Protecting Users From “Themselves”

William Enck, Sandra Rueda, Joshua Schiffman, Yogesh Sreenivasan,
Luke St. Clair, Trent Jaeger, and Patrick McDaniel
Systems and Internet Infrastructure Security Laboratory
Department of Computer Science and Engineering
The Pennsylvania State University
University Park, PA 16802
{enck, ruedarod, jschiffm, sreeniva, lstclair, tjaeger, mcdaniel}@cse.psu.edu

ABSTRACT

Computer usage and threat models have changed drastically since the advent of access control systems in the 1960s. Instead of multiple users sharing a single file system, each user has many devices with their own storage. Thus, a user’s fear has shifted away from other users’ impact on the same system to the threat of malice in the software they intentionally or even inadvertently run. As a result, we propose a new vision for access control: one where individual users are isolated by default and where the access of individual user applications is carefully managed. A key question is how much user administration effort would be required if a system implementing this vision were constructed. In this paper, we outline our work on just such a system, called PinUP, which manages file access on a per application basis for each user. We use historical data from our lab’s users to explore how much user and system administration effort is required. Since administration is required for user sharing in PinUP, we find that sharing via mail and file repositories requires a modest amount of administrative effort, a system policy change every couple of days and a small number of user administrative operations a day. We are encouraged that practical administration on such a scale is possible given an appropriate and secure user approach.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and protection —*Access Controls*

General Terms

Security

Keywords

Access Control, Policy

1. INTRODUCTION

When access control was invented, computers were expensive and limited resources. Each computer supported several users who shared not only the CPU, but also the storage of these machines. Early access control systems were designed to protect the secrecy and integrity of each user’s files from all the other users’ processes on a single computer [16, 17]. The main concerns at this time were that: (1) a user’s buggy program may modify the files of another user and (2) a nosy user may be able to browse another user’s secrets by scanning her files.

The world of computing is very different now. Two major differences are: (1) the increased variety and lower cost of computing devices[1] and (2) the increased variety of threats against our computing devices. First, the advent of many inexpensive devices has created a situation where each user owns multiple devices, so there is no other user to restrict. Second, new threats have emerged due to the increased connectivity and ease of appropriating software that results from that connectivity. Now, users must be more concerned with the threat that their own processes may be malicious or have a vulnerability that a remote attacker can leverage. For example, a web browser client executes a variety of programs (e.g., plugins) to process browser content, but all these programs run with the full rights of the user (i.e., as users “themselves”). Some of these programs may be malicious, some may have vulnerabilities, but the user must trust all these programs with all their data.

We claim that the access control problem of the early days of computing has morphed into a new problem. In the current environment, users are isolated from one another by default and the main challenge is to manage the access of each user’s applications. Sharing among users does occur, of course, but we claim that sharing can be modeled by a small number of mechanisms: email, web, and version control repositories. Thus, we believe that future access control models should leverage such natural isolation of users to simplify policy, provide a reliable control of user’s data based on applications, and enable limited sharing without complicating policy significantly. Towards this end, we have developed the PinUP access control system [7], a Linux Security Module that binds permissions to applications, provides a rule language for expressing how files are shared among applications, and treating inter-user sharing as an exceptional case.

In the future, we envision that user administration should more closely mirror sharing among isolated users. Our access control infrastructure should be setup such that normal, predictable operation is handled by system policy (i.e., policy specified by system administrators and/or general-purpose policy rules). System administrators may have to make some changes to system policy to support variations in behavior, but these should be quite infre-

[1]The notion of a computing device is much broader than that of a computer of the 1960’s and 1970’s. We consider any device that may be programmed or whose software may be reconfigured, including cell phones and PDAs, as a computing device.

quent. In this vision, users will have to administer exceptional sharing, but this sharing is limited to a few, well-defined mechanisms: email, web, and version control repositories. Only when users apply these mechanisms do they need to consider the sharing implications. Otherwise, user files are isolated from other users. The PinUP system supports default isolation policies, so it is an ideal candidate to implement this vision as we discuss.

The approach above raises the following question, *"In a world in which we approximate acceptable system behavior through user isolation, how many exceptions to that approximation will occur in practice and how difficult will it be to correct the policy given that approximation?"* This question highlights the key tradeoff in the PinUP system. Inasmuch as the system can predict all uses of a file, no user or system interactions are necessary. Where such approximations are insufficient, the user is required to administer the policies (e.g., using PinUP -supplied tools) that diverge from the norm. Consider for example a user that creates a to-do list from using an ASCII editor such as `vi`. Later, the user may want to share that list with another user by emailing it to her or placing it in a shared file repository. Such behaviors are not predictable in any practical sense, and *must* be driven by user-specified interactions with the policy system. Just how much interaction the user has with the system is key to the usability of the system and access model.

The following sections attempt to answer this question by looking at historical data in our laboratory to assess the number of interactions administrators and users would have with the policy system, and at some level attempt to understand the viability of the administrative model. By looking at the use of file repositories and email behavior, we collect the number of operations that would require unpredictable sharing with other users, thus requiring exceptional policy changes. A central distinction that we make between *system* and *host* policy is that the system policy requires the efforts of a system administrator (i.e., someone not directly involved in the application), whereas individual users change the host policy. In the examination of email and repository modifications, we find that the number of policy changes required of users and administrators is not large: administrators must perform a policy change every 2.6 days for a lab of 65 people and users must update their policies every 8.6 days on average. This data motivates optimism that an access control system with the isolation of users at its core may become a practical approach enabling effort to focus on how users manage their own data among applications.

2. RELATED WORK

Recent operating systems provide Mandatory Access Control via SELinux [14], AppArmor [15] and TrustedBSD [1]. However, writing policies to describe the files and objects accessible from a domain requires a deep understanding of access control and operating systems, a task far beyond the abilities of most users. Moreover, such systems are oriented toward protecting mainly system files and not user's files.

Specifying which files an application may access, known as sandboxing, is a common approach to isolate applications [5, 15, 9, 3, 12]. However, sandboxing techniques operate solely on filesystem abstractions like files and directory paths. Ko et al. [10] attempt to overcome such limitation by specifying policies of expected application behavior. However, this approach is more appropriate for developers, as the policies are slightly complex and require knowledge about the expected behavior of the application. Similar to the PinUP model, LIDS [12] allows files to be bound to specific applications, but the interface is restricted to system administrators. Contrary to these systems, the goal of PinUP is specifically to offer a user-oriented platform.

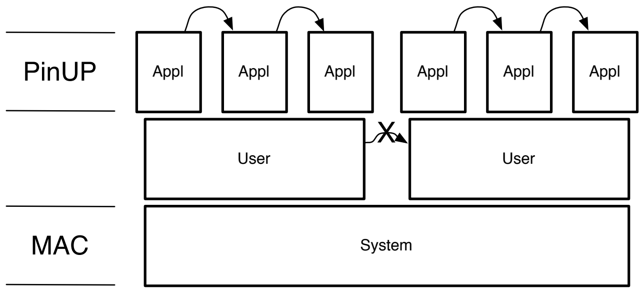

Figure 1: PinUP provides control over user applications, assuming users are isolated by default and a mandatory access control approach protects the system.

Lai and Gray [11] propose a mechanism to allow the user to specify the list of files an application may access. Unfortunately, the list must be specified each time an application is started. TRON [2] offers a similar approach, where a user's initial shell has capabilities for the entire home directory, and each user must explicitly create child processes with less capabilities. A major drawback to both Lai and Gray and TRON is that the user is by default responsible for setting up the environment in a proper way. Additionally, these operations must be performed every time an application is run. The RBAC model was also applied to allow users to run application inside defined subdomains [8]. While this approach allows users to specify the rights of each one of their subdomains only once, the user is still responsible for the initial setting and then for switching to the proper subdomain every time an application is run.

More recently, Polaris [18] extends Microsoft Windows allowing users to sandbox an application by indicating the set of files that will be accessed. Polaris uses "installation endowments" to provide applications capabilities to access system files, thereby focusing policy specification on user files. In an effort to provide greater flexibility, Polaris allows user to specify multiple sandboxes for different instances (pets) of the same application. Usability studies indicate that this option leads to user error where the wrong pet is selected, thereby compromising file security [6].

We claim these previous approaches make user administration difficult because users have to work with filesystem abstractions to sandbox applications. Our experience is that users only share files via a small number of mechanisms, such as email and repositories. Our goal is to determine whether user administration can be made tractable if users are isolated by default and can only share files using these mechanisms. We examine the hypothesis using an access control system called PinUP that isolates users by default. In our evaluation we demonstrate the administrative effort system administrators must exert to configure the default, general policies that PinUP provides for isolation and the administrative effort of users for sharing files between users.

3. PINUP

In this section, we examine the PinUP system to make clear the type of policy administration that results from PinUP's approximation of system behavior. Figure 1 shows PinUP's view of access control. PinUP is not another discretionary access control (DAC) approach, but rather, it is an overlay on a mandatory access control (MAC) base. PinUP is designed under two key assumptions: (1) the MAC enforcement, such as SELinux [14] and AppArmor [15], protects the system and (2) interaction between users is not allowed by default. First, since MAC enforcement protects the system, PinUP only protects user data. Second, since users do not interact by default (i.e., user interaction is not the norm), PinUP focuses

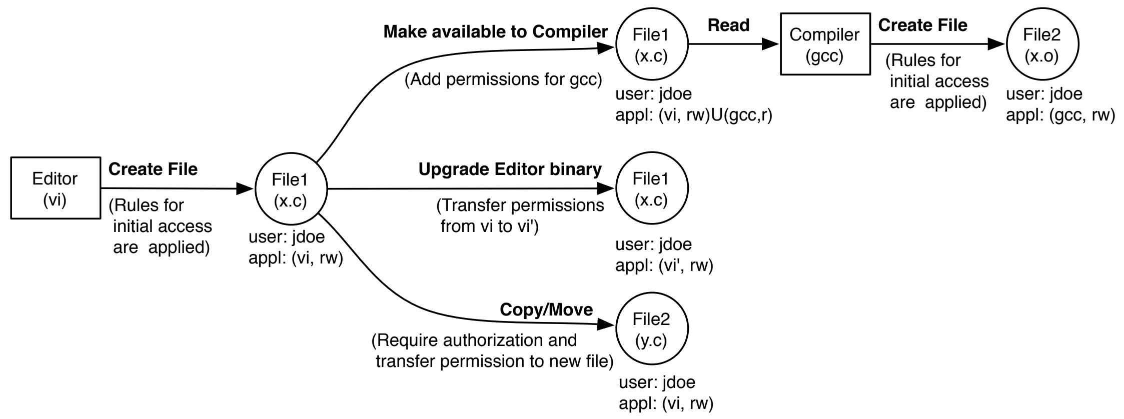

Figure 2: A summary of the administrative tasks for managing application access to files.

on managing access between each user's own applications. Since user interaction is infrequent, PinUP requires that the user or administrators perform manual operations to allow sharing. In this section, we examine the PinUP mechanisms, and in the next section whether the assumption of infrequent, manual administration appears feasible.

PinUP manages file access by controlling which applications can access which files for each user. When a file is created, only the creating application has access to the file by default, but PinUP *access automation rules* can describe how files created by one application can be accessed by another and how files of certain types can be accessed by certain applications. Changes to these PinUP policies can be made, but every change requires per-use user authentication (similar to the administrative interfaces of the OS X operating system). In particular, **user processes do not have the ability to modify PinUP policy**. We detail the implementation of the PinUP system in an extended technical report [7], and defer design issues to that text.

We now illustrate PinUP via the example in Figure 2. Assume that the user `jdoe` creates a file `x.c` using the editor `vi` . The identity of the user, creating application, and other attributes such as file extension are used to identify the PinUP policy to be applied to the file, e.g., `jdoe`, `vi`, and `.c`. The policy explicitly states which applications will be allowed to subsequently access that file. In this case, `vi` is given subsequent read and write access. `x.c` is a source file, so `jdoe` wants to make it read-only accessible to a compiler through a command line tool. Note that such a policy need not be manual–in PinUP policy rules can be stated that would dictate that all `.c` files created by `vi` would made available to the `gcc` compiler automatically. The compiler can then create a new object file `x.o` that it can write and that can also be read by the linker. Similar policy enforcement will occur as applications cascade over data to consume and create protected files.

Also illustrated in Figure 2, PinUP presents a number of other interesting design and implementation challenges. For example, how one identifies applications and propagates their identity across updates is key to ensuring correct policy enforcement. The issue of attaching, tracking, and propagating PinUP polices associated with user files is also daunting. This latter process is closely related to label management in mandatory access control systems. However, because of policy semantics and form, PinUP policy tracking requires different machinery.

3.1 Host-Level Policy

At a very high level, PinUP exists to enforce a rudimentary intra-application information flow policy. In this, there are two facets of policy that are relevant to safe operation. First, the enforcement must determine how the output files created by an application should be automatically associated by other applications, e.g., the `.o` object file output by `gcc` should be automatically readable by the `ld` linker. In another example, a `.qdf` Quicken file should only be accessible via the Quicken program. The set of policy rules that govern these permissions is a reflection of the workflows and environmental practices of the user/host. Constructing these policies is essential to closing the vulnerabilities presented by existing access controls, and is an open problem. However, we expect that many application usage policies will be common across all users.

Where workflows are not known or where the user needs to perform atypical manipulation of files, PinUP tools must be used to modify the application associations. For example, a user who wishes to burn an encrypted version of the Quicken file onto a CD would need to modify the PinUP permissions to make associations between the files and the program that will manipulate them. Such policy changes are implemented in the current system through `pinmod`, which requires the user to enter her password before changing the internal policy associated with a file.[2]

Note that these operations apply only to those files the user deems to be of "high-value". PinUP access automation rules will describe which files are to be governed by PinUP. Other files not being governed by PinUP will only be subject to the normal system access controls.

3.2 Distributed System Policy

Since users each have their own computing devices, enabling sharing among users becomes a distributed systems policy issue. Thus, in addition to some user administration to enable sharing, some system administration will also be required. In this section, we examine the types of system administration that the PinUP approach needs.

In the distributed case, each host in the distributed system would apply a common, system-administered policy that states relationships between applications and the rights they have to read, write, and execute the file content. The only additional systemic requirement this extended model would place on the host is a service for obtaining and updating the access policies to be enforced. Put another way, an environment-wide policy would be obtained from a central authority and would supersede the local user/host policy.

The distributed service would need to extend the model of software identity to include versioning. For example, many different

[2]This ensures that a malicious application cannot circumvent the protections by launching the `pinmod` without the aid of user.

Table 1: Subversion Stability - the administrative operations observed over 114 weeks (each period was two weeks) within the SIIS Laboratory at The Pennsylvania State University.

Operation	Description	Ops	Ops/Period	Post/Ops	Post/Ops/Period
NEW_REPO	Creation of a Subversion repository, i.e., version controlled filesystem tree	169	2.9649	115	2.1698
REPO_CHANGE_AUTHS	Changing of user permissions on existing Subversion repository	0	0.0000	0	0.0000
DEL_REPO_USER	Remove user from permissions of a Subversion repository	15	0.2632	15	0.2830
NEW_REPO_USER	Add a new user to the permissions of a Subversion repository	53	0.9298	53	1.0000
NEW_USER	Introduce a new user into the Subversion system	65	1.1404	51	0.9623
DEL_USER	Remove a user from Subversion system	1	0.0175	1	0.0189
CH_PASSWD	User change of a Subversion password	2	0.0351	2	0.0377
DELUSR_GROUP	Remove user from a Subversion group	0	0.0000	0	0.0000
ADDUSR_GROUP	Add user to a Subversion group	0	0.0000	0	0.0000
NEW_GROUP	Create a new Subversion group	2	0.0351	0	0.0000
DEL_GROUP	Delete a Subversion group	0	0.0000	0	0.0000
All	All operations observed on the Subversion system	**307**	**5.3860**	**237**	**4.4717**

versions (and patch levels) for a single editor within the environment may exist–and unlike hosts, such versions must be able to access the file simultaneously. This would require an extension to the semantic meaning of application identity (which in the host case is simply a hash of the executable) to encompass version histories and equivalences between versions.

In a general sense, the administrative model of this application-oriented access policy is similar to traditional models. Like other kinds of environmental policy, it is highly desirable to set global invariant policy that states security relationships between files and applications [13]. Secondly this *system policy* must identify the set of hosts that may participate in access system and identify singular hosts that have dominion over subsets of the governed data.

The *host level* policy operates as described above; all PinUP operations that set or change permissions operate in substantively the same manner, with the exception that a) permission changes must be consistent with the system policy, and b) updates are synchronized with the centralized policy database. In very much the same way as permission changes in distributed file system are currently implemented [19], all updates must be propagated to the other (interested) hosts in the system.

4. EVALUATING ADMINISTRATION

The above discussion demonstrates the functionality of PinUP, thus identifying where user and system administration may be possible. We envision that PinUP's access automation rules will cover nearly all application-level sharing, but users and system administrators will still need to direct sharing among users. Users themselves will declare the files to be shared with other users since this cannot be predicted from application behavior. System administrators will have to configure the means of sharing, system repositories, email, and web systems, to provide controlled sharing, where required.

To better understand the viability of the PinUP administrative model, the following sections attempt to quantify the amount of effort required in such administration. To answer this question, we look at historical data in our laboratory to assess the number of interactions administrators and users would have with the policy system. The central distinction we make in the next two sections between *system* and *host policy* is that the system policy is not malleable by the host and can only be modified by system administrators. From an overall perspective, these two studies attempt to understand how often an administrator will need to modify the distributed system policy, and how often users will be required to manage the host policy.

4.1 System Administrator Effort

The central (and often singular) means of sharing between users and hosts within the SIIS Laboratory for over 2 years is through Subversion [4]. As the central sharing mechanism, its access controls represent the best approximation of a global system access policy for our laboratory. Therefore, one can view the historical changes within that policy as a model for the changes one would expect to see for a global PinUP policy.

Subversion is a multi-user, distributed version control system. As configured locally, this widely used suite of tools delivers content via webserver to hosts over a HTTP/SSL (`https`) connection. Each user communicates with the Subversion server via the `svn` client, or alternatively through a web browser or OS-local file manager application. Clients are authenticated using passwords/username setup specifically for Subversion on the webserver and entered into the Subversion client. Read and write authorizations for Subversion content are assigned to users or groups. Groups are implemented as managed collections of user lists.

The primary organizing resource in Subversion is a *repository*. Allocated within an administrator-defined directory tree, each repository represents a unit of sharing that contains files and directories. In our environment, we create repositories for each new project, paper, or other administrative effort such as the laboratory web content. Users also have personal repositories to store local environments, e.g., startup files, personal tools, etc., which is used to relocate user environments on many machines quickly. Users *checkout* (obtain the current files), edit, *add* and *delete* files and directories, and *commit* (push changes to repositories). Administrator rights are also given to individual users via server-side configuration to manage sets of users, groups, and repositories.

Table 1 enumerates the administrative operations occurring on the Subversion webserver. Note that all of these operations require the intervention of the administrator. In particular–as an oddity of the environment–user password changes can only completed by the administrator replacing the password in the `htpasswd` file on the webserver. All other listed operations require the administrator to

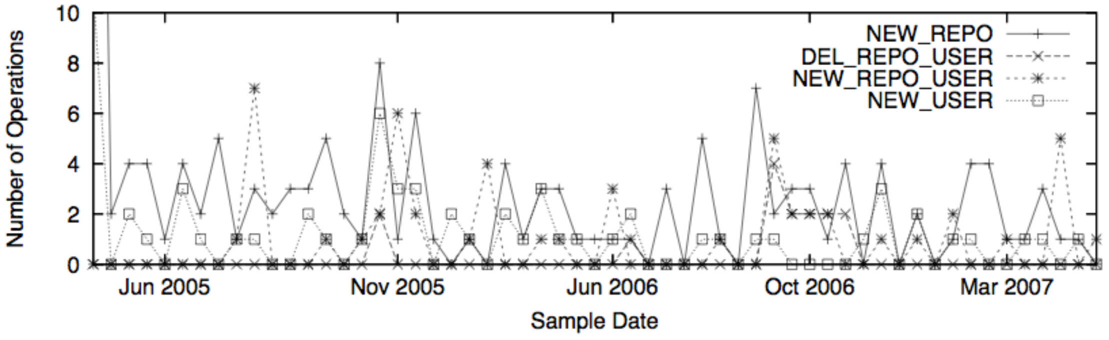

Figure 3: Frequent Operations - administrative operations occurring repositories over the 114-week sample period.

modify local configuration files on the webserver.

In order to assess the frequency of these operations, we evaluated the evolution of Subversion configuration files on Subversion over the 114 weeks since its introduction. The results that follow describe the analysis (as implemented by custom processing scripts which observed changes in configurations on consecutive samples). Note that there is a startup period in which a large number of administrator operations are necessary; this included an initial configuration process that created repositories, added users, etc. Therefore, we present two measurements, all operations and those *post*-setup operations. The latter measurements attempt to assess the steady-state operational activity by ignoring the first month of activity in determining the total and average number operations per two-week sample period.

Returning to Table 1, the operations bifurcate into those that are (relatively speaking) used frequently, and those that are use infrequently or not at all. The most frequent operation is the creation of a repository, where the administrator creates a new repository to track a project–this is most often the creation of a directory to track a new paper, proposal, or coding project. This is analogous to the establishment of new efforts, which is likely universal to all similar environments. Interestingly, we also see adding users to the environment as a frequent occurrence. After initial setup and ignoring the slow introduction of new students and faculty into the laboratory, this is largely a reflection of external collaboration. We have active collaborations that use Subversion as the primary sharing mechanisms with 14 different universities and laboratories. Such is the nature of research, and we find the Subversion is an effective metaphor supporting collaboration. Finally, we see frequent addition of users to repositories, and a less but noticeable frequency in the removal of users from repositories.

Conversely, there are several operations that are used infrequently or not at all. Principal among these is the use of groups. We initially setup "administrator" and lab "insiders" groups, which essentially never changed. Also, we found that repository permissions were never changed. Users were universally given both read and write permission, and no attempt was ever made to change them. Finally, essentially no users ever change their password. This is likely due to the difficulty of the interface, i.e., involving the administrator.

Figure 3 further illuminates the operational churn of the Subversion configuration. Here, the figure shows the four most frequently used operations (add repository, add/remove user from repository, and add user to Subversion) as a function of time. Apart from the initial configuration burst, the number of administrative operations is exceptionally constant. At no time are there more than 16 operations in a two-week period. On average, an operation occurs about once every 2.6 days (or 0.378 operations/day). Considering there are around 170 repositories being used by 65 users, this seems like an extremely stable configuration.

What does this suggest about the system policy in a PinUP - style system? This characterization suggests that the system-level view of sharing evolves constantly and very slowly. In this environment, users tend to control access not through fine-grained access privileges, but through governance of access to entire "projects". Groups and low-level permissions were ignored in deference to the simplest access policy available. Either users were trusted in the project or they were not—no other access formulation was necessary. This is encouraging to our thesis, as it indicates that administrators will not be needed to actively manage the PinUP policy–global sharing policies are simple and stable.

Another interesting characteristic exposed by this study is that rights are largely monotonically increasing. That is, users very infrequently lose rights, but very frequently are given new rights (e.g., added to a repository). It is telling that the only one user was removed from the system, and only 15 times was a user removed from a repository over the two-year period. These were the *only* occurrences in system were rights were revoked. Thus, at least at the system level, the closed world approach adopted by PinUP is consistent with current behaviors (where the user increases rights in exceptional cases).

4.2 User Administrative Effort

To estimate the effort needed to specify host policies, we examine the two most common modes of sharing used in the SIIS Lab: (1) adding files to Subversion repositories and (2) emailing files as attachments. In both cases, we observe a modest amount of user administration of one policy change every 8.4 and 2 days for Subversion and email, respectively.

4.2.1 Repository Sharing

Because we seek to measure how often a user must change his host policy, we need only count the number of Subversion operations that would elicit a host policy change. In Subversion, users can invoke the `svn add` command to include a new local file into a repository. In PinUP, this would require a change in host policy to give the `svn` binary read access to the file. As this is the sole operation that would require an update to the host policy, our analysis examines just file additions.

Our analysis seeks to characterize the *effort* required for host specific policy administration. In doing so, we investigate both raw file additions, as well as larger batch *requests* that correspond to a single semantic change. In general, the number of requests is the same as the number of raw operations in the worst case, but in practice the number of requests is much smaller.

We first examined individual file additions made to our SIIS Laboratory Subversion repository. Figure 4 illustrates data from the

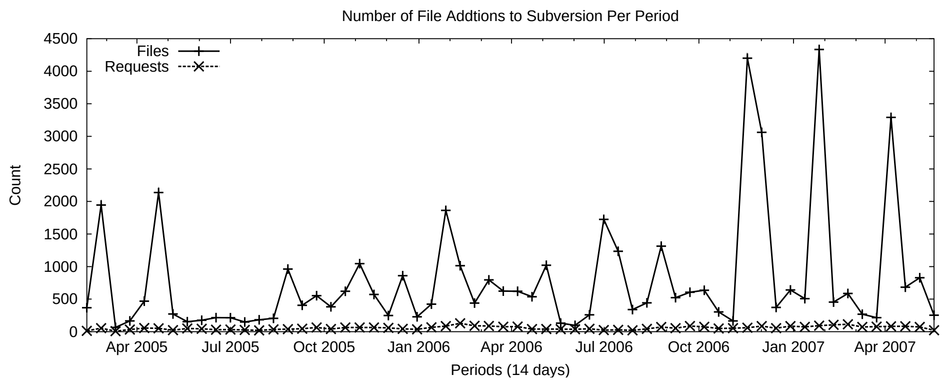

Figure 4: Frequency of Subversion Add Operations - Number of files added to Subversion repositories observed over 114 weeks (each period was 2 weeks) for all users within the SIIS laboratory at Pennsylvania State University.

collective log files for all users in all repositories in two-week intervals over a two-year period from April 2005 to April 2007. As is evident from the several large spikes (especially during 2007), the sheer number of individual files added is large. Since we equate each file addition to a manual change in the host policy, it would appear that each user must perform a prohibitive number of host policy updates. Even after averaging these results over all users, periods of high activity could incur significant administrative overhead.

Fortunately, the majority of addition operations are performed as a single `svn add` operation. Looking again at Figure 4, we see the line representing the number of requests initiated by users is not only consistent, but promisingly small. For example, the period around January, 2007 showed a spike of 4335 file additions, but this only corresponded to a modest 92 requests. In fact, on average, each user only issues 0.12 requests for additions per day with the busiest users reaching 0.9 requests per day. By modifying PinUP utilities to perform batch policy changes, each request would only involve a single change to the host policy.

These usage patterns are indeed feasible in a PinUP system. While sharing may result in a large number of raw operations, the amount of actual user administration is reduced to a few requests. With so few operations requiring a change to the host policy, users will only be interrupted to change the policy at an infrequent and manageable rate.

4.2.2 Email Sharing

Lab members also communicate frequently via email. In a system using PinUP, email clients initially will not have access to the files to be attached. This restriction is mainly to prevent a compromised email client from sending files without proper authorization. Similar to the Subversion case, the email client must be given access every time the file is to be attached. In fact, even if we attach the same file multiple times to different emails, we would want to authorize access only for that particular email. To characterize the administrative effort required for such fine-grained access, we study the *sent* mailboxes of a number of SIIS lab members.

Figure 5 shows the number of email attachments of seven lab members over a period of 18 months[3]. From the study, we observe that, with minor exceptions, attachments are sent very infrequently. On average, users send 1 email attachment every 2 days with high volume users averaging 3 per day. This study confirms our intuition that the sharing via email is indeed very little and can easily be administered using application-oriented access control policies like PinUP with minimal policy changes.

Upon further investigation of our email data, we discovered a semantic separation within the results. Users 4 and 5 have sent significantly more attachments throughout the observed duration. Interestingly, users 4 and 5 are professors, while the others are graduate students. Such discrete classification indicates that a user's role significantly impacts sharing patterns.

We further classified users based on their roles to gain a better understanding of this relationship. Figure 6 shows the number of email attachment for each role: professor and graduate student. From our data we observe that professors send significantly more email attachments than graduate students. This is easily attributed to their administrative role in the system (working with many students on many projects and also departmental overhead) in contrast to students who are more confined to research oriented activities. This characterization suggests that sharing patterns are very much driven by the overall role in the system. This further validates our claim, as it indicates that normal users require only a modest amount administrative effort for managing a system using PinUP and even the high volume users require only a small number of policy changes.

5. NEXT STEPS

This paper aims to show that the design of access control systems no longer fits the current model of the isolated single user device. The main characteristics of today's systems are:

- Users run multiple applications that require distinct access rights, creating an environment in which the user's own programs may pose a threat greater than that of other user's programs.
- Sharing of resources between users is limited and should be treated as the exception rather than the rule.

To facilitate these factors, access control systems should prohibit all access to files except where explicit permissions are granted to applications designated by the user.

To test our hypothesis and the feasibility of our model, we explored the administrative overhead required to implement such a system. We analyzed the behavior of a set of users in our lab and identified the number of policy changes resulting from their actions. We discovered, from a system-wide perspective, that the administrative operations needed for sharing are minor and the policy itself

[3]Note that some users were not in the University at the start of the study, thus resulting in no attachments per user early in the study.

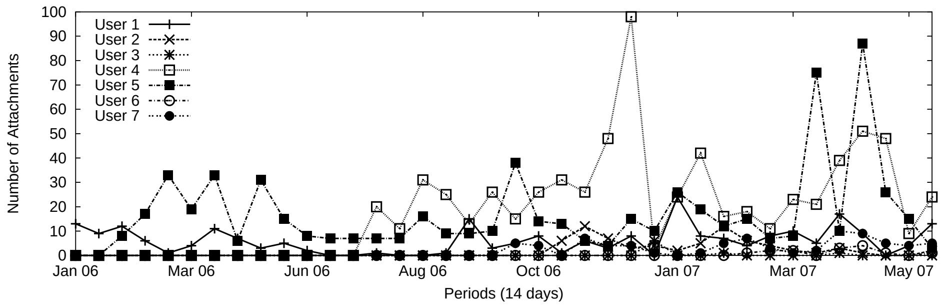

Figure 5: Frequency of Email Attachments - the number of files shared as email attachments observed over 78 weeks (each period was two weeks) for six individual users within the SIIS laboratory at Pennsylvania State University.

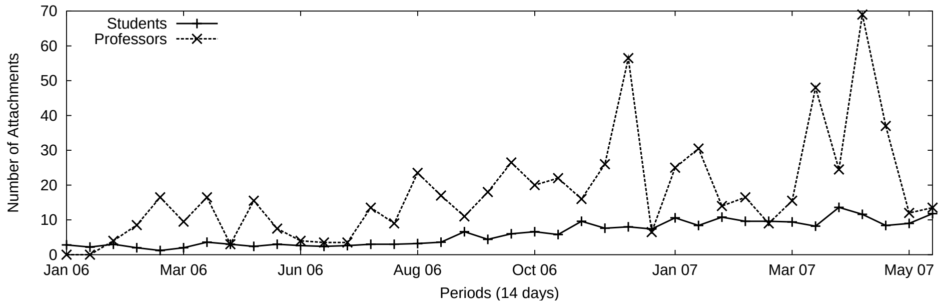

Figure 6: Frequency of Email Attachments - the number of files shared as email attachments observed over 78 weeks (each period was two weeks) for two different roles within the SIIS laboratory at Pennsylvania State University.

was generally monotonically increasing, while the number of revocation operations remained minimal.

The administrative burden a user faces was also observed to be small. Despite the large number of file additions we observed in Subversion, users only issue one actual host policy change per 8.4 days on average. For email, we observed users required a daily maximum of 3 operations on average depending on the user's role. As each administrative operation in our system would demand some user intervention, the minimal cost a user would incur per day validates that our model of access control does not cause undue burden on real users.

Current access control mechanisms no longer reflect modern day usage models. As computing platforms shift to support single user environments, the threat of harm has changed from malicious users to applications running as the user. In a PinUP system, file access is mediated on a per application basis. However, to support interaction between users, policy exceptions are required. Our observations from Subversion and email logs demonstrate that the frequency of these exceptions is small. In our future work, we intend to develop mechanisms to further simplify the ad hoc policy update process.

6. REFERENCES

[1] TrustedBSD. `http://www.trustedbsd.org`.

[2] Andrew Berman, Virgil Bourassa, and Erik Selberg. TRON: Process-specific file protection for the UNIX operating system. In *Proceedings of the USENIX Technical Conference*, pages 165–175, 1995.

[3] S. Chari and P. Cheng. Bluebox: A policy-driven, host-based intrusion detection system, 2003.

[4] CollabNet. Subversion. `http://subversion.tigris.org`.

[5] Crispin Cowan, Steve Beattie, Greg Kroah-Hartman, Calton Pu, Perry Wagle, and Virgil Gligor. SubDomain: Parsimonious server security. In *Proceedings of the 14th USENIX conference on System administration*, New Orleans, Louisiana, December 2000.

[6] Alexander Dewit and Jasna Kuljis. Aligning usability and security: A usability study of polaris. In *Proceedings of the 2nd Symposium On Usable Privacy and Security*, 2006.

[7] William Enck, Patrick McDaniel, and Trent Jaeger. Protecting User Files by Reducing Application Access. Technical Report NAS-TR-0063-2007, Network and Security Research Center, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA, February 2007.

[8] Christian Friberg and Achim Held. Support for discretionary role based access control in ACL-oriented operating systems. In *Proceedings of the second ACM workshop on Role-based access control*, pages 83–94, 1997.

[9] Ian Goldberg, David Wagner, Randi Thomas, and Eric A. Brewer. A secure environment for untrusted helper applications. In *Proceedings of the 6th Usenix Security Symposium*, San Jose, CA, USA, 1996.

[10] Calvin Ko, George Fink, and Karl Levitt. Automated detection of vulnerabilities in privileged programs by execution monitoring. In *Proceedings of the 10th Annual Computer Security Applications Conference*, pages 134–144, 1994.

[11] Nick Lai and Terence Gray. Strengthening discresionary access controls to inhibit trojan horses and computer viruses.

In *Proceedings of the 1988 USENIX Summer Symposium*, pages 275–286, June 1988.

[12] Linux intrusion detection system (LIDS). `http://www.lids.org`, accessed January 2007.

[13] Patrick McDaniel and Atul Prakash. Methods and Limitations of Security Policy Reconciliation. In *2002 IEEE Symposium on Security and Privacy*, pages 73–87. IEEE Computer Society Press, May 2002. Oakland, CA.

[14] National Security Agency. Security-enhanced Linux (SELinux). `http://www.nsa.gov/selinux`.

[15] Novell. AppArmor application security for linux. `http://www.novell.com/linux/security/apparmor/`, accessed December 2006.

[16] D. M. Ritchie and K. Thompson. The UNIX time-sharing system. *The Bell System Technical Journal*, 57(6 (part 2)):1905+, 1978.

[17] M. D. Schroeder, D. D. Clark, J. H. Saltzer, and D. Wells. Final report of the MULTICS kernel design project. Technical Report MIT-LCS-TR-196, MIT, March 1978.

[18] Marc Stiegler, Alan Karp, Ka-Ping Yee, and Mark Miller. Polaris: Virus safe computing for windows xp. Technical report, HP Laboratories Palo Alto, 2004.

[19] A. Tanenbaum and M. Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2002.