By Marc Stiegler, Alan H. Karp, Ka-Ping Yee,
Tyler Close, and Mark S. Miller

# POLARIS: VIRUS-SAFE COMPUTING for Windows XP

*It limits the damage a virus can do by using the operating system's own security mechanisms to enforce the Principle of Least Authority on individual applications.*

Viruses—those nasty pieces of software that sometimes run when a user launches an email attachment, edits a file with macros, or visits a Web page that uses scripts—are clearly a problem. Unlike some malicious programs that depend on programming mistakes, viruses don't exploit inadvertent security holes; they use the system the way it was designed to be used. But how can that be?

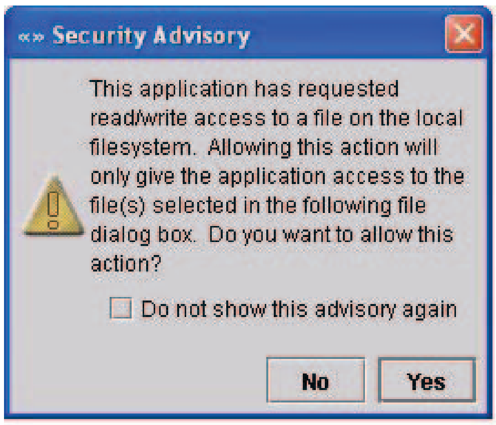
All widely used operating systems, not just Windows, base their security on the identity of the logged-in user. This means every program we run can do anything we can do, whether we want it done or not. This is the flaw in the basic design of our systems that viruses exploit, doing things we are allowed to do that we don't want done.

The problem is the excess authority that every program gets from the operating system. There's no reason Solitaire, for example, needs to be able to search our hard drives for secrets and send them to our competitors. There's no reason Excel needs to be able to put a Trojan horse in our startup folders. Yet in the current generation of operating systems that's simply the way things work. This view is so widespread that the first of Microsoft's 10 Immutable Laws of Security [5] says, "If a bad guy can persuade you to run his program on your computer, it's not your computer anymore."

"Sandboxing," or producing a set of rules for each program, as in Java 2 Security [4], is a common means of dealing with this problem. But the rules in sandboxing are static. Adding authorities to a running program (such as to open a file) is often difficult in these systems.

The alternative is to control access to individual resources, many such systems nagging the user with "May I?" dialogue boxes (such as the one in some Java Web Start applications [3], as in Figure 1a). Although we can hide this advisory for the duration of the run, the fact that it is needed at all indicates there is no distinction between requests made by the user and those made by the software. Hence, hiding

(a)

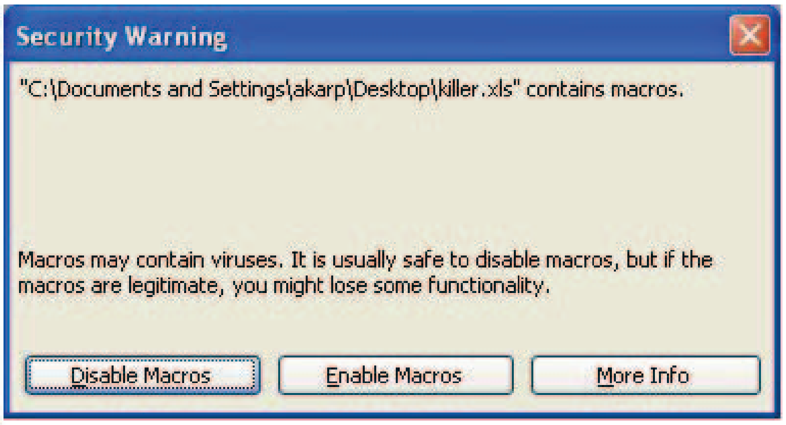


**Security Advisory**

This application has requested read/write access to a file on the local filesystem. Allowing this action will only give the application access to the file(s) selected in the following file dialog box. Do you want to allow this action?

Do not show this advisory again

(b)



**Security Warning**

"C:\Documents and Settings\akarp\Desktop\killer.xls" contains macros.

Macros may contain viruses. It is usually safe to disable macros, but if the macros are legitimate, you might lose some functionality.

**Figure 1. Standard dialogue boxes ask users to make security decisions without giving them a basis for understanding the implications of their decisions: (a) "May I?" request from Java Web Start; (b) dilemma posed by Excel.**

the advisory may allow the software to take actions counter to our wishes.

This assumption—that a system with fine-grain control is unusable—translates into the belief that the security policy must group authorities into relatively large chunks [2]. Such systems appear to be easy to use; issue a command and it runs. Experience has shown, however, that they become difficult to use when the security policy is configured to prevent viruses from abusing excess authorities. Virus scanners must be updated and run on a regular basis. Firewalls must be configured. Worse, blocking the attacks reduces functionality. Security advisories tell us "Don't launch email attachments"; "Disable macros in documents"; and "Turn off scripting on Web pages." The result is feature starvation.

The final blow is that systems that grant large chunks of authority present us with hopeless dilemmas. We're all familiar with dialogue boxes (see Figure 1b) that ask users to choose between not getting their work done and losing control of their machines. Worse, these dialogue boxes don't give sufficient information to make an intelligent decision. Why is the macro needed? What damage might it do? There is no way to tell.

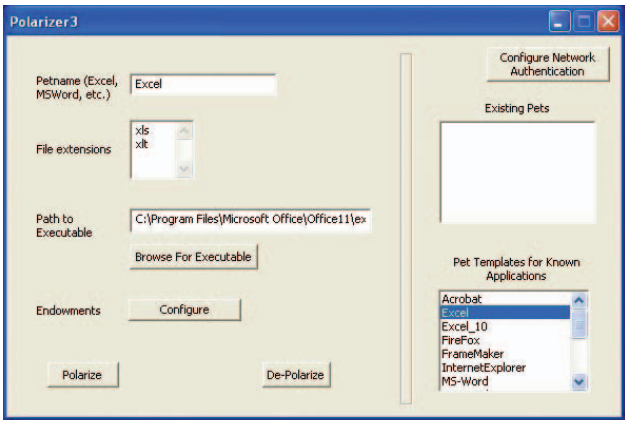
Vista, Microsoft's follow-on to Windows XP (scheduled for general release in early 2007) takes an intermediate approach called User Account Protection [6]. A set of programming guidelines helps application developers write programs that can be run by unprivileged users. In addition, certain powerful functions can be invoked only by using mechanisms safe from malicious code running with user permissions. Combined with virtualizing parts of the registry and file system, User Account Protection lets users get their work done without needing administrator privileges. While this approach adds important protections for system resources, it does little to protect user data. Identity thieves aren't interested in system files, and ransomware [11] encrypts user data while demanding payment for the decryption key.

Polaris is a package for Windows XP being developing by a skunk works group called the Virus Safe Computing Initiative at Hewlett-Packard Laboratories in Palo Alto, CA, that allows users to configure most applications so they launch with only the rights they need to do the job the user wants done. This step—enforcing the Principle of Least Authority—provides so much protection from viruses that there is no need to pop up security dialogue boxes or ask users to accept digital certificates. Moreover, there is less risk in launching email attachments, using macros in documents, or allowing scripting while browsing the Web. Polaris demonstrates we can build systems that are more secure, more functional, and easier to use than those already in common use.

We found from our earlier work on a secure distributed desktop environment called CapDesk [9] that combining designation with authorization lets users manage fine-grain authorities while making almost all security decisions disappear into the background. For example, double-clicking on the icon for a spreadsheet to launch Excel is an act of designation. In Polaris, the act of designation is also treated as an act of authorization, adding the authority to edit the designated spreadsheet to an instance of Excel initially configured without authority to edit files. In Polaris jargon, an application configured this way is said to be polarized.

The process running polarized Excel needs access to more than just the file being edited. It also needs access to, for example, its own executable. Most programs also have a large number of auxiliary files (such

(a)



Polarizer3

Petname (Excel, MSWord, etc.): Excel

File extensions: xls, xlt

Path to Executable: C:\Program Files\Microsoft Office\Office11\ex

Endowments: Configure

Existing Pets

Pet Templates for Known Applications: Acrobat, Excel, Excel_10, FireFox, FrameMaker, InternetExplorer, MS-Word

(b)

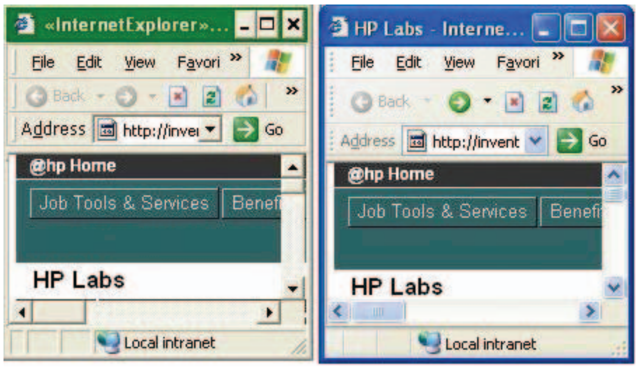


Figure 2. It's easy to configure an application in Polaris and know which windows are running programs protected by Polaris: (a) polarizing an application; (b) visual indication of protection state.

as shared libraries or fonts) and often create temporary files in some folder. Since access to these files is needed each time the application runs, regardless of which file it is editing, Polaris gives each application an installation endowment consisting of the ability to use the files—another concept carried over from CapDesk. This coupling of an installation endowment and combining designation with authorization makes the security decisions part of the user's normal activity.

Unlike static sandboxing or fine-grain access control in Java Web Start, Polaris is able to add to the authorities available to a process without requiring any extra effort by the user. The user simply clicks the File Open icon. Polaris detects the dialogue box and substitutes one from the PowerBox, a process with access to all of the user's files. After the user selects a file, Polaris makes that file accessible to the running program. No further security decisions are required. Polaris infers what authorities the user wants to grant by detecting the user's acts of designation in the PowerBox. The PowerBox is the third concept Polaris adopted from CapDesk, along with installation endowment and using acts of designation as acts of authorization.

Users of Polaris need not worry about getting their work done when encountering problems. Polarizing applications doesn't prevent them from running the standard version. They can either launch the application directly or right-click on an icon for the file and select Open instead of OpenSafe. Launched this way, the application runs in the user's account with all of the user's permissions. However, if a virus runs in an unpolarized application, it can abuse any of the user's authorities.

## Polarizing Applications

Polaris developers call an instance of a polarized application a pet. Figure 2a shows the dialogue box set up to configure a pet to run Excel. The user selects a petname for the pet [8] to appear in the title bar of each window running it, giving the user a convenient verification that the program being run is safe from viruses. If the user specifies file extensions, the pet will launch when the user double-clicks on the icon for a file with one of these extensions.

It may make sense to have more than one pet for a given application. For example, a user might have one browser pet for a particular intranet and another for the Internet. Since each pet runs in a separate account, the user can have the intranet pet remember passwords without worrying that visiting some external Web site with the Internet pet will reveal them.

Users are better off if they are aware of the security environment around them, but the cues should not be obtrusive [12]. As shown in Figure 2b, Polaris modifies the title bar of the window running the application. If a pet is running in the window, the petname appears, <<InternetExplorer>>, as shown on the left. If the application was not launched under Polaris, the petname is blank, and there is no <<>>, as shown on the right. The difference in style, which is due to a

ENFORCING the Principle of Least Authority gives so much protection from viruses there is no need to pop up security dialogue boxes or ask users to accept digital certificates.

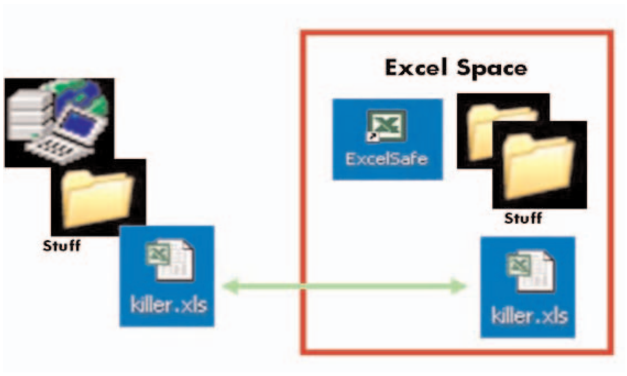


Figure 3. Starting an Excel pet.

Windows artifact, is another visual cue.

These same visual cues appear in all subwindows and are important when more than one application is open. For example, a macro virus running in Excel could open a file dialogue box that overlaps a window running Word. Without a visual cue, the user might select a file without knowing which application would be granted permission to edit it.

## How It Works

Polaris doesn't change the operating system or the applications; it changes only the way applications are launched. Instead of starting an application in the logged-in user's account, a polarized application is launched in a restricted user account with few permissions. This procedure uses the operating system's own security mechanisms to limit what the software, including any viruses it contains, can do. A program launched this way lacks permission to read or modify the user's data or for a keystroke logger to see what is being typed.

When the user double-clicks on the icon for a file, Polaris launches the application in steps (see Figure 3). First, it copies the file the user designated to a folder accessible to the restricted account. Next, it sets up a synchronizer to keep the copy and the original file consistent. Finally, it launches the application using a feature of the Windows API that lets the user start a process in the restricted account.

If a virus runs in the restricted account, the only thing it can damage with the privileges of that account is the file being edited. It lacks the ability to modify the user's startup folder, nor can it read other files looking for secrets. If the browser is polarized, malicious scripts can't use the browser's privileges to plant spyware and adware on the user's system outside the area available to the restricted account. Furthermore, malware is able to change only the restricted account's part of the Windows Registry, since access to registry entries is controlled through the same mechanism used for files. Any such changes are easily undone by polarizing the application again. A number of users at Hewlett-Packard in the pilot study we have been running since 2005 who have visited Web pages containing viruses can attest to the protection provided by Polaris. Any damage these users have

## VIRUSES AND WORMS

The terms virus and worm are used interchangeably to describe somewhat different types of malware. We use a loosely followed distinction that worms propagate on their own, while viruses are spread only by people. The prototypical worm was released in 1988 by Robert T. Morris, a student at Cornell University at the time [1]. An early virus called Love Letter was released in 2000 (www.cert.org/advisories/CA-2000-04.html), inducing people to open its attachment, a Visual Basic Script. Once launched, the script makes several modifications to the machine and sends copies of itself to all entries in all Outlook address books in the system. The definitions we use here are officially accepted by the U.S. Government. The Jargon File, an online technology dictionary (www.eps.mcgill.ca/jargon/jargon.html), also supports our definitions:

***worm n.*** [from "tapeworm" in John Brunner's novel *The Shockwave Rider* via Xerox PARC]. A program that propagates itself over a network, reproducing itself as it goes. Compare virus. The term has negative connotations, as it is assumed that only crackers write worms. Perhaps the best-known example was Robert T. Morris's Great Worm of 1988, a "benign" one that got out of control and hogged hundreds of Suns and VAX machines across the U.S.

***virus n.*** [from the obvious analogy with biological viruses, via science fiction]. A cracker program that searches out other programs and "infects" them by embedding a copy of itself in them, making them Trojan horses. When these programs are executed, the embedded virus is also executed, thus propagating the "infection." This is normally invisible to the user. Unlike a worm, a virus cannot infect other computers without assistance.

The question of what is a worm and what is a virus is hardly settled, though. For example, the title of the CERT advisory referenced in the context of defining "virus" is "Love Letter Worm."

### Reference

1. Spafford, E. The Internet worm program: An analysis. *ACM SigComm Computer Communications Review 19*, 1 (Jan. 1989), 17–57.

reported is limited to the pet account.

There are two reasons not to simply change the Windows Access Control List (ACL) and edit in place. First, many applications (such as Microsoft Word) create temporary files in the same directory as the documents they open. These applications would work properly only if they were granted write authority to the parent directory. Doing so would greatly increase the damage a virus could do. In addition, the user probably lacks permission to change the ACL of files residing on network shares.

Another reason for not editing in place has to do with the difference between permission and authority (see the sidebar "Viruses and Worms"). As implemented, the restricted user account has the authority to make changes in the original file because the synchronizer uses its permissions to copy updates to the original. The restricted account has no permission to change the original file. The advantage of not allowing the restricted account to change the original file is that the authority to make changes is revoked when the synchronizer is stopped, should, say, the machine crash. Using this mechanism means Polaris leaves no dangling permissions that must be cleaned up later.

SINCE JUNE 2005, 15 PEOPLE IN HEWLETT-PACKARD Labs, 10 at other Hewlett-Packard locations, and some not associated with Hewlett-Packard have been using the alpha release of Polaris. For the most part they are unaware of its presence. In fact, one executive used Polaris with no problems for several days before we told him what we'd done to his machine. Several of these early users have been saved from harm when viruses ran in polarized applications. Users who consistently surf with a polarized browser report finding little or no spyware or adware on their machines.

The beta version of Polaris was released in June 2006 and is still available. We added four new users at Hewlett-Packard Labs with the rollout of the beta release and plan to add 10 to 20 more once we resolve any issues raised by this group. We also plan to expand our outside testing beyond the current pilot studies at the School of Public Policy at George Mason University, Fairfax, VA, and at the U.S. Navy (responsible for the Department of Defense Horizontal Fusion Project), Monterey, CA. We will use reports from these early testers to decide when and how quickly we add additional users. Hewlett-Packard had no plans to turn Polaris into a product at press time.

The beta release included numerous improvements over earlier versions. Most important, it closed the GUI hole. The issue is that Windows allows any application to send GUI events to any window on the desktop. These messages can be used by the receiving application to run commands with the privileges of the receiving process instead of the privileges of the process sending the message (see the sidebar "Privilege, Permission, and Authority"). These Windows messages can even be used to exploit flaws in system services to gain full control over the machine [7]. We have found a feature of the Windows API that lets Polaris block such attacks [1]. Unfortunately, using it exposes some bugs in Windows that need workarounds. For example, users of polarized applications can cut and paste bitmaps but not text when using this feature. We have implemented workarounds for this and for other problems we've encountered.

Polaris uses acts of designation to determine authorization, avoiding the usual trade-offs between usability and security. By presenting the user with fewer dialogue boxes, Polaris makes Windows somewhat easier to use. Beyond this improvement, we would like to make using a machine protected by Polaris identical to using a standard Windows desktop. We're close but can do better. For example, the beta version

## PRIVILEGE, PERMISSION, AND AUTHORITY

The security community often cites the Principle of Least Privilege [2], though exactly what constitutes a privilege isn't clear, even to these experts. One attempt at clarification [1] introduced the distinction between permission and authority, defining permission as the set of rules written down in, say, an access control list, and authority as the set of consequences a process can cause to happen. Authority combines the set of permissions with the behavior of parties having these permissions.

Consider a Web server. The process running it has permission to read the files of the Web site; there is a specific entry in the ACL for each file. Someone visiting the Web site has no entry in the ACL but still can read the contents of the file because the server presents the information. Hence, the Web surfer has authority to read the files, even though no explicit permission grants such access.

Security analysis that considers only permission is incomplete. Security analysis that includes authority is necessarily constrained by the ability to understand the behavior of programs. Fortunately, it is often possible to get a usable bound on the authority available to any process [1].

### REFERENCES

1. Miller, M. and Shapiro, J. Paradigm regained: Abstraction mechanisms for access control. In *Proceedings of the Eighth Asian Computing Science Conference (ASIAN 2003)* (Mumbai, India, Dec. 10–13). Tata Institute of Fundamental Research, Mumbai India, 2003, 224–242; erights.org/talks/thesis/index.html.
2. Saltzer, H. and Schroeder, M. The protection of information in computer systems. *Proceedings of the IEEE 63*, 9 (Sept. 1975), 1278–1308.

POLARIS doesn't change the operating system or the applications; it changes only the way applications are launched.

does not handle linked files (such as spreadsheets containing references to other spreadsheets) very well. When there is a difference, we need to identify only an act of designation they can use to change the authorization. We have solutions to the linked files and other problems that did not make it into the beta version.

We've also been unable to solve some problems. For example, Direct 3D is incompatible with the security machinery inside Polaris. Hence, many games don't work if polarized. PGP won't run polarized. And some operations of the Cygwin command shell modify access control lists in a manner that is incompatible with Polaris.

We also haven't been able to block some attacks. For example, the beta version does nothing about limiting network access, meaning that a virus could send the contents of the document being edited to a competitor. We believe we have identified a possible solution to this problem by using a custom firewall.

## Conclusion

By bundling designation with authorization in order to apply the Principle of Least Authority to individual programs, Polaris provides protection against entire families of viruses with minimal impact on usability and functionality. We've found that such viruses running in polarized applications are able to do much less harm than when they run in their non-polarized counterparts. The parts of the system these viruses attack (such as the Windows directory, the user's startup folder, and most of the Windows registry) are safe from them. Polaris lets users, as well as application developers, take advantage of the effort that went into developing powerful macro languages, use email to send programs to one another, and enable the true power of Web scripting, all without opening up our systems to attack. C

## References

1. Close, T., Stiegler, M., and Karp, A. Shatter-proofing Windows. In Black Hat USA 2005 (Las Vegas, July 23, 2005); www.blackhat.com/presentations/bh-usa-05/BH_US_05-Close/tyler close_whitepaper_US05.pdf.
2. Kamp, P.-H. and Watson, R. Building systems to be shared securely. *ACM Queue 2,* 5 (July/Aug. 2004), 42–51.
3. Kim, S. *Java Web Start: Developing and Distributing Java Applications for the Client Side.* White Paper, IBM Corp. Armonk, NY, Sept. 1, 2001; www-106.ibm.com/developerworks/java/library/j-webstart/.
4. McGraw, G. and Felten, E. *Securing Java: Getting Down to Business with Mobile Code, 2nd Edition.* John Wiley & Sons, Inc., New York, 1999.
5. Microsoft Corp. *10 Immutable Laws of Security.* Redmond, WA; www.microsoft.com/technet/archive/community/columns/security/essays/10imlaws.mspx.
6. Microsoft Corp. *Developer Best Practices and Guidelines for Applications in a Least Privileged Environment: Understanding User Account Protection in Microsoft Windows Vista Beta 1, Windows Security Access Control.* Redmond, WA, Sept. 2005; msdn.microsoft.com/windowsvista/default.aspx?pull=/library/en-us/dnlong/html/AccProtVista.asp.
7. Paget, C. Click next to continue. In Black Hat 2003 (Las Vegas, July 2003); blackhat.com/html/bh-media-archives/bh-archives-2003.html#USA-2003.
8. Stiegler, M. An introduction to petname systems. In *Advances in Financial Cryptography, Volume 2,* I. Grigg, Ed., 2005; www.financial-cryptography.com/mt/archives/000499.html.
9. Stiegler, M. and Miller, M. *A Capability-based Client: The DarpaBrowser.* Technical Report, Focused Research Topic 5. Combex, Inc., Meadowbrook, PA, June 2002; www.combex.com/papers/darpa-report/index.html.
10. U.S. General Accounting Office. *Technology Assessment: Cybersecurity for Critical Infrastructure Protection.* GAO-04-321, Washington, D.C., May 2004, 27.
11. Websense Security Laboratories. *Cyber Extortion Attack.* May 2005; www.websensesecuritylabs.com/alerts/alert.php?AlertID=194.
12. Yee, K.-P. User interaction design for secure systems. In *Proceedings of the Fourth International Conference on Information and Communications Security* (Singapore, Dec.). Springer-Verlag, 2002, 278–290.

**Marc Stiegler** (marc.d.stiegler@hp.com) is a visiting scholar in the Virus Safe Computing Initiative of the Advanced Architecture Program at Hewlett-Packard Laboratories, Palo Alto, CA.
**Alan H. Karp** (alan.karp@hp.com) is a principal scientist in the Virus Safe Computing Initiative of the Advanced Architecture Program at Hewlett-Packard Laboratories, Palo Alto, CA.
**Ka-Ping Yee** (ping@zesty.ca) is a Ph.D. student in the Computer Science Division at the University of California, Berkeley.
**Tyler Close** (tyler.close@hp.com) is a research scientist in the Mobile and Media Systems Laboratory at Hewlett-Packard Laboratories, Palo Alto, CA.
**Mark S. Miller** (erights@hp.com) is a visiting scholar in the Virus Safe Computing Initiative of the Advanced Architecture Program at Hewlett-Packard Laboratories, Palo Alto, CA.