A Note On High Integrity PC Bootstrapping

by Dr. Frederick B. Cohen ‡

In this paper, we describe two techniques for assuring a high integrity startup in a PC based computing environment. We begin with background information on PC startup procedures and current integrity threats against normal PC startup. We then describe a sound technique for assuring a high integrity startup and the basis for its soundness. Next we show a second method which is not sound, but which works well against attacks not specifically directed against this defense.

‡ This research was funded by ASP, PO Box 81270, Pittsburgh, PA 15217, USA

1 Background

In recent years, numerous computer viruses [1] and Trojan horses have been introduced to the computing environment [2], many of them designed to attack the DOS operating system at a very low level, so as to modify the bootstrapping of the operating system itself. In order to understand how these attacks operate and how to defend against them, we first describe the typical DOS bootstrap process in some detail [34].

1.1 How DOS Bootstraps

At hardware initialization, the 8086 (and subsequent similar microprocessors) begins executing instructions that start at memory location $FFFF0_16$ (normally a ROM location). This normally starts a hardware diagnostic, loads the first sector of the bootable disk into an arbitrary memory ares and jumps to it. This sector contains position independent machine code that checks to see if the disk contains a copy of the DOS operating system by looking for the first two operating system files in the root directory, and loads one or both of these files into memory, starting program execution at the entry point of the first of those files. If any of these steps fail, the bootstrap fails and the user is prompted to try again.

On most systems, the next step is for the executing program (the *'Basic Input Output System'*) to relocate itself into the lowest available area of memory, execute hardware dependent initialization code, and load the operating system into the next higher memory area. It them makes space available for I/O buffers, file control blocks, and installable device drivers as specified in the \config.sys' file. Finally, the command interpreter (\command.com by default) is loaded into memory and executed. \command.com then interprets the \autoexec.bat system initialization file, and after completion, starts interpreting user commands.

1.2 Common Variations on the Bootstrap Process

There are a few fairly common modifications to this bootstrap process used to include logical device drivers, operating system enhancements, special interfaces, and security mechanisms that have a considerable impact on the normal startup procedure.

The lowest level protection software modifies the 1st sector of a disk so as to change the entire bootstrap process. This technique is normally used to protect a disk from external access, and to provide automatic disk or file encryption. It has the advantage of being so

low level that most attackers are unaware of it, and it forces an attacker to spend time and effort to manually trace through enough of the protection mechanism to access the disk.

No widely available legitimate software that we are aware of modifies the rest of the DOS bootstrap process prior to the loading of device drivers. We believe this is because the device driver loader is adequate to fulfill any legitimate needs, and since each version of DOS has different operating system files, it is quite difficult to assure portability for a modification at an intermediate level.

Device drivers are quite common. The most popular device drivers replace the default DOS device drivers activated through the "interrupt vectors" used to call DOS and hardware services, by replacing default interrupt vector addresses with the addresses of the replacement routines. The drivers remain in memory and pass control back to the loading program. When subsequent device I/O or DOS calls are performed, the replacement drivers intercede. This technique is commonly used for modified keyboard operations, memory management, access control, transparent encryption, and network I/O.

After device drivers are loaded, the command interpreter is run. Only a few products modify the command interpreter because there are so many DOS based programs that require the default interpreter. It is quite common to modify the system initialization process in the \autoexec.bat file.

1.3 How Attackers Modify the Bootstrap Process

The lowest level attacks modify the boot sector [2]. This is most easily done by copying the previous boot sector onto another area of the disk and installing a memory resident operating system modification. Upon bootstrap, the attack code is loaded and control is then returned to the original bootstrap program.

No widely spread attacks modify the operating system itself for the same reasons legitimate programs tend to avoid this part of the process. It is relatively unreliable and difficult to do correctly. Very few attackers modify device drivers themselves because they too are difficult to modify safely, but it is certainly feasible to load independent device drivers in this part of the bootstrap process and we should expect this to be used for attack.

It is common for attackers to modify the command interpreter because this is a normal DOS program and is thus far better behaved than most of the programs in the bootstrap process [2]. Furthermore, the same command interpreter is used by virtually every DOS user. Attacks that modify \autoexec.bat are somewhat less common, presumably because they are more easily detected and bypassed by people.

Beginning in 1989, virus and Trojan horse writers became quite sophisticated in their attempts to bypass integrity products. For example, several attackers went to a great deal of effort to mask the presence of an attack by generating fake responses to system calls that would otherwise tend to reveal its presence [2]. Ultimately, such an attack could simulate the entire PC environment in order to avoid any sort of detection. Previous results have indicated that no defense can be reliable against such an attack unless there is a secure channel between the user and the defense and some part of the defense is non-modifiable [5]. The only way we know of to provide this assurance in current DOS based systems is with hardware based protection or simulation technologies.

2 Sound PC Bootstrap Protection

The historical problems with hardware mechanisms are that they are very expensive to install and maintain unless they are designing into the system from its inception, and they tend to interrupt normal operation unless they are very well designed. Most systems prior to PCs had hardware mechanisms to at least protect the operating system from modification, but in the zeal to make the PC cheap and simple, operating system protection was widely ignored.

Simulation based defenses are systems that simulate a secure hardware facility. They normally do this by interpreting instructions rather than using the hardware execution mechanism. This dramatically reduces system performance and depends on the accuracy and integrity of the simulation system. Unfortunately, even a simulation based defense cannot operate properly if the underlying bootstrap process is not protected.

There is one built-in hardware based protection mechanism available on the PC which does not interfere in any way with normal operation and can be effectively used to assure that available defenses are sound against bootstrap attacks; the write protect mechanism on floppy disks. From our observations and numerous systems administrator reports, most users don't use their floppy disks once their hard disk is loaded with a bootable operating system except for rare backups and occasional loading of new software. Keeping a write protected bootable floppy disk in the disk drive therefore involves no considerable expense or inconvenience.

This hardware mechanism can be used to protect the entire bootstrapping process, and is commonly used by systems administration, maintenance, and security personnel to assure a known operating system state. The same mechanism can be effectively used in almost every DOS based system. Furthermore, by placing a select set of protection software on the write

protected floppy disk, we can assure to a very high degree that the system we are using is clean at bootstrap and that subsequent changes will be quickly and efficiently detected.

The technique for a PC is quite simple. We place the boot block, operating system files, device drivers, command interpreter, a customized initialization file, and the core of an integrity checking defense [5] on the floppy disk, write protect the disk, and use it to bootstrap the computer.

At bootstrap, a clean version of the operating system is assured by purely hardware means. The initialization file then uses the defense mechanism to establish a secure channel, perform self-test, verify that key elements of the hard disk are unmodified, transfer control to the hard disk, and allow normal operation to proceed. Since hardware protection is used throughout the initialization process, there is no way to corrupt the operating system until control is transferred to unverified programs on the hard disk. Similar results can be attained through ROM based bootstrap procedures.

To quickly review, we have a hardware based protection mechanism for the PC (and many other systems) that can be easily exploited for integrity protection using existing tools with virtually no added cost or inconvenience. This technique provides a secure bootstrap process, but it has several problems. The most damaging problem is that it requires procedural compliance by the user, which is ineffective [6] in many environments. Another problem is that the number and size of device drivers may exceed available floppy disk space. Bootstrap performance may also be substantially degraded because floppy disks are not as fast as hard disks. Device drivers often make assumptions about disks, and the default DOS command interpreter is designed to always load from the booted disk drive, which makes performance still worse. Most of these problems can be circumvented by simulating the ROM bootstrap process after performing startup checks of the hard disk, thus giving control to the hard disk bootstrap process after assuring its integrity.

3 Second Best PC Bootstrap Protection

As we see above, the best practice on a PC is to start with hardware based protection and try to extend the influence of the protection mechanism through software to the remainder of the operating environment. This is of course a general property of operating system protection as it exists today. We begin with a small trusted mechanism and try to extend its influence to the larger environment.

Bootstrapping from a floppy disk seems impractical or causes inconvenience in some environments. As an alternative, we have devised a less secure method called a "snapshot".

Snapshots are not new. In fact, they are very similar to a common recovery technique in fault tolerant computing [7], where we take periodic 'checkpoints' of a process state and 'rollback' to the last known state for continued operation in case of a failure. The difference in the case of a 'snapshot' of a PC during bootstrap is that we do not update the state information periodically for retries. Instead, we take a one-time 'snapshot' of the relevant state information, and restore from that state to assure that the known 'golden' state exists at a specific time in every bootstrap of the machine.

The snapshot should be placed as close to the end of the bootstrap process as possible in order to assure that as much of the process as possible is covered. In order to provide assurance, the snapshot should be placed in as secure a manner as possible within the process, so as to avoid accidental or malicious avoidance. In the PC bootstrap process, this is best done between loading the last device driver and invoking the command interpreter.

Assuming that the hardware and drivers are unchanged between bootstraps, a snapshot taken at this point should be restorable at the same point in every bootstrap process without changing any vital portion of the system state. If we restore from this snapshot and then immediately check the integrity of all bootstrap related files, we can provide a high degree of assurance that no operating system modification has been made unless it explicitly attacks the snapshot mechanism. Furthermore, we can reliably correct these operating system files at this time using integrity shell techniques [8], thus providing a reliable use of redundancy for integrity.

In experimental use, a snapshot based PC protection mechanism has been in use for some time, and such a mechanism has recently been released into the wider environment with no ill effects. It appears that this technique successfully counters all attacks relating to the operating system initialization process except those explicitly directed towards the defense itself.

A partial snapshot similar to this technique has been in widespread use for detecting and recovering from corruptions of key components of DOS since 1987, but because the designers of DOS have explicitly stated that any portion of the resident DOS operating system may legitimately change at any time, it is impossible to reliably use this technique for periodic operating system assurance except in the bootstrapping process. Even in DOS based systems running on CPUs with hardware capabilities for operating system protection, the designers of the operating system have refused to use the protection to provide any assurance for the operating system in memory.

Finally, we note that a high speed secure bootstrap from the floppy disk may be used once a snapshot has been established. To implement this, we take a snapshot the first time we use the operating system. We then save the snapshot on the floppy disk, and create a specialized floppy based bootstrap that loads PC memory and registers from the snapshot

and continues from that point. This is far faster than the standard PC bootstrap, and provides a high degree of assurance that the system is in a known state after bootstrap.

4 References

1. F. Cohen, 'Computer Viruses - Theory and Experiments', DOD/NBS 7th Conference on Computer Security, originally appearing in IFIP-sec 84, also appearing as invited paper in IFIP-TC11, 'Computers and Security', V6#1 (Jan. 1987), pp 22-35 and other publications in several languages.
2. E. Wilding, Ed. 'The Computer Virus Bulletin', 21 The Quadrant, Abingdon Science Park, Oxon, OX14-3YS, England, ISSN#0956-9979
3. J. Mueller and W. Wang, 'The Ultimate DOS Programming Manual', Windcrest Books, Blue Ridge Summit, PA 17294-0850, USA, ISBN#0-8306-3534-3
4. R. Duncan, 'Advanced MS-DOS', Microsoft Press, 16011 NE 36th Way, Box 97017, Redmond, WA 98073-9717, USA, ISBN#0-914845-77-2
5. F. Cohen, 'Models of Practical Defenses Against Computer Viruses', IFIP-TC11, 'Computers and Security', V7#6, December, 1988.
6. F. Cohen, 'A Short Course on Computer Viruses', ASP Press, PO Box 81270, Pittsburgh, PA 15217, USA, 1990, ISBN#1-878109-01-4
7. D. Siewiorek and R. Swarz, 'The Theory and Practice of Reliable System Design' Digital Press, Bedford, MA 01730, USA, 1982, pp172-174, ISBN#0-932376-13-4
8. F. Cohen, 'Automated Integrity Maintenance for Viral Defense', IFIP-TC11 'Computers and Security', 1990
9. F. Cohen, 'The ASP 2.5 Technical Users Manual', ASP Press, PO Box 81270, Pittsburgh, PA 15217, ISBN#1-878109-12-X