Privacy Analysis for Data Sharing in *nix Systems

Aameek Singh **Ling Liu** **Mustaque Ahamad**

College of Computing, Georgia Institute of Technology
{aameek, lingliu, mustaq}@cc.gatech.edu

Abstract

Linux and its various flavors (together called *nix) are growing in mainstream popularity and many enterprise infrastructures now are based on *nix platforms. An important component of these systems is the ingrained multi-user support that lets users share data with each other. In this paper, we analyze *nix systems and identify an urgent need for better privacy support in their data sharing mechanisms. In one of our studies it was possible to access over 84 GB of private data at one organization of 836 users, including over 300,000 emails and 579 passwords to financial and other private services websites. The most surprising aspect was the extremely low level of sophistication of the attack. The attack uses no technical vulnerabilities, rather inadequacies of *nix access control combined with user/application's privacy-indifferent behavior.

1 Introduction

With increasing popularity of open source operating systems like Linux, many enterprises are opting to set up their intranets using such *nix systems. The market research firm IDC expects linux server sales to hit US $9.1 billion by 2008 and that linux servers will comprise 25.7% of total server units shipped in 2008. Consequently, many big companies like IBM, Dell actively support the open source *nix movement.

One of the important features of *nix systems is their ingrained multi-user support. These operating systems are designed for simultaneous multiple users and provide seamless mechanisms to share data between different users. For example, user *alice* can set up appropriate access "permissions" on the data she wants to share with her group *students* by executing a simple `chmod` command [7]. These access permissions are dictated by the *nix *access control model*.

In this paper, we take a critical look at the *nix access control model. Our objective is to analyze the **privacy** support in its data sharing mechanisms. For example, how does the system assist a user to share data only with desired users and prevent private information from being leaked to unauthorized users? In order to successfully do this, we also need to look at the **convenience** of using *nix data sharing mechanisms in typical situations. This is due to the fact that lack of convenience typically leads to users compromising (intentionally or mistakenly) their security requirements to conveniently fit the specifications of the underlying access control model.

Please note that we use the seemingly oxymoronic phrase "*private sharing*" to indicate the desire of sharing data only with a select set of authorized users.

As part of our analysis, we looked at how users use the access control for their data sharing needs in practice. We conducted experiments at two *nix installations of a few hundred computer-literate users each. Surprisingly, we found that large chunks of private data was accessible to unauthorized users. In many cases, the user's definition of an "authorized user" does not match the underlying system's definition, that leads to such a breach. This observation is best exemplified in the following scenario. Many users attempt to privately share data by using execute-only permissions for their home directories. This prevents other users from listing the contents of the directory, but any user who knows the name of a subdirectory can `cd` into it. The data owner *authorizes* some users by explicitly giving them the names of the subdirectory through out-of-band mechanisms like personal communication or email. However, this authorization is not the same as the system's authorization. From the underlying system perspective, it is assumed that any user who issues the command with the right directory name is authorized. Thus, users who simply guess the subdirectory name can also access the data. Along with that, setting execute-only permissions on the home directory to share *one* subdirectory, puts all other subdirectories (sibling to the shared directory) also at risk, which if not protected appropriately, can be accessed.

We were able to effectively exploit these shortcomings in our studies. At one organization of 836 users, over 84 GB of data was accessible, including more than 300,000 emails and 579 passwords to financial websites like *bankofamerica.com* and other private websites like medical insurance records. Importantly, the attack does not always need to *guess* directory names, but can find actual names from unprotected command history files (*.history, .bash_history*) or standard application directory names (*.mozilla*). The reason for this surprisingly large privacy breach without exploiting technical vulnerabilities like buffer overflows or gaining elevated privileges, is the combination of lack of system support and user or even applications' privacy-indifferent behavior either mistakenly or for lack of anything better. Also, as we discuss later in Section-2, even for an extremely privacy-conscious user with all available tools, it is tough to protect private data in many situations.

The attack described in this paper is a form of an **insider** attack, in which the attacker is inside the organization. The attacker could be a disgruntled employee, contractor or simply a curious employee trying to access the salaries chart in the boss's home directory. According to a recent study by the US Secret Service and CERT [6], such attacks are on a rise with 29% of the surveyed companies reporting[1] having experienced an insider attack in the past year [4]. Also, in complete congruence to our attack, the report finds that:

"`Most incidents were not technically sophisticated or complex ...`"

The rest of the paper is organized as follows. In Section-2, we discuss multiple *nix issues, including usability, that lead to privacy breaches. In Section-3, we present our case studies at two *nix installations which demonstrate these breaches. We briefly describe possible enhancements to these breaches in Section-4. We conclude in Section-5.

2 Data Privacy: Vulnerability Analysis

In this section, we discuss various privacy breaches that occur in *nix systems. The discussion below primarily explores the popular {*owner, group, others*} *nix paradigm. The advanced mechanisms like ACLs [5] enhance privacy in only a few cases and we will demonstrate a clear need of a new, more complete solution.

2.1 Selective Data Sharing

The first privacy breach occurs due to the need to "*selectively*" share data. The selectivity can be of two kinds:

- **Data Selectivity**: Data selectivity is when a user wants to share only a few (say one) of the subdirectories in the home directory. So, an authorized user is allowed to access only the shared subdirectory, but not any of the sibling directories. In order to do this correctly, the owner needs to follow two steps - (a) set appropriate permissions to the shared subdirectory (at least execute permissions on the entire path to the subdirectory and the sharing permissions on the subdirectory), and (b) *remove permissions from the sibling subdirectories*. The second step is unintuitive, since the user needs to act on objects that are not the focus of the transaction. Also, any new file being created needs to be protected.
- **User Selectivity**: In many situations, users need to share data with an adhoc set of users that do not belong to a single user group or are only a subset of a user group, and not the entire *others* set. In this situation, the permissions for *group* or *others* are not sufficient. Creating a new group requires administrative assistance which is not always feasible.

In order to do selective data sharing (a common example is to allow access to `public_html`), currently owners mostly use execute-only permissions on the home directories. The perception is that since users can not list the contents of the directory, they cannot go any further than traversing into the home directory unless they know the exact name of the subdirectory. Now, the owner can authorize desired users by giving them the names of the appropriate subdirectories. Those *authorized* users can traverse into the home directory and then use the subdirectory name to `cd` into it (without having to list the contents of the home directory). From data selectivity perspective, it is assumed that they cannot access the rest of the contents and from user selectivity perspective, unauthorized users cannot access any contents.

However, the underlying system cannot distinguish between such authorized or unauthorized users. Any user who can **guess** the subdirectory name can actually access the data. For an attacker inside the organization, this is not a herculean task. For example, for a computer science graduate school, it is highly likely that users will have directories named *research*, *classes* or *thesis*. An easy way of creating such a list of names is by collecting names from users that actually have *read* permissions on the home directories. Within the context of a single organization, or in general human psychology, it is likely that many users have similar directory names.

Secondly, many times directory names do not need to be guessed at all. The names can be extracted from **history** files (like *.history* or *.bash_history*), that contain the commands last executed by the owner, like `cd`, which will include real directory names. In fact, in our experiments we found around 20-30% of all users had readable history files and around 40% of the total leaked data was obtained from the directory names extracted from these history files.

Thirdly, it is not always user created directories that leak information. Many **applications** use standard directory names and fail to protect critical information. For example, the famous Mozilla web browser [2] stores the profile directory in `~/.mozilla` and had that directory world-readable [8] in many cases, till as late as 2003. Many *nix installations with the browser installed before that have this vulnerability and we were able to obtain around 575 password to financial and private websites (because users saved passwords without encrypting them). In addition, their browser caches, bookmarks, cookies and histories were also available. The browser Opera [3] also has a similar vulnerability, though to a lesser extent. While it can be argued that it is the responsibility of application developers to ensure that this does not happen, we believe that the underlying system can assist users and applications in a more proactive manner.

The POSIX ACLs [5], if used help in achieving only user selectivity. They do not address the data selectivity requirements or prevent leaking of application data.

2.2 Metadata Privacy

So far, we have only talked about the privacy breach for file *data*. However, there are many situations in which users are interested in protecting even the metadata of the files. The metadata contains information like ownership, access time, updation time, creation time and file sizes. There are scenarios where a user might obtain confidential information by just looking at the metadata. For example, an employee might be interested in knowing how big is his annual review letter or did the boss update it after the argument he had with her?

The *nix access control does not provide good metadata privacy. Even if users only have execute permissions on a directory, as long as they can guess the name of the contained file, its metadata can be accessed even if the file itself does not have any read, write or execute permissions on it. Thus, if a user has to share even a single file/directory within the home directory (thus, requiring atleast execute permissions), all other files contained in the home directory lose their metadata privacy if their names are known or can be guessed.

2.3 Data Sharing Convenience

User convenience is an important feature of an access control implementation. If users find it tough to implement their security requirements, they are likely to compromise them to easily fit the underlying access control model. This can be seen as one of the reasons why encryption file systems are not in widespread use, even though they guarantee maximum security.

From our analysis of the *nix access control, along with some of the issues discussed earlier, we found the following two data sharing scenarios in which there is no convenient support for privacy.

- **Sharing a Deep-Rooted Directory**: For a user to share a directory that is multiple levels in depth from the home directory, there needs to be at least execute permissions on all directories in the path. This in itself (a) leaks the path information, (b) puts sibling directories at risk and (c) leaks metadata information for sibling directories. In order to prevent this, since most operating systems do not allow hard links to directories anymore, a user would have to create a new copy of the data. And since users are more careless with permissions for deep rooted directories (they protect a higher level directory and that automatically protects children directories), a copy of such a directory could have privacy-compromising permissions.
- **Representation of Shared Data**: In many circumstances the way one user represents data might not be the most suitable way for another user. For example, while an employee might keep his resume in a directory named `job-search`, it is clearly not the most apt name to share with his boss. The employee might want her to see the directory simply as `CV`. Changing the name to meet the needs of other users is not an ideal solution. This again shows the lack of adequate system support for private and convenient data sharing.

It is important to recognize that even an extremely privacy-conscious user can not protect data at all times. Exhaustive efforts to maintain correct permissions for all user and **application created** data will still be insufficient to protect metadata or allow private sharing of deep rooted directories with user-specific representation.

3 Case Studies

As part of our study, we conducted experiments at two geographically and organizationally distinct *nix installations. Users at both installations (CS graduate schools) are highly computer literate and can be expected to be familiar with all available access control tools. For our analysis, we consider the following data to be private:

- All user emails are considered private.
- All data under an execute-only home directory is considered private.
- Browser profile data (saved passwords, caches, browsing history, cookies) is considered private.

The second assumption above merits further justification. It can be argued that not every subdirectory under an execute-only home directory is meant to be private (for example, a directory named *public*). However, the semantics of the execute-only permission set dictate that any user other than the owner cannot list the contents of the directory and since the owner never *broadcasts* the names of the shared directories, an unauthorized user *should not* be able to access that data. And since we do not include in our measurements any obviously-private data from home directories of users with *read* permissions (for example, world-readable directories named `personal`, or `private`), we believe the two effects to approximately cancel out.

3.1 Modus Operandi

Next, we describe the design of our attack[2] that scans user directories and measures the amount of private data accessible to unauthorized users.
The attack works in multiple phases. The first step is to obtain directory name lists which can be tried against users with execute-only home directories. Three strategies are used to obtain these lists:

- *Static Lists*: These are manually entered names of directories likely to be found in the context of the organizations - CS graduate schools. For example, "*research*", "*papers*", "*private*" and their variants in case ("*Research*") or abbreviations ("*pvt*").
- *Global Lists*: These lists are generated by obtaining the directory names from home directories of users that have *read* permissions.
- *History Lists*: These are user specific lists generated by parsing users' history files, if readable. We used a simple mechanism, parsing only `cd` commands with directory names. It is possible to do more by parsing text editor commands (like `vim`) or copy/move commands.

In the next step the tool starts a multi-threaded scanning operation that attempts to scan each user directory. For users with **no** permissions, no scanning is possible. For users with **read** permissions, as discussed earlier, since there is no precise way of guessing which data would be private, we only measure email and browser profile statistics. Finally, for users with **execute-only** permissions, along with email and browser statistics, we also attempt to extract data using the directory name lists prepared in the first step.

Evaluating Email Statistics
This is done by attempting to read data from standard mailbox names - "*mail*", "*Mail*", "*mbox*" in the home directory and the mail inboxes in */var/mail/userName*. A `grep` [7] like tool is used to measure (a) number of readable emails, (b) number of times the word "*password*" or its variants appeared in the emails.

Evaluating Execute-only Data Statistics
For users with execute-only permissions on the home directory, the scanner uses the combination of static, global and the user's history lists to access possible subdirectories. Double counting is avoided by ensuring that a name appearing in more than one list is accounted for only once and by not traversing any symbolic links. While scanning the files, counts are obtained for the total number of files and the total size of the data that could be accessed.

Evaluating Browser Statistics
The mozilla browser [2] stores user profiles in the `~/.mozilla` directory. This directory used to be world-readable till as late as 2003 when the bug was corrected [8]. Within that profile directory, there are subdirectories for each profile that has been used by that user. Within the profile directory, there is another directory with a randomized name (for security against remote attacks) ending in "*.slt*" [1]. Within this directory, the following files exist and (with this bug) were readable:

- **Password Database:** A file with the name of type "*12345678.s*" containing user logins and passwords saved by mozilla. Since many users do not encrypt their passwords, mozilla stores the passwords in a base-64 encoding (indicated by the line starting with a ~ in the passwords file), which can potentially be trivially decoded to get plaintext passwords.
- **Cookies:** The `cookies.txt` file contains all browser cookies. Many websites including popular email services like Gmail, Hotmail allow users to automatically login by keeping such cookies. Hijacking this cookie can allow a malicious user to login into these accounts. For many other cookies related attacks, see [9].
- **Cache:** This is a subdirectory that contains the cached web pages visited by the user.
- **History Database:** Web surfing history, which many sophisticated viruses and spyware invest resources to collect, are also readable.

3.2 Results

The complete characteristics of the two organizations are shown in Table-1. At both the organizations, a significant number of users (68% and 77%) used execute-only permissions on their home directories.

Org.	# Users	# ReadX	# NoPerms	# X-only
Org-1	836	198	54	573
Org-2	768	136	39	593

Table 1: Case Study Organization Characteristics. # ReadX is the number of users with read and execute permissions to their home directories, # NoPerms are users with no permissions and # X-only are the users with only execute permissions

Org.	# Hit Users	# Hits	# Files	Data Size
Org-1	462	2409	983086	82 GB
Org-2	380	911	364932	25 GB

Table 2: Data extracted from X-only home directory permissions. # Hit Users is the number of users that leaked private information. # Hits is the total number of directory name hits against all X-only users. # Files is the number of leaked files and Data-Size is the total size of those files

Table-2 lists the amount of data extracted from execute-only home directories at Org-1 and 2.

As can be seen, a large fraction of users indeed leaked private information - 55% and 49% of total users respectively. Recall that we do not extract any data from users with *read* permissions on their home directories; so a more useful number is the fraction of X-only users that revealed private information. That number is 80% and 64% respectively. Also, on an average, 2127 files and 177 MB of data is leaked in the first organization for each X-only user and 960 files and 65 MB of data is leaked in the second organization. A partial reason for the lower numbers in the second organization could be the fewer number of users with *read* permissions, which would have impacted the global name lists creation. Overall, we believe this to be a very significant privacy breach.

As mentioned earlier, many times the names of the subdirectories do not need to be guessed and can be obtained from the history files in the user home directories. Table-3 lists the success rate of the attack in exploiting history files. As it shows, around 40% of X-only users had readable history files which led to 40-50% of total leaked data in size.

Email Statistics

Table-4 presents the results of the email data extracted from users in both organizations. Recall that this data is obtained for both X-only users and the users with *read* permissions on their home directories.

As can be seen, a large number of emails are accessible to unauthorized users (especially at Org-1). Also, the number of times the word "password" or its variants appear in these emails is alarming. Even though we understand that some of these occurrences might not be accompanied by actual passwords, by personal experience, distributing passwords via emails is by no means an uncommon event.

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

Table 3: Exploiting History Files. # History Hits is the number of users with readable history files. # Files is the number of private files leaked due to directory names obtained from history files and Data-Size is the size of the leaked data

Org.	# Folders	# Emails	Size	# *Password*
Org-1	2509	315919	4.2 GB	6352
Org-2	505	38206	120 MB	237

Table 4: Email Statistics. # Folders is the number of leaked email folders. # Emails is the total number of leaked emails. Size is the size of leaked data and # *Password* is the number of times the word "*password*" or its variants appeared in the emails

Browser Statistics

The second organization did not have the mozilla vulnerability since they had a more recent version of the browser installed, by which time the bug had been corrected. So the results shown in Table-5 have been obtained only from the first organization. Looking at the results, the amount of accessible private information is enormous. Figure-1 contains a sample of the websites that had their passwords extractable and clearly most of these websites are extremely sensitive and a privacy breach of this sort is completely unacceptable.

Note that some obtained passwords were for accounts in other institutions and a few of them are likely to be *nix systems. Thus, it is conceivable that this password

# Users with accessible `.mozilla`	311 (54%)
# Users with readable password DB	149 (26%)
# Passwords Retrievable	579
# Users with readable cookies DB	207 (36%)
# Cookies Retrievable	19456
# Users with accessible caches	233 (40%)
# Cached Entries	20907
# Users with readable browsing histories	256 (44%)
# URLs in History	130,503

Table 5: Browser Statistics at Org-1

Financial Websites	Personal Websites
www.paypal.com	adultfriendfinder.com
www.ameritrade.com	www.hthstudents.com
www.bankofamerica.com	www.icers911.org
Email Accounts	**Other Institutions**
mail.lycos.com	cvpr.cs.toronto.edu
my.screenname.aol.com	e8.cvl.iis.u-tokyo.ac.jp
webmail.bellsouth.net	systems.cs.colorado.edu

Figure 1: Sample accounts with retrievable passwords

extraction can be used to **expand to other *nix installations** and thus be much more severe in scope than a single installation.

3.3 Attack Severity

It is important to highlight the severity of this attack:

- **Low Technical Sophistication**: The attack is extremely low-tech; the commands used in a manual attack would be `cd`, `ls` and such. This aspect makes the threat significantly more dangerous than most other vulnerabilities.
- **Low Detection Possibility**: In absence of extensive logging, as typically is the case, this attack has a very low probability of detection and even if detected by means like modified last access time, the attacker can not be identified.
- **No Quick Fix**: Unlike most other security vulnerabilities, this attack uses a **design** shortcoming combined with user/application carelessness and no patches would correct this problem overnight.
- **High Success Rate**: The attack had a high success rate at installations where most users are computer literate. With increasing mainstream penetration of *nix systems, ordinary users cannot be expected to fully understand the vulnerabilities. This makes this attack a very potent threat.

4 Privacy Enhancements

The results presented in the previous section clearly establish the fact that there needs to be much better privacy protection in *nix installations. We are currently exploring two enhancement approaches. The first solution is a Privacy Auditing Tool that monitors the privacy health of an organization and can alert users/administrators of potential threats. Analogous to the enterprise security applications that monitor virus and other malicious activity, the privacy auditing tool periodically monitors user home directories and identifies potential data exposures. The second approach is a more proactive approach that modifies the file system hierarchy by virtualizing it differently for different users. Ensuring the most private user data stays only in the owner's view, many forms of inadvertant exposure can be avoided.

5 Conclusions

In this paper, we critically analyzed the *nix access control model for privacy support in its data sharing mechanisms. We identified a number of design inadequacies that, combined with user or application's privacy-indifferent behavior, lead to privacy breaches. We tested two *nix installations of a few hundred users and found that a massive amount of private data is inadequately protected including emails and actual passwords to financial and other sensitive websites. We briefly proposed two enhancement approaches that can improve data condentiality in *nix systems and we continue to develop these approaches for our future work.

References

[1] Mozilla Contents http://www.holgermetzger.de/pdl.html.

[2] Mozilla web browser http://www.mozilla.org.

[3] Opera web browser http://www.opera.com.

[4] D. Cappelli and Michelle Keeney. Insider threat: Real data on a real problem. *USSS/CERT Insider Threat Study, 2004.*

[5] A. Grunbacher and A. Nuremberg. POSIX Access Control Lists on Linux. *http://www.suse.de/ agruen/acl/linux-acls/online.*

[6] Carnegie Mellon Software Engineering Institute. CERT.

[7] Linux Manual Pages. *man command-name.*

[8] Mozilla Bug Report. Bug 59557. *https://bugzilla.mozilla.org/show%5Fbug.cgi?id=59557.*

[9] Emil Sit and Kevin Fu. Web Cookies: Not Just a Privacy Risk. *Communications of the ACM*, 44(9), 2001.

[10] United States Secret Service and CERT Coordination Center. E-Crime Watch Survey. *2004.*

Notes

[1] It is believed that such attacks are usually much under-reported for lack of concrete evidence or fear of negative publicity [10]

[2] Due to the sensitive nature of the task (measuring *private* content) we took precautions to ensure that our study does *not* violate user privacy by anonymizing users, randomizing scan orders and only collecting aggregates