Where Does Security Stand? New Vulnerabilities vs. Trusted Computing

Shay Gueron
University of Haifa and Intel Corporation

Geoffrey Strongin
Advanced Micro Devices

Jean-Pierre Seifert
University of Innsbruck and Samsung Electronics

Derek Chiou
University of Texas at Austin

Resit Sendag
University of Rhode Island

Joshua J. Yi
Freescale Semiconductor

How can we ensure that platform hardware, firmware, and software work in concert to withstand rapidly evolving security threats? Architectural innovations bring performance gains but can also create new security vulnerabilities. In this panel discussion, from the 2007 Workshop on Computer Architecture Research Directions, Shay Gueron, Geoffrey Strongin, and Jean-Pierre Seifert assess the current state of security and discuss possible routes toward trusted computing.

Moderator's introduction: Shay Gueron

In any discussion of the PC platform, the most important point to remember is that it is intended to benefit users. The PC experience today, as it is viewed by our users, is the result of several components operating together, with each component playing its own unique role.

The hardware and, in particular, the processors provide the "compute power." This capability involves two factors—architecture and microarchitecture. The quality of the processor architecture is measured by the flexibility and usability of the instruction set that it offers. The instruction set provides building blocks for use by the software. The quality of the microarchitecture is measured by the efficiency from the performance, power consumption, and cost.

Processors have various design vectors—for example, power, performance, debugging, and cost. Security is one of the factors in the equation, and the processor is a critical component of an overall platform solution to security. Innovations in the microarchitectural features improve the processor's quality from the power/performance viewpoint. Two obvious examples are the caches and the branch predictors that are parts of any modern processor, where the different micro-implementations provide the competitive edge for each processor. On the other hand, security is a bit different: The security of the platform is the result of cooperation between the hardware, firmware, and software on the platform. Therefore, all of the platform components must work together to deliver security.

When facing a security issue, it is necessary to find the best place in the platform to address it. For instance, if an issue affects only a few applications, it is probably better to rewrite those applications

0272-1732/07/$20.00 © 2007 IEEE Published by the IEEE Computer Society

than to make a hardware change that could affect performance of all applications. In some cases, a processor's features can solve security problems. Indeed, various chip manufacturers have added security features to processors over time—for example, ring protections and the eXecute Disable (XD) bit.

Side-channel attacks

Consider the most recent type of side-channel attack in the news—the branch prediction analysis (BPA) attacks.[1] These attacks exploit the branch prediction mechanism. Processors improve performance using branch prediction to predict the most probable code path to execute and then fill the pipeline with the corresponding instructions. Sometimes a processor mispredicts a branch, requiring it to restart from that branch once it detects the misprediction. Good branch predictors are essential to achieve high performance in modern microprocessors with deep pipelines and multiple-instruction issue. Performance, however, varies depending on whether a branch is predicted correctly or incorrectly; an incorrectly predicted branch means the processor must roll back the effects of instructions after the mispredicted branch and restart at the that branch. The BPA security attack capitalizes on that performance variation. A spy process running on the target machine, together with another application process, can use these timing differences as side-channel information and deduce the precise execution flow performed by this "victim" process.

This could potentially lead to a complete break of a cryptosystem—but under what exact circumstances? Such a theoretical exploit could occur only under all three of the following circumstances:

- The malicious code (spy) was allowed to run on the system either by some operating system (OS) automatic upload or by the user inadvertently launching it.
- The victim cryptosystem is written in such a way that knowledge of the execution flow provides the attacker with information that lets him compromise the security.
- The OS supports multiprocessing, and the scheduling mechanism enables a sophisticated ping-pong game between the spy and its victim.

Here is an obvious way to eliminate the problem: Processors and operating systems do not multiprocess, and processors do not use branch prediction. It's not difficult to guess that such solutions would not be very attractive to the user, who—remember—needs to get services from the platform. This raises an important question: Will users be willing to give up some of the platform's performance capabilities and the multitasking user experience, just to solve a (theoretical) threat that can be easily handled by the software?

Indeed, in the case of BPA attacks, careful analysis indicates a potential vulnerability in some unexpected part of the RSA algorithm.[2] However, this analysis shows that a change of algorithm in the crypto-applications that carries a minimal performance penalty (and potentially even improves that performance) can be simply deployed to solve the problem. The software-algorithmic change has already been offered (by Intel) to the open-source community.

Another educational example is a potential vulnerability in OpenSSL 0.9.8 in the "exponent scanning" phase,[2] where the code implementation writes

```
return((a->d[i]&
  (((BN_ULONG)1)<<j))?1:0);
```

Here, the branch could be exploited if the compiler does not perform the optimization to eliminate it. But by just replacing the line with

```
return((a->d[i] >> j) & 0x01);
```

the problem is eliminated, regardless of the OS scheduling properties and the particular processor. This demonstrates how a programmer who is not aware of all security implications in crypto-programming can inadvertently insert a potential vulnerability through one innocent line of code.

About this article

Derek Chiou, Resit Sendag, and Josh Yi conceived and organized the 2007 CARD Workshop and transcribed, organized, and edited this article based on the panel discussion. Video and audio of this and the other panels can be found at http://www.ele.uri.edu/CARD/.

Virtualization and trust boundary

Attacks that exploit the hardware must first successfully exploit the OS to obtain ring-0 privilege. But, if the attacker obtains this privilege, he can in fact perform many malicious acts that do not involve the hardware. Thus, major efforts should be concentrated on preventing ring-0 access. As long as the OS is using the virtualization layer, it's not possible for an adversary to install *rootkit malware*. (A rootkit is a tool that allows administrators access to maintain or control a computer system, without having the user be aware of it.) With a rootkit, it's possible to execute files on the target machine, change the system configuration, access and edit system log files, and monitor computer usage. The main difference between a rootkit and any other management tool is that on top of having root-level privileges, a rootkit is also undetectable (to the user). Rootkits are used for multiple legitimate purposes, such as law enforcement and monitoring of employees' activity in an organization. Parental supervision is another legitimate usage for a rootkit. It is clear, however, that a malicious rootkit (rootkit malware) is a very serious security and privacy threat if it gets installed on a user's platform.

One recent example is the Blue Pill attack, in which the rootkit uses virtualization to go under the OS. Software solutions could most likely detect such activity—for example, by having the OS monitor the time differences in execution with and without a hypervisor.

Buffer overflows

The most dominant software attack, the so-called *buffer overflow attack,* capitalizes on pushing attack code onto the C function call stack. Simply marking the stack area as nonexecutable with a special bit (called the NX or XD bit) eliminated many of those very simple software attacks. Thus, the XD bit is an example of a *hardware-based assist* for a problem that is inherently a software problem. However, even before the XD bit was implemented, the security community realized that it could protect only against the exploits that existed at that time, and not against all potential buffer overflow vulnerabilities.

Software (OS) vendors are also working to prevent buffer overflows; Microsoft's canary bit, which can detect some overflows, is an example.

Toward a united security strategy

The PC platform has made tremendous advances in the past 20 years, owing to great progress in architecture, microarchitecture, OS, and software. However, from the security perspective, the PC platform has become a target for attacks, and the problem today seems to be worse than it was 20 years ago. Attackers have an advantage with their ability to develop 0-day exploits versus the time it takes for patches to be widely installed. The solution to this problem cannot come from a single source. The PC platform community needs to step back and look at how to solve the larger problem first, and then determine the roles each provider can play in the solution.

All players in the PC platform community need to work together and improve the platform's quality from all aspects, including security. The processors need to continue improving computational efficiency. It is the role of software to guard against malicious attacks. The PC platform will continue to evolve, and the user experience can benefit through the cooperative efforts of the industry.

Position statement: Geoffrey Strongin

Only recently has the mainstream computing community recognized security as one of the biggest issues facing computer architecture. Three decades ago, only a few security experts were thinking about how to make general-purpose computing more secure. Today, virtually everyone, including the cashier at Starbucks, knows that computer security is a serious issue. We are, however, far from solving the problem.

Though trusted computing has been the focus of a dedicated group of researchers who appear to have left very few stones unturned, until there is a significant commercial deployment we have no way of measuring the actual progress and the actual benefits. AMD is working on the third

iteration of trusted-computing hardware support, yet there is still no commercial OS support for trusted computing. Other microprocessor vendors are in the same boat.

We need to look pragmatically at what trusted computing is capable of accomplishing and what it cannot accomplish. The current direction of trusted computing implementations is to leverage secure virtualization technology as the core technology for creating isolation between guest partitions. The problem is that the guest partitions are themselves large monolithic targets. The virtualization approach does not improve the security of the guest itself; each guest remains vulnerable. Secure virtualization may allow other guests to continue to function when one is corrupted, but this does not help if there is critical data within the guest that is corrupted.

We need more robust protection tools. For example, some viruses kill tool alarms, essentially creating silent failures of the security programs. One proposed technique to improve guest OS security are tamper-resistant external health monitors living outside of a guest OS that provide higher assurance warnings of a penetration or failure inside a guest. Such research is promising and should be pursued.

More importantly, we need to look at how to provide a finer level of protection within a given guest. Most legacy operating systems are monolithic, contain millions of lines of code, and have a simple user-mode/kernel-mode view of protection. If we could rewrite operating systems, security could be done much better—but we cannot, because of legacy constraints.

One strategy to incrementally improve security is to partition a legacy monolithic operating system into smaller submodules that are internally compartmentalized to effectively introduce firewalling between those submodules. Such partitioning could be assisted using the virtualization layer, or it could be accomplished by other techniques. I think that a critical area of combined hardware and software research is to develop workable solutions that allow for the effective decomposition of the monolithic OS kernels without incurring the massive code rewrites that would make this logistically impractical.

Position statement: Jean-Pierre Seifert

Despite massive trust and security engineering efforts by the Trusted Computing Group (TCG) around the PC architecture, new potential security vulnerabilities of the x86 architecture have attracted much recent research activity and vibrant public discussion. As a few vulnerabilities that potentially could undermine TCG's general computer architecture security efforts, consider the following:

- Virtualization hardware support could function as a tool for undetectable malware—that is, reversing the security promises of virtualization, as Figure 1 shows. (As Shay Gueron points out, a software solution could only *most likely* detect the potentially malicious virtualization. This likelihood depends on the perfection of the virtualization. A perfect virtualization could also maliciously virtualize the required timestamps to hide some timing differences. At least theoretically, such a distinguisher algorithm between a native and a virtualized environment is provably not existing.[3])
- New (old?) side-channel attacks could capitalize upon caches, branch prediction units, keystroke tables, and so on, potentially circumventing the TCG trust boundaries.
- Buffer overflow attacks could circumvent even the XD or No eXecute (NX) bit.

I also see several new challenges ahead:

- With an architecture not originally intended as a security-aware platform, security cannot be baked in afterwards—without dramatic architecture changes. Because security is not a cheap-and-simple add-on after the fact, the underlying architecture must be completely overhauled for security.
- Due to massive implicit and explicit parallelism deeply hidden in today's very complex miroarchitectures, the

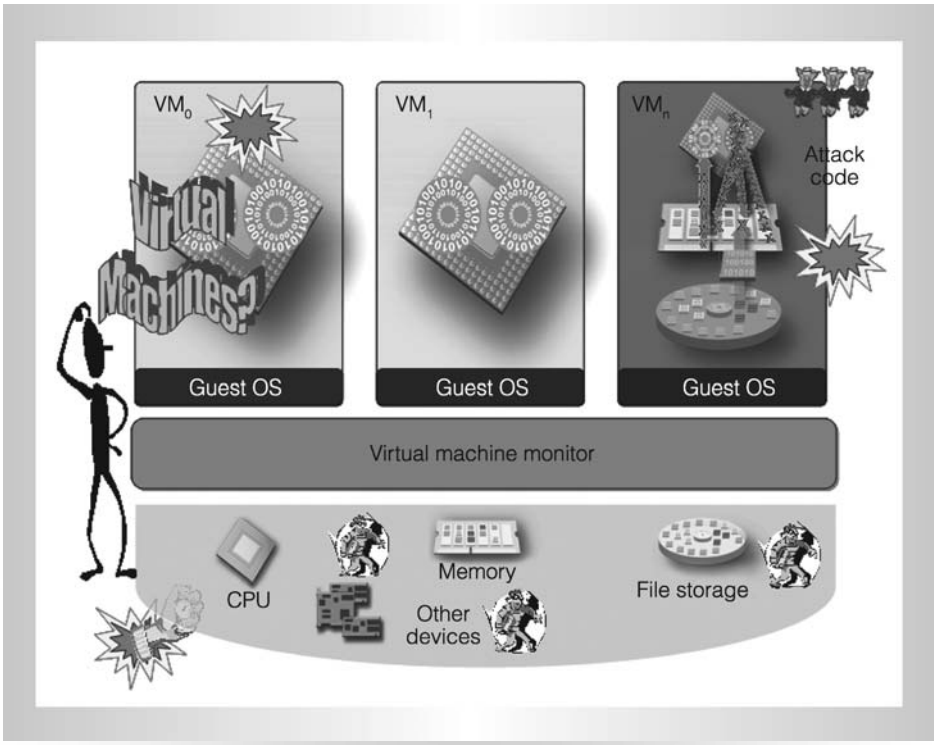

Figure 1. Undermining the trust boundary of virtual machines via side-channel attacks.

isolated execution requirement—that is, the confinement guarantee—must be carefully evaluated. To support Lampson's confinement[4] at the OS level, we must tag resources like caches, branch prediction units, translation look-aside buffers (TLBs), and so on at process-level granularity.

- The most dominant software attack—the so-called buffer overflow attack—capitalizes on pushing attack code onto the C function call stack. Marking the stack area with a special bit (called the NX or XD bit) as nonexecutable eliminated many of those very simple software attacks. This is a correct step but only a very tiny one in the direction of a capability-based computer-architecture paradigm.
- Although the potential security advantages of a capability-based computer-architecture are tremendous, its practical realization is a daunting research project. But, as evidenced by the amount we have modified the x86 architecture over the last 25+ years, the giant effort toward a capability-aware x86/x86-64 architecture is perhaps not unprecedented and thus not completely hopeless.

I agree with Geoffrey that trusted platforms are not currently available. There are two vectors of research in security: new vulnerabilities and new trusted computing. It is very hard to get trusted platforms today with built-in hardware support. We are also lacking software support. Because of the current lack of real systems, we cannot say anything about the real value of trusted-computing platforms. But we can already derive from the existing trusted-computing specifications their shortcomings and speculate on new attacks. As architectures introduce new mechanisms for reasons such as improving performance or improving security, new attacks are developed to exploit those new mechanisms.

For example, new, very sophisticated attacks use performance variations, such as cache eviction latencies and branch prediction timing differences, caused by microarchitectural performance enhancements. Some of these attacks are only academic

today, but so were buffer overflow attacks 20 years ago. Such attacks can overcome the known promises of trusting computing, especially in the presence of the isolation promise. There are also new attacks on virtualization hardware and other trusted-computing hardware being proposed by the security community. Thus, introducing virtualization hardware support could actually increase vulnerability vectors rather than reduce them.

As a specific example, the NX bit was recently added to the x86 instruction set architecture (ISA). Today, many known security vulnerabilities leverage that bit. It is important to examine the security implications of any proposed new hardware feature, because once it is in the architecture, it is very difficult to remove it.

Security is not only a processor manufacturers' problem; we do also need better trusted-computing software support. The operating system especially must be written to use the secure hardware correctly. To give you an example, let's consider the x86 timestamp counter instruction RDTSC. This instruction is the source of many new and old side-channel attacks because it is available in ring 3 (user mode), although it has been well known for some time that such an instruction *must* be a privileged instruction. However, this ring-3 availability can no longer be removed, as all major operating systems rely on the instruction's availability at the application level. Thus, this legacy architecture flaw is a clear software issue. To make it short, starting from a commodity standard architecture that was not designed for security, it is very hard to make little changes that make the processor secure.

Where should security be solved?

Strongin: Side-channel and covert channel attacks must be addressed at the operating system level, not at the architecture level. (A ''covert channel'' can be described as any communications channel that can be exploited by a process to transfer information in a manner that violates the system's security policy. Essentially, it is a method of communication that is not part of an actual computer system's design but can be used to transfer information to users or system processes that normally would not be allowed access to the information.) Such attacks cannot be easily addressed at the hardware architectural level. Systems become even more vulnerable when hyperthreading and multicore designs are introduced. All shared resources have vulnerabilities to side-channel attacks. There is no point in worrying about side-channel attacks unless you are already running a secure OS. The Rainbow series of specifications defined the requirements for implementing a secure OS, or trusted computing base, assuming it was designed that way from scratch. Unless the OS is secure, trying to add architectural features to close covert- and side-channel attacks is like trying to soundproof the walls of a barn whose front door, back door, and windows are all open.

Audience member: We had the same issues with the same solution—put it in the OS—30 years ago. Have we come very far?

Strongin: No, we have not come far. The Rainbow project concluded the same thing—namely, that the problem has to be solved in the OS. Hardware is inherently vulnerable.

Seifert: Security is not addressed in mainstream operating systems of today. The problem, however, cannot be solved quickly. We can isolate boxes from each other, but what can we do within a single box? We need more work within a single box. Perhaps we could use one of the many new cores in a multicore system to do security monitoring? The more functionality, however, the greater the probability of additional back doors.

Security cores

Gueron: The ubiquity and cost of the PC platform benefits the user. Using a core for security means it cannot be used for something else, which increases system cost. Software writers should provide security; hardware should just be as fast as possible.

Strongin: One environment checking another environment, potentially using a vir-

tualization layer, can be done on a single core. The binding of a virtual core to a physical machine is a different discussion topic. If we have isolated execution environments, then the issue becomes one of software overhead, software management, and the interfaces to let multiple tool vendors write compatible and interchangeable tools. This is an area that is potentially very fruitful for research for tool vendors and the academic community. To exploit and monitor these isolated environments, however, we need OS interfaces. Those same interfaces for monitoring become attack surfaces. We do not want to make things worse while trying to make them better.

Seifert: But, at some point, perhaps at eight cores, it might be an interesting academic project to try moving monitoring into one of the cores or to try implementing the monitoring in an external FPGA.

Strongin: I agree that exploration of these architectural directions is interesting and potentially useful. Keep the exploration at the logical level and avoid binding too early to physical hardware. What are the unique aspects of a security processor compared to other cores?

Audience member: Perhaps only certain cores should have special security capabilities.

Seifert: It is very difficult to provide security to a system with lots of chips and an OS. There must be some way to bring in new functionality, to fix some issue, to provide some update. Once you have an update facility, you have potentially created a big hole. Fixed functionality solves the security problem but eliminates the ability to change that functionality. A core with special capabilities opens potential security questions.

The user as a threat

Audience member: The user is part of the threat, even if there is hardwired security. How much should we pay—in dollars, functionality, and inconvenience—for security?

Seifert: Two years ago, there was a big company that decided to try to go back to big dinosaur computing with dumb terminals to improve security. They discovered that such an architecture kills mobility. We have gotten spoiled. Unless we focus on the hard problems, as Yale Patt said in the single-threaded vs. multithreaded panel, we cannot get good solutions.

Strongin: It is possible to build systems that are protected against the user. Preventing a user from using administrator privileges provides some protection. If a user has physical access to a machine, however, it is difficult to make that machine secure. Cryptocards that are used in banking servers are very resistant to attack but are very expensive—in the $10,000 range. Such cards are designed to resist user hacking, even with a user who has access to the machine.

Why is the user attacking the system? These days, it is mostly to break intellectual property protection such as digital rights management (DRM), to copy DVDs, and so on. It is a tough balancing act, however, because most users do not attack their own systems, and additional security just makes the system harder to use.

Seifert: Even with a good security infrastructure, a user can mistakenly misconfigure the system. Making the system more idiot-proof also burdens the average user to protect against the rare stupid user. If there is OS support, it shifts the borderline, but does not solve the problem.

Strongin: It is very difficult to make any system idiot-proof, since idiots are surprisingly ingenious and often do things that we never think about.

The costs of security

Gueron: Highly tamper-resistant security drives up the cost to the point where it is too high. Take, for example, password security. If passwords are required to have a capital letter, a lowercase letter, a number, and a symbol, and have to be at least 20 characters long, and need to be changed every week, passwords wind up being written down on sticky notes. Those sticky

notes get lost, requiring IT support to force the passwords. Overall, security is far less. Hardware tamper-resistant keys make a lost password a real problem. All security solutions have associated costs. Linking with the administration costs is the job of system vendors.

Seifert: I agree there must be a be a recovery system, to deal with a lost key, lost passwords, and so on, but such a recovery system, of course, opens up security holes. Even hardware and board testing facilities, such as JTAG, open security holes. This suggests another line of hardware research, which is, how can you fully test a secure system but keep it secure? You cannot lock down everything. The ability to test the system later is necessary. A back door is mandated. Thus, there are many new constraints and new issues for security. But, first things first: One cannot solve all of the problems at once.

Current microarchitectural hacks

Audience member: Here is a well-known example of a security attack: There was a paper published that demonstrated that you could get an RSA key by the way the RSA library was written.[5] Was that a micro-arch attack or an implementation attack?

Strongin: That was a problem of the OS running higher security-level tasks with lower security-level tasks. Such problems shouldn't happen if you're serious about avoiding side channels. You should never allow the simultaneous execution of processes of different security levels or different security domains. Even allowing the snooping thread to run is an OS architectural failure. Snooping threads of lower security should not be allowed to run simultaneously with the encryption thread.

Seifert: Last week, I got confirmation that the authors of the operating system took responsibility for it and want to ramp up research on such security holes. However, for a more complete and updated story on this fascinating topic of microarchitectural attacks, I refer the audience to a nice and recent survey.[6]

Gueron: The encryption library routine could have been written more robustly to avoid this problem. The hardware needs to provide maximum throughput. Software writers should be responsible for writing secure code. Every line of code needs to be scrutinized for security issues. For example, one should not use different routines for squaring and multiplication.

Seifert: I agree, it's the responsibility of the software community to write secure code. The hardware architecture gets more and more complicated to get more and more throughput. We must educate the software community about the hardware architecture and the consequences of doing certain things in specific architectures. Just a single bit changed in the microarchitecture could have significant consequences for the OS. One cannot simply take software off the shelf and expect it to be secure. There are no easy solutions. Any known solution is very people intensive. Is it possible to build an architecture that will be less vulnerable to such attacks? One suggestion is to buffer up the effects of instructions to avoid the cache attack.

Two levels of protection are just not enough

Strongin: We have to remove covert channels and side channels from our answer. We are talking about an OS vulnerability. The OS should keep other applications from running at the same time as crypto-applications. Maybe our current privilege levels are too coarse, looking to the security domain of the threads to ensure that only the user who can see the key can run while the key is in use. In a monolithic OS with just two modes, kernel and user as shown in Figure 2, the machine is completely open. Any kernel-mode thread can see everything else, including the state of other kernel threads and drivers. Nothing separates out the different drivers.

We need to develop architectures that can, without forcing a total rewrite, ensure isolation between kernel threads and drivers. The challenge is to provide such functionality without throwing out the OS and starting over. The original core x86 ISA had

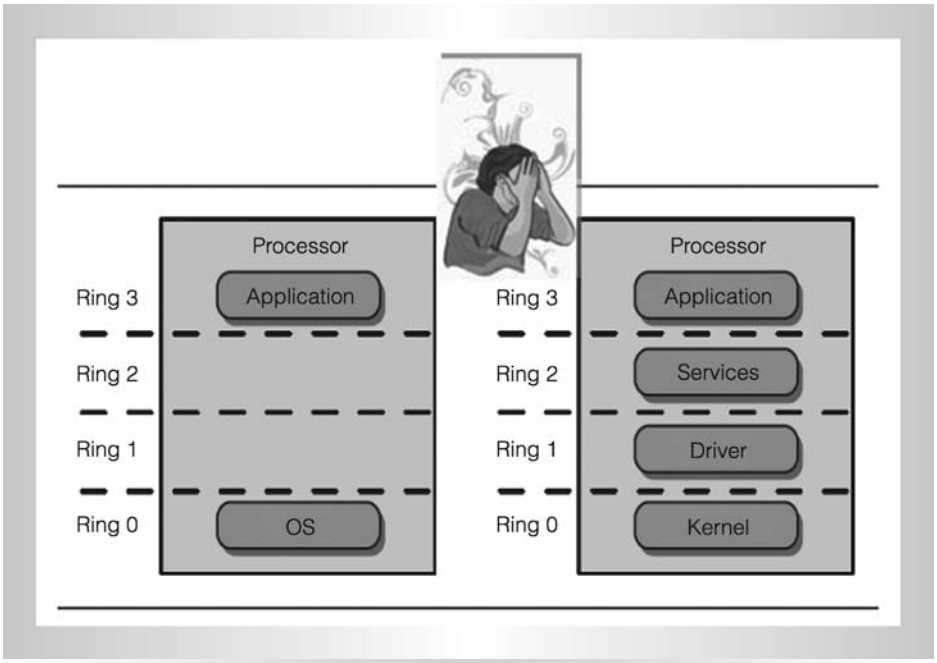

Figure 2. Operating system reality (left) versus intended ring use (right) in the x86 architecture.

the ability to isolate using segmentation, but that ability was not used as the transition to paging occurred. Reinventing segmentation, however, is not necessarily the right answer. We need to think about how to get closer to a secure architecture. Solving the monolithic OS problem can go a long way toward solving user-visible security issues.

Seifert: Though more security rings are available, only user and super-user are in use today in most standard operating systems. We need to find a transparent way to split the OS into modules that the hardware can handle. Ideally, such partitioning would be automatic and transparent.

Audience member: Hardware and software solutions have been around for a long time. It has not been done yet, because millions of lines of code would need to be changed and no one is willing to pay for it.

Seifert: Ideally, we can find an ingenious hardware solution to transparently solve the problem.

Gueron: Jean-Pierre and I wrote a paper on software mitigation for certain microarchitectural attacks.[2] I am on the side of software handling security, not hardware.

But let me elaborate a little bit more on this very interesting subject. Security holes are unexpected and often unforeseen. Figure 3 (taken from the paper I just mentioned[2]) illustrates one recent example. Figure 3 shows the information flow for computing modular inverse using the Binary Extended Euclidean Algorithm (BEEA). As seen, the branches in this flow are input data dependent. In a possible software side-channel attack[1] a spy process saturating the branch target buffer can track which branches are taken (or not taken) by the other (crypto) process that runs in parallel.

In addition, we proved that from having knowledge of these branches for a computation of greatest common divisor, GCD (u, v)—with u and v being coprime but *totally unknown*—one can compute both u and v (in polynomial time).[2] It turns out that OpenSSL (version 0.9.8a) used the BEEA for modular inversion. This modular inversion was used for masking the base during modular exponentiation, and this masking technique is critical for avoiding

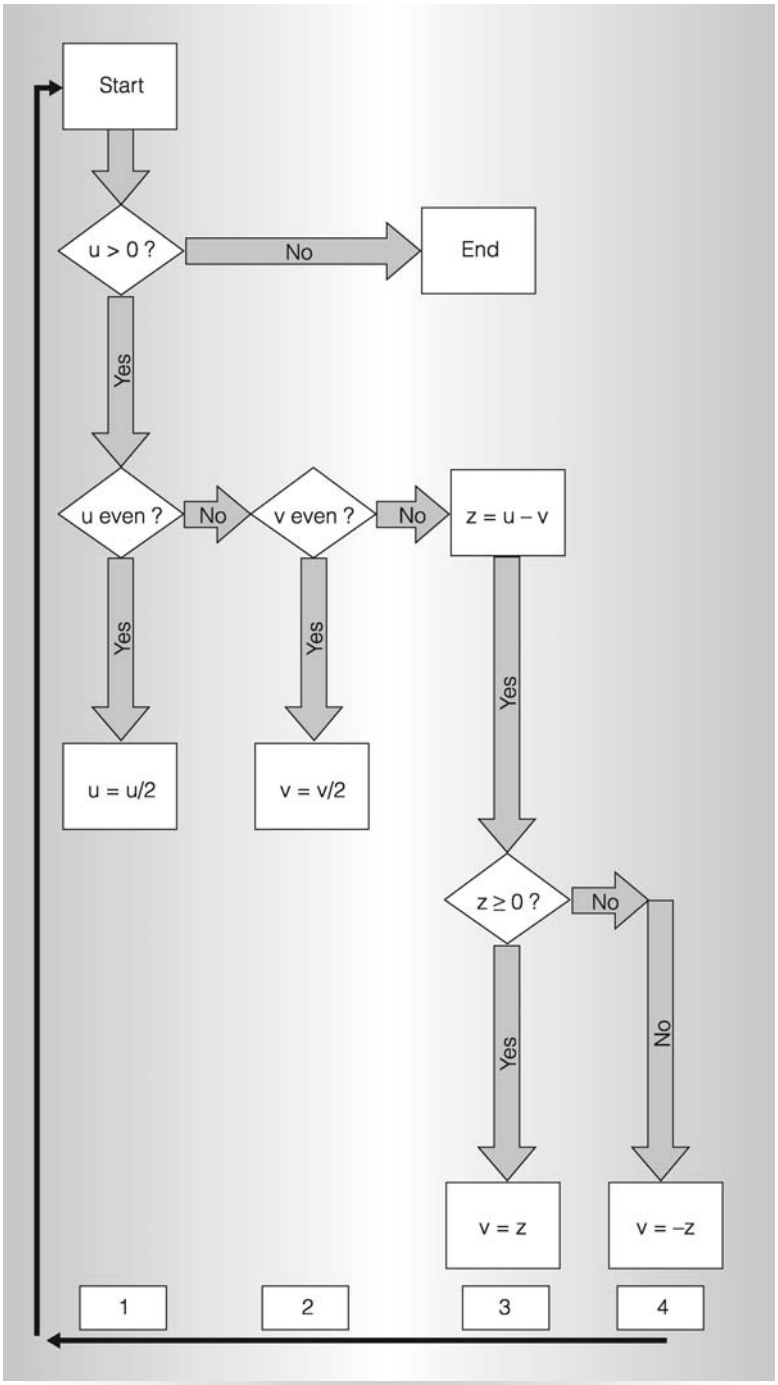

Figure 3. Information flow of Binary Extended Euclidean Algorithm (BEEA) (courtesy O. Acıiçmez, S. Gueron, and J.-P. Seifert).

some well-known timing attacks. Thus, under a side-channel spy-based threat model, an innocent looking algorithm (BEEA) can lead to compromising the whole system security. OpenSSL was notified about this vulnerability (by Intel Corporation, based on our input[2]), and the current OpenSSL implements some mitigations.

This surprising combination of facts teaches us a very important lesson. Probably, the best defense against current and future microarchitectural attacks is to let the cryptographic software community become aware of the security implications of writing cryptographic software that is going to be executed on throughput-optimized general-purpose processors. This will help software be written using a proper side-channel attack-aware methodology.

Seifert: In our paper, the final software solution was even faster than the original, insecure code. It is important to focus on the tough solution that will solve all of the problems. MICRO

References

1. O. Acıiçmez, C.K. Koç, and J.-P. Seifert, ''On the Power of Simple Branch Prediction Analysis,'' *Proc. ACM Symp. Information, Computer and Communications Security* (ASIACCS 07), ACM Press, 2007, pp. 312-320 (also available in the Cryptology ePrint Archive, Report 2006/351, Oct. 2006; http://eprint.iacr.org/2006/351).
2. O. Acıiçmez, S. Gueron, and J.-P. Seifert, ''New Branch Prediction Vulnerabilities in OpenSSL and Necessary Software Countermeasures,'' to appear in *Proc. 11th IMA Int*'l Conf. on Cryptography and Coding, LNCS 4887, Springer (also available at Cryptology ePrint Archive, Report 2007/039, Feb. 2007; http://eprint.iacr.org/2007/039).
3. S. Gueron and J.-P. Seifert, ''On the Impossibility to Detect Virtualization,'' unpublished manuscript, Aug. 2007 (available from S. Gueron, shay@math.technion.ac.il; or J.-P. Seifert, seifert@mi.informatik.uni-frankfurt.de).
4. B.W. Lampson, ''A Note on the Confinement Problem,'' *Comm. ACM*, vol. 16, no. 10, Oct. 1973, pp. 613-615.
5. C. Percival, ''Cache Missing for Fun and Profit,'' 2005; http://www.daemonology.net/papers/htt.pdf.

6. O. Acıiçmez, J.-P. Seifert, and C. Koç, ''Micro-Architectural Cryptanalysis,'' *IEEE Trans. Security and Privacy*, vol. 5, no. 4, Jul.-Aug. 2007, pp. 62-64.

Shay Gueron is a professor in the Department of Mathematics, Faculty of Science and Science Education, at the University of Haifa, Israel, and a security architect at Intel Corporation (Israel Design Center). Gueron's research interests include applied security, cryptography, and algorithms. He has a PhD in applied mathematics from Technion—Israel Institute of Technology.

Jean-Pierre Seifert is affiliated with the Universities of Haifa and Innsbruck and is directing Samsung's Trusted Platform Research Lab in San Jose. He previously worked for Intel and Infineon Technologies, where as a principal engineer he led research and development in the Smartcard and SecurityIC business unit. There, he received the Inventor of the Year award and initiated and managed several EU-funded projects.

Geoffrey Strongin is an AMD Fellow and AMD's chief platform security architect. He was a founding board member of the Trusted Computing Group. He has a BSEE degree from Arizona State University.

Biographies of **Derek Chiou, Resit Sendag**, and **Joshua J. Yi** appear on p. 24.

Direct questions and comments about this article to Shay Gueron, Department of Mathematics, University of Haifa, 31905 Haifa, Israel; shay@math.haifa.ac.il.

For more information on this or any other computing topic, please visit our Digital Library at http://computer.org/csdl.