DOI:10.1145/2018396.2018418

# Technical Perspective
# Making Untrusted Code Useful

By Butler Lampson

THE FOLLOWING PAPER combines two important themes in secure computing: *assurance* and information *flow control*. Assurance is evidence that a computer system is secure, that is, obeys its security specification, usually called a security *policy*. A system's Trusted Computing Base (TCB) is all the parts that must work for the system to be secure. For high assurance, the TCB needs to be small and the policy simple. Flow control is one such policy; it specifies how information is allowed to move around in a system.

In the late 1960s the military began to worry about a "multilevel" computer system in which secrets leak from a user process handling classified data. It is impractical to require every application program to come from a trusted source; databases, math libraries, and many other essential tools are too big and complicated to rebuild, or even to audit. So the secret process might contain a Trojan horse from the KGB, which leaks the secrets to Moscow via another, unclassified process.

Flow control solves this problem by requiring that no action of a secret process can affect the state of an unclassified one. Formally, it models the system as a state machine; the state is a set of variables (including process state such as the program counter), and every step sets some variables to functions of others. So each step has the form

$$\text{var}_1, \ldots \text{var}_n := f_1(\text{args}_1), \ldots, f_n(\text{args}_n)$$

The only flows are from $\text{args}_i$ to $\text{var}_i$.

Each variable $v$ has a *label* $L(v)$. Labels form a lattice, a partial order $\sqsubseteq$ in which any two elements have a least upper bound or max. The flow rule is that information can only flow from weaker (unclassified) labels to stronger (secret) ones, so each step must satisfy

$$\max_{v \in \text{args}_i} L(v) \sqsubseteq L(\text{var}_i)$$

Typically a label is a set of atomic elements called *categories*, the ordering is set inclusion, and max is set union.

The flow rule is good because it composes: if each step obeys the rule, the whole computation does so. Hence the label on every data item is at least the max of the labels on everything that affected it; the rule is end to end. It is certainly simple, and assurance is just evidence that each step obeys it.

In the early 1980s research on flow led to the "Orange Book," which defines the security of a computer system by how well it implements flow control and how good its assurance is. The government said that it would require all multilevel systems to be secure in this sense, and several vendors developed them. Sadly, they all turned out to have rather large TCBs and to be slow, clumsy to use, and years behind less secure systems in functionality. The requirement was frequently waived, and it was finally abandoned. After this discouraging experience people lost interest in information flow, especially since a personal computer is not usually multilevel. A networked computer is always multilevel, however, and today all computers are networked (though most adversaries are much weaker than the KGB).

Ten years ago Myers and Liskov[2] revived the field by pointing out that categories need not be predefined. Instead, any process can create a new category, which it owns. An owner can declassify a category, that is, remove it from a label. This is appealingly decentralized and Internet friendly, and makes it possible to *isolate* any program by running it with a new category in its label and then declassifying the result; the flow rule ensures there are no other visible effects.

This paper describes HiStar, a system that enforces decentralized information flow directly. The variables are OS objects, files, threads, etc., rather than integers. Security depends on the simple flow control policy and a small TCB, a new 2,000 line kernel with six object types, each with just a few methods. This is much less than an existing code base retrofitted for strong security. HiStar implements access control using control flow, the reverse of previous practice. Flows through shared resources, such as using up a resource or deleting an object, are tricky problems whose solutions add complexity. Containers enable sharing by holding hard links to objects. Unreachable objects are garbage collected, so there is no deletion.

**Security depends on the simple flow control policy and a small TCB.**

Unix applications run on a library OS[1] that is not part of the TCB, using containers to represent directories, file descriptors, and other protected state. It's surprising that such a minimal kernel suffices, but similar results have recently been reported for Windows.[3] Isolation can clone a whole Unix; a container argument provides the environment: the file system root, address space, resources, and so on. The clone can run most Unix programs, and it can't affect anything outside of itself.

HiStar works well for tasks in which untrusted code reads sensitive data, such as setting up an SSL connection or running a virus scanner. It can isolate a lot of untrusted code using a little trusted code as a wrapper that decides how to declassify the results. The general-purpose computing that failed in the 1980s has not been tried.

This is the latest step in the long and frustrating journey toward secure computing. It is a convincing solution for some serious practical problems. Of course the devil is in the details, which you can read about in the paper. C

**References**

1. Kaashoek, F. et al. Application performance and flexibility on exokernel systems. *ACM Operating Systems Review 31*, 5, (Dec. 1997), 52–65.
2. Myers, A. and Liskov, B. Protecting privacy using the decentralized label model. *Trans. Comput. Syst. 9*, 4 (Oct. 2000), 410–442.
3. Porter, D. et al. Rethinking the library OS from the top down. *ACM SIGPLAN Notices 46*, 3 (Mar. 2011), 291–304.

**Butler Lampson** (Butler.Lampson@microsoft.com) is a Technical Fellow at Microsoft Research and is a Fellow of ACM.