# PACL's: An Access Control List Approach to Anti-Viral Security[†]

David R. Wichers[††] Douglas M. Cook Ronald A. Olsson
John Crossley Paul Kerchen Karl N. Levitt Raymond Lo

Division of Computer Science
Department of Electrical Engineering and Computer Science
University of California, Davis
Davis, CA 95616

(916) 752-7004

**Abstract**—Almost all attempts at anti-viral software have been a reaction to specific viruses that have infected the user community. These solutions attempt to protect against a specific strain or strains of viruses rather than provide general protection against a wide variety of viruses. This paper describes a new, conceptually simple approach that provides a more general solution to the virus problem. Our approach associates with each file in a system an access control list (ACL) that explicitly specifies which programs can modify the file. Thus, a virus cannot modify arbitrary files and its possible effects are greatly reduced. Our approach is unique in the way it uses ACL's to specify which *programs* can access a file; other schemes use ACL's to specify which *users* can access a file and how. We use the acronym *PACL*'s, for Program ACL's, to refer to these ACL's and to our scheme. To see how our ideas can be incorporated into an existing operating system, we have designed an extension to the UNIX[†††] kernel. We also constructed a simulator that has allowed us to gain operational experience with our ideas in a typical user environment. The results indicate that our scheme is a promising approach for preventing the spread of viruses without being too intrusive on users.

[†] This research supported by Lawrence Livermore National Laboratory, the State of California MICRO program, and Deloitte, Haskins, and Sells, Inc.

[††] David Wichers' present address is: Arca Systems, Inc., 2841 Junction Ave, Suite 201 San Jose, CA 95134 (408) 434-6633

[†††] UNIX is a registered trademark of AT&T.

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

## 1. Introduction

A computer virus is a program that can 'infect' other programs by modifying them to include a possibly evolved copy of itself [6]. One attribute of viruses that allows them to spread so easily is that a virus inherits all of a user's privileges when the user runs an infected program. Typical operating system protection schemes provide no help in such a case—they protect a user's files from other users, but not from him/herself. Thus, a virus can quickly infect all of a user's files. Even worse, if the user has special system privileges (e.g., 'superuser'), the virus can infect all files on a given system.

A typical virus propagates itself by searching for an uninfected program and copying the viral part of its code into that program so that when the newly infected program is run, the viral code will be executed. To prevent propagation, viruses must be prevented from inserting themselves into other programs. (We assume that the operating system prevents programs, including viruses, from writing directly to disk.)

Two simple observations form the basis of our approach. First, the typical virus carrier is unrelated to the programs that it infects. Second, programs executing on behalf of a user have more privileges than are necessary to complete their assigned task. For example, an infected game program might have the privilege to access all of a user's files. Yet it should only have access to those related to the game, e.g., a score file. Our approach, then, is to restrict a program's privileges to the minimum needed to complete its assigned task. Then, if a program is infected, it will not be able to infect unrelated programs (files).

To impose this *least privilege* restriction, we associate an access control list (ACL) [7, 9] with each file in the system. In our scheme, a file's ACL contains the names of all the programs that may modify the file. We use the acronym *PACL*'s, for Program ACL's, to refer to these ACL's and to our scheme. Thus, to modify (write, append, delete, etc.) the file, a program must be on the file's PACL. Our use of PACL's differs from that found in standard ACL schemes: we store names of *programs*, as opposed to the names of *users*, that can access each file.

The notion of least privilege fits well with common system usage. Users create files using a number of different programs. These files are usually modified only by the programs that create them. For example, consider the typical steps involved in creating, compiling, and linking a C program. To create the program, the user uses his/her favorite editor to create source files. During the entire life of those files, they are only modified by the same editor that created them. When these files are compiled, the compiler generates object files. Each time the program is recompiled, these object files are written over by the same compiler, and not by any other program. Similarly, the linker creates the executable and writes over the executable file each time the program is relinked. This usage suggests that normal files are modified by a small number of programs, usually only one. Of course, more complicated usages exist, but they are less common.

Since the number of programs that need to modify a single file is usually very small, we can keep track of these programs in order to prevent other programs from deliberately or accidentally modifying files. This method is similar to existing computer protection mechanisms based on access control lists. The standard ACL scheme is designed to control how each user's files can be accessed by other users. That is, a file's ACL indicates what users may access the files, and in what ways. If the ACL does not explicitly state that a user is allowed to perform the function requested, then it is not allowed. The difference between this security problem and the virus problem is that a virus security system needs to protect a user from him/herself, not from other users. The virus problem is inherently a problem of integrity, not security. Our *PACL-Integrity* scheme is therefore simpler, associating with each file a list of all programs that can modify the file.

To see how our ideas can be incorporated into an existing operating system, we have designed an extension to the UNIX kernel that incorporates our PACL scheme. We have also constructed a simulator to allow us to gain experience with the PACL-Integrity model without requiring actual changes to the kernel. The experience we have gained shows that the scheme seems reasonable to implement and is not too intrusive on the user.

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

The remainder of this paper is organized as follows. Section 2 discusses our PACL-Integrity model in more depth. Section 3 describes how the model can be realized in the UNIX kernel. Section 4 presents the simulator and section 5 describes our experience using it. Section 6 discusses the tradeoffs involved in our approach and outlines future work. Section 7 summarizes related work. Finally, section 8 contains some concluding remarks.

## 2. The PACL-Integrity Model

The PACL-Integrity model associates a PACL with each file on the system. The PACL for a given file names all programs that have the privilege to modify the file. When a file is created, its PACL is set to contain the name of the program that created the file. During the life of the file, the file's PACL can be changed only by a trusted utility program. This utility allows a user to tailor the protection mechanism to meet his/her needs.

The success of a protection scheme depends on how intrusive users find it. A scheme that is too intrusive will effectively render a system unusable. For example, requiring an explicit acknowledgement from a user each time any file is to be accessed might be a secure scheme, but it is not usable. Moreover, if a scheme that is too intrusive provides a means by which the user can disable it, then users will simply run with security checks disabled, effectively rendering a system insecure.

To make our approach secure yet usable, we include a number of 'user-friendly' features. These features simplify common usages of the PACL-Integrity mechanism. The first feature is an inheritance mechanism that allows a user to define a default PACL for a directory. Any file (or subdirectory) created in this directory inherits the directory's default PACL, as well as the name of the program that created the file. This feature allows the user to tailor a directory to the type of work being done in it. An entire system (or account) can be tailored in this manner by creating a default at the root (or home) directory and then building directories below it.

The second feature allows a user to specify a global inheritance policy. The user can define a default PACL for any file based on its extension (suffix). For example, a UNIX object file is typically created by an assembler or compiler and given the extension '.o'. Later, the linker reads in a number of object files, links them together, and generates executable code. When it has successfully generated an executable, it sometimes will remove the object files as they are no longer needed. Since the object files were created by the compiler, their PACL's will contain the name of the compiler, but not that of 'ld' (the linker). With the extension-based default mechanism, the user can define a default for '.o' files that contains 'ld', thereby allowing the linker to remove unwanted object files after it has created the executable.

The third feature allows the user to enable/disable the PACL mechanism for a particular file. This feature is provided by associating a flag with each file. If this flag is enabled, the normal PACL security rules will be applied to that file. If the flag is disabled, then all PACL security rules for the file are ignored and only the 'normal' security rules will be used when the file is accessed; i.e., any program with appropriate access rights can modify the file.

The final feature allows a user to temporarily disable the PACL mechanism for all of his/her files. It also allows the system administrator to temporarily disable the PACL mechanism for the entire system. This feature is needed to facilitate programs that need to modify many or all of a user's or system's files. For example, a utility program that restores files from backup tapes will typically modify many files during its execution.

These features are provided to allow the system to be tailored to meet each individual user's needs. Once defaults have been set up correctly, each user should be able to use the system while being protected from viruses, without being unduly inconvenienced by the PACL mechanism.

The PACL-Integrity mechanism makes several basic assumptions about the underlying hardware and operating system. The devices on which programs are stored (e.g., disk) must be protected so that they can only be accessed by kernel code. Without such protection, a virus could write directly to a device, bypassing all protection mechanisms. This requirement rules out the possibility that this type of

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

system would be viable in some personal computer environments where direct disk access is not protected, for example. The operating system itself must check all file accesses to make sure the PACL security rules are enforced. It must also protect the PACL's themselves from illegal modification. The hardware and operating system must also protect against standard attacks, such as modifying system buffers or kernel code.

## 3. A PACL-Integrity Model for UNIX

### 3.1. Overview

Our PACL-Integrity model can be implemented for UNIX by extending the kernel. The PACL scheme must be included in the kernel to ensure that all file accesses are checked. The current UNIX protection mechanisms, based on user names, are still enforced. If an attempt to write satisfies the existing security rules, the PACL mechanism then further verifies the validity of the access.

When a file is created, its PACL is created as well. A file's PACL is stored as part of the header information (i.e., *inode*) of the file, just like the mode bits, owner, size, date, and time fields. Since the PACL is part of a file's inode, the PACL information for a file is removed when the file is deleted, which simplifies the task of PACL maintenance.

The kernel builds the PACL for a new file from three items. The first item put in the PACL is the name of program that creates the file. In UNIX, a program's name is its complete pathname. For example, the editor program 'vi' in the directory '/usr/ucb' has the name '/usr/ucb/vi'. The second item put in the PACL is the default PACL of the directory in which the file is created. The final item put in the PACL is the default, if any, for the new file's extension. (Note that the defaults put in the PACL are those in effect when the file is created; if the defaults are later changed, the PACL's of existing files are not modified automatically.)

The specific kinds of access for which the kernel must check include opening a file for writing and unlinking a file. The former gives the program the privilege to modify the file in any manner while the latter deletes the file. We consider deletion a form of modification.

### 3.2. New System Calls

Nine new system calls give programs the ability to interact with the PACL mechanism. The first system call, *setppriv()*, is a privileged call that sets the state of the current process into a mode that allows it to call the other new system calls. (This method is analogous to a process setting its user-id to root in regular UNIX.) Without executing this initial call, a process is not allowed to use any of the other system calls that interact with the PACL's, with one exception described below; in such a case, they simply return an error to the calling process. The only programs that are allowed to use *setppriv()* are the programs listed in the file '/etc/paclprivs'. One example of an entry in this file is the utility program described later.

The second call, *paclenable()*, is used to enable or disable (based on its argument) the entire PACL mechanism for the given process and its children. If the initial system process (*init*) disables the PACL mechanism, then the effect is that the PACL mechanism is disabled for the entire system since all processes are children of *init*.

The third call, *clrppriv()*, removes a process from PACL privileged mode. It allows the process to relinquish its privilege when no longer needed. The two calls *setppriv()* and *clrppriv()* allow programs to create critical regions in their code where they have privilege to access PACL's. Outside of these regions, PACL privileges are not necessary and hence should not be enabled.

The fourth call, *getppriv()*, is the only call that will not return an error if *setppriv()* has not been previously called. It tells the currently running process whether or not it is currently in PACL privileged mode, i.e., the process successfully called *setppriv()* without calling a corresponding *clrppriv()*.

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

The next two new system calls allow a program to manipulate PACL's. Only the owner of a file can change its PACL. The first, *addpacl()*, adds a program name to a given file's PACL. The second, *delpacl()*, deletes a program name from a given file's PACL. These system calls also allow a file's owner to enable/disable the PACL mechanism for a particular file.

The remaining three system calls allow a program to query a file's PACL in various ways. These calls can only be executed by the file's owner. The first, *getpacl()*, returns a list of all the program names in a file's PACL. The second, *verpaclm()*, determines if a specified program has the privilege to modify a given file. It compares the program name with those in the file's PACL, handling links if the filename provided is a link to another file. The third, *verpaclr()*, determines if the specified program has the privilege to remove a given file. It is similar to *verpaclm()* except it does not traverse links because any remove reference to a link would be removing the link, and not the file to which the link points.

## 3.3. The *ch* Utility

The above eight system calls provide the means for a system program to manage PACL's. The utility program, *ch*, described below uses these calls and is an example of a type of user interface that can be provided for user interaction with this mechanism. *ch* is listed in '/etc/paclprivs' so that it is authorized to use these PACL system calls on the user's behalf.

To use the *ch* utility, the user must first enter his/her password. We make the assumption that a virus can assume a user's login name but it does not know the user's password. Otherwise, we cannot distinguish a virus from a legitimate user.

*ch* allows the user to:

- add/remove program names from PACL's;
- display the contents of PACL's;
- set/clear the enable flag in PACL's;
- modify the default PACL's for directories and file extensions; and
- temporarily turn off the entire PACL mechanism (e.g., for that user during a single login session).

These features correspond to those described in section 2. Several additional features make the utility more usable. One feature allows the user to traverse the directory structure; a user can, therefore, move to different directories without exiting the utility. A second feature is that *ch* provides all the remove privileges that exist in a normal shell. The 'rm' (remove) program may not have privilege to remove most files; i.e., it may not be in the PACL for every file. The utility, therefore, provides an 'rm' command with functionality equivalent to that of the 'rm' program. Without such a command, the user would need to add the 'rm' program to a file's PACL, exit the utility, and then use the 'rm' program to remove the program. For the same reason, the utility also provides an 'rmdir' (remove directory) command. Basically, *ch* provides a subset of the normal shell commands along with the features described above that allow the user to tailor the PACL security system. If the user executes a program from within *ch*, a new process is created to execute that program. This process is subject to the rules that apply to the new program, not those that apply to the *ch* program. Other utility programs can easily be generated by the system administrator by writing programs using these system calls and then adding the program names to '/etc/paclprivs'.

## 3.4. The Role of the Superuser

In existing UNIX systems, the superuser—e.g., the 'root' account—may bypass the normal protection mechanisms. Having root privilege is not sufficient to override the PACL protection mechanism in our system. In particular, a user (or would-be virus) executing as root can only disable the PACL mechanism using the *ch* utility, for which it must give the root password. A program running as root must, therefore, be listed in a file's PACL in order for that program to have the privilege to modify that particular file.

DOCKET ALARM Find authenticated court documents without watermarks at docketalarm.com.

# Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

## Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

## Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

## Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

## API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

**LAW FIRMS**

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

**FINANCIAL INSTITUTIONS**

Litigation and bankruptcy checks for companies and debtors.

**E-DISCOVERY AND LEGAL VENDORS**

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | sales@docketalarm.com | 1-866-77-FASTCASE

fastcase® Smarter legal research.