TightLip: Keeping Applications from Spilling the Beans (presented with demo)

Aydan Yumerefendi[*], Benjamin Mickle, and Landon P. Cox
Duke University, Durham, NC
{aydan, bam11, lpcox}@cs.duke.edu

Managing the permissions of any shared space is challenging, even for highly skilled computer users. This task is particularly daunting for untrained PC users, for whom access control errors are routine and can lead to damaging privacy leaks. A 2003 usability study of the Kazaa peer-to-peer file-sharing network found that many users share their entire hard drive with the rest of the Internet, including email inboxes and credit card information. Over 12 hours, the study found 156 distinct users who were sharing their email inboxes. Not only were these files available for download, but other users could be observed downloading them.

This and similar compromises such as users inadvertently copying sensitive data into their public web space present a different threat model than is normally assumed by the privacy and security literature. In these cases, data leaked due to access control *misconfigurations* rather than malice or buggy software. For example, companies in the UK were reported to have banned their employees from using Google Desktop because it allows users to search across machines and can store sensitive files on remote Google servers.

Neither secure communication channels nor host-based intrusion detection would have prevented these exposures. Furthermore, the impact of these leaks extends beyond the negligent users themselves since the leaked data can and often does include previous communication and transaction records involving principals. No matter how careful any individual is, her privacy will only be as secure as her least competent confidant. Prior approaches to similar problems either rely on new programming language features, making them incompatible with legacy code or track "tainted" within a running process, leading to prohibitively poor performance.

Thus, we are exploring a new approach to preventing leaks due to access control misconfigurations through a privacy management system called *TightLip*. TightLip helps users define *what* data is important and *who* is trusted, rather than forcing them to understand the complex dynamics of *how* data flows among their software packages. We have divided the problem into two complementary goals—first, to create file and host meta-data, such as labels and group membership, and second, to use this meta-data to alert users about potential leaks. TightLip transparently labels sensitive files by searching for data such as email and tax returns. Users only need to be prompted when a potential breach occurs, which should be rare.

Our current focus is on TightLip's second goal of using labels to stop privacy leaks. Our approach relies on a new operating system object called a *doppelganger*[1] process. Doppelgangers are sandboxed copy processes that inherit most, but not all, of the state of an *original* process. In TightLip, doppelgangers are spawned when a process tries to read a sensitive file. The TightLip kernel returns sensitive data to the original and scrubbed data to the doppelganger.

The doppelganger and original then run in parallel while the operating system monitors the sequence and arguments of their system calls. If output buffers for the two processes are the same, the output is independent of the sensitive input with very high probability and is allowed to pass. However, if output buffers are different, they likely depend on their respective inputs. Routines for handling divergence are policy-specific and based on the trust level of the destination and the state of the doppelganger.

We have implemented doppelgangers in a prototype TightLip kernel and evaluated our approach through micro-benchmarks and SpecWeb99 web server benchmarks. The performance overhead in our micro-benchmarks is several orders of magnitude less than comparable taint-checking systems. SpecWeb99 results show that Apache running on TightLip exhibits a negligible 5% slowdown in request rate, response time, and transfer rate compared to an unmodified server environment.

[*] Student

[1] From the Oxford English Dictionary: "the apparition of a living person; a double, a wraith."