Magic Boxes and Boots: Security in Hardware

Sean Smith, Dartmouth College

software via an encrypted channel into a tamper-resistant module, which then decrypts and executes it inside its private space.

In 1990, Steve White and Liam Comerford presented their Abyss secure coprocessor architecture for software protection. To overcome the physical space limits of the small coprocessor, they proposed *partitioned computation*: The magic box housed the critical—and, hopefully, hard-to-reverse-engineer—portion of the software, which interacted with the rest of the software running on the unprotected host.

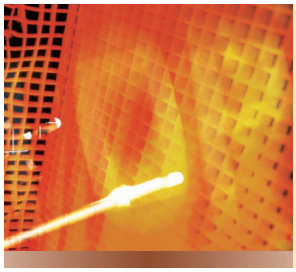
An old friend of mine once observed that he'd never seen software do much without some hardware to run it on. Or he might have said that he'd never seen computer hardware do much useful without some software; the point is the same.

Computer users tend to think of computation—even the globally distributed computation that constitutes the Internet—in terms of what we see: the browser user interface, the text editor, the Gnome or OS X or Windows desktop.

Similarly, we computer security folks tend to think of computer security problems solely in terms of what users see: the application software, perhaps the OS underneath it, or perhaps even the "end to end" environment from one application installation to another.

Although we lament the endless stream of security problems that emerge, we limit our playing field to these upper computation layers. Computation exists, however, not in the application alone, but also in the context of a software stack, the hardware, and the physical environment in which the hardware resides.

How can we use hardware to change the security game?



Moore's law helps a long history of research to support hardware-enforced system security.

SECURE COPROCESSORS

From a security perspective, the first idea that comes to mind is to add a "magic box" to the computer—one that can hide secrets and computation even from an adversary with direct physical access.

Such magic boxes have a long history of names—tamper-resistant module, secure coprocessor, trusted platform module—to go with the history of technology both to build them and to break them (S. Smith, "Fairy Dust, Secrets, and the Real World," *IEEE Security and Privacy*, Jan./Feb. 2003, pp. 89-93). The "Further Reading in Hardware-Based Security" sidebar lists a few reference papers chronicling developments in this field.

Software protection

The main branch of secure coprocessor work emerged in the context of software protection. In the late 1970s, Robert Best proposed a CPU that encrypted its address and data buses so that only it knows the plaintext.

In his 1980 PhD thesis, Steve Kent generalized this approach to *encrypted storage*: The software vendor ships the

Security services

White led a team that further refined the Abyss architecture into the Citadel design, which brought two changes worth noting.

First, it showed a shift in thinking. Instead of being a place for locking up proprietary software, Citadel's magic box provided security services to the rest of the host machine.

Second, Citadel matured into a series of real hardware prototypes, many of which actually worked. Bennet Yee and Doug Tygar used some of these prototypes to build their Dyad system at Carnegie Mellon University. The coprocessor now ran a modified Mach microkernel. The internal space limitations were mitigated by *cryptopaging*, which provides virtual memory for the coprocessor's internal applications. The coprocessor uses the host as its backing store and employs cryptographic hardware and FIFOs to encrypt the contents on the way out and to decrypt them on the way back in.

Having a trusted magic box use an untrusted storage area—as in Best's design for instructions and Yee and Tygar's design for memory pages—raises some questions. Encryption protects the plaintext contents from an adversary, but what about the access patterns? Oded Goldreich noticed this vulnerability in 1987, and his observation led to a series of theoretical results in *oblivious RAM*. However, these techniques were deemed too impractical to find real-world use.

Yee and Tygar also developed a series of applications for their Dyad platform: decentralized electronic currency, postal meters, secure audit logs, and electronic contracts. In addition, they developed the idea of using the magic box to check the software's integrity while participating in the host's boot process. Previous host-integrity-checking approaches, such as Tripwire, did not have the trusted verifier that a magic box provided.

A business opportunity

In the mid-1990s, I was doing security analyses at Los Alamos National Laboratory for public-sector entities that wanted to migrate their services to the emerging Web environment.

Intrigued by the potential of using secure coprocessors to address trust issues in these new scenarios, I asked the Citadel group at IBM if they had any more platforms available. They said no, but suggested that I come and do my proposed research there. After I signed the employee confidentiality agreement, they explained why.

IBM had found another use for hardware-based security—protecting cryptographic accelerators. Financial institutions constitute a sizable market for crypto accelerators—one that is quite comfortable with the idea of armored boxes. IBM had decided that its next-generation crypto accelerator product needed to have the features the secure coprocessor researchers had been advocating. (Secure personal tokens, such as Fortezza cards, represent another branch—but that's another story.)

Further Reading in Hardware-Based Security

Magic box design and use in security applications have a long history. Here are some representative publications:

- W.A. Arbaugh, D.J. Farber, and J.M. Smith, "A Secure and Reliable Bootstrap Architecture," *Proc. IEEE Symp. Security and Privacy*, IEEE CS Press, 1997, pp. 65-71.
- D. Asnonov, *Querying Databases Privately: A New Approach to Private Information Retrieval*, LNCS 3128, Springer-Verlag, 2004.
- R.M. Best, "Preventing Software Piracy with Crypto-Microprocessors," *Proc. IEEE Spring Compcon 80*, IEEE CS Press, 1980, pp. 466-469.
- P.C. Clark and L.J. Hoffman, "BITS: A Smartcard-Protected Operating System," *Comm. ACM*, Nov. 1994, pp. 66-70.
- O. Goldreich and R. Ostrovsky, "Software Protection and Simulation on Oblivious RAMs," *J. ACM*, vol. 43, no. 3, 1996, pp. 431-473.
- A. Iliev and S.W. Smith, "Private Information Storage with Logarithmic-Space Secure Hardware," in *Information Security Management, Education, and Privacy* (Proc. i-NetSec 04: SEC2004 Embedded Workshop on Privacy and Anonymity in Networked and Distributed Systems), Y. Deswarte et al., eds., Springer-Verlag, 2004, pp. 201-216.
- N. Itoi and W.A. Arbaugh, "Personal Secure Booting," *Proc. 6th Australasian Conf. Information Security and Privacy*, LNCS 2119, Springer-Verlag, 2001, pp. 130-141.
- S. Jiang, S.W. Smith, and K. Minami, "Securing Web Servers against Insider Attack," *Proc. 17th Ann. Computer Security Applications Conf.*, www.acsac.org/2001/abstracts/thu-1030-b-jiang.html.
- S.T. Kent, *Protecting Externally Supplied Software in Small Computers*, doctoral dissertation, Laboratory of Computer Science, Massachusetts Inst. of Technology, 1980.
- R.R. Sailer et al., "Design and Implementation of a TCG-Based Integrity Measurement Architecture," *Proc. 13th Usenix Security Symp.*, Usenix, 2004, pp. 223-238.
- S.W. Smith and S. Weingart, "Building a High-Performance, Programmable Secure Coprocessor," *Computer Networks*, special issue on computer network security, Apr. 1999, pp. 831-860.
- J.D. Tygar and B.S. Yee, *Dyad: A System for Using Physically Secure Coprocessors*, tech. report CMU-CS-91-140R, School of Computer Science, Carnegie Mellon Univ., 1991.
- S.R. White et al., *Introduction to the Citadel Architecture: Security in Physically Exposed Environments*, research report RC 16672, IBM T.J. Watson Research Center, 1991.
- S.R. White and L. Comerford, "ABYSS: An Architecture for Software Protection," *IEEE Trans. Software Eng.*, June 1990, pp. 619-629.

So we were given a budget and free rein to produce a secure coprocessor platform, provided we delivered it on time and in a form that could be turned into a crypto box supporting IBM's application.

The real thing doing the right thing

My main goal was to develop a platform that would enable the deployment of secure coprocessor applications by all sorts of people, including young

security researchers at Los Alamos.

One main principle drove the resulting architecture: An untampered magic box should always be able to prove it's "the real thing doing the right thing," despite a variety of mutually suspicious software developers and potentially adversarial code. This work resulted in the IBM 4758 secure coprocessor platform (and quite a few war stories, but that's a different subject). IBM even released free tools for software development, which enabled me to continue experimenting in application development at Dartmouth.

The 4758 architecture persisted in a follow-on product released recently, but developers' tools aren't available yet. Consequently, my students spend too much time trying to fit large objects in small places.

One example of how secure hardware changes things: Combining the trusted environment with the 4758's fast crypto led to a series of results—in my group, then over to Dmitri Asonov at Humboldt Unversity and IBM, and back to my group again—that included building real-world instantiations of oblivious RAM techniques previously deemed impractical.

SECURE BOOTSTRAP

Dyad's host-integrity checking application also started another branch of work: using special hardware in a host's boot process to help ensure its software's integrity.

In 1994, Paul Clark and Lance Hoffman used a smart card to help in a host's boot process. In 1997, Bill Arbaugh developed and prototyped Aegis, which used a modified BIOS and an external PROM board to systematically verify each software component cryptographically and replace ones that failed the check. Nao Itoi later extended Aegis to use smart cards.

More recently, the multivendor Trusted Computing Platform Alliance (TCPA)—now renamed the Trusted Computing Group (TCG; www.trustedcomputinggroup.org/), just to keep you on your toes—has developed an evolving series of specifications that build on this idea. A Trusted Platform Module participates in the boot process to verify integrity, ensure that TPM-protected secrets are released only to the appropriate software stack, and attest various properties to a remote party.

In some eyes, TCPA—the term that has come to denote this architecture and its vision—is intimately tied to having someone other than the user control what software a PC runs—a topic that has roots in Kent's thesis (and perhaps earlier) but continues to raise heated discussions.

Researchers are also looking at hardware techniques to combat buffer overflow.

TPM chips

Some chip vendors have been producing TCPA-compliant TPMs; some PC vendors have been including them on commercial platforms. Nothing stops a researcher from buying such a platform and experimenting—nothing, that is, except such obstacles as outdated BIOS devices, TPMs that lag behind the latest specs, TPMs that don't quite comply with the documentation, and documentation that is often vague and puzzling.

I speak from experience here. My group noticed that the secure boot approach of a TPM-enhanced PC might transform it (with a bit of work) into a box that's "the real thing doing the right thing"—specifically, a more powerful, less physically secure, but cheaper and potentially far more pervasive cousin of the 4758.

We started a project to make this possible. We used IBM's published device driver to write our own TPM library, but IBM scooped us by publishing theirs in summer 2003; we liked it better, so we switched to it.

As a target application, we considered the problem of how a Web user decides to trust a remote server. The server proves knowledge of a long-lived private key, but this key is bound only to the server's identity, not to the actual service being provided.

The WebALPS project was our attempt to solve this on the 4758. We moved the key and the application it attests to into a secure coprocessor platform, but the small 4758 environment made using WebALPS awkward, and it never found uses in the field.

TPM-equipped workstations

What about using a TPM-equipped workstation instead?

Solving this trust problem requires binding the Secure Sockets Layer private key to the application. Although TPM can bind the SSL key to a particular software stack, this approach does not address a lifetime mismatch: A server key pair lives a long time, but the server software deemed currently trustworthy changes often, and the Web pages and CGI scripts that constitute a service may change even more often than server software.

To work around this, we had the TPM verify a minimal Linux kernel. The kernel includes a Tripwire-like module that verifies the rest of the system against a signed configuration file. If things match, Apache gets its private key. The SSL certificate authority (CA) testifies to the server's identity and the configuration signer's continued good judgment. Potentially, the same businesses that sign SSL server certificates might get into this configuration-signing business as well.

We developed this platform, called the Bear/Enforcer, which is available as open source (http://enforcer.sourceforge.net). Arguably, it might be the first open-source TCPA/TCG platform, and we've had more than 1,000 downloads to date. Consequently, we've achieved our real goal—experimentation is taking root.

In our lab, we've extended the platform to use SELinux, which lets users run a digital rights management (DRM)

application in a platform compartment that attests to a remote vendor exactly what's in the compartment—and nothing else. The vendor can then trust that the DRM application will run as intended, and users can trust that the vendor learns nothing else about the machine.

We're also merging this with OpenCA—so that the operating party, and perhaps the sites certifying or cross-certifying it, can have some assurance that the CA private key is released only to a platform that complies with appropriate policy.

In concurrent research, Reiner Sailer and others at IBM Watson have extended the TCPA/TCG attestation process up through the application.

FUTURE ENHANCEMENTS

Of course, using hardware to change—and hopefully simplify—the security game need not be confined just to the secure coprocessor/secure bootstrap story. Arbaugh's group is using hardware coprocessors to check integrity not just at boot time, but at regular intervals during runtime. When prompted, certain graybeards will preach the virtues of capability-based systems—and the tagged architectures that gracefully support them—as technology that ought to be reconsidered.

Ongoing academic efforts—such as the XOM project at Stanford and Toronto, another project called Aegis (this one at MIT), and work in Ruby Lee's group at Princeton—focus on building CPUs with additional security enhancements, such as the ability to bind an encrypted chunk of code to an encrypted chunk of data.

Researchers are also looking at hardware techniques to combat buffer overflow. Half the participants in the "Software Security" panel at the NSF Cyber Trust meeting in August 2004 even suggested using hardware to help with software security issues—such as providing acceleration for type-checking.

Some colleagues scoff at the idea of using hardware to secure software. I respectfully disagree. Looking back, hardware developments such as kernel-user CPU modes and hardware-enforced memory management contributed greatly to producing programming environments that make it much easier to produce secure systems.

Moore's law is in our favor. I look forward to seeing what happens next. ■

Sean Smith is an assistant professor of computer science at Dartmouth College and director of the Cyber Security and Trust Research Center at the Institute for Security Technology Studies. His book, Trusted Computing Platforms: Design and Applications *(Kluwer), will appear in January 2005. Contact him at sws@cs.dartmouth.edu.*

Editor: William A. Arbaugh, Dept. of Computer Science, University of Maryland at College Park; waa@cs.umd.edu