Keylogger Detection and Containment

Ortolani, S.

2013

document version
Publisher's PDF, also known as Version of record

Summary

Our recent history inspired many novelists with espionage stories where two or more spies played the delicate and dangerous game of secretly stealing private information. Essential skill was the ability to wiretap phones and telexes without the other party realizing. Typically, this required implantation of tiny devices directly into the communication hardware. With the rise of computers, those tiny devices became what are known as hardware keyloggers: tiny dongles placed in-between the keyboard and the motherboard designed to log and save all the user keystrokes. Also in this case, however, physical access was required. This allowed the audience (us) to feel relatively secure, as we believe that it is in our power to prevent a trespasser from breaking in. Some of us by rigging their apartment, some others by keeping a pistol under their pillow.

Software keyloggers are the "mass-market" version of these hardware devices. Installed on users' computers, they monitor the user activity by surreptitiously logging all the keystrokes and, in some cases, by delivering them to a third party [36]. If designed to be run in user-space, they are also easy to implement: all modern operating systems offer documented sets of unprivileged APIs that can be leveraged by user-space programs to intercept all the user keystrokes. Contrary to keyloggers running as kernel modules, no permission is required for deployment and execution. A user can erroneously regard the keylogger as a harmless piece of software and being deceived in executing it. Kernel keyloggers, besides depending upon privileged access for both execution and deployment, require the programmer to rely on kernel-level facilities to intercept all the messages dispatched by the keyboard driver; this undoubtedly requires a considerable effort and knowledge for an effective and bug-free implementation.

In light of these observations, it is no surprise that 95% of the existing keyloggers run in user space [42]. Despite the rapid growth of keylogger-based frauds (i.e., identity theft, password leakage, etc.), not many effective and efficient solutions have been proposed to address this problem. Traditional defense mechanisms use fingerprinting strategies similar to those used to detect viruses and worms. Unfortunately, this strategy is hardly effective against the vast number of new keylogger variants surfacing every day in the wild.

In this thesis, we address the problem of detecting user-space keyloggers by exploiting their peculiar behavior. In particular, we leverage the intuition that the keylogger's activity strictly depends on the user's input. To generalize our approach, we discard any assumption on the internals of the keyloggers, and we develop black-box approaches vetting processes strictly in terms of system activity. To meet the ease of deployment and execution of user-space keyloggers, we also explore both privileged and unprivileged solutions. Unprivileged solutions, although bound to rely on less powerful system characterizations, allow for deployment scenarios which are normally rarely considered; those are the many cases in which the user does not have a super user account at his disposal: Internet cafés, business laptops, or borrowed terminals are all sound examples. It is our belief that a first line of defense shall be granted regardless of the privileges available.

This dissertation starts by proposing KeySling and NoisyKey, two unprivileged solutions to detect and tolerate user-space keyloggers. In KeySling we correlate the I/O of each process with some simulated user activity, and we assert detection in case the two are highly correlated. The rationale is that the more intense the stream of keystrokes, the more I/O operations are needed by the keylogger to log the keystrokes to a file. NoisyKey upgrades the user's first line of defense by allowing him to live together with a keylogger without putting his privacy at stake. By confining the user input in a noisy event channel, the keylogger is fed with random keystrokes that can not be told apart from the real user input. We then introduce Klimax, a privileged infrastructure able to monitor the memory activity of a running process. This new behavior characterization enables detection of keyloggers delaying the actual leakage (I/O activity) as much as possible. This is the case of privacy-breaching malware, or spyware, which also strives to conceal its presence by restraining from unnecessary system activities. We conclude the dissertation by considering the case of keyloggers in the form of application add-ons rather than separate and isolated processes. In more details, by relying on the new system characterizations offered by Klimax, we propose a new cross-browser detection model able to detect add-ons turning the host browser into a keylogger.