Rethinking Computers for Cybersecurity

Article *in* Computer · April 2015

DOI: 10.1109/MC.2015.118

1 author:

Ruby B. Lee
Princeton University
266 PUBLICATIONS **10,120** CITATIONS

SEE PROFILE

Rethinking Computers for Cybersecurity

Ruby B. Lee, Princeton University

Cyberattacks are growing at an alarming rate, even as our dependence on cyberspace transactions increases. Our software security solutions may no longer be sufficient. It is time to rethink computer design from the foundations. Can hardware security be enlisted to improve cybersecurity? The author discusses two classes of hardware security: hardware-enhanced security architectures for improving software and system security, and secure hardware.

Computers are the engines of the Information Age, the machines behind all cyber activity. We perform financial transactions, access our medical records, and maintain critical infrastructures through cyberspace, using computers that range from smartphones to cloud-computing servers. Unfortunately, while our dependence on cyber transactions is increasing, so are cyberattacks. Yet security, which is essential for such cyber transactions, has not been seriously considered in the basic design of computers. How can this be, given our reliance on computers for our daily activities, our commercial competitiveness, and our national security?

One problem is that for decades, computers have been designed to improve performance and reduce energy consumption, cost, and size, while introducing compelling new features—but not to improve security. Currently, in cyberspace, attackers have the upper hand over defenders. An attacker only has to find one attack path into a system while the defender has to defend on all fronts. Once an attack path is discovered, it will work on a large majority of computers, due to their similarity. When security patches

0018-9162/15/$31.00 © 2015 IEEE

> SECURITY SHOULD BE IMPLEMENTED IN COMPUTER SYSTEMS WITHOUT DEGRADING THEIR PERFORMANCE, ENERGY EFFICIENCY, AND USABILITY.

are announced to fix vulnerabilities, users do not immediately install them, resulting in a surge of attacks exploiting these vulnerabilities.

Since there are so many possible attacks, a security defense strategy must be flexible and adaptable to new attacks and environments. This suggests security defenses implemented in software, since software is more flexible and adaptable than hardware. But why wouldn't a smart attacker target these software defenses first? Can we design hardware that first protects software security mechanisms, which then protect other software applications and systems? What strategies might we use for hardware-enhanced security, and what can we do to make hardware itself more secure and trustworthy?

PROMOTING SECURITY HEALTH

One strategy is to promote good *security health* in our computers. We know that the best way to prevent people from becoming sick is to ensure that they are healthy in the first place. Similarly, we should build mechanisms into our computers for good security health, better immunity against viruses, and the ability to quickly recover from attacks. This requires being proactive and building security into the basic hardware and software platform, rather than reacting to each new attack by patching the system.

However, we must have realistic expectations. There is no such thing as absolute security. As defenses get better, attackers also get smarter; it is an ongoing arms race. Moreover, mechanisms used for defense can also be exploited for new types of attacks. Hence, we should consider whether the advantages of a new security mechanism outweigh how it might potentially be abused.

Basic security properties and user concerns

So what sort of security should a commodity computer provide? Faced with many types of security breaches, what should our priorities be?

From the security perspective, the cornerstone security properties are confidentiality, integrity, and availability (CIA):

- Confidentiality breaches occur when secret or sensitive information is leaked to unauthorized recipients.
- Integrity breaches occur when information is modified by unauthorized parties, without detection.
- Availability breaches occur when legitimate users cannot get required or requested services and resources.

Providing these fundamental CIA properties is a first step toward building in security health. While it might not be possible to create an all-in-one solution, we can try to make it significantly harder for attackers to carry out security breaches. Furthermore, to gain market acceptance, security should be implemented in computing systems without degrading their performance, energy efficiency (important for battery life, particularly in mobile devices), and usability.[1] This is a tall order. Still, concrete security mechanisms can be built into computer architectures to achieve some of these goals.

From the user's perspective, surveys show that the top concern for both smartphone and cloud computing users is protecting private or proprietary data. Second is protecting important programs and data when all the surrounding software seems vulnerable to attacks. Cloud computing users have a third concern about adversarial virtual machines (VMs) running in the same cloud server.

Hardware's role in security

One possible way to address both cornerstone security properties and user concerns about security is through *hardware-enhanced security*. I will illustrate this by describing some security architectures developed at the Princeton Architecture Lab for Multimedia and Security (PALMS)—including DataSafe, Bastion, Hyperwall, and NoHype—which differ from typical software-only solutions in that they use hardware to improve software security and thus system security. Contrary to common misconceptions, hardware-based solutions need not be rigid and can adapt to new situations. The vision is to provide "general-purpose" security enablers for arbitrary security defenses, just as hardware processors provide the instruction primitives for realizing arbitrary software functions in general-purpose computers.

In addition, PALMS research suggests possibilities for *secure hardware*—how hardware itself can be made more secure. An example of this approach is Newcache, a secure cache architecture that is both fast and resistant to side-channel information leakage.

HARDWARE-ENHANCED SECURITY

How can a software–hardware architecture provide protection for data even when it is used with untrusted third-party programs? How can new hardware features provide secure

environments for executing software security protection mechanisms, even when other software applications and the operating system itself are untrusted? And how can cloud computing be made more secure for customers? Problems like these can benefit from hardware-enhanced security.

SELF-PROTECTING DATA

One common scenario today involves users downloading third-party software applications (called apps) into a smartphone via the Internet. Such apps could be free or cost as little as a couple of dollars. You have no idea what the software really does, but you are willing to give the app access to private or sensitive data because you believe the functions the app performs on the data will be useful to you; if you are diabetic, for example, the app could present graphs of your insulin measurements. But you certainly do not want this private information leaked into cyberspace.

Some app writers might inadvertently introduce security vulnerabilities in their code that attackers can exploit. Alternatively, malicious app developers could hide malware that leaks user information or inserts backdoors into the smartphone. How, then, do you allow an unvetted app to use your sensitive data constructively and yet prevent it from leaking this sensitive data? The question is tricky because clearly you have given the app permission to access private data.

Of course, an app could ask permission to access your GPS location but, if you decline, it will not work. Even if you allow such access to, say, Google Maps to get information about traffic density, you do not want your GPS location sent to an unauthorized party.

How to protect sensitive data while allowing unvetted apps to process the data is an area of active research in the security community. One potentially effective strategy requires data owners to define both an access control policy and a usage policy for their data: the former provides initial authorization for an app or other entity to access their data, while the latter defines how the app may use the data after initial access is authorized. These policies should always be attached to the data and enforced throughout its lifetime, no matter which apps access the data on which computers. This is not possible today but could be achieved by implementing architectural support for *self-protecting data*.

Owner access control and usage policies

A basic premise for self-protecting data is that a data owner can best understand the data's security requirements and so is best qualified to define its access control and usage policies. The software developers who create an app can implement basic security measures but might not recognize or be able to predict the nuanced differences among users regarding how and to what extent they want their data protected.

One problem with owner control is that many users may not be able to define appropriate data access control and usage policies. Assuming that users know at least what data should be protected, however, it is possible to provide default policies for different types of sensitive or confidential data, with facilities that enable the data owner to modify or extend these policies.

For a distributed system with many computers to support self-protecting data, we must design new hardware and software mechanisms that enable the persistent binding of the data with its associated security policy and enforce this throughout the lifetime of the data—whenever it is accessed and on any computer, including mobile devices. However, this presents multiple challenges.

Challenges of self-protecting data

One basic challenge of self-protecting data is making it easy for users to define security policies at a level they understand—that is, to express policies in a high-level language ubiquitously supported by commodity computer systems.

Another challenge is lack of access to an app's source code. Users can only install the app and execute it; there is no way to rewrite the app at the source-code level or recompile it for more secure code or for built-in security checks.

In addition, the underlying OS software, itself large and complex, is likely to contain bugs and security vulnerabilities that attackers can exploit. Attackers who achieve root privilege—that is, access at the OS level—can see an app's entire memory space and thus all data accessible to the app. The many security updates required for any commodity OS attest to the fact that no matter how much effort OS developers put into security, exploitable vulnerabilities will always exist in complex software.

A further challenge is ensuring that the attached security policy for protected data is enforced in a manner that cannot be bypassed over the data's entire lifetime.

Finally, the performance of the new security measures must be acceptable

to users; otherwise, users will simply find ways to circumvent them.

DataSafe: A software–hardware architecture for self-protecting data

Meeting these challenges optimally requires a combined software–hardware architecture: software to provide usability and flexibility, and hardware to enhance performance and provide mechanisms that cannot be easily bypassed. If the OS can be compromised, then a more powerful software layer below the OS needs to be trusted; if this layer can also be compromised, then the hardware is the final defense: processors are the engines that execute all software.

Underlying concept. Figure 1 shows the general concept underlying DataSafe, an architecture designed to support self-protecting data.[2] Protected data is encrypted, and access control and usage policies are attached to the data. The only way to transfer protected data between machines is via this data package. Once an app or its user is authorized to access the data according to the access control policy, DataSafe software translates the high-level usage policy into low-level hardware tags. These are then associated with the memory locations where the decrypted data will be placed.

The tags are propagated along with the data as it is being processed. When any output is requested, the hardware checks the tags to determine whether output for the data is allowed. This step prevents information leaking out of the machine in which access has been given. When the app completes its operation on the data, any data that has been modified is re-encrypted and repackaged with its original access control and usage policies, before being written back to storage or transferred to another machine.

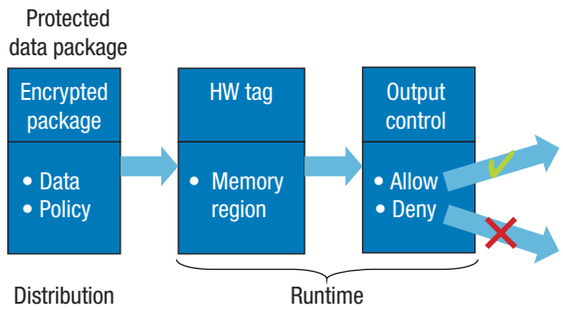

FIGURE 1. Basic concept for DataSafe, a software–hardware architecture designed to support self-protecting data.

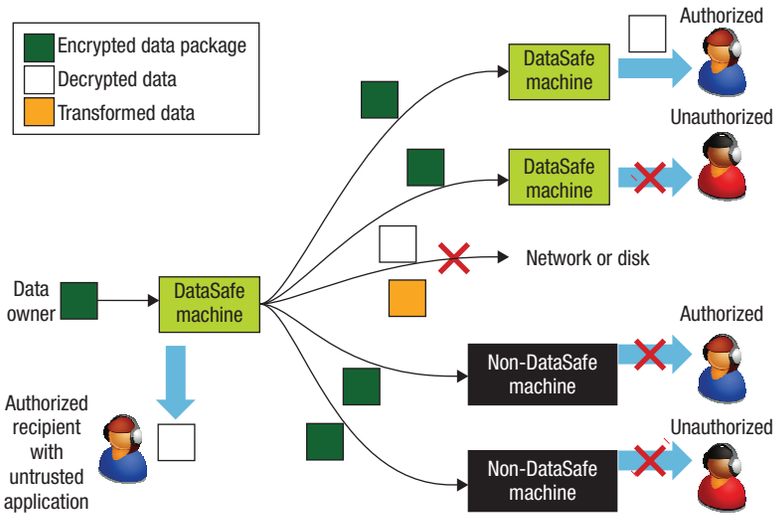

FIGURE 2. Distributed DataSafe operation coexisting with non-DataSafe machines. Output control protects decrypted information from being leaked out of the machine where access was initially allowed.

Distributed operation. Figure 2 illustrates a distributed operation with both DataSafe-enabled computers and non-DataSafe computers. The authorized user on the figure's left can get decrypted plaintext data from a DataSafe encrypted package and use this data with any app. However, an authorized user may not send plaintext data or transformed plaintext data to the network or to persistent storage (right side of figure); DataSafe-protected data is re-encrypted and sent in a DataSafe package. This output control protects decrypted information from being leaked out of the machine where access was initially allowed.

If the destination is a DataSafe machine, authorized users and apps can access the decrypted data. Unauthorized users and apps will not pass initial access-control policy checking and will only be able to access the encrypted data. If the destination is a non-DataSafe machine, only the encrypted data can be accessed, not the plaintext version. This is important because not all computers, especially older ones, will be DataSafe-enabled. Details of the key management performed in each trust domain before a DataSafe machine gets the key to decrypt a DataSafe packet are discussed elsewhere.[2]

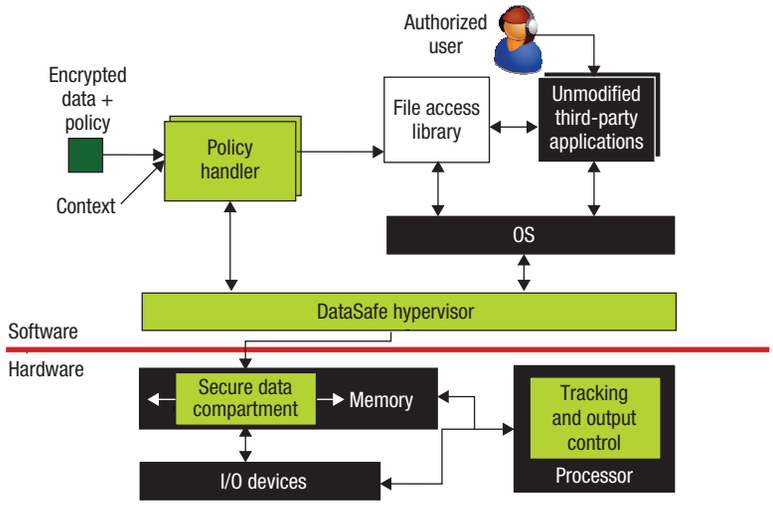

FIGURE 3. DataSafe software and hardware components (green boxes) added to existing components (black boxes).

DataSafe machine operation. Figure 3 shows the operation of a DataSafe machine. Black boxes represent untrusted software or hardware components, while green boxes represent DataSafe trusted components. A dark green box signifies a DataSafe encrypted package with attached access control and usage policies.

First, an authorized user requests permission for an app to run on sensitive data, which is retrieved as a DataSafe package. The DataSafe policy handler is a flexible software component that supports arbitrary security policies expressed in a high-level language, such as XML. It performs initial access control checking and, if access is authorized, translates the high-level usage policy into low-level hardware tags.

The computer's hypervisor, which runs below the VM running the untrusted OS and apps, decrypts the data and associates hardware tags with every memory word of the memory allocated to hold the decrypted data—called a *secure data compartment* within the memory. The app is unmodified, and its calls to access the data are transparently intercepted by the file access library or some other mechanism. The app is unaware whether it is accessing protected or unprotected data.

The DataSafe hardware reserves memory space for the hardware tags and propagates the tags whenever protected data, or data derived from protected data, is used. When an output request is made, the DataSafe hardware first checks to see if this output is allowed by the hardware tags associated with the requested data. If not, a protective measure is taken, such as outputting random data, issuing an error message, or stopping the app.

Application-transparent lifetime protection of data. So how does the DataSafe software–hardware architecture meet the challenges of self-protecting data described earlier? By interpreting any security policies regarding access control and usage and translating them into hardware tags at runtime, DataSafe software bridges the semantic gap between flexible, high-level software policies and specific hardware tags. As for lack of access to app source code, DataSafe imposes the stronger constraint that an app requesting data is not modified at all, and so is oblivious to whether that data is—or is not—protected. To deal with OS vulnerabilities, DataSafe assumes that the OS is untrusted, and its protection features do not rely on the OS. It does, however, assume that a trusted hypervisor is managing the VMs in which the untrusted OS and apps execute. To ensure that security policies can never be bypassed, DataSafe uses hardware for dynamic information flow tracking to propagate the security tags and for checking all outputs. Finally, since hardware tag propagation is very fast, there is essentially no performance penalty for users.

Figure 4 illustrates a specific real-world use of DataSafe: processing sensor data for a city. Here, security policies are attached to different types of data, such as that from surveillance cameras or from environmental sensors that detect the presence of toxic chemicals. Sensor data is sent to a cloud server for processing. Various third-party analysis programs can be used, without modification, while the original sensor data remains protected from unauthorized parties.

SECURE ENCLAVES FOR TRUSTED SOFTWARE

Security mechanisms implemented in software require a secure environment in which to execute their trusted components. For example, DataSafe's policy handler is a trusted software component that needs to execute in a secure environment. To achieve a flexible, general-purpose security architecture, new hardware trust anchors must be designed that provide *secure enclaves* for trusted software modules from various trust domains, within the sea of untrusted software.

Bastion security architecture

Figure 5 illustrates Bastion, an architecture that supports secure enclaves for trusted software modules.[3] The schematic shows two of many VMs running on top of a hypervisor. Any number of trusted software modules—here labeled A, B, and C—can be

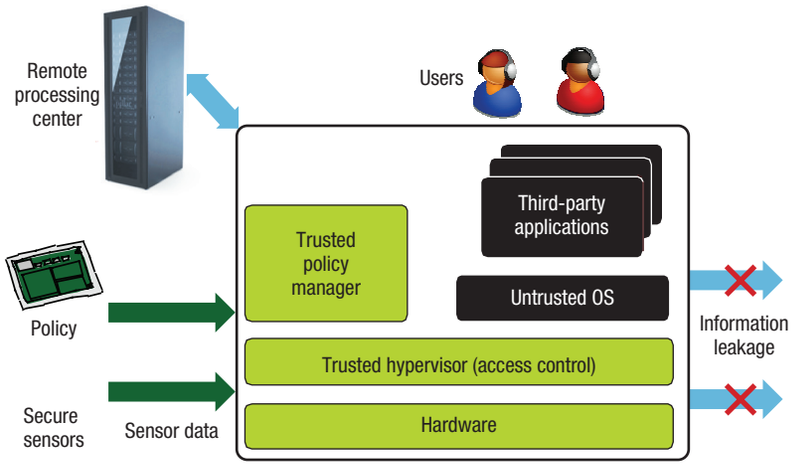

FIGURE 4. Example of DataSafe architecture in processing a city's sensor data. Data can be processed by third-party analytic apps, but remains protected from unauthorized parties.

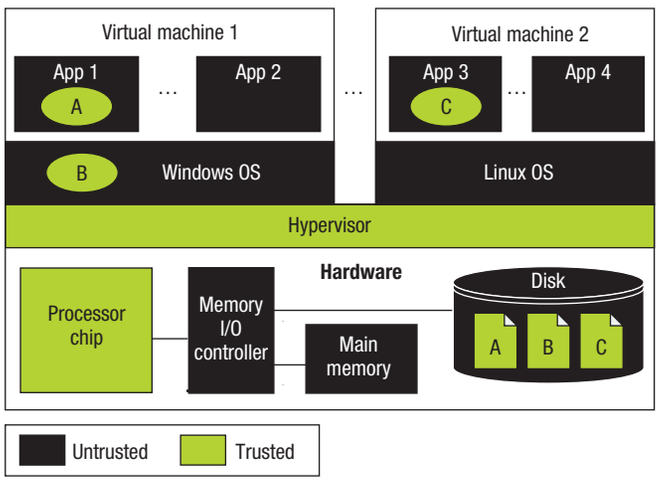

FIGURE 5. Bastion security architecture. With multiple virtual machines running on top of a hypervisor, any number of trusted software modules—here labeled A, B, and C—can be simultaneously supported in user apps or in the OS. Bastion provides a secure execution environment, or enclave, for trusted code rather than a sandbox for untrusted code.

simultaneously supported in user apps or in the OS. These modules can come from different, perhaps mutually suspicious, trust domains and so must be isolated from one another, from other apps, from the rest of the app, and from any untrusted OS.

The Bastion hypervisor functions as a *super* trusted software module, directly protected by the hardware. The hypervisor then provides similar security protection for the trusted software modules, any number of which can exist in a VM. Each trusted software module also has an associated *persistent secure storage* area accessible only by that trusted software module but not by other apps, the rest of the app, or the OS. This storage area is secured cryptographically and so can be implemented using commodity, insecure storage media; it is persistent in that the data is preserved when power to the computer is turned off. The area can be used for storing crypto keys and other secrets accessible only to this trusted software module. Secure enclaves for trusted software modules can be dynamically created, as needed, for executing security-critical software.

Detailed descriptions of Bastion can be found elsewhere.[3,4]

Bastion versus SGX

Commodity microprocessors are beginning to incorporate security features—for example, the Software Guard Extensions (SGX) in future Intel microprocessors.[5–7] SGX has goals similar to those of Bastion but does not trust the hypervisor and so requires more extensive hardware additions.

One advantage of SGX over Bastion is its reduced trusted computing base: the trusted hardware can still protect secure enclaves even when the hypervisor is compromised—although compromising the hypervisor could be significantly more difficult than attacking the OS, since hypervisors are orders of magnitude smaller than a typical full-feature OS.

A disadvantage of SGX compared to Bastion is less flexibility and scalability. SGX's all-hardware implementation implies a fixed number of new hardware registers and a maximum number of bits per hardware field—which further implies a maximum number of secure enclaves and trust domains, as well as a maximum number of pages per enclave, above which performance might degrade significantly. In contrast, with Bastion's trusted hypervisor an unlimited number of software-implemented registers can be instantiated in the hypervisor's hardware-protected memory and persistent storage, enabling secure enclaves and trust domains of

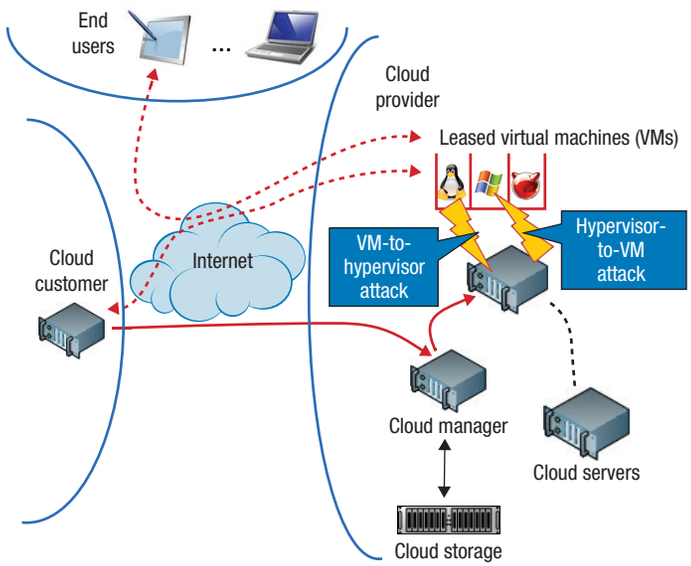

FIGURE 6. Infrastructure as a service (IaaS) paradigm for cloud computing. Cloud customers lease virtual machines that run on a cloud provider's servers.

unlimited number and size without performance degradation.

Another disadvantage of SGX is that hardware has to implement complex memory management functions normally implemented in software. Post-release bugs for complicated functions are more easily fixed in software than in hardware/firmware implementations in the field.

However, the key advantage of SGX, if implemented in all Intel chips, is that it will be widely available in commodity x86 microprocessors.

Architectures like Bastion or SGX, when properly used, will make it much harder for malicious actors to attack security-critical software running in secure enclaves, in contrast to the ease with which they can attack regular apps and the OS today. This is a significant step toward realizing the vision of a general-purpose security architecture.

Software isolation with Trustzone

A secure enclave is a software isolation mechanism that is finer grained than that offered by VMs. Until SGX is commercially available, the state of the art for software isolation in existing microprocessors is ARM Trustzone,[8] implemented in most newer ARM chips.

Trustzone implements two worlds: a *normal world* in which all current software executes, and a *secure world* in which only trusted applets execute. The secure world is like one permanent secure enclave. This is great for infrequently performed security-critical tasks, such as logging in or changing critical wireless communication parameters in mobile handsets, or when only one trust domain is active at any time. However, performance may be significantly degraded if frequent world switches are required.

Alternatively, if it has to support multiple, mutually suspicious trust domains concurrently, the secure world OS will have to become much more complex—and thus more error-prone and vulnerable. Another issue with Trustzone is that requests and parameters passed to it from the normal world cannot be trusted. In many ways, Trustzone resembles the precursor to the Bastion architecture, called the SP (Secret Protection) architecture,[9,10] which provided minimalist hardware support for a secure execution environment for one trust domain at a time.

Trustzone is widely available in smartphones and tablet computers, and a software ecosystem has been built around it. It provides runtime security that complements the bootup and platform integrity provided by an industry consortium's earlier Trusted Platform Module (TPM) chip.[11] Real-world security can be considerably improved with proper use of Trustzone protection.

VM SECURITY IN THE CLOUD

Today's computing landscape consists of client computers or smartphones performing computations using cloud servers in a large datacenter. Figure 6 shows a cloud customer leasing VMs from a cloud provider to host a website that performs various data computations and services; end users can access the website in the cloud. This cloud computing paradigm is referred to as infrastructure as a service (IaaS).

With IaaS, cloud customers provide their own OS image and apps to run in the leased VMs. In two other common paradigms, platform as a service (PaaS) and software as a service (SaaS), the cloud provider supplies more of the software components such as the OS and the apps.

Cloud customers would like computing in the cloud to be at least as secure as computing on machines at their own site. In theory, computing in a VM is indistinguishable from computing directly on the hardware, since the virtualization layer—the virtual machine monitor or hypervisor—manages hardware resources such that each VM performs as though it had its own machine. However, it is possible for VMs leased by adversaries to be colocated with a victim VM on the same server and attack the hypervisor, as also illustrated in Figure 6. Since the all-powerful hypervisor can access everything in a VM, an adversary who attains hypervisor-level privileges can breach the confidentiality and integrity of a victim VM's data and code.

One defensive approach is to use the commodity hypervisor for resource

NEW HARDWARE TRUST ANCHORS MUST BE DESIGNED TO PROVIDE SECURE ENCLAVES FOR TRUSTED SOFTWARE.

allocation, but not trust it with a VM's confidential data and code. The Hyperwall security architecture[12] achieves this with new hardware mechanisms that can prevent the hypervisor from accessing all or part of a VM's memory during runtime, based on a security policy specified by the cloud customer. Since the hardware is below the hypervisor, it can protect a VM from attackers who manage to attain hypervisor-level privileges. Hyperwall also has hardware that collects trust evidence to send back to cloud customers attesting that their sensitive memory pages are protected.

Another approach is embodied in the NoHype architecture.[13,14] NoHype uses the hypervisor only to allocate resources and start the VM, after which the hypervisor gets out of the way and the VM runs on bare hardware. At the end of the VM's lease period, the hypervisor comes in to terminate the VM, reclaiming its resources. Unlike Hyperwall, NoHype does not need new hardware mechanisms, but it does have some resource-allocation constraints. For example, a processor core is assigned only one VM at a time, thus removing the need for a hypervisor to schedule VMs at runtime; however, there is no restriction on the number of processor cores that can be assigned to one VM. Also, memory is not over-provisioned—that is, each VM is always given the full quota of memory it has contracted for in the service-level agreement.

The hardware virtualization mechanisms now built into Intel and AMD processors detect whether a VM accesses outside of its allotted memory—which, with NoHype, indicates a security breach. Also, I/O device and network interfaces are assigned directly to each VM through, for example, hardware-virtualized devices like SRIOV (single root I/O virtualization), wherein input and output queues are multiplexed to the physical device or network port. Each VM is assigned one input queue and one output queue to connect directly to a device or port.

NoHype provides static but adaptable assignment of resources to a VM, which is reasonable for many highly secure environments, especially given many-core processor chips and lots of server memory. NoHype's lower scalability is offset by its simplicity and increased security. Other researchers have shown that a hostile VM can be scheduled on the same cloud server as its victim VM, without any insider information regarding the cloud provider's allocation algorithms.[15] By removing the hypervisor attack surface during the VM's runtime,[14] NoHype aims to make computing in the cloud, even with other untrusted VMs present, equivalent to running on one's own computers.

SECURE HARDWARE

Correctly functioning hardware doing exactly what it was designed to do—such as improve a computer's performance or reduce power consumption—can be exploited by an attacker to breach security. For example, measurements of hardware runtime characteristics can be used by attackers to leak secret information through *side-channel attacks*. Unlike exploitable software due to software bugs, such attacks on hardware components are not due to buggy hardware. By *secure hardware*, I mean hardware that is designed and built to thwart attacks on hardware.

Attackers tend to reach for low-hanging fruit first and thus target software and networking systems before they consider attacking hardware. When they do attack hardware, their initial strategy is usually to launch software attacks on hardware resources, which can be accomplished remotely with no extra equipment, rather than physical attacks, which typically require proximity to the victim computer and equipment for taking physical measurements, like power, electro-magnetic, or acoustic measurements. Hence, software attacks on hardware caches are some of the most dangerous attacks on hardware that should be addressed, because they are easy to perform and also because of the ubiquity of caches in all modern computers.

Information leaks via cache side-channel attacks

Caches are faster, smaller memories between the processor and the slower, larger main memory that help reduce effective memory access time. When data the processor requests resides in a cache—called a *cache hit*—it is returned immediately to the processor, with a latency of only one or two processor cycles. If data is not in the cache—called a *cache miss*—it must be fetched from the main memory, transported into cache, and then sent to the processor, requiring hundreds of processor cycles. Modern processors have multiple levels of caches to bridge this "memory wall." Caches are probably the most critical hardware component for achieving high performance in today's computers. Caches are shared by all the software running on a machine—independent of any software isolation mechanisms used.

Attackers exploit the timing difference between a cache hit and a cache miss in a cache side-channel attack.[16] For example, an attacking software

process that is legitimately accessing its own memory space can determine which cache lines have been accessed by a victim process. Because all caches today use a fixed, static mapping from memory addresses to cache lines, the attacker can then infer the memory addresses used by the victim, and from these addresses, the attacker can infer the value of secret encryption key bits used by the victim process.

Such cache side-channel attacks can be used to leak secret encryption keys, thus breaching whatever confidentiality strong encryption provides. These attacks can also leak private keys used in public-key ciphers and thereby enable attacks based on masquerading and false authentication. For example, a side-channel attack on an e-bank, Realbank, which leaks its private key, could enable a fake bank, Badbank, to pretend to be Realbank. While software isolation mechanisms isolate memory used for different processes and different VMs, these mechanisms do not isolate the available caches, so information can still leak through cache side-channel attacks. Research has shown that even in a cloud computing environment, a VM can perform a side-channel attack on another VM to obtain its private key.[17]

Because caches are so ubiquitous, such information leakage is possible in essentially all computers—including smartphones. Cache side-channel attacks thus constitute a serious security problem.

Limitations of software-based solutions

Many software mitigations have been proposed for cache side-channel attacks. However, these require modifying each piece of software that performs cryptography with changes specific to the pertinent crypto algorithm rather than across all crypto algorithms. While this can be done for commonly used crypto libraries, it is impossible to change all legacy applications that embed cryptography. Furthermore, proposed software solutions exact a huge performance penalty, slowing down performance by as much as 10 times.

Nor can software-based solutions guarantee security, because software has no direct control over hardware caches and cannot prevent hardware or system asynchronous events that modify the cache state. Consider two examples. In one defensive strategy, before accessing the entry it wants, software will cycle through every entry of a security-sensitive table (preloading) to get a constant time (cache hit) for each table access. However, hardware cache mechanisms can cause such cache lines to be evicted before actual table access is achieved. Another constant-time defense makes every memory access a cache miss, essentially disabling the cache; this would cause performance to drop precipitously.

COMPUTER ARCHITECTS SHOULD RETHINK THE DESIGN OF ALL HARDWARE SUBSYSTEMS, LEADING TO A NEW SCIENCE OF SECURE HARDWARE DESIGN.

Secure caches

It might be possible to devise a hardware solution that prevents cache side-channel attacks without impacting the performance gains cache use provides—if we rethink cache design.

Unfortunately, the obvious solution—cache isolation, which partitions the cache so there is no cache sharing between a potential victim and a potential attacker process[18]—typically degrades system performance. A less obvious approach is to introduce dynamic randomization in the mapping of memory addresses to cache lines: the attacker can still share the cache and observe its state, but cannot get any useful information about what cache lines a potential victim process is using.[18,19] This method applies the "moving target defense" strategy to cache design: memory-to-cache mapping changes for each execution of the same program on the same hardware. To this end, Newcache[19] uses randomization techniques that can improve performance for some programs and at the same time improve security by defeating cache side-channel attacks.[19,20]

If chip vendors replace all conventional caches with secure ones like Newcache, dangerous cache side-channel attacks can be proactively defeated before attackers decide to exploit them seriously. In addition, computer architects should rethink the design of all hardware subsystems, especially performance and energy optimization features, to ensure that they do not inadvertently make

See **www.computer.org/computer-multimedia** for multimedia content related to this article.

attacks on hardware easier. This can lead to a new science of secure hardware design.

The work reported here suggests the potential scope of hardware security, describing both hardware-enhanced security architectures and secure hardware designs. While challenges for cybersecurity are enormous, it is promising to see growing interest among hardware vendors in building security mechanisms into chips and computers, and among researchers for creating better hardware–software security defenses. As threats proliferate and user demand for protection increases, computer designers will be called on to build security into the hardware foundations of all computing devices.

ABOUT THE AUTHOR

RUBY B. LEE is the Forrest G. Hamrick Professor of Electrical Engineering at Princeton University and formerly served as chief architect at Hewlett-Packard. Her current research interests include secure hardware–software architectures, secure cloud computing, secure processors and caches, and secure smartphones. Lee received a PhD in electrical engineering from Stanford University. She is an IEEE Fellow and ACM Fellow, and holds numerous US and international patents. Contact her at rblee@princeton.edu.

REFERENCES

1. R.B. Lee, *Security Basics for Computer Architects*, Morgan & Claypool, 2013.
2. Y.-Y. Chen, P. Jamkhedkar, and R.B. Lee, "A Software-Hardware Architecture for Self-Protecting Data," *Proc. ACM 19th Conf. Computer and Communications Security* (CCS 12), 2012, pp. 14–27.
3. D. Champagne and R.B. Lee, "Scalable Architectural Support for Trusted Software," *Proc. IEEE 16th Int'l Symp. High-Performance Computer Architecture* (HPCA 10), 2010, pp. 1–12.
4. D. Champagne, "Scalable Security Architecture for Trusted Software," PhD dissertation, Dept. Electrical Eng., Princeton Univ., 2010.
5. F. Mckeen et al., "Innovative Instructions and Software Model for Isolated Execution," *Proc. 2nd Int'l Workshop Hardware and Architectural Support for Security and Privacy* (HASP 13), 2013; doi:10.1145/2487726.2488368.
6. M. Hoekstra et al., "Using Innovative Instructions to Create Trustworthy Software Solutions," *Proc. 2nd Int'l Workshop Hardware and Architectural Support for Security and Privacy* (HASP 13), 2013; doi:10.1145/2487726.2488368.
7. Intel, *Software Guard Extensions Programming Reference*, Sept. 2013; https://software.intel.com/sites/default/files/329298-001.pdf.
8. ARM Security Technology, *Building a Secure System Using Trustzone Technology*, 2009; http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf.
9. R.B. Lee et al., "Architecture for Protecting Critical Secrets in Microprocessors," *Proc. 32nd Int'l Symp. Computer Architecture* (ISCA 05), 2005, pp. 2–13.
10. J.S. Dwoskin and R.B. Lee, "Hardware-Rooted Trust for Secure Key Management and Transient Trust," *Proc. 14th ACM Conf. Computer and Communications Security* (CCS 07), 2007, pp. 389–400.
11. Trusted Computing Group, *TPM Main Specification*; www.trustedcomputinggroup.org/resources/tpm_main_specification.
12. J. Szefer and R.B. Lee, "Architectural Support for Hypervisor-Secure Virtualization," *Proc. 17th Int'l Conf. Architectural Support for Programming Languages and OSs* (ASPLOS 12), 2012, pp. 437–50.
13. E. Keller et al., "NoHype: Virtualized Cloud Infrastructure without the Virtualization," *Proc. 37th Int'l Symp. Computer Architecture* (ISCA 10), 2010, pp. 350–361.
14. J. Szefer et al., "Eliminating the Hypervisor Attack Surface for a More Secure Cloud," *Proc. 18th ACM Conf. Computer and Communications Security* (CCS 11), 2011, pp. 401–412.
15. T. Ristenpart et al., "Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds," *Proc. 16th ACM Conf. Computer and Communications Security* (CCS 09), 2009, pp. 199–212.
16. D.A. Osvik, A. Shamir, and E. Tromer, "Cache Attacks and Countermeasures: The Case of AES," *Proc. Cryptographers' Track RSA Conf. 2006* (CT–RSA 06), 2006, pp. 1–20.
17. Y. Zhang et al., "Cross-VM Side Channels and Their Use to Extract Private Keys," *Proc. 19th ACM Conf. Computer and Communications Security* (CCS 12), 2012, pp. 305–316.
18. Z. Wang and R.B. Lee, "New Cache Designs for Thwarting Software Cache-Based Side Channel Attacks," *Proc. 34th Int'l Symp. Computer Architecture* (ISCA 07), 2007, pp. 495–505.
19. Z. Wang and R.B. Lee, "A Novel Cache Architecture with Enhanced Performance and Security," *Proc. 41st IEEE/ACM Int'l Symp. Microarchitecture*, 2008, p. 89–93.
20. F. Liu and R.B. Lee, "Security Testing of a Secure Cache Design," *Proc. 2nd Int'l Workshop Hardware and Architectural Support for Security and Privacy* (HASP 13), 2013; doi:10.1145/2487726.2487729.