Improving Cyber Security

Ruby B. Lee

Cyber security is essential given our growing dependence on cyberspace for all aspects of modern societies. However, today, attackers have the upper hand. In this chapter, I discuss the security properties needed, and some key strategies that may have the potential to level the playing field between attackers and defenders. These research strategies were developed at the National Cyber Leap Year summit, with experts from industry, academia, and government working collaboratively. These broad research thrusts can be interpreted at different levels of the system, and in different application domains. Because a promising direction explored at the summit is the use of hardware architecture to enhance security, I provide a hardware-enhanced interpretation of the proposed research thrusts, with the goal of illustrating how new security features can be built into future commodity computers to improve system security. The goal is to be able to ensure essential security features for critical tasks, even in the presence of malware and software vulnerabilities in the system, and users who are not

Ruby B. Lee, "Improving Cyber Security", in Advances in Cyber Security: Technology, Operations and Experiences, ed. Frank Hsu and Dorothy Marinucci, ISBN 978-0-8232-4457-7, Fordham University Press 2013, pp. 37-59.

security-savvy. Feasibility examples are given to show how new hardware security features can help improve software security and also how hardware itself can be designed to be more trustworthy. These examples illustrate that by rethinking the fundamental design of computers with security as one of the key requirements, we can design future *secure, trustworthy computers* without necessarily sacrificing performance and other goals.

Cyber Security Today

In its early days, the Internet was used as a means of enhancing research among collaborating scientists. Internet protocols were designed to enable seamless *inter-network* communications between sender and receiver across heterogeneous networks, resulting in the name "Internet." Researchers worked hard to define the basic Internet protocols[1] and make them work, hiding the physical complexity of the different physical network technologies being used while allowing full flexibility to implement arbitrary applications across heterogeneous networks. The success of this design can be seen today in the various applications built on top of the Internet, such as the World Wide Web, search, web mail, e-commerce, e-banking, and social network applications. Today, our social lives, our economic competitiveness, our national security, and in fact all aspects of our lives depend on the correct functioning and ubiquitous availability of the Internet and wireless networks. This dependence on the Internet and on cyberspace transactions is increasing at the same time cyber attacks are escalating.

With society's growing dependence on cyberspace, cyber security becomes a critical issue. The technologies that make up cyberspace were not designed with security in mind. Internet protocols were designed for friendly parties to communicate and collaborate with each other—they were not designed with malicious adversaries in mind. Similarly, computer technology, both hardware and software, was not designed with attackers in mind. Hence, it should not be surprising that the basic network, soft-

1. For example, supporting the TCP/IP (Transmission Control Protocol/Internet Protocol) protocols enables the fundamental interoperability across different types of networks.

ware, and hardware technologies underlying cyberspace are full of security vulnerabilities that can be exploited by malicious parties. In addition, attackers only need to find one path into the system to infiltrate it, whereas defenders have to defend on all fronts. Furthermore, since most computer systems use the same systems software and hardware, the attacker will also have found a way to infiltrate a large majority of today's computers.

AN EXAMPLE OF AN INTERNET-SCALE ATTACK

An example of an Internet-scale attack is a *distributed denial-of-service* (DDoS) attack. In a DDoS attack, an army of zombies (also called a *botnet*) is harnessed to attack a *primary victim*, which could be a website, computer, or network (see Figure 1a). The zombies are actually *secondary victims*, since they are innocent computers whose owners are unaware that their machines have been infected with malware that can turn their machines into "zombies" (also called "bots" or "agents") that can be used in a DDoS attack on a primary victim.

Such stealthy infiltration of computers and installation of zombie programs can be done months before an actual DDoS attack is launched. They are possible due to security vulnerabilities, most often in the software, that can be exploited by an attacker to infiltrate a computer and silently install a zombie program without being detected. They even allow for updating such bot code in the zombie computer and sending the computer's configuration updates to the adversary. During a DDoS attack, the zombie programs are invoked to send innocuous requests to the primary victim website. Such requests are essentially indistinguishable from legitimate requests to this website. However, a flood of zombie requests from a large number of computers to a victim website can overwhelm it and its surrounding network, making it unable to provide service for legitimate requests (see Figure 1b). Hence, this is a denial-of-service (DoS) attack on the availability of services. It is called a *distributed* denial-of-service (DDoS) attack because the flood of requests comes from frontline zombie computers, which can be distributed all over the world, making it hard to drop traffic from just a few sources.

A DDoS attack is disturbing because it is like using innocent citizens without their consent to form an ad-hoc army to cause damage to one's

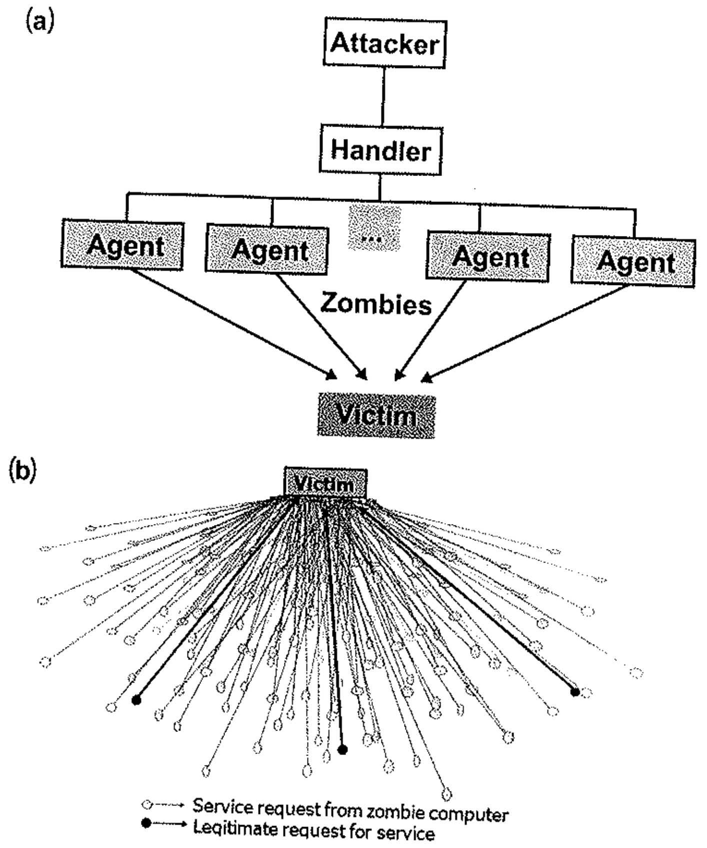
(a)
Attacker
Handler
Agent
Agent
...
Agent
Agent
Zombies
Victim
(b)
Victim
Service request from zombie computer
Legitimate request for service

Figure 1. Distributed denial-of-service (DDoS) attack: (a) Example of a DDoS attack network; (b) DDoS attack flooding a primary victim site.

own country or other countries. Tracing the true attacker is difficult, since he will typically hide behind many levels of indirection, using *handlers* (which are themselves infected computers) to infiltrate other computers and install handler or zombie code on them. The zombie code and the actual requests made during a DDoS attack often use very little storage, computing, and networking resources, so they essentially run "under the radar" and are invisible to the user of the zombie computer. With software and networking enabled in trillions of devices and embedded systems, rather than just millions of computers, DDoS attacks can become orders of magnitude more potent in flooding network bandwidth and computer resources.

FUNDAMENTAL SECURITY PROPERTIES

There are many security properties required or desired in computer and communications systems. Three such properties have been called *cornerstone* security properties: confidentiality, integrity and availability; their acronym, CIA, is the same as a famous U.S. agency and hence is easy to remember. *Confidentiality* is the prevention of the disclosure of secret or sensitive information to unauthorized users or entities. *Integrity* is the prevention of unauthorized modification of protected information without detection. *Availability* is the provision of services and systems to legitimate users when requested or needed. This is similar to providing reliability (or fault-tolerant systems), except that availability, in the security sense, is harder because it has to also consider intelligent attackers with malevolent intent whose behavior cannot be characterized by a probability distribution—unlike device failures and faults.

In addition to the fundamental CIA triad, there are many other important aspects of security, some of which I define briefly here. *Access control*, comprising both *authentication* and *authorization*, is essential to ensure that only authenticated users who are authorized to access certain information can in fact access that material, while denying access to unauthorized parties. *Attribution* support should ideally be built into secure systems in order to find the real attackers when a security breach has occurred. *Accountability* support holds vendors, owners, users, and systems responsible for vulnerabilities that enable successful attacks. *Non-repudiation* is desirable to ensure that a user cannot deny that he has made a certain request or performed a certain action. *Attestation* is the ability of a system to provide some non-forgeable trust evidence of its hardware and the software it is currently running. *Anonymity* is the ability to perform certain actions without being identified or tracked. *Privacy* is the right to determine how one's personal information is disseminated, or redistributed by an authorized recipient. Confidentiality and privacy are different in the sense that confidentiality is the obligation to protect secret information, while privacy is the right to protect one's personal information. However, it is possible that they may share similar defense mechanisms.

I define a *secure computer system* as one that provides at least the three cornerstone security properties of confidentiality, integrity and availability.

A *trustworthy computer* is one that is *designed* to be dependable and to provide such security properties, to do what it is supposed to do and nothing that may harm itself or others. In contrast, what has conventionally been proposed is a *trusted computer*—one that is depended on to enforce security policies, but if it is infiltrated, then all bets are off for enforcing security policies. More precisely, this has been described as a computer with a *trusted computing base* (TCB), which if violated means that security policies may not be enforced. Unfortunately, no COTS (commodity off the shelf) computer system today can achieve a dependable trusted computing base. In fact, commodity computers just have not been designed to be trustworthy. Security has typically been bolted on, after the fact, resulting in degradation of performance, cost, time-to-market, and ease-of-use. Furthermore, security issues are not even in the core computer architecture, hardware or software curriculum in colleges in the United States today, whereas they were more routinely taught thirty to forty years ago. This has created a crisis in educated manpower in the research, design and development of secure and trustworthy computer systems that are so sorely needed for cyber security. Hence, there is an urgent need to remedy this and to consider both the research and the educational needs for improving cyber security.

Strategies for Improving Cyber Security

Today, cyber attackers have a highly asymmetric advantage over defenders. How can we level the playing field so that attackers do not have such an upper hand? How can we increase the work factor for mounting successful cyber attacks by orders of magnitude?

Among many public and private sector initiatives, there was a government-sponsored initiative called the National Cyber Leap Year (NCLY) initiative, whose goal was to find game-changing strategies for improving cyber security, leaping ahead of current incremental or reactive strategies. Multiple rounds of calls for proposals for research directions were issued, and hundreds of responses came from both companies and universities. From these, a few major research directions were honed and discussed in an NCLY summit held in August 2009 to which security experts from industry and academia were invited. The summit was hosted by NITRD, representing thirteen government agencies, with many security experts from these government agencies present. It promoted constructive discussion of five research directions that were considered to have game-changing potential: hardware-enabled trust, digital provenance, nature-inspired cyber health, moving target defense, and cyber economics. The findings were summarized in the NCLY Summit Co-Chairs' Report [1]. While these approaches were considered to be promising directions where cyber security research is required, this does not in any way imply that other approaches are not promising.

The summit brought together experts from industry, academia, and government, with very different backgrounds and viewpoints, to rally under the common goal of improving cyber security. As an invited co-chair, I found the collaboration promoted by the summit quite refreshing, as did other co-chairs and participants. Indeed, such collaboration is needed to make real inroads in improving cyber security. More importantly, it is gratifying to note that this summit has had significant impact on government funding for cyber security research. Within about eighteen months, many government agencies, including NSF, DARPA, DHS, and DOD, to name a few, put out calls for proposals for research funding in these and related cyber security research areas.

Some of the promising research directions discussed at the summit can be combined under a broader umbrella of what might be termed "proactive design strategies" to:

enable tailored trustworthy spaces within the generally untrusted cyberspace;
thwart attackers with moving target defenses; and
reward responsible behavior with economic and other incentives.

These three major thrusts were presented at the NCLY kickoff meeting in May 2010 [2]. Below, I will first discuss these three strategies, in general, for improving cyber security. Then, to illustrate how these research strategies can be applied not only to the more visible layers of software and networking, but also to fundamental hardware design, I will give some feasibility examples on how to build future trustworthy computers that can help improve cyber security with hardware-enhanced trust.

ENABLING TAILORED TRUSTWORTHY SPACES

The goal of this research direction is to create technology that can provide trustworthy spaces on demand, tailored to the needs of the usage scenario.

This is in contrast to the conventional strategy where a *sandbox* is set up for executing untrusted applications. A sandbox is a constrained execution environment where the untrusted application is unable to use all the resources normally available on the computer system. While this may prevent it from accessing or modifying critical system resources, it also means that many existing applications will not run as they did before, leading to user dissatisfaction with such "security-upgraded" systems.

The fundamental difference of the proposed strategy is that untrusted applications run as before, with the same access to the system resources that they had before. However, trusted applications that are security-critical should be enabled to run in newly created *trustworthy spaces*, tailored to their needs. This recognizes that there is no one-size-fits-all trustworthy space. Instead, different trustworthy spaces should be created based on the security requirements of the particular application, or of the secure, sensitive, or proprietary data that are to be accessed. For example, the confidentiality levels required are different for protecting YouTube videos, movie rentals, online medical records, bank accounts and nuclear weapons. Similarly, confidentiality, integrity, availability, and privacy requirements differ from one application to another, from one environment to another, and from one use-case to another. Hence, the goal is to provide a secure execution environment for trusted applications without constraining untrusted applications, which can then run as before and also use the full functionality provided by commodity systems.

While this can be achieved in many ways, one approach is to set up a *secure execution compartment*, tailored to the security needs of the particular application, user, environment, and context. The security-critical part of an application then runs only in this secure execution compartment, where it is protected from other applications and also from system software such as the operating system or hypervisor, which might want to snoop on its confidential data or code. In addition to secure setup and secure execution, it is also important to provide secure termination of such secure execution compartments, to prevent an a posteriori leak of confidential information or contamination that may affect its future use.

An example of how small changes to the hardware and software architecture can enable this is discussed later, in the Bastion Security Architecture section.

In another approach, it would be great if we could enable *self-protecting data*—that is, data that can protect themselves from security breaches. This can be interpreted as data that have an attached security policy that cannot be violated no matter what application uses the data. Hence, the application does not have to be certified as trusted before it can be allowed to access protected data. Such data would have confidentiality and integrity policies that are either predefined, or can vary dynamically based on the context, the time, the location, the application, the system, and the user accessing the data. New hardware may be needed to enforce such policies, but for older computers without this new hardware, the self-protecting data might be accessible only in encrypted form, for example, with no access to the decryption key.

Data provenance, where the chronological history of the ownership of the data can be reliably obtained, is also highly desirable, both to establish data authenticity and to track adversaries when security has been breached.

It is also desirable for the system to be able to give some *trust evidence* to users that the protections they requested for their trustworthy spaces have indeed been set up and are being enforced. This may require the system to be able to perform *tailored attestations* [3] to give unforgeable assurances to the requester of the identity, type, or properties of the hardware and the software that is running on the system.

These are just a few examples of some promising research directions that may help enable the dynamic establishment of tailored trustworthy spaces within the untrusted cyberspace—without crimping the flexibility and functionality of existing applications or future applications that do not need such security.

It is not easy to set up such tailored trustworthy spaces, on demand, over the public Internet in insecure cyberspace. But if it can be done, it can enable the resilient execution of security-critical tasks, even in the presence of active attacks or malware previously introduced into the computer system.

THWARTING ATTACKERS WITH MOVING TARGET STRATEGIES

The goal of this research direction is to significantly increase the attacker's work factor for a successful attack and to reduce the number of machines that succumb to the same attack path.

One problem with today's homogeneous computing environments is that once an attacker finds a penetration path into a system, he can penetrate a huge number of similar systems. The majority of today's desktop and notebook computers use the Microsoft Windows operating system and Intel microprocessors, and the same web browsers, e-mail programs, and database software. Hence, many computers will have the same security vulnerabilities in either the software or the hardware. The idea of a moving target defense strategy is to have each system look different to an attacker, and even have a single system look different over time. This means that the attacker must learn to penetrate each system separately, and relearn how to penetrate a system he has already found a path into before—thus significantly increasing the attacker's work factor for a successful attack. A major challenge is to ensure that the system is as easy for legitimate users to use as it was before, while making it orders of magnitude harder for attackers. This moving target defense strategy also changes the game from today's reactive defense posture triggered by new attacks, to a preemptive posture where systems are specifically designed to look like moving targets to potential attackers.

A promising approach in moving target defense strategies is to use randomization in system design. This could potentially improve both the security and the performance of the system, rather than trading off one for the other. Randomness between systems, and within a system, can thwart would-be attackers from doing vulnerability mapping attacks that apply to many machines, or to one machine over a long period of time. Past research has also shown how randomization in algorithms can improve its performance. If we can combine these benefits by applying randomization theory in the creative design of new systems that have built-in moving target defenses, we can indeed achieve the previously conflicting goals of improving security and improving performance at the same time. In the Hardware Security Examples section, I show an example of applying the moving target defense strategy to design hardware cache architectures that improve both the performance and the security of cache memories.

Another approach is to use biologically-inspired defenses, where diversity helps prevent the extinction of a species due to rapidly-spreading viruses and diseases. Concepts of diversity, randomness, and moving target defenses could all be explored, as well as their potential and real impact on improving cyber security.

REWARDING RESPONSIBLE BEHAVIOR

The goal in this research thrust is to realign cyber economic incentives for promoting socially responsible behavior in cyberspace and for deterring malicious behavior.

Today, cyber crime pays: small investments of money and time can yield a large return on investment (ROI). How can we turn the tables so that cyber crime does not pay—or at least does not pay so well? How can we reward responsible behavior in cyber space that may prevent someone else from being attacked, even if we ourselves are not affected? It is said that fear and avarice are the greatest motivators. Since we do not want to wait for a "cyber 911" attack (fear) to trigger improvements in cyber security, can we use economic incentives (avarice) instead to strongly encourage software, hardware, and network vendors and cloud computing providers to build more secure products using the best security design principles and practices? Also, what economic incentives will persuade individual users, companies, and organizations to buy—and even demand—secure computers and services, and to implement security best practices?

Improving cyber security is a bit like improving the environment—although no single entity is ultimately responsible, everyone should be made aware of the consequences and should do their part. For example, the DDoS attacks discussed earlier in this chapter can be significantly mitigated if individual users and corporations do their best to prevent their machines from being used as zombies or bots in such attacks. It may be possible for fines to be levied on corporations with zombie machines, or more generally, for machines where the "security health" metrics are unacceptable. Exhortation alone may not work; note that legislation was needed to enforce car seat use to protect infants in car accidents. However, rather than punitive legislation, the spirit of this research thrust is more toward rewarding responsible behavior. What legislation could provide economic incentives for responsible cyber security behavior, comparable to incentives for using alternative energy sources for conserving energy in "green" buildings and automobiles? But even without any legislation, if the general public is made aware of the security risks, their costs and their potentially disastrous outcomes, they may provide the market pull for vendors to provide more secure IT (Information Technology) products and systems as competitive advantages.

In addition to economic incentives, research into other incentives for responsible cyber security practices is also highly encouraged. This may include the incentive to protect one's reputation in cyberspace. Fear of losing future business if one's reputation is damaged seems to have worked well in promoting responsible behavior when selling goods through web-based sites like eBay.

Hardware Security Examples

General strategies are good, but it is always helpful to see some actual examples. Also, since hardware-enhanced security solutions are less familiar to the general public, and even to the security community, it may be interesting to see some hardware feasibility examples using these general research strategies.

Hardware security mechanisms can satisfy one of the fundamental NCLY cyber security goals—it can significantly increase the work factor for attackers, since it is often orders of magnitude harder to launch a successful attack on hardware than on software. Hence in this section, I discuss the Bastion and Newcache architectures, where hardware security features can be built into future commodity computing devices to enable the construction of more secure computer systems. The Bastion architecture provides hardware and software mechanisms to facilitate the creation of tailored trustworthy spaces, while the Newcache architecture uses a moving target defense strategy to provide more trustworthy hardware that cannot be used to leak confidential information.

ARCHITECTURAL SUPPORT FOR TAILORED TRUSTWORTHY SPACES

Software virtualization technology can be used to protect a trusted application from untrusted applications, and is the core technology underlying cloud computing. Software virtualization creates virtual machines (VMs), which run on a system software layer called a virtual machine monitor (VMM), also called a hypervisor. This hypervisor manages all the hardware resources and isolates one virtual machine from another, giving each the illusion that it has the machine to itself. Trusted virtual machines can be used to run security-critical applications, while untrusted virtual machines run untrusted applications. This allows existing programs to run as normal, using a commodity (untrusted) operating system in the untrusted VM. Meanwhile, trusted applications are isolated by the hypervisor in a new trusted VM.

While software virtualization can provide adequate security for many applications and use cases, it has some unresolved security and performance issues. First, it requires that the entire software stack, including the operating system, be trusted in the trusted virtual machine. However, commodity OSes that are publicly available today with the functionality demanded by applications, are not trusted. While there are some verified OS microkernels like seL4 [4], or ones designed to be trusted like seLinux [5], they are not widely deployed. Second, performance is degraded as costly *world switches* are needed to switch from an untrusted virtual machine to the hypervisor to the trusted virtual machine and back. Third, the isolation of virtual machines executing on the same physical machine provided by software virtualization cannot prevent the leaking of confidential information through the shared hardware resources (e.g., through cache-based side-channel attacks, discussed in the "Mitigating Hardware Information Leaks" section later in this chapter). This can leak secret encryption keys with correctly functional hardware resources, thus defeating any cryptographic protection for confidentiality or integrity.

Furthermore, typically only a small part of an application is security-critical. Rather than trying to certify that the entire large application is trusted, it may be easier to verify only the security-critical modules, i.e., verify that they perform the required functions correctly, do not leak confidential information, and do not contain bugs (security vulnerabilities) that can be exploited by attackers. These security-critical modules are typically much smaller in size than the entire application, and therefore amenable to advances made in static code analysis techniques and theorem proving techniques used to verify their trustworthiness [4]. Frequently, these security-critical modules must run in the same virtual address space as the untrusted application as, for example, in a security monitor running alongside the untrusted application it is monitoring. Hence, a virtual machine may be too coarse a granularity for a trustworthy space; we may need a finer granularity trustworthy space *within a virtual machine.* Possible hardware solutions to these problems are discussed below.

Bastion Security Architecture

Bastion is a hardware-software architecture [3, 6] that can provide secure execution environments for executing trusted software modules in an untrusted software stack. The trusted software modules encapsulate security-critical tasks and must be used to access protected data. In a Bastion system, the operating system in a virtual machine need not be trusted; only the microprocessor and the hypervisor must be trusted in order to enforce the security policies for confidentiality and integrity, for the trusted software modules and their data.

Figure 2 shows a block diagram of the Bastion architecture in a typical virtualized environment. It shows a computer with a hypervisor managing the hardware resources for two virtual machines: one running a Windows OS and the other running a Linux OS. Bastion leverages this virtualized environment, since it is very common in both client computers and cloud computing servers today. Rather than expect the entire hardware platform to be trusted, Bastion only requires that the microprocessor chip be trusted. In particular, the main memory is not trusted (unlike the assumptions for the systems using the Trusted Platform Module (TPM) chip [7], where the entire hardware box is considered trusted, including the main memory and buses). Similarly, rather than requiring the entire software stack to be trusted, Bastion only requires the hypervisor to be trusted. In particular, the guest OS in the VM is not required to be trusted to run the trusted software modules A, B, and C (shown in gray) within untrusted applications on an untrusted commodity OS.

For example, module A could be a security monitor running inside the virtual address space of Application 1, which it is monitoring for security breaches. Clearly, it is important to protect the security monitor itself—that is, protect module A from an attacker exploiting some security vulnerability in Application 1, some other application, or the OS to infiltrate the

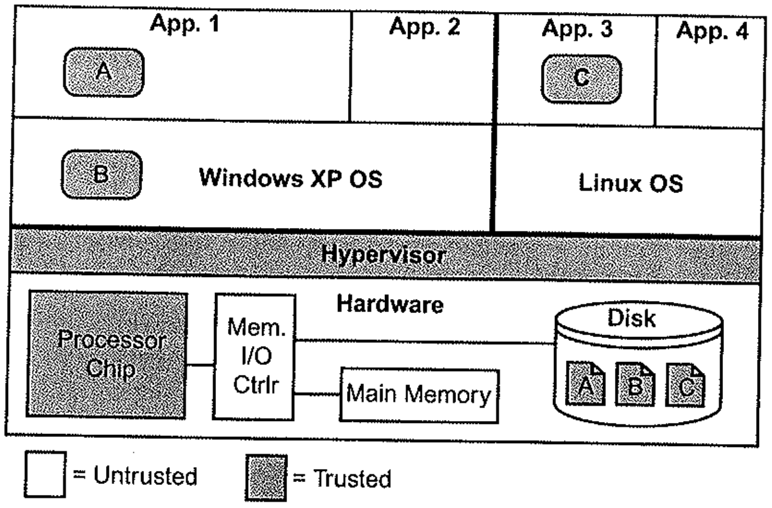

Figure 2. Bastion hardware-software security architecture.

system and attack module A. In general, an application writer may be very motivated to write a secure application (or secure modules within a large application) to protect his security-critical data, but he has no control over other applications or the OS. While OS vendors are highly motivated to secure their OS, it is such a large and complex piece of code that this is an extremely difficult task. Frequent security updates are evidence of the continuing vulnerability of an OS to attacks, despite the best efforts of OS providers. Bastion thus does not require the OS to be bug-free, but instead protects trusted software modules (e.g., A, B, and C) from a compromised OS.

Bastion architecture also provides module A with its own secure storage (shown in Figure 2 as a grey component A, in the local disk or remote storage). This is not a fixed contiguous portion of the disk or online storage, but rather the secure storage consists of any portions of the regular storage medium that are cryptographically secured (i.e., encrypted for confidentiality and hashed for integrity). Only module A has access to the keys to decrypt and verify the hash of its secure storage A. These keys are protected by the trusted hypervisor, which is itself directly protected by the trusted microprocessor in Bastion. This secure storage is persistent (a nonvolatile memory resource) in that it survives power on-off cycles.

Secure and authenticated memory is also provided to module A during its execution. This is volatile memory that does not survive power on-off cycles, but must be protected in order to provide a secure execution environment for a trusted software module like A, B or C in Figure 2. This and other hardware mechanisms [3, 6] provide a fine-grained, dynamically instantiated, secure execution compartment for trusted software. Hence, Bastion can be used to enable tailored trustworthy spaces within a sea of untrusted software, with hardware-hypervisor architectural support.

Figure 2 also illustrates that Bastion can support any number of trusted software modules from many different trust domains simultaneously. For example, Bastion supports trusted software modules in either application space (modules A and C) or OS space (module B). Module B can also have its own secure storage B, which is not accessible to the rest of the OS or any other software. Module C is another trusted software module in a different virtual machine.

Bastion also provides *trustworthy tailored attestation* [3], which enables a remote party to query the state of the trusted components it requires to perform a security-critical task. This can provide an unforgeable report of the integrity measurements of the trusted software modules required for the security-critical task, and of the hypervisor. Input parameters, configurations, and output results of a secure computation can also be included in the attestation report. Attestations can be requested any time, and are very fast.

Minimal Trust Chains and Secure Trust Anchors

An important difference with conventional TPM-based systems [7] is the minimal trust chain used by Bastion, which can skip layers of untrusted software, like the OS and some middleware. Bastion only requires that the microprocessor, the hypervisor, and the relevant trusted software modules form a secure trust chain, skipping layers of untrusted software in between. This minimal trust chain is illustrated on the bottom in Figure 3, while the traditional full trust chain where every layer of software must be trusted and unmodified is shown on the top. Minimal trust chains can enable more resilient execution of security-critical tasks, since they will not be prevented from executing due to updates or malware in unrelated software.

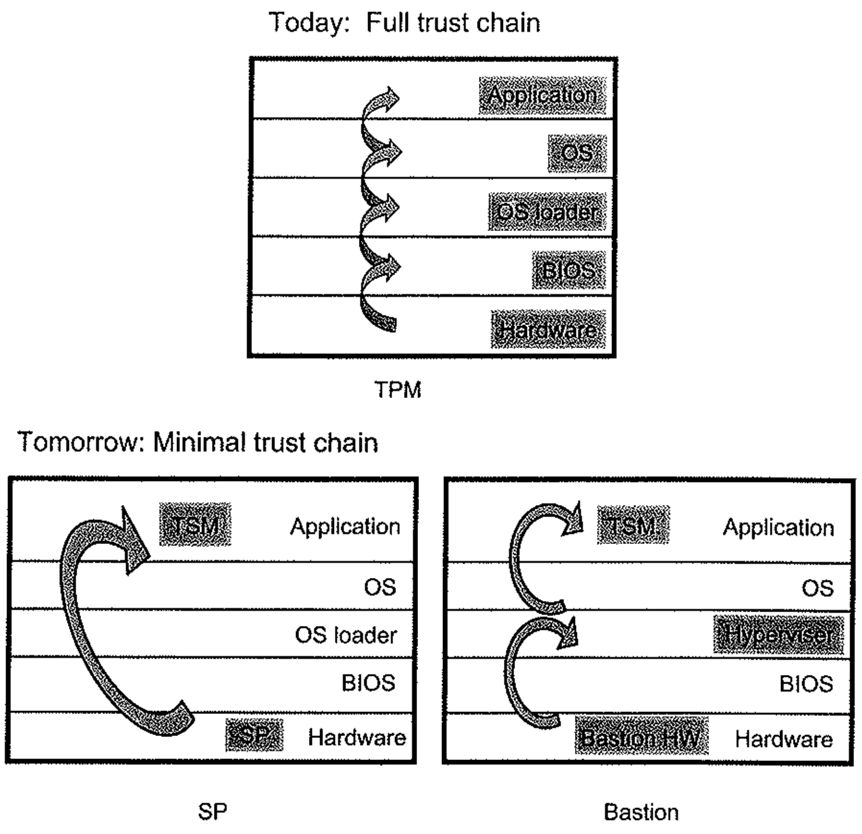

Figure 3. Conventional and minimal trust chains.

However, it is the responsibility of the applications designer to ensure that all the needs of the security-critical task are encapsulated in the trusted software modules and the hypervisor. The integrity of this minimal chain of trusted hardware and software components can then be reported upon a *tailored* attestation request.

While not every software layer has to be verified and attested to in a minimal trust chain, it is important that this chain is securely rooted. In Bastion, this is securely rooted in the trusted hypervisor and the new hardware registers and mechanisms used to protect the trusted hypervisor.

The concept of minimal, layer-skipping trust chains was first proposed in an earlier secure processor architecture called the Secret Protection

(SP) architecture [8, 9]. Here, a trusted hypervisor is not needed, and the untrusted OS runs directly on hardware. SP uses only two new registers as hardware trust anchors to anchor a minimal trust chain for protecting a trusted software module, as shown in Figure 3. It is a simpler security architecture than Bastion, useful for embedded or closed systems, but is not scalable as Bastion for simultaneously protecting multiple, mutually-suspicious trust domains.

MITIGATING HARDWARE INFORMATION LEAKS

New hardware architecture can be used to enhance the security provided by software. For example, hardware can assist software in providing tailored trustworthy spaces and secure trust anchors, as illustrated above by the Bastion and SP architectures. In addition, hardware itself should also be designed to be more trustworthy. For example, it should not leak secret information when operating correctly. Below, I discuss how this can happen, and show an example where a moving target defense is used in hardware design, to design leak-free caches that also improve performance.

In hardware design today, a fundamental strategy used for performance optimization is what I will characterize as a strategy to "make frequent paths fast, but allow infrequent paths to be slow." This enables the average execution time taken to be much closer to the fast path. For example, the frequently referenced memory items are stored in a fast cache memory that is much faster but smaller than the main memory. If the processor wants an item from memory, it gets this item very quickly if the item is found in the fast cache (called *a cache hit*). Otherwise, a *cache miss* occurs and it takes a long time (currently hundreds of processor cycles) to fetch the item from the main memory. This simple strategy has worked well for hardware performance optimizations, reducing the execution time taken.

Unfortunately, the two possibilities of a fast path and a slow path, which can be observed by anyone, or any program, that can time such operations, can be used to encode a binary "0" or "1." Hence, such performance optimization hardware mechanisms can be used to leak secret information through both covert channels and side channels. In a *covert channel attack*, an authorized insider leaks secret information to an unauthorized recipient using mechanisms not intended to be communications channels. In a *side-channel attack*, an insider is not even needed—for example, correctly functioning hardware can leak information without violating any usage or security policies.

In a power analysis side-channel attack [10], for example, an attacker can measure the power consumption of a program, and thus infer secret information such as a secret cryptographic key embedded inside the computing device. While most hardware side-channel attacks require physical access or proximity to the device, cache-based side-channel attacks do not. They can be triggered remotely, and they also do not need any special equipment to obtain the side-channel information [11, 12, 13, 14]. For example, a simple software side-channel attack using the cache can be constructed easily by having a "listener" program run on the same computer as a victim program, accessing data in a large data structure that uses up all the cache lines in the shared hardware cache, and timing each access. When the victim program is scheduled to run, it will replace some of these cache lines with the data it needs. When the listener program is scheduled to run again, it will access its large data structure again and time each access: a fast cache hit indicates his data has not been replaced in the cache, but a slow cache miss indicates that it has been replaced—most likely by the victim program's data. This can be used to leak information about which memory locations the victim program has accessed, since all computers use a static, fixed, memory-to-cache mapping for all programs. From this, the attacker can deduce which table entries were used by the victim program. If the victim program was an encryption program like the Advanced Encryption Standard (AES) [15], the attacker can then deduce which AES table entries were accessed in the cipher and hence what key bits were used to index these tables. This can be used to infer some or all of the bits of the secret encryption key.

Such cache-based side-channel attacks can undermine the strong cryptography used to encrypt secret information to protect its confidentiality. They can also undermine the isolation of different virtual machines provided by hypervisors. In general, the attacker's strategy is simple: exploit shared hardware performance optimization mechanisms like caches or branch prediction mechanisms to leak information. Hardware power optimization mechanisms can also be exploited similarly. Hence, we recommend that new strategies for performance optimization should be researched that can improve performance without sacrificing security.

Newcache: Moving Target Defense in Hardware

The Newcache architecture uses a new moving target strategy in hardware cache design to achieve both performance and security improvements simultaneously [16]. In particular, rather than use the conventional fixed, static mapping of memory addresses to cache lines, it uses a dynamic, randomized mapping of memory addresses to cache lines. Hence, even if the same program is executed on the same computer, it will have a different memory-to-cache mapping each time it executes. Also, the same program executing on a different machine of identical configuration will have a different and unpredictable memory-to-cache mapping. Hence, even if the attacker can observe which physical cache lines hit or miss, he or she cannot deduce any information about the memory locations actually used by a victim program. This thwarts these fast and dangerous cache side-channel attacks—without requiring any software changes in the application programs, nor any changes in the compiler or the instruction set architecture (ISA) of the microprocessor executing the programs. Hence, it is a hardware solution for a hardware information leakage vulnerability. It is also an example that shows that hardware can itself be designed to be more secure and trustworthy; in this case, to not leak information.

In the past, design for security has been at odds with design for performance. One had to choose either a secure system or a high performance system. The Newcache example shows that this need not be the case if we are willing to rethink some basic aspects of computer design. Here, permitting the freedom to do a *clean-slate* design is needed, although the resulting design may in fact not need to change too much of established designs. In Newcache, we only need to change the address decoder—something that has not been changed in the multitude of cache optimization designs proposed in the last three to four decades.

Newcache architecture also shows that using randomization in hardware design can in fact improve both security and performance. This is a surprising result, since cache performance is actually improved due to rethinking the design of caches for security [16]. It is especially surprising in such a mature area as cache design, which has been thoroughly studied by both researchers in academia and practitioners in industry, since caches are perhaps the most important component in the performance of modern computer systems.

Conclusion

With the growing complexity of distributed computer systems and also of computing devices like smartphones, it is impractical to assume that we can achieve perfectly secure systems. Rather, tomorrow's systems must operate securely in the presence of vulnerabilities and malware in the system. They should be resilient and should provide availability of systems and services to security-critical tasks, even under attack.

This chapter discusses some promising game-changing strategies for improving cyber security: enabling tailored trustworthy spaces, thwarting attackers with proactive moving target strategies, and rewarding responsible behavior in cyberspace with economic or other incentives. It is important for academia, industry, and government to work together to research, develop, and incentivize the ubiquitous use of more secure and trustworthy computer products.

Some concrete examples are given to show that these strategies can be used in hardware design to significantly improve cyber security: by either using hardware to enhance the security provided by software, or improving the trustworthiness of the hardware itself. For example, the Bastion hardware-software architecture can be used to help software systems achieve tailored trustworthy spaces, while the Newcache architecture shows how to build more trustworthy hardware—in this case, secure caches that cannot be used to leak information to adversaries—by using a moving target defense in hardware design. Surprisingly, Newcache can improve performance and power-efficiency at the same time as it improves security.

These hardware feasibility examples demonstrate that it is possible to build new hardware features into future commodity processors to enhance cyber security. They are research architectures that should be further evaluated for their validity in different application domains, their robustness against new attacks, and the impact they will have on the current software and hardware ecosystems. Industrial-strength design validation and security verification by teams of professionals are needed before deployment.

The good news is that it may be possible to build security into the hardware and software of future commodity computing and communication devices that can be used to improve cyber security significantly, without degrading their performance or versatility.

REFERENCES

[1] "National Cyber Leap Year Summit 2009 Co-Chairs' Report, Sept 16 2009" (2009, September 16) [Online]. Available: http://www.cyber.st.dhs.gov/docs/National_Cyber_Leap_Year_Summit_2009_Co-Chairs_Report.pdf

[2] Federal Cybersecurity Game-Change R&D Themes (2010, May 19). Kick-off Event 2010 held in conjunction with Security and Privacy Symposium, Oakland, California [Online]. Available: http://www.nitrd.gov/CSThemes/player.html http://www.nitrd.gov/fileupload/files/CSIAIWGCybersecurityGameChangeRDRecommendations20100513.pdf

[3] D. Champagne (2010, April). "Scalable Security Architecture for Trusted Software," PhD Thesis, Electrical Engineering Department, Princeton University [Online]. Available: http://palms.ee.princeton.edu/publications

[4] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, "seL4: Formal verification of an OS kernel," In *Symposium on Operating Systems Principles (SOSP)*, pp. 207–220, October 2009.

[5] seLinux. Web site and papers [Online]. Available: http://www.nsa.gov/research/selinux/

[6] D. Champagne and R. B. Lee, "Scalable Architecture for Trusted Software," Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA), January 2010.

[7] Trusted Computing Group, *TPM Main Specifications* [Online]. Available: http://www.trustedcomputinggroup.org/resources/tpm_main_specification

[8] J. Dwoskin and R. B. Lee, "Hardware-rooted Trust for Secure Key Management and Transient Trust," *ACM Conference on Computer and Communications Security (CCS)*, pp. 389–400, October 2007.

[9] R. B. Lee, P. Kwan, J. P. McGregor, J. Dwoskin, Z. Wang, "Architecture for Protecting Critical Secrets in Microprocessors," *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pp. 2–13, June 2005.

[10] C. Kocher, J. Jaffe, and B. Jun. "Differential power analysis," *Advances in Cryptology – CRYPTO'99*, Vol. 1666 of Lecture Notes in Computer Science, pp. 388–397, 1999.

[11] D. J. Bernstein, "Cache-timing Attacks on AES" [Online]. Available: http://cr.yp.to/antiforgery/cachetiming-20050414.pdf

[12] C. Percival, "Cache Missing for Fun and Profit" [Online]. Available: http://www.daemonology.net/papers/htt.pdf

[13] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and Countermeasures: the Case of AES," *Cryptology ePrint Archive*, Report 2005/271, 2005.

[14] Z. Wang and R. B. Lee, "New Cache Designs for Thwarting Software Cache-based Side Channel Attacks," *Proceedings of the 34th International Symposium on Computer Architecture (ISCA 2007)*, pp. 494–505, June 2007.

[15] NIST, "FIPS-197: Advanced Encryption Standard," November 2001.

[16] Z. Wang and R. B. Lee, "A Novel Cache Architecture with Enhanced Performance and Security," *Proceedings of the 41st. Annual IEEE/ACM International Symposium on Microarchitecture (Micro)*, pp. 88–93, December 2008.