Nexus: A New Operating System for Trustworthy Computing

Alan Shieh Dan Williams Emin Gün Sirer Fred B. Schneider

Cornell University
{ashieh,djwill,egs,fbs}@cs.cornell.edu

Tamper-proof coprocessors for secure computing are poised to become a standard hardware feature on future computers. Such hardware provides the primitives necessary to support *trustworthy computing* applications, that is, applications that can provide strong guarantees about their run time behavior.

Current operating systems, however, lack the architecture and abstractions required to support trustworthy computing. Traditional operating systems, whether monolithic or based on a microkernel architecture, rely on a large trusted computing base (TCB) that is error-prone, expensive to audit, and inherently difficult to trust. Current abstractions they provide do not support the newly emerging secure coprocessing hardware. The construction of trustworthy applications, whose run time properties can be queried, inspected and assured, remains ad hoc in the absence of general-purpose system abstractions for trusted computing.

We are building a new operating system, called the Nexus, to support trustworthy computing applications. The Nexus has three unique properties: a novel architecture to reduce the trusted computing base, a general-purpose and flexible attestation mechanism to establish statements about the current state of a computation, and a strong, high-performance isolation mechanism to enable reasoning about future behavior based on statements about the present.

The Nexus system organization reduces the size of the hardware and software TCB in two ways. First, functionality that is traditionally found inside the kernel is accomplished in isolated, fine-grain, unprivileged service components outside of the Nexus. For example, the Nexus executes device drivers in virtual user space sandboxes, which reduces the size of the kernel significantly. A *driver containment policy* specifies the types of operations the device driver can perform. The Nexus enforces these policies and ensures that device drivers cannot violate isolation semantics. Device driver isolation enables the Nexus to not have to rely on the implementation of the device driver for correctness. A device driver that attempts to circumvent memory protection by, for instance, manipulating DMA buffers, will be caught during containment policy enforcement. As another example, the Nexus further reduces the size of the TCB by eliminating the need to trust any secondary storage in the system. This admits a more realistic threat model, where a malicious user could rewrite or replay information on the disk at any time.

The Nexus provides a flexible and comprehensive attestation mechanism to enable reasoning about software properties. While traditional attestation chains identifies applications by their hash, the Nexus generates descriptive names for applications by running signed code, known as labeling functions, over the applications' run-time state, including the state of reference monitors acting on an application. *Authentication by attestation* is extensively used to authorize access to other processes and system services; resources protected in such a fashion are accessible only if a matching attestation chain is presented.

Labeling functions can represent a wide range of assertions about applications, and can even distinguish families of programs via generic properties. In an anti-spam system, an e-mail client can be labeled with the number of keystrokes and file accesses, as determined by a reference monitor; such a name represents an execution state where the user has manually composed a message. In a peer-to-peer application, a remote client can be labeled with the number of packets that have been forwarded recently, and with the hashes of the data blocks about to be sent. A client can use these names to determine whether the remote node is properly following the protocol.

The Nexus provides novel secure memory region abstractions to enable trustworthy computing applications. Secure memory regions guarantee integrity and optionally support confidentiality, can be stored persistently on secondary storage, and are resilient against secondary storage attacks such as replay. The Nexus decouples the integrity and confidentiality mechanisms from the implementation of the persistence mechanism, enabling applications to use any storage mechanism, and thereby eliminating the filesystem from the TCB.

The memory region abstractions are simple, efficient, and most importantly, sufficiently flexible for constructing a range of higher-level services. For instance, we have implemented a capability

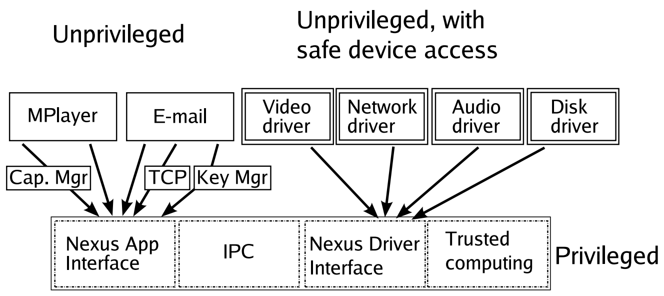

Figure 1: Nexus system architecture

manager for managing general-purpose object capabilities based on the confidential, integrity-protected memory region abstraction. This capability manager can implement general-purpose security automata [6], which can capture, encode, and act on past state in making access control decisions. Implementing such a service with current systems is difficult, as they are vulnerable to replay attacks on the disk. The Nexus thus supports *linear capabilities*, which provide a hard limit on the number of times a capability can be used. The flexibility of the low-level memory region abstraction provided by the Nexus enables the capability manager to be implemented entirely in user space. In contrast, capabilities are typically implemented in the kernel on traditional systems, increasing TCB size and forcing applications to use a specific interface and implementation.

We have implemented the Nexus kernel and user services on x86 hardware, along with several applications that use the new trustworthy computing abstractions that the Nexus and middleware services provide. One such application is a media player, which uses the capability manager to obtain a linear capability to play a movie a limited number of times. Only a media player that matches the labeling function authorized to obtain a linear capability, for instance, one that runs in a reference monitor that blocks network and file writes, can request the capability, and the capability will only be granted for the first k viewings. We have also implemented a spam-resilient e-mail system, which labels e-mail clients with a labeling function (as described above) indicating that a message has been typed by a human. The attestation chain is sent with the message, which can be examined at the recipient, who then knows that the message is unlikely to be spam, and presents the message to the user.

The Nexus differs fundamentally from past work on creating a trustworthy execution environment. The XOMOS system [3, 4] explored new processor architectures to enable trustworthy computing, and posited a system design where the secure processor would obviate trust in the operating system. However, since the required changes affect the central processor, and have numerous performance implications, this approach remains unadopted by industry. In contrast, the Nexus is designed to work on top of the commonly-deployed secure coprocessors, known as trusted platform modules, that are an industry standard [8]. Pioneer provides a dynamic root of trust for an attestation-like mechanism called *verifiable code execution* for legacy platforms that lack a TPM [7]. It may be possible to combine verifiable code execution with a trusted dispatcher infrastructure to provide sufficient attestation guarantees to replace the TPM from the Nexus TCB. The TCGLinux system [5] proposed adding attestation capabilities to Linux. This approach fails because it does not take steps to reduce the large trusted computing base that a Linux system entails, or to provide isolation between applications. It is all too easy to use the latest Linux security advisory to compromise the system and produce falsified attestation certificates (by, for instance, attesting to an intact but disabled version of the original service, while executing a different, compromised version). Terra [2] is an execution environment for trustworthy computing based on virtualization. The TCB is composed of the Terra virtual machine monitor (VMM), which provides isolation between trusted applications by creating distinct virtual execution on the same physical computer for each application, and attests to the load-time contents of each virtual machine. Attestation in Terra generates a single hash describing an entire virtual machine. Since each application requires a full-blown virtual machine, this approach does not scale well with increasing numbers of domains (e.g. applications). Terra executes drivers at high privilege, so its TCB is larger than that of Nexus. While Asbestos [1] does not directly address trustworthy computing, its characterization of the label passing interactions between components is a useful run-time policy that can be encoded within Nexus attestation chains.

Acknowledgments

Supported in part by AFOSR grant F49620-03-1-0156, National Science Foundation Grants 0430161 and CCF-0424422 (TRUST), ONR Grant N00014-01-1-0968, and a grant from Intel Corporation.

References

[1] P. Efstathopoulos, M. Krohn, S. VanDeBogart, C. Frey, D. Ziegler, E. Kohler, D. Mazieres, M. F. Kaashoek, and R. T. Morris. Labels and Event Processes in the Asbestos Operating System. In *Proceedings of the Symposium on Operating Systems Principles*, Brighton, UK, Oct. 2005.

[2] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A Virtual Machine-Based Platform for Trusted Computing. In *Proceedings of the Symposium on Operating Systems Principles*, Bolton Landing, NY, Oct. 2003.

[3] D. Lie, C. Thekkath, and M. Horowitz. Implementing an Untrusted Operating System on Trusted Hardware. In *Proceedings of the Symposium on Operating Systems Principles*, Bolton Landing, NY, Oct. 2003.

[4] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz. Architectural Support for Copy and Tamper Resistant Software. *ACM SIGPLAN Notices*, 35(11):168–177, 2000.

[5] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and Implementation of a TCG-based Integrity Measurement Architecture. In *Proceedings of the 13th Usenix Security Symposium*, pages 223–238, August 2004.

[6] F. B. Schneider. Enforceable security policies. *ACM Transactions on Information and System Security*, 3(1):30–50, 2000.

[7] A. Seshadri, M. Luk, E. Shi, A. Perrig, L. van Doorn, and P. Khosla. Pioneer: Verifying Integrity and Guaranteeing Execution of Code on Legacy Platforms. In *Proceedings of the Symposium on Operating Systems Principles*, Brighton, UK, Oct. 2005.

[8] Trusted Computing Group. TPM Specification Version 1.2.