Polaris: Virus Safe Computing for Windows XP

Marc Stiegler, Alan H. Karp, Ka-Ping Yee[1], Mark Miller[2]
Mobile and Media Systems Laboratory
HP Laboratories Palo Alto
HPL-2004-221
December 1, 2004*

security, virus, Microsoft Windows

Polaris is a package for Windows XP that demonstrates that we can do better at dealing with viruses than has been done so far. Polaris allows users to configure most applications so that they launch with only the rights they need to do the job the user wants done. This simple step, enforcing the Principle of Least Authority (POLA), gives so much protection from viruses that there is no need to pop up security dialog boxes or ask users to accept digital certificates. Further, there is little danger in launching email attachments, using macros in documents, or allowing scripting while browsing the web. Polaris demonstrates that we can build systems that are more secure, more functional, and easier to use.

Polaris: Virus Safe Computing for Windows XP

Marc Stiegler, Alan H. Karp, Ka-Ping Yee[1], Mark Miller[2]
Hewlett-Packard Laboratories
Palo Alto, California

Viruses, those nasty pieces of software that can run when you launch an email attachment, edit a file with macros, or visit web a page that uses scripts, are an ongoing problem.   Unlike some malware that depends on security holes in a piece of code, these kinds of viruses aren't exploiting flaws; they're using the system the way it was designed to be used. How can that be?

All widely used operating systems, not just Windows, base their security on the identity of the logged in user. That means every program you run can do anything you can do, whether you want it done or not. It is this flaw in the basic design of our systems that viruses exploit. They do things you're allowed to do that you don't want done. The problem is the excess authority that every program gets. There's no reason Solitaire needs the ability to search your disk for secrets and send them to your competition. There's no reason Excel needs the ability to put a Trojan horse in your startup folder. Yet, on today's systems that's simply the way things work. This view is so ingrained in the thinking of computer users that the first of Microsoft's 10 Immutable Laws of Security [12] states, "If a bad guy can persuade you to run his program on your computer, it's not your computer anymore."

A common means of dealing with this problem is *sandboxing*, which involves setting up a set of rules for each program, as in Java 2 Security [10]. A failing of sandboxing is that the rules are static. Adding authorities to a running program, such as to open a file, is often difficult in such systems. The alternative is to control access to individual resources, but to date such systems have been too hard to use. Many such applications constantly nag the user with "May I?" dialog boxes, such as one from Java Web Start [11] shown in Figure 1.

Figure 1. "May I?" request from Java Web Start.

Although the user can hide this advisory for the duration of this run, the fact that it is needed at all indicates that there is no distinction between requests made by the user and

---

[1] Also, University of California, Berkeley.
[2] Also, Johns Hopkins University.

those made by the software. Hence, hiding the advisory may allow the software to take actions counter to the user's wishes.

This view that a system with fine-grained control is unusable translates into the belief that we must group authorities into relatively large chunks [6]. Such systems appear on the surface to be easy to use; issue a command, and it runs. Experience has shown, however, that such systems become quite hard to use when we try to prevent viruses from abusing these excess authorities. There are virus scanners to be updated and run on a regular basis. There are firewalls to be configured. Even worse, blocking the attacks reduces functionality. Security advisories tell us "Don't launch email attachments." "Disable macros in documents". "Turn off scripting on web pages." The final blow is that systems that grant large chunks of authority are hard to use. We're all familiar with dialog boxes such as the one shown in Figure 2.

This dialog box asks you to choose between not getting your work done and losing control of your machine. Even worse, it doesn't give you sufficient information to make an intelligent decision. Why is the macro needed? What damage might it do? You have no way of knowing.

Figure 2. Dilemma posed by Excel.

Polaris is a package for Windows XP that demonstrates that we can do better. Polaris allows users to configure most applications so that they launch with only the rights they need to do the job the user wants done. This simple step, enforcing the Principle of Least Authority (POLA), gives so much protection from viruses that there is no need to pop up security dialog boxes or ask users to accept digital certificates. Further, there is little danger in launching email attachments, using macros in documents, or allowing scripting while browsing the web. Polaris demonstrates that we can build systems that are more secure, more functional, and easier to use.

Using an Application under Polaris

We have found from our earlier work on CapDesk [7] that combining *designation* with *authorization* allows us to manage fine-grained authorities while making almost all of the security decisions disappear into the background. Double clicking on the icon for a spreadsheet to launch Excel is an act of designation. In Polaris, we treat the act of designation also as one of authorization. Polaris allows users to configure most applications so they launch in *polarized* mode, that is with only the rights they need for the job the user wants done.

Of course, the process running polarized Excel needs access to more than just the file being edited. It needs access to its own executable, for one. Most programs also have a large number of auxiliary files, such as shared libraries or fonts. Many times they create temporary files. Since access to these files is needed every time the application runs, regardless of which file it is editing, we give each application an *installation endowment* consisting of the ability to read these files, another concept we carried over from CapDesk [7]. It is this coupling of an installation endowment and the combining of designation with authorization that makes the security decisions part of the user's normal activity.

Unlike sandboxing or Java Web Start, Polaris is able to add to the authorities available to a process without any extra effort on the part of the user. The user simply clicks the File Open icon. Polaris detects the dialog box and replaces it with one from the *PowerBox*, a process that has access to all the user's files. After the user selects a file, Polaris makes that one file accessible to the running program. There are no extra security decisions to be made. 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 [7].

Polarizing an Application

An application needs to be configured to run safe from viruses, a process we call *polarization*. We call an instance of a polarized application a *Pet*. Figure 3 shows the dialog box set up to configure a Pet to run Excel. The user selects a *Pet Name* for the pet. This pet name will appear in the title bar of each window running the pet, 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 clicks on the icon for a file with one of these extensions.
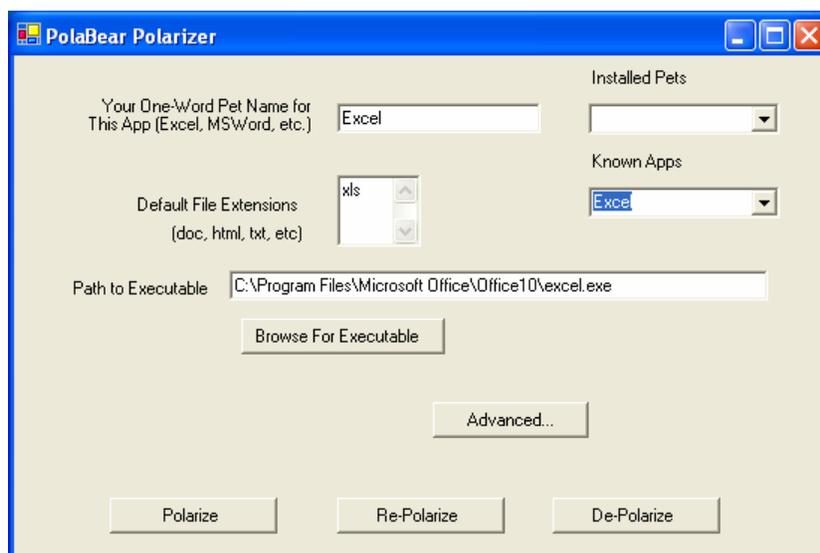
Figure 3. Polarizing an application.

It often makes sense to have more than one pet for a given application. For example, a user might have one browser pet for the Intranet, another one for the Internet, and a third one for reading files from disk. Since each pet runs in a separate user 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. If the browser can be configured to treat all web sites as untrusted, the pet for reading files can't accidentally run a malicious script.

Visual Cues

It is important that the user be aware of the security environment, but the cues should not be obtrusive [13]. As shown in Figure 4, Polaris modifies the title bar of the window running the application. If a pet is running in the window, the pet name appears. If the application was not launched under Polaris, the pet name is left blank. We also take advantage of a feature of Windows XP that let's us change the color of the title bar of windows running pets.
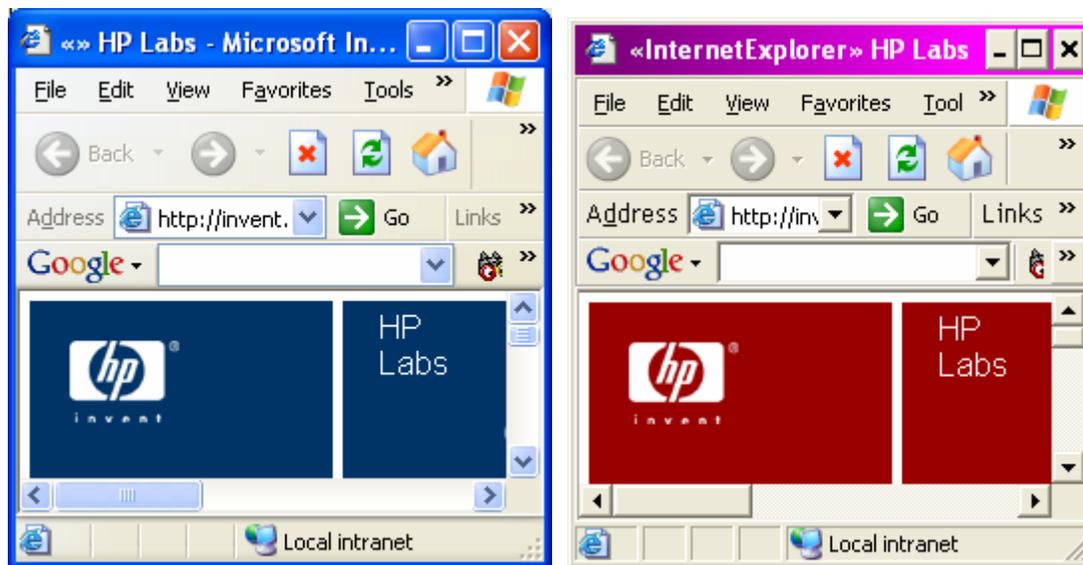
Figure 4. Visual indication of protection state.

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

Just because an application has been Polarized doesn't mean the user is prevented from using the unsafe version. The user 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 the user's permissions. However, if a virus runs in an unpolarized application, it will be able to abuse any of the user's authorities.

4

How Polaris Works

Polaris doesn't change the operating system or the applications; all that changes is the way applications are launched. Instead of starting the application in the logged in user's account, a polarized application is launched in a restricted user account that has very few permissions. This procedure uses the operating system's security mechanisms to limit what the software, including any viruses it contains, can do.

As Figure 5 shows, Polaris launches the application in steps. First, it copies the file to a folder accessible to the restricted account. Next, it sets up a synchronizer to keep the copy and the original file consistent. Finally, Polaris launches the application under the restricted user account using a variant of the Windows RunAs command [8].
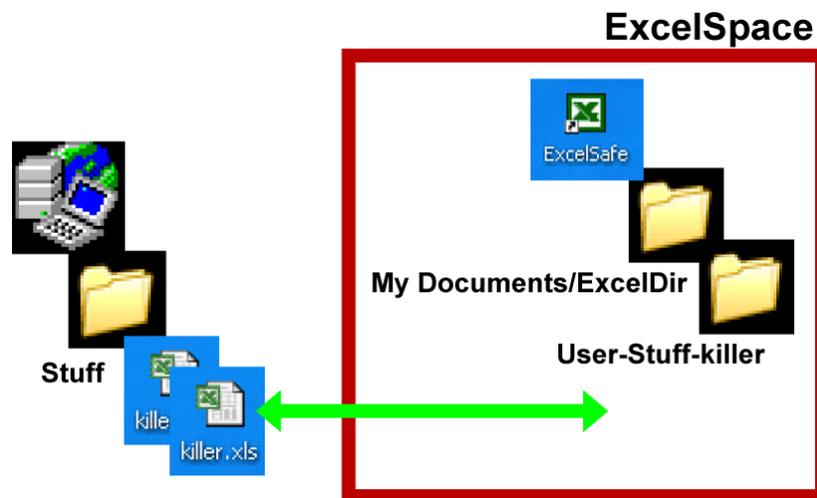

Figure 5. Starting an Excel pet.

If a virus runs in the restricted account, the only thing it can damage is the file it's in. It has no ability to modify the user's startup folder, nor can it read other files looking for secrets. If the browser has been polarized, malicious scripts can't plant executable spyware and adware on the user's system, nor can they harm the machine. A number of users in our pilot study who have visited web pages containing viruses can attest to this last claim.

There are two reasons not to simply change the Windows Access Control List (ACL) and edit in place. First, many applications, for example Microsoft Word, create temporary files in the same directory as the documents they open. These applications would only work properly if we granted both read and write authority to the entire directory containing the document. Doing so would greatly increase the damage that a virus could do. The second reason has to do with the difference between *permission* and *authority* (see sidebar). As implemented, the restricted user account has the authority to effect changes to the original file, but it never gets permission. The advantage is that the authority is revoked when the synchronizer is stopped, for example when the machine

crashes. By properly distinguishing authority from permission, Polaris doesn't leave any dangling permissions to be cleaned up later.

Status

The pre-Alpha version of Polaris has been used by about 20 people in HP Labs, some of them for six months or more. This pre-Alpha version has some significant shortcomings, most noticeably when adding authorities to a running program. Nevertheless, for the most part our users aren't aware of its presence. In fact, one executive used Polaris with no problems for several days before we had a chance to tell him what we'd done to his machine.

The Alpha release of Polaris is currently available under a controlled roll-out. We have added additional users at HP Labs and started pilots at the School of Public Policy at George Mason University and in a group in the US Navy. While we'd like additional reference accounts, we're limiting the number of test sites for the time being.

Future Work

The most noticeable problem with this release is that launching applications is somewhat slow. In addition, this version does not handle linked files very well, such as spreadsheets containing references to other spreadsheets. Also, Java applications will shut down after a brief period of operation under Polaris. We have solutions to these problems that we plan to incorporate in the Beta version.

There are some problems we don't know how to solve. Direct 3D is incompatible with the security machinery inside Polaris. Hence, over half of all game software is incompatible with Polaris. Less significant is the fact that PGP won't run polarized and some operations of the Cygwin command shell, which attempts to emulate a Unix bash shell on Windows, modifies access control lists in a manner that is incompatible with Polaris.

There are some attacks we haven't yet blocked. The current release does nothing about limiting network access, which means that a virus could send the contents of the document being edited to a competitor. We have a solution to this problem that we plan to incorporate into the Beta release. A problem we don't have a solution for is the *GUI hole*. Due to a fundamental design flaw in Windows, any application can read GUI events sent to any window on the screen. This flaw is exploited by keyboard sniffers, for example. Any application can send GUI events to any window on the screen. That means that a virus could send requests for additional authorities to the PowerBox, and select any file on the system. However, if Polaris gets adopted widely enough that virus writers are attacking the PowerBox, we'll have achieved our goal of making the world far safer from viruses than it is today.

Summary

By applying the Principle of Least Authority to individual programs, Polaris provides effective protection against viruses while simultaneously improving usability and functionality. Running Polarized applications will prevent any viruses that run in them from doing very much harm. Parts of the system that viruses attack, such as the Windows directory, the user's startup folder, and most of the Windows registry, are safe. With Polaris, we can start to take advantage of all the money that was spent developing powerful macro languages, use email to send programs to each other, and enable the true power of web scripting, all without opening up our systems to attack.

References

1. Spafford, E. H., "The Internet Worm Program: An Analysis", *ACM SigComm Computer Communications Review*, **19**, #1, pp. 17-57, January 1989
2. CERT, "Love Letter Worm", Advisory CA-2000-04, http://www.cert.org/advisories/CA-2000-04.html, May 2000.
3. Raymond, E. "The Jargon File", http://www.eps.mcgill.ca/jargon/jargon.html. There's no date on the web page, but the language makes it clear that the text was written quite some time ago.
4. Saltzer, H. H. and Schroeder, M. D., "The Protection of Information in Computer Systems", *Proceedings of the IEEE*, **63**, #9, pp. 1278-1308, September 1975
5. Miller, M. S. and Shapiro, J. L., "Paradigm Regained", 2003
6. Kamp, P.-H. and Watson, R., "Building Systems to Be Shared Securely", *ACM Queue*, **2**, #5, pp. 42-51, July/August 2004
7. Stiegler, M. D. and Miller, M. S., "E and CapDesk", http://www.combex.com/tech/edesk.html.
8. RunAsX, http://www.incog.freeserve.co.uk/runasx.html.
9. GAO, "Technology Assessment: Cybersecurity for Critical Infrastructure Protection", GAO-04-321, pg. 27, May 2004
10. McGraw, G. and Felten, E. W., *Securing Java: Getting Down to Business with Mobile Code, 2nd Edition*, Wiley:New York, 1999
11. Kim, S., "Java Web Start: Developing and Distributing Java Applications for the Client Side", http://www-106.ibm.com/developerworks/java/library/j-webstart/.
12. Microsoft, "10 Immutable Laws of Security", http://www.microsoft.com/technet/archive/community/columns/security/essays/10imlaws.mspx.
13. Yee, K-P, "User Interaction Design for Secure Systems". In *Proceedings of the International Conference on Information and Communications Security*, Springer-Verlag LNCS 2513, p. 278-290, 2002.

Sidebar: Viruses and Worms

The terms *virus* and *worm* have been used interchangeably to describe somewhat different types of malware. We use a loosely followed distinction that worms propagate on their own, but viruses are only spread by people. The prototypical worm was released

by Morris in 1989 [1]. An early virus is Love Letter [2], which induces people to open its attachment, a VBS script. Once launched, this script makes several modifications to the machine and sends copies of itself to all entries in all the Outlook address books on the system. The definitions we use here are officially accepted by the US Government [9]. "The Jargon File"[3] also supports these 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**. Nowadays the term has negative connotations, as it is assumed that only **cracker**s 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 VAXen across the U.S.

virus n.

[from the obvious analogy with biological viruses, via SF] A cracker program that searches out other programs and `infects' them by embedding a copy of itself in them, so that they become **Trojan horse**s. When these programs are executed, the embedded virus is executed too, thus propagating the `infection'. This normally happens invisibly to the user. Unlike a **worm**, a virus cannot infect other computers without assistance.

The question is hardly settled, though. The title of a CERT advisory is "Love Letter Worm" [2].

Sidebar: Privilege, Permission, and Authority

The security community often refers to the Principle of Least Privilege [4]. However, exactly what constitutes a *privilege* isn't clear. One attempt at a definition [5] introduces the distinction between *permission* and *authority*. The authors define permission to be the set of rules as written down, say in an access control list, and authority to be the set of actions a process can cause to happen. The latter combines the set of permissions with the behavior of parties having these permissions.

Consider a web server. The process running the server 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 visitor has authority to read the files even though there is no explicit permission granting the access.

Security analysis that considers only permission will be incomplete. Security analysis that includes authority is necessarily limited by our ability to understand the behavior of programs. Fortunately, it is often possible to get a usable bound on the authority available to any process [5].