Tagged/Capability Architectures

Shamik Das

28 March 2000

Conventional computer architectures provide little or no hardware support for enforcing data security. Access control, if implemented, is left to the operating system, which typically means that access lists are maintained by the system on a per-object basis. However, a class of machines called *capability architectures* exists in which hardware support for security checking has been implemented. Capability concepts have also been implemented at the operating-system level so that they may be utilized on processors without such hardware support.

A *capability* is defined as a tag, token, or key that gives the possessor of the capability permissions on an entity or object [2]. It is specific to the object, not the possessor, and is usually implemented as an object identifier paired with an identifier for the particular access rights on that object. Thus, a capability system is one in which security is enforced by issuing object or process capabilities to the entities that require them rather than maintaining access control lists for each object or process.

There are several benefits and consequences to such a security implementation. For example, capabilities must be unforgeable, since they are not associated with particular processes. Also, capabilities must be unmodifiable – it should not be possible for a process to change its "read" capabilities into "write" capabilities without express permission. Another property is context-independence, as any use of a capability means the same thing for any user, in whatever way it is used. Capabilities are easily transferred from one entity to another, so a process may grant its capabilities to any other process. However, once such capabilities are granted, they are not easily revoked, since the capabilities that belong to both processes are identical. Finally, by encapsulating accesses to memory, filesystems, devices, etc. with capabilities, a capability architecture can present a uniform means of access to the shared resources of a system.

To visualize capability models versus access-control models, it is helpful to consider the *system object access matrix*. Along the horizontal axis, the system objects (files, devices, etc.) are listed, and along the vertical axis, those entities that may access objects are listed. At each intersection, access rights for that particular entity to that particular object are listed.

Within this framework, the traditional access-control list consists of a column of this matrix – for each object, the system maintains a list of access permissions for the users that have access. Capability

systems, on the other hand, use rows of the matrix, as each user has a capability for some objects in the system.

A more formal framework for analyzing capability systems is the *take-grant* model. In this model, entities may possess **read** and **write** capabilities on objects and **take** and **grant** capabilities on other capabilities. The state of access rights within a system can thus be described via a directed *access graph* where the vertices represent entities and the edges are tagged with the capabilities that the entities possess.

It is possible to show that systems using this framework cannot be secure [3]. However, using a refined framework called *diminshed-take*, it is possible to alleviate the security issues with *take-grant*. In particular, the Extremely Reliable Operating System (EROS) attempts to implement this security model [3].

EROS is, from the ground up, a capability-based operating system. It implements all capabilities in software, thus allowing it to be used on commodity-processor-based systems. An interesting feature of EROS is that in order to preserve capabilities across system failures (an issue with all capability systems), EROS writes its entire state to disk every five minutes. This feature, called *universal persistence*, presents some interesting programming issues (for example, the entire state of a long-term program can be kept in memory rather than writing it to disk!)

For comparison purposes, benchmarks run versus the Linux operating system indicated that EROS is faster on many basic operations such as context switching and syscalls [4].

Capabilities may be utilized to augment parts of an architecture rather than the whole. The Symbolics 3600 Lisp machine [5] is one example. The hardware is designed with a Lisp-based instruction set; thus, most Lisp operations take one cycle. Capabilities are implemented via data tags. This tagging allows for run-time type-checking on the data. Additionally, it allows for the design of the architecture around generalized instructions; that is, instructions such as ADD may be issued with arbitrary numerical data, and the hardware will select the proper operation for the particular type of data. Further, memory pages are also tagged, which allows for hardware-assisted garbage collection. The page tags allow the hardware to find quickly any pages with temporary data. The hardware then uses data tags to identify quickly any pointers into the tagged pages, and the data that is pointed to can be copied out of the page.

Another architecture in which capabilities are used is the M-machine [1]. The M-machine memory subsystem has hardware support for *guarded pointers*. In this implementation, a guarded pointer is a 64-bit word, where 4 bits are used for permissions and 54 bits are used for segment-plus-offset memory reference. The remaining 6 bits are used to indicate the segment size, which is variable. In addition, there is a one-bit tag used to differentiate a guarded pointer from an ordinary 64-bit value. To accommodate these pointers, the M-machine has special instructions for pointer creation and pointer arithmetic. This implementation allows for cycle-by-cycle context switching with no overhead, as there is no thread-dependent virtual-to-physical address mapping involved and the guarded pointer contains all the permissions information necessary to execute instructions on the referenced data.

Finally, there are several other architectures and operating systems that have implemented capabilities in some form. These include the Burroughs B5000, MIT PDP 1, IBM System 38, Intel i432, and the Mach and Amoeba operating systems.

References

[1] Carter, Nicholas P., Keckler, Stephen W., and Dally, William J. "Hardware support for fast capability-based addressing," *6th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VI)*: San Jose, CA, 1994.

[2] Levy, Henry M. *Capability-Based Computer Systems.* Digital Press: Bedford, MA, 1984.

[3] Shapiro, Jonathan Strauss. "EROS: A capability system." Ph.D. Thesis, ch. 4. University of Pennsylvania: 1999.

[4] Shapiro, Jonathan S., Smith, Jonathan M., and Farber, David J. "EROS: a fast capability system," *17th ACM Symposium on Operating Systems Principles (SOSP '99).* Published as *Operating Systems Review,* 34(5):170-185, Dec. 1999.

[5] *Symbolics 3600 Technical Summary.* pp. 3-11, 78-103.