Data Privacy: Are Operating Systems Doing Enough?

Aameek Singh
College of Computing
Georgia Institute of Technology
aameek@cc.gatech.edu

Ling Liu
College of Computing
Georgia Institute of Technology
lingliu@cc.gatech.edu

Abstract

In this paper, we evaluate operating systems for their support for user-data privacy. Our studies at three different installations indicate that current levels of user education and OS support are astonishingly poor. For example, at one installation of 836 users, we were able to access over 86 GB of private *data, including more than 300,000 emails, using simple scanning techniques. Along with this analysis, we also present a number of OS support mechanisms to alleviate this problem and discuss their pros and cons.*

1 Introduction

The growing use of technology in day-to-day affairs has brought about an immediate need for user privacy. While a lot of privacy issues are regularly researched for new and upcoming technologies like RFID [4, 3] and P2P [5], little has been discussed on the support for data privacy in operating systems and most of the original access control mechanisms still work the same way. This problem is exacerbated by the archaic nature of user education in protecting their data. For example, newer mechanisms like Access Control Lists (ACL) for Unix-like OSes, which can provide better data privacy, have failed to penetrate into regular usage[1].

In this paper, we discuss the current state of data privacy in Unix-like operating systems. Multi-user operating systems like Unix, Linux allow users to share their data and in order to control access, POSIX-standard [2] file/directory permissions are used. Each file and directory is associated with three kinds of permissions: *read (r)*, *write (w)* and *execute (x)*. This set of permissions is defined for three kinds of users: (1) *owner* - the owner of the file, (2) *group* - users with the same group as the owner, and (3) *others* - every other user.

Users maintain their data privacy using these file permissions. For example, users can decide to give no permissions to *group* and *others* for certain data like email, thus keeping it absolutely private, or can provide certain access rights to the *group* for appropriate sharing of data (can be considered as *private* from *other* users). Another popular technique is the use of *execute-only (x-only)* permissions for the home directory. This prevents users from reading the content of the directory (the files/subdirectories it contains), though authorized users, who know the names of the lower level directories, can access that data.

However, inappropriate usage of these permissions, partly due to lack of convenient support from OSes, can lead to *private* data to be accessible by unauthorized users. For example, in our study we found many users without appropriate permissions for their mailboxes, which could allow other users to read them. In addition, usage of popular directory names allowed scanners to penetrate *x-only* permissions and led to access to private data. Since it is believed the nearly 80 percent of information security breaches and resulting losses originate from **inside** an organization [1], we contend

[1] Latest Linux GUIs do not let users set ACLs, for example.

that it is absolutely essential to plug this hole.

Our main contributions in this paper are:

1. We identify issues that cause privacy compromise and discuss privacy-enhancing features that an OS should provide. We believe this paper to serve as an important tool for user-education in understanding this problem.

2. We present results from three multi-user installations measuring the amount of private data accessible to unauthorized users[2].

3. We discuss a number of solutions at user, OS and filesystem levels that can provide much greater privacy support and discuss their pros and cons.

The rest of the paper is organized as follows. We describe in detail the issue of data privacy in operating systems in Section-2. We present and analyze our measured privacy results at deployed user installations in Section-3. Later, in Section-4 we discuss a number of solutions.

2 Privacy: Expectation and Reality

In this section we discuss how users try to protect their data and how it can be compromised. We begin with the description of what we consider as *private* in this paper.

2.1 What is Private?

An important part of this study is the definition of the term *private*, i.e. the question - what kind of user data is considered private? This is a tough question to answer since any data with permissions allowing access to other users can be readily deemed public, even though it is due to improper settings by the data owner. Thus, it is important to reach a reasonable definition of *private data*. In this paper, we consider the following two kinds of data to be private:

- All email is considered private. This is consistent with current social norms.

- Any data contained in a directory which has an *x-only* parent or ancestor in the directory path.

The second assumption merits further justification. As described earlier, a *x-only* directory is one that has execute-only permissions for group or others. As an example, consider the home directory */home/aameek* with full permissions to the owner and execute only permissions to the group and others. Such a directory would be listed as *drwx−−x−−x*[3] by the *ls -l* command. The semantics of this permission set dictate that any user other than the owner cannot list the contents of the directory, though he/she can traverse the subdirectories by executing a *change-directory (cd)* to that subdirectory (using its name). For example, the *"ls -l /home/aameek"* would return a permission-denied error, while a *"cd /home/aameek/research"* would work and its contents would be visible (if the subdirectory *"research"* has read permissions).

We consider any data contained in these subdirectories to be *private*, since only authorized users who know that a directory named *"research"* exists are **supposed** to access that data. The authorization is usually provided to users by telling them the subdirectory names using out-of-band mechanisms like email or other personal modes of communication.

While it can be argued that not every subdirectory under an *x-only* parent is meant to be private (for example, a directory named *public*), we believe our definition to be a practical one. Other than requiring user participation in auditing and declaring their accessible private information, there seems to be not many practical ways of measuring private content. Also, if we look at the semantic nature of the permission sets and the fact that a user never broadcasts the details of the subdirectories he/she does not consider private under

[2]It is worthwhile to mention that the scanning techniques used for measurement, are *regular* Unix commands and do not use any technical exploits (hacks).

[3]This permission set is also called 711.

an *x-only* parent, it can be concluded that an unauthorized user *should* not be able to access that data.

2.2 Privacy Compromise

In a multi-user OS installation, the privacy compromise occurs due to many users having generic names for the subdirectories. For example, names like *research*, *personal*, *home*, *work*[4] are common as first level subdirectories in a research campus installation. Therefore, an unauthorized user can try and **guess** these names for an *x-only* home directory and access private information.

This problem is similar to the old problem of unauthorized users guessing user passwords, though more fundamental in nature. It is due to the fact that passwords can be enforced to be tough-to-remember cryptic clues, but a file/directory name is supposed to convey its contents and thus fundamentally meant to be semantic in nature. Also, there can be thousands of files for a single password (account), making the option of naming files in unreasonable ways, practically impossible.

To access private content, an automated scanner can be developed that guesses directory names. The guessing can be done in multiple ways:

- **Static Lists:** A static list of popular directory names is used. For example, names like *research*, *home*, *work* etc.
- **Global Lists:** The scanner looks at user directories with read permissions and compiles a name-frequency list and scans *x-only* home directories for those names.
- **Adaptive Lists:** The scanner looks at users in the same user group for the frequency lists. Another adaptive mechanism is to look at users with same higher level directories[5].
- **History Lists:** For each *x-only* user, the scanner reads its history files like *.history*, *.bash_history* for any commands that lists out directory names. As discussed later in Section-3, information leaked through history files was fairly significant.

This form of *attack* can also be thought of as *automated social engineering*. Social engineering is referred to the process of exploiting the human element in mounting attacks. For example, a renowned hacker Kevin Mitnick used to befriend system administrators and attempt to get account passwords from them. Our privacy attack can be similarly mounted manually by trying to get subdirectory names from a careless user. The scanning process described above automates this process by looking at information readable from a chunk of similar users (those with read permissions).

3 Measurement Study

To measure the amount of private data accessible to unauthorized users, we developed a prototype scanner and ran it at three geographically different organizations across the country. Users at these installations are divided into three categories:

1. *Read-Users* The users that have read permissions on their home directories, thus allowing listing of subdirectories. Such users are used for compiling frequency lists and scanned only for readable emails.
2. *No-Users* The users that have no permissions on their home directories for other users. Such users are not used in the scanning process.
3. *X-only-Users* Users that have *x-only* permissions on their home directories. This is the most interesting set of users for our study.

The characteristics of the three installations were as shown in Figure-1[6]. The unusually low number of *x-only* user homes in Org-3 is due to the fact of it being an installation of non-primary

[4] along with their abbreviated forms and/or upper case initials like *Research*, *pvt* etc.

[5] In many settings, users entering an organization in the same year are based in a single higher level directory

[6] Names of the organization have been withheld.

Org.	OS	# Users	# Read-Users	# No-Users	# *X-only*-Users
Org-1	HP-UX	836	198	54	573
Org-2	HP-UX	768	136	39	593
Org-3	Solaris	2000	443	1491	66

Figure 1: Installation Characteristics

accounts for users with default permissions and umask set to 700.

The scanner used all guessing mechanisms for *x-only* user homes described earlier in Section-2.2. Along with an input of a static list of directories, it automatically compiled a global list, an adaptive usergroup based list and also an adaptive higher-level-directory list. In addition, history files *.history* and *.bash_history* were used to extract directory names. Symbolic links outside the user home were not followed and duplicates in the lists were removed to prevent double-counting. Guessing was done only for first level subdirectories For every successful guess of a subdirectory name, a crawler-like operation is initiated which looks at all the data in that subdirectory and calculates various statistics.

We measured the number of files and size of data which were private but accessible by unauthorized users. A static list of email directories and mailboxes names was also used to measure the number of email folders, number of emails[7] and size of emails readable by unauthorized users. In addition, we also calculated the number of times the word "password" appeared in these emails. The results are presented in Figure-2. Again, the low numbers for Org-3 are explained by the much lesser number of X-only-Users. In addition, the low number of Read-Users makes it tougher to compile effective frequency lists.

History files also had a significant contribution into leaking this private information. Figure-3 shows the number of occurrences of readable history files, the number of private files it led to be accessed and the size of data it leaked, for Org-1 and Org-2. Org-3 did not leak any information due to history files.

A number of other statistics like the kind of user shells most used (which influences history files), frequency of directory names, evaluation per each guessing method (static/global/adaptive) were also collected. We omit those due to space constraints.

4 Solutions

In this section, we present a number of potential solutions to this privacy compromise issue. The solutions are broadly divided into two categories:

1. *Administrative and Educational*: Solutions like setting default user permissions and umask to 700, educating users to newer access control mechanisms like ACLs (setfacl, getfacl commands) and enforcing their usage. In addition, an auditing tool can be developed which informs an administrator of the levels of privacy compromise and can also list main victims. The code for the auditing tool is similar to the scanner developed for our study. Another form of auditing is to allow data owners access to information of accesses to their data by other users. However, this raises an important ethical issue of whether such information is violation of privacy of the user accessing that data.

2. *Technical*: Solutions that enhance OS-support for data privacy by making changes to the OS code, for example, changing access control mechanisms.

In this paper we discuss only technical solutions for its suitability to the publication forum.

[7]Number of emails was measure by searching for "Subject:" at the beginning of a line

Org.	# Files	Data Size	#Email Folders	#Emails	Email Size	# Pwd
Org-1	983086	82 GB	2509	315919	4.2 GB	6352
Org-2	364932	25 GB	505	38206	120 MB	237
Org-3	1288	146 MB	87	559	1.7 MB	11

Figure 2: Accessible Private Data

Org.	# Readable History Files	# Private Files Leaked	Private Data Size
Org-1	253	561254	35 GB
Org-2	237	155826	14 GB

Figure 3: Private Data Leaked by History Files

As described earlier, we consider two kinds of data to be private - emails and data hidden by x-only parents. Emails can be protected by enforcing appropriate permissions to the default mail inboxes (for example, in */var/mail* directories). Using mechanisms similar to our scanner, user mailboxes can be identified and permissions corrected. A simple multi-threaded[8] daemon process can efficiently perform this operation periodically.

For data hidden under *x-only* parents, we propose a more sophisticated access control mechanism. Note that the scanning attack is successful only due to the ability of an attacker to **guess** a subdirectory name. The aim of our solution is to take away that ability, while protecting the semantics of names and convenience of data owners.

Our proposed solution can be implemented at various level of OS stack - user level, Virtual File System (VFS) level or a File System (FS) level. The common feature at all levels is that it distinguishes between a file/directory name seen by the data owner and by other users. The name seen by data owner is a simple semantics based named like "research", while the name seen by other users is the actual name appended with a random string like "research-1453d3dfg45gy". Such a random string can be generated doing a secure hash on a key appended with the actual name, i.e. $Hash(Key||research)$ for our example. This mechanism of generating random strings helps in sharing of data (only need to authorize users by telling a key). We *briefly* discuss the solution at each level below:

– *User Shell Level*: The actual name of the file/directory is the random name. However, whenever a data owner executes a command that interacts with the file system (listing directories, cd to a directory), he/she uses a semantics based name which is mapped to the actual name by the user shell. Such commands can be identified by using mechanisms similar to Bourne shell's name completion feature. Clearly this is a rudimentary solution and only works for primitive commands (Name completion feature does not work while running scripts).

– *VFS Level*: Many OSes like Unix, Linux, Solaris have a Virtual File System (VFS) interface layer which defines how a file is opened after an *open* command is issued. This function can be modified by first doing a check of whether a data owner is executing the open command and then doing mapping of the semantics based name to the actual name in that case. For any other user, the name of the file is the random string and thus, an unauthorized user would need to guess the entire name (which is unlikely and can be made practically impossible by increasing the length of the key). Users can be authorized by informing them of the name and a key (using similar out-of-band mechanisms).

– *FS Level*: Instead of enforcing a new access control mechanism over all FS supported by the OS,

[8] for parallelizing across multiple users

the above feature can be pushed down to the FS level, thus letting users mount their home directories only on such private FS.

5 Conclusions

In this paper, we have raised the issue of data privacy support in popular operating systems. We identified various privacy compromising features and proposed a range of solutions. Based on a measurement study at three different installations, we also demonstrated the immediate need for enhancing OS support for data privacy.

References

[1] http://nsi.org/SSWebSite/TheService.html.

[2] http://www.pasc.org.

[3] A. Juels and J. Brainard. Soft blocking: Flexible blocker tags on the cheap. In *Workshop on Privacy in Electronic Society*, 2004.

[4] A. Juels, R. L. Rivest, and M. Szydlo. The blocker tag: Selective blocking of rfid tags for consumer privacy. In *ACM CCS*, 2003.

[5] A. Singh and L. Liu. Agyaat: Providing mutually anonymous services over structured p2p networks. In *Technical Report, CERCS*, 2004.