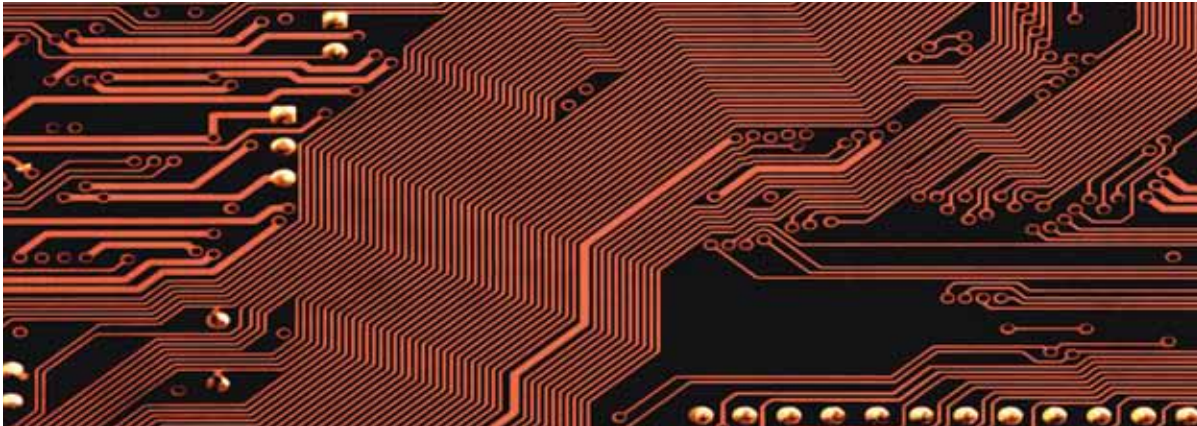


# SOFTWARE INSECURITY

By Rik Farrow



While awareness of software security principles has improved tremendously over the past ten years, computer systems have remained exploitable. The reason for this is simple: the ability to write software is common, while the ability to write secure software is very rare. As this is unlikely to change, the future of computer security requires approaches that allow most programmers to write more securely, as well as isolating their code so exploitation results in less harm.

We have become accustomed to routine announcements of software updates that address security issues, with Microsoft's Patch Tuesday being the most recognized example. Apple and other vendors often bury security fixes in an effort to make their software appear more secure, when it is merely less commonly used. Enterprise systems companies do even worse, with Oracle announcing security patches only twice a year. And every vendor knows about exploitable security issues that remain unpatched for years because doing so would break other software, or destroy backwards compatibility.

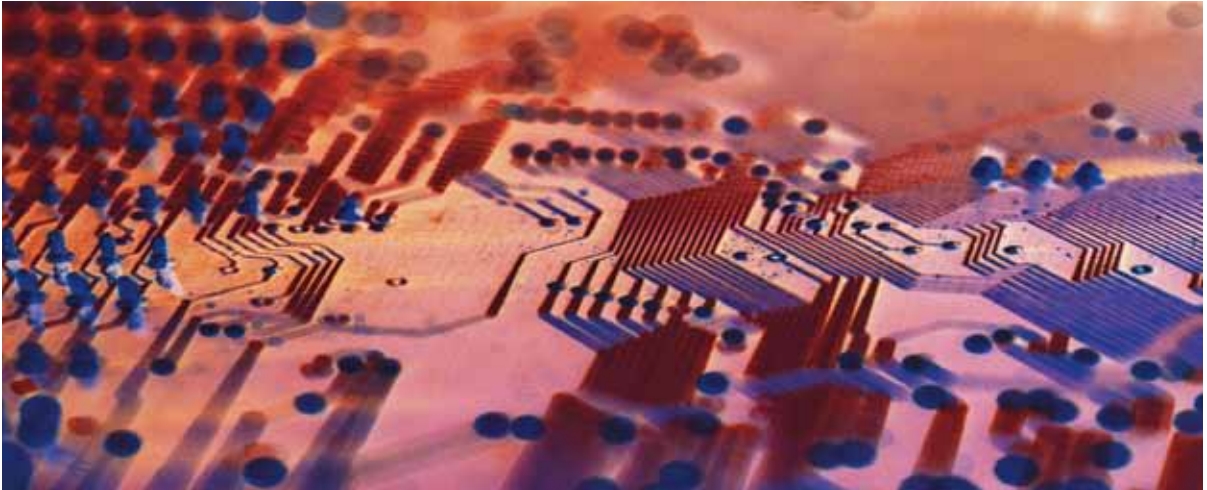
The anti-virus industry is thriving, but doomed to fail. The users of malware modify the code they send out on the order of hours, making signature-based analysis of potential malware only useful for viruses that have been around for days or weeks. Heuristic-based anti-malware does exist, and has a much higher success rate because it looks for behavior that indicates malicious activity rather than signatures. But heuristic-based defenses are rarely used because of their significant impact on performance. Anti-virus/anti-malware is a reactive form of security that can

slow infection rates by detecting attacks that were new days ago.

## History

Like any other industry, designers of computer hardware and software have followed familiar paths, and this is a large cause of the problems we face today. Our security is based on models designed for time-sharing systems, where the goal was preventing one program from crashing the entire computer. A secondary goal was to limit user access to only files and resources that that user had been granted permission. While these are wonderful goals, and certainly desirable, they poorly match our current use case.

Creating computers that don't crash when used relies on two hardware features, also part of the time-sharing heritage: privileged mode and memory management. Privileged mode is reserved to the operating system itself, while memory management sandboxes running programs, preventing them from interfering with other programs and the operating system.



The most commonly used computers today — PCs, laptops, and the many mobile computing devices — are used by single users. That makes time-sharing security totally inappropriate for today's uses, as it is the user's own activity, web browsing, messaging, reading imported documents, that is the source of exploitation. It has also been common to grant the user administrator privileges, making actions of the user capable of exploiting the operating system as well.

### **A New Start**

While it would be wonderful to be able to start over, there is an enormous investment in current software, as well as familiarity in how to use software. What we will see are frameworks that make using existing software more secure, and provide support for new software that is built to use this framework.

There are already examples of web browsers that isolate principals, where a principal represents an Internet domain that is the source of the code and data that supplies web pages. Microsoft's IE8 and Google's Chrome are the first commercial examples, while the research browser Opus2 has gone farther with isolation. These web browsers are using memory management and software techniques to isolate (sandbox) user activities, so that visiting a website that is infected with malicious code will not affect all of the user's future activities.

Principal isolation in web browsers represents an approach that is layered on top of our already current systems. What will work much better are systems that are designed so that principals, whether they are websites or the source of classified documents, remain isolated even within the same computer. Past

attempts at doing this, such as Mandatory Access Control (MAC) systems, have proven unwieldy to manage and use, and work poorly because of the dynamic nature of networked access. I expect that we will see operating systems and hardware that provides this isolation as a primitive, not as something layered on top of already overly complex systems.

Virtual machine monitors (VMMs) and microkernels come close to providing the proper level of support for securely encapsulating applications and data from different principals. Virtual machines, as currently used, are much too heavyweight, as running a single program within the protection of a virtual machine (VM) requires running an entire operating system and all the supporting libraries. One group of researchers has created MirageOS, a minimal operating system running on top of the Xen VMM, permitting a much leaner approach for isolating applications using VMs. The MirageOS approach works for servers, but is a poor match for user applications that share a single display, keyboard, and mouse. And VMMs themselves are large and complex, and the source of vulnerabilities that compromise not only themselves but all VMs running within them.

Microkernels represent a similar, but different way, of isolating principals. True microkernels rely on a very small trusted code base, giving them an inherent advantage over today's operating systems. Microkernels banish device drivers from the operating system's privileged mode, making any computer using microkernels more robust and secure right from the start. Other system services, such as access to filesystems (directories and files), are also treated like user programs, and run in isolation.

Microkernels use message passing for communicating between the different parts of the operating system, and this support lends itself very well to running applications designed with principal isolation in mind. SeL4 (OK Systems) represents a microkernel already in commercial use, while Barrelfish and Minix3 are research microkernels that are recent developments. Barrelfish is focused on running applications on many-core computers and has already matched the performance of current high-performance, monolithic operating systems.

Interest in microkernels waned after the late 1980s because of poor performance. Hardware has changed, and experimental manycore processors, like Intel's Single Chip Cloud (SCC), have made message passing, the Achilles heel of microkernels, a very fast, hardware supported feature. Building an operating system that runs on hardware like the SCC supports the ability to run many programs from many principals, using true hardware isolation, while taking advantage of fast on-chip communications.

I expect to see first-class support for message-passing to improve. First class support means that microkernels will have CPU hardware that supports microkernel design principles of isolation of operating system services and device drivers without sacrificing performance. As operating system researchers and vendors are accustomed to dealing with the old, mainframe-like CPUs, I know this change will take time and will be best done by smaller, nimbler groups and firms that have less investment in legacy systems.

## Isolation

New operating systems and hardware in themselves will not deal with the problem we have today — one user running insecure programs exposed to data and code from many principals. Bridges that allow running current software securely on top of a more secure base will be developed.

Existing software will be run in true isolation, consisting of the software necessary to run a particular application running in a VM or over a

microkernel. New computers, both desktops and smartphones, will be like having many computers connected together over a switch (KVM) to the input and display devices (keyboard, mouse, and screen). Each computer will operate independently of others, and not interfere with the input destined for other applications, all because of CPU support for isolation using message-passing. Researchers will need to solve two problems (beyond the design of efficient CPU hardware): one technical and the other financial.

Running a Microsoft application like Office requires a large number of support libraries, called DLLs in Windows and shared objects in other systems. In current systems, once a DLL or shared object is in use, its code gets shared with all running programs that need that support library. Sharing libraries means that less memory gets used, an important goal in all systems.

If isolation is used, each isolated CPU would have its own copy of each library, resulting in an increase in memory use. VMMs, both commercial and open source, already have mechanisms for deduplicating memory so that duplicate copies of libraries would only exist once in physical memory, solving that problem. This solution will be used in a microkernel approach as well.

The second issue, financial, requires a political solution. Running Microsoft Windows applications in separate VMs or isolated CPUs today requires an operating system license for each VM or CPU. This will be changed, given the appropriate amount of persuasion.

## Conclusion

In the present time, the best security practice requires the capture and storage of all network traffic. Because a successful attack today might not be recognized until days or weeks later, knowing which systems received the attack from the network is very important, even if it is also a bonanza for storage vendors. In the longer term, I expect that systems that isolate principals, instead of users, will become the foundation for secure computing. **Q**

---

**Rik Farrow** has worked as a consultant, instructor, and author focusing on computer security since 1984. He wrote the second book on Unix security, and has designed and taught courses for U.S. government agencies. Mr. Farrow is also the editor of ;login., the magazine of the USENIX Association for Advanced Computing.