Secure Computing Through Hardware-Enforced Fine-Grain Data Ownership

Jeremy H. Brown
(jhbrown@ai.mit.edu)

The Problem: Popular modern computer systems such as UNIX and Windows lack many important security and correctness features. Although processes are protected from one another by address-space sandboxing, they are not at all protected from malicious data entry, leaving many important programs vulnerable to attacks such as buffer-overflow. Data access is controlled at the rather coarse granularity of files stored on disk; data dissemination is completely uncontrolled. Privileged "root" processes have access to all sensitive functions, including all files, regardless of their actual ownership; there are no gradations between normal, unprivileged processes and "root" processes.

Motivation: Commercial and government agencies entrust computers with important, proprietary information, but because of the flaws described above, that information is vulnerable to unintended dissemination, whether through human error (for instance, government employees placing copies of classified material onto unclassified laptops) or malice. Malice is perhaps the greater concern: crackers routinely break into corporate systems to steal credit card information, and it is not unheard of for disloyal employees to give copies of critical information to rival companies or nations.

Previous Work: Programming languages such as Lisp and Java provide unforgeable pointers to objects, perform automatic memory management, and dynamically verify that a reference to an object falls within the object's boundaries. These features prevent information dissemination or system corruption due to memory-mismanagement problems such as buffer-overruns.

Recent work by Andrew Myers [3] describes a sophisticated model for tracking the ownership of data and preventing its release to unauthorized entities. Variables are labeled with ownership information. A trusted compiler verifies that information isn't leaked via any of the following means: explicitly copying data into less-restricted variables; implicitly copying data through conditional branching and subsequent writes; or writing data to less-restricted output channels.

On the hardware side, capability architectures ([1], [2]) are to conventional architectures much as Java is to C: capabilities (pointers) are hardware-distinguishable from integers, and accesses through capabilities are bounds-checked. Capability architectures avoid the all-or-none privileges of "root" by giving a process capabilities for exactly the set of objects upon which it needs to operate, without giving it access to any data irrelevant to its intended function.

Approach: Although better languages and compilers are inarguably good for security and correctness, we believe that to produce a simple, fast, and trustworthy computing base, we must begin with the hardware. Therefore, we are designing a modern capability architecture with support for a dynamic variant on Myer's data ownership labeling.

We will implement capabilities as pointers which include embedded object bounds; thus, verifying the legality of a memory operation via a capability will require no additional memory accesses. By exploiting the high transistor count of modern VLSI processes, we will place the verification mechanism in parallel with the normal datapath, thus eliminating any slowdown compared to conventional architectures.

We will place ownership labels on the data words themselves, rather than the variables into which they are stored. Each register in the processor, including the program counter, will contain both the primary data, and the data's ownership label. Rather than label every word in memory, however, we will only place labeling information between labeling domains, thus avoiding significant memory bandwidth overhead.

A hardware label-derivation cache will provide labels for newly derived data, based on the constituent data's labeling. This scheme is shown in the figure below. A miss in the label-derivation cache will be handled by software, much as a miss in a TLB is handled by software.

To prevent the label on the program counter from becoming monotonically more restrictive every time a conditional branch is taken, a hardware stack is provided upon which a target address, labeled with the current PC's security level, can be pushed. The hardware pop operation pulls the top address and label from the PC stack, and installs them as the new PC. This allows a less-restricted segment of code to set a less-restricted return-address before increasing PC restriction via conditional branches.

In addition to this hardware, a small amount of trusted software will be necessary to handle system-wide garbage collection, and to fill the label-derivation cache on a miss.

Difficulty: We are proposing to implement a an entire computer system, from the chip level up, containing a variety of experimental features. Although it is our goal to keep each element as simple as possible, our design goals require a certain amount of complexity. For instance, we propose to implement a fairly simple processing element, but one with at least three parallel *domains of execution*: the primary datapath, the capability-verification path, and the ownership-label derivation and verification path. We anticipate the construction of the entire system to be a major effort requiring significant investments of time and money.

Impact: A modern capability architecture eliminates a variety of common memory management errors at the hardware level, dramatically improving the security not only of new applications, but even of existing applications which are easily ported. Additionally, implementations of "safe" languages like Lisp and Java will run much faster on our capability architecture than on conventional architectures, since there is no need for software object bounds-checking.

The powerful, dynamic ownership model enables arbitrary compartmentalization in a lattice model which subsumes traditional linear classification schemes. A single computer can thus be used for multiple, distinct projects with no risk of inter-project information leakage. Declassification authority is not centralized: owners control access to their own data. Ownership is inherited: owners control access to copies of their data.

Total information-dissemination control means that organizations can give their employees access to corporate data and the Internet without enabling the employee to email corporately-owned data out over the Internet.

Future Work: Nearly the entirety of this project remains as future work. The next few steps are: produce a formal proof that our dynamic ownership model does not leak information via explicit or implicit channels[1]; develop a simulator for the processing element; develop the trusted code base for manipulating ownership labels and performing garbage collection; develop schematics and layout for the processing element; and so forth.

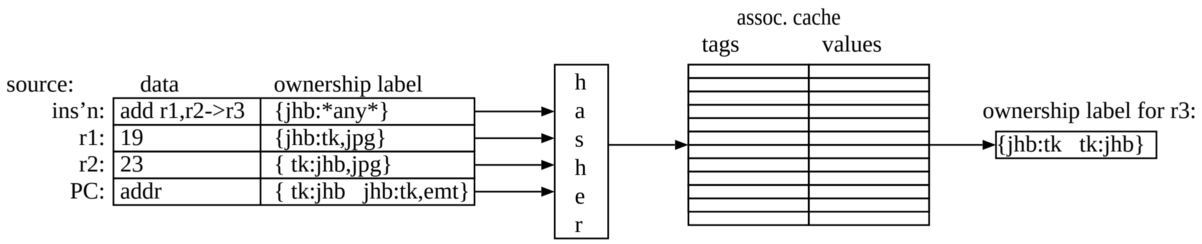

Figure: Looking up a derived ownership label in the label cache.

Research Support: This material is based upon work supported under a National Science Foundation Graduate Fellowship. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. Support for this research was provided in part by the Defense Advanced Research Projects Agency under Rome Labs Contract Number F30602-98-1-0172.

References:

[1] R. S. Fabry Capability-Based Addressing *Communications of the ACM*, Vol 17, No. 7, July 1974.

[2] H. M. Levy Capability-based computer systems Digital Press, 1984.

[3] A. C. Myers Mostly-Static Decentralized Information Flow Control PhD Thesis, Massachusetts Institute of Technology, February 1999.

[1] Note that our approach is still subject to timing attacks.