Isolating Drivers without Tears

Nathanael Paul David Evans

Department of Computer Science
University of Virginia
Charlottesville, VA
{nate, evans}@cs.virginia.edu

A fundamental problem with the current operating systems design is that device driver code executes in the kernel address space with the same level of permissions as the rest of the kernel. Vulnerabilities in device drivers can be exploited to manipulate kernel data structures, and bugs in device driver implementations often lead to system wide crashes. Swift et. al, found that 85% of Windows XP failures are due to flaws in device drivers [SBL03]. Seventy percent of the Linux 2.4.1 kernel is device driver code, and device driver code was found to be three to seven times more error prone than the rest of the kernel [Cho01]. The security advisories database at securityfocus.com reports 51 incidents related to device drivers since April of 2003. When a user-level process calls a device driver executing in kernel space, it presents attackers with an attractive target for privilege escalation. Increasing emphasis on security and reliability has led operating systems designers to seek ways of executing drivers more securely. Moving drivers out of the kernel has obvious security advantages, but serious practical and performance challenges. All current mainstream operating systems, including Windows XP and Linux, run drivers in the kernel for performance reasons.

Substantial research has focused on developing general mechanisms for supporting general purpose kernel extensions, but little consideration has been given specifically to device drivers. In fact, the performance requirements and kernel access requirements of device drivers are typically quite different from those of general purpose code. Hence, we believe there is an opportunity to provide support for secure and efficient device drivers without the dramatic redesigns or performance penalty needed to support general kernel extensions.

By recognizing the specific requirements of common device drivers, we introduce a new approach to providing isolation we call *driver space*. Our design seeks to isolate driver code from the rest of the kernel by combining software and hardware mechanisms, while introducing as few changes as necessary while providing a fast interface to user code without sole reliance on user space isolation. Driver space accomplishes isolation by using the intermediate protection levels provided in the Intel x86 architecture, and we will implement an environment for type safe drivers in kernel space. By dividing drivers between protected, type safe code that runs directly in the kernel, and unprotected, type unsafe code that runs in driver space, we endeavor to provide a migration path towards type safe drivers. Legacy drivers run entirely in driver space, but code fragments that are rewritten in a type safe way can move into the kernel. Our approach is similar in spirit to the one proposed by Windows Longhorn (Microsoft's next major operating system release, expected in 2006) which will restrict legacy device drivers from using hardware acceleration, so device vendors will be encouraged to rewrite their drivers in user space [Lan03, Pur03]. By providing efficient mechanisms for isolating device drivers from the kernel, driver space offers the potential to greatly improve the reliability and security of mainstream operating systems.

References

[Cho01] A. Chou, J. Yang, B. Chelf, S. Hallem, D. Engler. *An Empirical Study of Operating Systems Errors*. ACM Symposium on Operating Systems Principles (SOSP), October 2001.

[Lan03] B. Langley. *Graphics Drivers For Windows Longhorn*. Windows Hardware Engineering Conference (WinHEC), August 2003.

[Pur03] M. Puryear. *Windows Longhorn Audio Architecture*. Windows Hardware Engineering Conference (WinHEC), August 2003.

[SBL03] M. Swift, B. Bershad, and H. Levy. *Improving the Reliability of Commodity Operating Systems*. ACM Symposium on Operating Systems Principles (SOSP), October 2003.