InkTag: Secure Applications On An Untrusted Operating System

Owen S. Hofmann[1,2], Michael Z. Lee[2], Alan M. Dunn[2], Emmett Witchel

The University of Texas at Austin {*osh,mzlee,adunn,witchel*}*@cs.utexas.edu*

Building secure systems is incredibly difficult in part because the operating system (OS) is a shared vulnerability for *every* program on the system. Many of the most visible computer worms of the past decade (such as Stuxnet, Nimda, SQLSlammer, Code Red, Sasser and Conficker) are particularly damaging because they can compromise the OS. We describe InkTag, a system where processes can run without trusting the OS, while still running at user level and making use of OS services such as scheduling and I/O. Processes instead rely on communication with a small trusted hypervisor to verify the results of OS operations. InkTag makes building secure systems easier, because it eliminates the OS from the trusted computing base (TCB).

Building secure applications without trusting the operating system, while audacious, is plausible because although operating systems provide services with complex implementations, these services often have simple specifications (e.g., a virtual address space). The vast amount of code in operating systems is due to providing these simple services simultaneously to many different processes—the global behavior and resource management problem is much more complicated than the specification for an individual process. Checking the operating system's work would remove much of it from the trusted computing base, and the smaller the TCB, the easier a system is to understand and secure.

In InkTag, *high-assurance processes*, or HAPs, are the principals. HAPs communicate with the trusted hypervisor to maintain privacy, integrity, and freshness of their data, even when using the services of a malicious operating system. Previous work, such as Overshadow [Chen et al. 2008], has proposed protecting processes from an untrusted OS. InkTag is the first to address essential systems features beyond simple isolation, such as establishing and enforcing access control policy. A key challenge for InkTag is to base the security of HAPs on the hypervisor *without* bloating the hypervisor's code or interface. The InkTag hypervisor does not include a security policy, but instead exports simple cryptographic operations that allow HAPs to define their own policies. HAPs then use cryptography to implement security much like a distributed system. For efficiency, InkTag uses shared key cryptography (encryption for privacy and hash-based message authentication codes (HMACs) for integrity) whenever possible. Key negotiation in InkTag is far easier than in a distributed system because the hypervisor is trusted, and hypercalls provide a completely secure, highly available channel between a HAP and the hypervisor.

Challenges of removing trust from the OS Removing trust from the OS creates unique challenges for secure systems. For example, access control is usually expressed in terms of OS-provided primitives, such as effective user ID and the current process ID. Trusted system applications such as `login` and system calls such as `fork` modify this operating system state—`login` changes the user identifier and `fork` changes the process identifier. However, HAPs cannot trust the OS as a central authority for holding state related to access control. Instead, they must maintain that state themselves, and it must be unforgeable by other applications. In addition, rather than implicitly trusting system applications because they run as the root user, access control in InkTag must place explicit trust in an application (such as `login`) to modify important access control state.

Removing trust from the OS also creates interesting opportunities. Access control in InkTag is delegated to HAPs, allowing for a wide variety of different schemes. HAPs can emulate traditional system access control by allowing a central authority to enforce policy. However, individual HAPs also have the ability to enforce application-level security requirements at the system level (the InkTag hypervisor). For example, a web application with many users can isolate those users' private data as if they were separate users on the local machine. This is not feasible in traditional operating systems, where adding a new protection domain (a new user ID) requires administrator-level access.

Isolating HAPs The most basic guarantee that InkTag must provide for a HAP is that its code, data and control flow are isolated from the operating system. This isolation is necessary, otherwise it would be impossible to distinguish the intended behavior of a HAP from behavior resulting from operating system interference. Users would be unable to trust the HAP any more than the untrusted operating system.

To provide isolation, the InkTag hypervisor provides the HAP with a *high-assurance address space*. A high-assurance address space is a sequence of (not necessarily contiguous) virtual pages with the following properties:

1. **Privacy.** Address space contents are private to the HAP. HAP code can read and write it, but the operating system or other processes can at most read data in encrypted form.

[1]Presenter

[2]Student Authors

2. **Integrity.** Only a HAP is allowed to correctly write to its secure address space; writes from the OS or other applications will be detected.
3. **Freshness.** A HAP that reads a page of data reads the data most recently written to that page. The OS cannot replace a page with a previous version.

Enforcing access control For both files mapped into the HAP address space, as well as accesses to files through system calls and the hypercall interface, the InkTag hypervisor must be able to determine if a HAP has access to a given resource. Furthermore, the result of that decision must be dynamic: access must be able to be granted to and revoked from different HAPs over time. Finally, the hypervisor mechanism should be simple and independent of OS file system abstractions. InkTag implements a capability system to grant access to HAPs.

For each file, InkTag tracks a *read secret*, a capability that grants a HAP read access to a file, and a *write secret*, a capability that grants a HAP write access to a file. A HAP that presents a file's read secret to the hypervisor will get back the initialization vector, shared encryption key and hash for any page of that file's data. A HAP that presents a file's write secret will be able to update a file page's hash. Future readers will check reads against the hash, ensuring data integrity. Access control in InkTag is made dynamic by allowing the read and write secrets to change in response to events such as revoking access to a resource. HAPs that have only previous read and write secrets cannot decrypt data written after the update, and they cannot update the file's contents.

Identifying HAPs Essential for usable access control policies are two pieces of information about a running HAP: the application that the HAP represents, and the context in which that HAP is executing.

InkTag allows HAPs to identify and group other HAPs by a *canonical name* that represents the application implemented by the running binary. For instance, the `/bin/cat` binary could be invoked simultaneously in many different contexts, each a separate HAP. Each of these instances should map to "cat", a canonical name for the `cat` application. To the hypervisor, a HAP is identified by the hashed contents of its initial code and data when it is loaded into memory. The trusted distributer of an application must sign this hash of the loaded binary along with a canonical name.

The context in which an application runs is as important as application identity in making access control decisions. For example, the `cat` application has access to different sets of files when invoked by different users. To enable access control decisions, InkTag allows applications to securely record the actions that affect access control. HAPs in InkTag maintain *histories* within their high-assurance address spaces (which are isolated from the operating system). Histories contain entries for events that affect access control state within a single process, such as calls to `setuid`. If a privileged program drops privileges, the relevant entry in the history indicates that any further actions occur in a less-privileged context. In addition to changes in privilege, the history contains entries for events that transfer control between HAPs, such as `fork` and `exec`.

Access control policy InkTag delegates access control policy to HAPs, enabling decentralized access control that is flexible enough to enforce application-level security requirements at the system level. Read and write secrets are the mechanisms by which individual HAPs prove to the hypervisor that they are allowed to access file information. High-level access control policies are implemented by the selective granting of read and write secrets to HAPs in accordance with a policy based on higher-level principals, such as users and groups.

HAPs grant read and write capabilities to other HAPs by possessing the *master secret*, a capability given to the creator of a file that allows the bearer to update the read and write secrets. Many access control schemes are possible. For instance, HAPs might rely on a central *access control daemon* (ACD) that possesses all master secrets and controls file access, similar to the centralized model of traditional operating systems. However, any HAP in InkTag can implement any security policy that can be mapped onto read and write capabilities, and have that policy be enforced by the InkTag hypervisor. For instance, a multi-user web application could assume responsibility for master secrets for its users' private files, and distribute read and write secrets to individual server processes handling requests on behalf of web users. By only releasing secrets for a single user to processes handling that user's requests, the application can make each user's files inaccessible to processes acting for other users. This guarantee holds even if a malicious user can inject code into a running process. In contrast, most web applications today handle all requests under a single OS user ID. A compromise in one application process can read the same secret data as any other application process. Varied access control schemes can coexist on the same system, for example allowing a central ACD to manage most access control while a web application implements its own policy.

Prototype We have implemented a prototype of InkTag within the KVM hypervisor. Through modifications to the C library, existing applications can be ported to run on InkTag with relative ease. Our prototype runs the Apache web server as a HAP, with full privacy and integrity for the process address space and file I/O, at a performance cost of about 1.7×.