Overview of BIOS Rootkits

Ross Schlaikjer (ross.schlaikjer@gmail.com)

December 13, 2013

Mentor Ming Chow

1 Abstract

One of the most critical systems on a modern machine is the BIOS, and yet it's inner workings are a complete black box. Theimportance of the BIOS combined with the fact that an end user cannot easily read or modify it makes it a difficult but rewarding target for exploitation, since it can transparently tweak how any OS installed on the computer behaves and persist no matter how often the computers drive is wiped. This paper will explore the methods through which a BIOS may be modified to bypass security in the OS that is running on the system.

2 Introduction

When it comes to recovering a system from a rootkit, virus or other malware, the nuclear option that is always on the table is to simply reformat and reinstall - no matter how persistent a software infection is, it cannot survive being completely written over. But the HDD is not the only place that persistent data can be stored on a computer. All computers have a BIOS or its EFI equivalent, and some models also have flashable embedded controllers dedicated to certain I/O tasks. If these are compromised, they could be used by an attacker to bug the system (for example logging keystrokes), override privileges or otherwise affect the OS.

2.1 Infection Methods

While one would think that it would be very difficult to write to something as important as the BIOS, there are plenty of applications that allow for writing, provided the user has sufficient privileges. If the BIOS attack is part of a larger infection, it is likely that the attacker has already infiltrated the system to the point where it is possible to operate as root. Writing an exploit into the BIOS can be used to guarantee reinfection should the system be wiped, or to redirect control from the OS kernel into malicious code, or even facilitate handing off to an entirely different kernel or OS by mapping disk pages to memory and then rewriting the IDT.

Attacks can be carried out without even rewriting much of the BIOS by taking advantage of ACPI, which replaces APM on most systems. ASL, the ACPI Source Language, is used to define handlers for devices and the control how the ACPI subsystem interfaces with the OS. To do so, it has a number of features, such as the ability to directly write system memory. If the location of kernel memory objects such as the syscall descriptor table are known, they can be overwritten. For example, the handler for undefined syscalls, which would normally just return -ENOSYS, could be overwritten with JMP %rbx, which would allow a normal user to execute code with kernel level permissions by calling syscall on a nonexistent call number with a function pointer as an argument. The function pointed to could setuid root and exec a root shell, compromising the system.

Since the BIOS is used for interfacing with all hardware while the system is still in unprotected mode (loading bootloader code, etc), it can also be used to surreptitiously modify or patch code that the system is loading. Again, this requires some knowledge about the target system, but one could override INT 13, the disk access functions, to check the data in each sector it reads, and if it matches a known pattern (say for example NTLDR) could instead return other data, resulting in a bootloader that looks fine on disk when the system is loaded (since high level disk access is done not through BIOS interrupts but more advanced device drivers), but is replaced at every boot with malicious code.

3 To the Community

The topic of firmware based exploits is relatively little-talked about, yes has huge potential for being used as a vector for hard to detect and hard to remove. With this paper I hope to raise awareness of the problem that is present in having unknowable but rewritable systems present in our computing devices. While the BIOS rootkit that grabbed headlines not long ago seemed outlandish (and does indeed appear to have been a hoax), users should not be lulled into believing that this is not a threat. In this day and age, it may not just be crackers on the internet trying to rewrite your BIOS but one government or another, since there is so much potential for gathering private information for a user with a compromised system.

4 Defenses

4.1 Hardware switch for BIOS flashing

Attackers without physical access to the machine could be prevented from modifying the BIOS or other firmware systems by including on the motherboard itself a DIP switch or jumper pins that prevent the BIOS from being written if in a certain position. Since even the best programmers cannot circumvent a hardware limitation, this would prevent any attacks that are carried out from the system itself. This is only effective, however, if people understand the necessity of ensuring that the BIOS is not rewritten maliciously. A lazy user could turn on flashing for a legitimate update and never turn it off, or even know to turn it on to begin with. It would also be a hurdle to pushing out prompt BIOS updates to users that are not comfortorable with opening up a computer and changing things. Form factor and accessibility would also be an issue in laptops and other mobile computing.

4.2 Firmware signing

In order to ensure that only legitimate code is installed on the BIOS, a second (non-programmable) subsystem could be included to verify that all BIOS images that a user attempts to flash are cryptographically signed by the manufacturer of the computer or authorized third parties. This is somewhat similar in principle to the idea of EFIs secure boot, and comes with a similar suite of problems - hobbyists and homebrewers will not be able to tinker (though admittedly the industry for BIOS tinkering is smaller and more dangerous than that of OS development), and the leak or cracking of the signing key would invalidate all protection immediately.

4.3 Open Source BIOSs

I believe that a significant part of why the BIOS is such a mystical thing, even amongst the computer literate, is that it is currently a black box, where updates go in and Phoenix only knows whats going on inside. If it were possible for users to see and compile for themselves the code that is present on their chipsets, it would be easier for people to not only spot bugs in the code itself but easily compile and install it themselves, making it much easier to clean up after an attack. Being able to compare checksums of the current BIOS to known hashes would also be useful for detecting issues.

5 Conclusion

While BIOS based attacks require that the attacker already have root on the system that they are modifying or have physical access to the hardware, they have serious power to alter the behaviour of a machine without the user knowing. EFI is unlikely to address issues of modifiability, and as a more complex software suite is more likely to simply have bugs of its own.

6 References

Implementing and Detecting a ACPI based BIOS rootkit, John Heasman
http://www.blackhat.com/presentations/bh-europe-06/bh-eu-06-Heasman.pdf

BIOS Based Rootkits, Wesley Wineberg
http://www.exfiltrated.com/research.php

ACPI: Design Principles and Concerns, Loc Duot, Olivier Levillain, and Benjamin Morin
http://www.ssi.gouv.fr/IMG/pdf/article_acpi.pdf

Advanced Persistent Attacks: BIOS Rootkit - Mebromi, Hamza Sirag, Nihant Bondugula, Rishabh Gupta
http://mason.gmu.edu/ msherif/isa564/fall11/projects/bios.pdf