POLA Today Keeps the Virus at Bay

Alan H. Karp
HP Laboratories Palo Alto

HPL-2003-191
September 8th , 2003*

E-mail: alan.karp@hp.com

security, access control, computer virus

The software industry is making a major effort to eliminate the flaws exploited by writers of malware. It is the premise of this essay that this strategy cannot succeed. Something else is needed, and that something is enforcing the Principle of Least Authority (POLA) at a finer granularity than we do today.

POLA Today Keeps the Virus at Bay

Alan H. Karp
Hewlett-Packard Laboratories
alan.karp@hp.com

September 3, 2003

I applaud the software industry's efforts to produce code that is less susceptible to attack, but I'm afraid the effort is futile because this strategy succeeds only when perfection is achieved. A suitable flaw in any piece of software, written by Microsoft or anyone else, can be used to grant an attacker all the privileges of the user. Reducing the number of such flaws will make finding points of attack harder, but once one is found, it is likely to be exploited.

The mistake is in asking "How can we prevent attacks?" when we should be asking "How can we limit the damage that can be done when an attack succeeds?". The former assumes infallibility; the latter recognizes that building systems is a human process. The answer to the first question is usually "By fixing our code." The answer to the second question is invariably "By enforcing the Principle of Least Authority."

Nearly all of today's systems, including Microsoft's and all Unix flavors, enforce POLA only at the level of the user. This approach is a good way to prevent me from doing something I am not allowed to do, but it does nothing to prevent a process acting on my behalf from doing something I am allowed to do but don't want to do. That's exactly what the virus does; it corrupts any file I can modify. There is no reason that the process displaying my email should be able to read my tax return; there is no reason that process should be able to open a network connection to send that data to an attacker. Both are possible if POLA applies only to the user.

A better approach is to enforce POLA at the level of the process or even at the level of objects within a process. Doing so limits the damage that a successful attack can do to the set of actions the process or object needs to do its job. It may destroy the message my email program is displaying, but it can't replace a command with a Trojan horse because it doesn't have access to that file. It may read my email, but it can't send it to the attacker because it can't open a network connection.

Such fine-grained control of permissions sounds like a user interface nightmare. Must we put up with thousands of nagging "May I?" prompts? Ka-Ping Yee of Berkeley has shown that the answer is "No"; user actions implicitly specify the desired permissions[5]. When I double-click on a Word document, I am telling the system that I want the process running Word to be able to read and write only this specific file. I don't have to worry that a macro virus will overwrite my

NORMAL.DOT and infect other documents. I don't have to worry that the virus will open a network connection and send my document to my competitor. That's because I never said that macros in that file should have access to NORMAL.DOT or the network.

The primary flaw in today's systems is that access control is identity based. This means that any process acting on my behalf necessarily runs as me and, therefore, has all my privileges. In such a system, I prove my identity to the system administrator who sets up an account for me. This account maps one-to-one with my identity and embodies my access rights in the form of an entry in the access control list (ACL) of every resource I'm allowed to use. Each process I run carries my account identity with it, and every request to the system presents this identity. Access is allowed or denied based on the entry associated with this identity in the ACL for the specified resource.

An alternative approach is to separate authentication (who I am), authorization (what I am allowed to do), and access control (whether or not to honor a request). I prove my identity to an authenticator and receive the set of authorizations I should have. These can be validated by the access control mechanism when I make a request. Since I have an explicit set of authorizations, I can choose which to give each process I start. Now, even if a virus takes complete control of the process I'm using to read my email, it cannot erase all my spreadsheets.

A system built along these lines is far more flexible and manageable than what we have today. There need not be a single authenticator. HP Corporate can handle my identity as an HP employee and give me the set of authorizations it administers; HP Labs can do the same for its authorizations. If I move from HP Labs to a product division, only those two organizations need to update my authorizations. In addition, the authenticator doesn't need to interpret the authorizations; it just hands them out. Similarly, the access control mechanism need not concern itself with who I am or how I got my authorization; it just needs to allow or deny access. Doing things this way allows the authentication and access control systems to evolve independently, and new kinds of authorization can be introduced without modifying existing systems.

Such a large change in the way we think about our systems seems to require that we throw out everything and start from scratch. Some people are taking this approach. Jonathan Shapiro's EROS project at Johns Hopkins University[2], for example, is building an entirely new operating system that enforces POLA as an inherent part of its architecture. Less disruptive are those approaches that merely require that we rewrite our applications in a POLA language, such as the E language being developed by Mark Miller of HP[1]. Marc Stiegler of Combex has written a fully functional desktop in E that has all the desired properties[3]. These two have also produced a prototype web browser under DARPA contract that can safely view a web page that would infect a machine if any other browser were used[4].

Less disruptive measures are needed if we're to maintain compatibility with existing software, but we must be willing to forego some of the finer points of POLA. Attacks almost always involve some interaction with the operating system, so we can do most of what's needed by filtering all kernel calls. Most flavors of Unix provide an appropriate interface, and a relatively small change to Windows would add this functionality. In fact, the Windows file system redirector already provides much of what is needed. Alternatively, we could run each process in a virtual machine and filter at the interface to the underlying operating system kernel. Even less intrusive is an approach based on dynamically creating accounts with exactly the permissions we want the process to have. Unfortunately, without changes to the process launcher, we don't have a way to tell what rights the new process should have. It's worth the effort, though, because if we solve this problem the worm can't propagate, and the virus can't install the backdoor.

Reducing exploitable flaws in our software has benefits, but it will not eliminate successful attacks. It is far better to reduce the exploitabilty of flaws. Only by remembering that we give privileges to people but enforce access control on processes will we be able to design systems less vulnerable than those we use today.

References

1. Mark Miller, "E Language", http://erights.org.
2. Jonathan Shapiro, "EROS: The Extremely Reliable Operating System", http://www.eros-os.org
3. Marc Stiegler, "CapDesk", http://www.combex.com/tech/edesk.html
4. David Wagner and Dean Tribble, "A Security Analysis of the Combex DarpaBrowser Architecure", http://www.combex.com/papers/darpa-review/security-review.html
5. Ka-Ping Yee, "User Interaction Design for Secure Systems". In *Proceedings of the International Conference on Information and Communications Security*, Singapore, 2002. http://zesty.ca/sid/