered in the remaining

[8] This is again not recommended : keeping iPXE as close to the original code as possible is desirable to prevent antivirii from finding any signature and stick to the golden rule of plausible deniability : "this code is absolutely legit, it is simply iPXE".

of this paper) to try to download from also increases resilience.

Last but not least : iPXE can chain configuration files by downloading them from the internet. Meaning that no direct reference to the final hostile bootkit needs to be made from Rakshasa itself (the code placed on the computer therefore contains 0 hostile code per si : there is absolutely no reason for an antivirus to flag it as such[9].).
It is worth noticing that if the remote bootkit is ever replaced by a BIOS flasher, Rakshasa can be updated remotely ! Flashing PCI firmwares is a bit tougher : first of, PCI flashing tools are (very) vendor specific. In order to flash the correct PCI firmware on a given network card, the remote web site acting as a command and control would need to know who is the manufacturer of the card. Fortunately, iPXE allows through a set of functions sending the MAC address of the ethernet card as a parameter in a url[10]. From the MAC address, a simple php script placed on the command and control server can extract the OUI number (first 1024 bits) and therefore deduce the manufacturer. From there, the command and control website can return the flashing tool and firmware suitable for the specific device to be flashed under the form of a bootable GNU/Linux environment that will emulate an interruption 0x19 and boot the main OS when done. The second problem comes from the fact that some PCI devices feature a physical switch that must be manually moved before upgrading the firmware. This is in fact not an issue since a PCI expansion ROM can be placed directly inside Coreboot and will have precedence over the one physically present on the device.

Finally, the same update mechanism can be used to disinfect the BIOS remotely by restoring the original BIOS ROM instead of performing a regular Rakshasa upgrade.

## 7 Embedded features of Rakshasa

Because of the design of Rakshasa, any bootkit can be used along with our backdoor without any modification, by simply changing the malicious payload downloaded from the command and control blog (without requiring any adjustment on the backdoored computer).

The active payload of Rakshasa being first executed as a main operating system (from 16b real mode), it can do anything an operating system can do. In particular, it can perform any operation state of the art bootkits can do, such as patching the authentication routine of Windows to allow login locally with any password. The same routine being used when authenticating via Remote Desktop on Windows, this could be enough to even take control of the computer remotely. Other techniques such as injecting an unsigned kernel driver that will in turn execute an arbitrary process (injected by the bootkit itself) with SYSTEM privileges in userland without touching the file system has also been proved practical (this is indeed more than enough to take full control of the computer remotely : a reverse meterpreter over HTTPS - possibly after injecting a dll into a web browser to bypass firewalling restrictions at OS level, and pulling the eventual proxy credentials from the registry or using internal Windows API is also practical and public code exists to perform all those actions).

For our proof of concept, we got a bit of help from Piotr Bania, who kindly customized a version of his great commercial Kon-boot bootkit to boot silently (that is to say without the fancy 16b demo-looking graphics, which are indeed impressive but quite detrimental to a backdoor's stealthiness...). Kon-boot is a very advanced bootkit capable of generically patching the authentication routine of any windows from Windows XP to Windows 7 and 2008, both in 32b and 64b.

If the OS is a Windows flavor, we can disable ASLR by patching the seed used for randomization with a chosen value[11]. We can also remove the NX bit from the Page Table Entry directly in RAM[12]. The combined effect of those two alterations is to leave the address space of any application with executable data

[9] If an antivirus ever did it, it would actually be... a false positive !!
[10] This is even documented on their website at: http://ipxe.org/cfg/mac.
[11] This exact location has been documented by Kumar and Kumar at HITB Malaysia in 2010.
[12] As demonstrated by Dan Rosenberg in his remote kernel exploit presented at Defcon 2011.

(that is: data/bss/heap/stack) and 100 per cent predictable mappings. In other words, any future vulnerability affecting this operating system is going to be trivial to exploit (weaponized by default !). This is equivalent to removing most of the security enhancements achieved by Microsoft in the past 10 years.

But this is not quite enough : we'd like to be able to attack other (possibly unknown at the time of backdooring) operating systems generically. Since we in fact control more than just the malicious payload but the entire booting process since the first bit of BIOS ROM is executed, we can do a lot more. In particular, we can remove cpu microcode updates from Coreboot. The BIOS is the place of choice to push cpu upgrades : if we remove those microcodes, then the cpu bugs and potential vulnerabilities normally fixed at this level will remain exploitable.

We can also remove anti SSM protections, which will have the net effect of letting the Operating System vulnerable to a generic public local exploit[20].

At this stage of the demonstration, it should be clear that once Rakshasa has been installed on a given hardware, the security of the Operating System cannot be ensured anymore.

## 8 How to properly build a botnet from the BIOS : BIOSBonets

.
Some may argue that the weakest point in the architecture presented so far lies in the blog used as a command and control. It is about time we demystify some belief in the insecurity of botnets : if vendors can perform secure updates, and Microsoft seems to have mostly done so for the past 10 years[13], then there is absolutely no reason why malware writers could not do the same.

The weakest point of botnets architectures today is certainly the availability of their command and control. Whenever a hostname is identified by law enforcement agencies as a C and C, they typically perform a DNS take over, redirect the DNS to a public IP they control, and send a shutdown command to any infected host connecting to this IP.

Denying law enforcement agencies (or whoever else) the capability to send a shutdown command to a botnet is relatively easy : all it really takes is to digitally sign the commands using strong asymmetric cryptography. The updates of Rakshasa could also be digitally signed, which would prevent modification of the updates by the same agencies. This way, integrity is ensured.

The only remaining problem is the availability (at least partial and sporadic : after all, we probably don't need to upgrade our backdoor at every reboot strictly speaking) of the command and control. A trivial way to solve this issue is to have a rotative command and control that is randomly picked by Rakshasa every time from a random address across all the internet IP range[14]. There again, the backdoor would be upgraded less often, but the command and control could simply not be shut down.

Finally, embedding client side SSL certificates on the backdoor would prevent trivial detection of command and control hosts by scanning all the internet for given files. It is worth mentioning that Coreboot is capable of using its own embedded CMOS image, hence leaving the real CMOS nvram available to store volatile cryptographic keys. By using this feature as well as hardware fingerprinting (typically the MAC address) to regenerate decryption keys at each reboot, it is possible to create a backdoor extremely hard to extract from the backdoored hardware (eg: it can't be simply copied into a virtual machine for analysis). We believe a BIOSBotnet relaying on such an architecture can literally not be shut down.

[13] Modulo the Flame worm, which seems to have spread, among other vectors, thanks to a rogue cryptographic certificate based on a rogue MD5 hash collision -using an unknown technique at the time of writing- and by hijacking DNS answers to the Microsoft update server.

[14] In practice, a large subset including a sufficient number of non controlled IP would suffice : law enforcement agencies are unlikely to ever shut down the whole AT&T range or say google.com.

## 9 Why (possibly hardware assisted) encryption won't solve the problem

At this stage of the demonstration, some may believe cryptography, and in particular full disk cryptography combined with TPM shall prevent the backdoor from doing any damage. We shall therefore discuss those technologies in the present chapter.

First of, Full Disk Encryption (FDE) in itself has been proved vulnerable to multiple implementation flaws[23][24], and can often be attacked via bruteforce[25], but in all generality, it as simply been proved not to prevent bootkitting at all when not associated with TPM[13]. We briefly propose a variant of the evil maid[26] attack applicable to the scenario of Rakshasa having to bypass such a security feature. First of, TPM being a passive chip, Rakshasa can still boot a remote OS silently, whether TPM is present or not.

Let's first assume TPM is not present. Instead of fetching a bootkit, Rakshasa can detect that the first bootable hard drive is encrypted, and boot a small operating system mimicking the login prompt of the FDE, wait for the user to enter their credentials, save the password to CMOS, optionally sending the password back to the command and control server. Once the password is known, Rakshasa can disable interrupt 0x10 (video) emulate an interruption 0x19 to reload the real boot loader, simulate keyboard typing[23] in 16b real mode by programming directly the PIC microcontrolers embedded in both the keyboard and the motherboard, and eventually let the system boot normally.

Let's now assume TPM is indeed present : if the machine was backdoored before the configuration was sealed with TPM (typically : the machine was backdoored by the manufacturer or by anyone in the supply chain before delivery), the attack is absolutely unchanged. Actually, removing the backdoor after sealing TPM would then change the content of the TPM register at boot time, resulting in the hard drive not being decrypted at all. So in fact, the backdoor is itself protected against tempering !

## 10 Conclusion

In this short white-paper, we have outlined some issues with modern pcs due to legacy and weak security architecture. Because pcs were designed in the early 80's, they weren't initially designed with the usage we have of computers today in mind, and focused more on the interoperability of their core peripherals rather than segregation and security. Unfortunately, there is no simple fix for this : making computers immune to hardware backdooring would require radical modifications of their architecture, which would result in breaking backward compatibility. It is worth noticing that even the most up to date technologies such as TPM and full disk encryption cannot prevent backdooring by someone in the supply chain. We hope this white-paper will help raise awareness among the security community, and help decision makers spend more on verifying the integrity of their hardware as well as software rather than investing in silver bullets that have proved to fail in the past 30 years. In particular, including PCI ROMs and BIOS firmwares as part of security audits both before usage and in case of forensics investigations would be a good idea, while certainly not sufficient.

## 11 Acknowledgements

We would like to thank: Florentin Demetrescu, member of the Coreboot project, without whom this research would have never started in first place. Piotr Bania, for kindly contributing a custom version of kon-boot to facilitate this proof of concept. The Toucan System team both in France and Australia : I wouldn't be doing research without your continuous moral and technical backup. The Hackito Ergo Sum team, Programming committee, Speakers and attendees for their constant passion. Matthieu Suiche, Mark Dowd, Tarjei Mandt, Snare, Twiz, Sid, Meder Kyderaliev, Cesar Cerrudo, Fyodor Yaroshkin Rodrigo "BSDaemon" Branco, Silvio Cesare, Tim shelton, Andrewg, Nemo, Mercy, Caddis, TGO, Dan Rosenberg, Morla, mxn, Xort, the Grugq, Nergal, Pipax, Spender, FX and the entire Phenoelite crew, for their friendship, inspiration, humility, passion and support. All those I haven't mentioned here but help me challenge myself (you know who you are), the HITB, Recon, H2HC, CBACert, Ruxcon teams, as well as the SVC. #social, #blacksec, #busticati. Thanks heaps : you made me write this ;)

## References

1. Duflot, L.: Security issues related to pentium system management mode, CanSecWest (2006)
2. Intel: Extensible firmware interface specifications v1.10 (2003)
3. Intel: Unified extensible firmware interface specifications (2005)
4. PCI-SIG: Peripheral component interconnect specifications lb3 v0.2.6 (2004)
5. Coreboot: (Coreboot - formerly linuxbios)
6. Coreboot: (Seabios - open source implementation of a 16bit x86 bios)
7. iPXE: (ipxe - open source boot firmware)
8. Northrop-Grumman-Corp: Occupying the information high ground: Chinese capabilities for computer network operations and cyber espionage (2012)
9. Saco, A., Ortega, A.: Persistant bios infection, CanSecWest (2009)
10. Heasman, J.: Firmware rootkits and the threat to the enterprise, Blackhat USA (2007)
11. Soeder, D., Permeh, R.: Bootroot, Blackhat USA (2005)
12. Kumar, Kumar: Bootkit 2.0 : Attacking windows 7 via boot sectors, HackinTheBox (2010)
13. Kleissner, P.: Stoned bootkit, Blackhat USA (2009)
14. Bania, P.: Kon-boot (2008)
15. Roux, P.L.: (Truecrypt - free open-source on-the-fly disk encryption software)
16. PaX: (Address space layout randomization)
17. IBM: (Connector bus isa (industry standard architecture))
18. PCI-SIG: Pci exptress base r3.0 v1.0 (2010)
19. AMD: (I/o virtualization technology (iommu) specification revision 1.26)
20. BSDaemon-Coideloko-D0nAnd0n: System management mode hack using smm for "other purposes". (Phrack magazine)
21. Trusted-Computing-Group: (Trusted platform module specifications)
22. Intel: (The preboot execution environment specification v2.1)
23. Brossard, J.: Bypassing preboot authentication passwords by instrumenting the bios keyboard buffer, Defcon (2008)
24. Brossard, J.: Bios information leakage. (2005)
25. Brossard, J.: Bruteforcing preboot authentication passwords, h2hc conference (2009)
26. Rutkowska, J.: Evil maid goes after truecrypt! (2009)