
Privacy analysis and enhancements for data sharing in *nix systems

Aameek Singh*

Storage Systems
IBM Almaden Research Center
650 Harry Road, San Jose, CA – 95120, USA
E-mail: aameek.singh@us.ibm.com
*Corresponding author

Ling Liu and Mustaque Ahamad

College of Computing
Georgia Institute of Technology
801 Atlantic Drive, Atlanta, GA – 30332, USA
E-mail: lingliu@cc.gatech.edu
E-mail: mustaq@cc.gatech.edu

Abstract: In this paper, we analyse the data sharing mechanisms of *nix systems and identify an immediate need for better privacy support. For example, using a simple insider attack we were able to access over 84 GB of private data at one organisation of 825 users, including 300 000 e-mails and 579 passwords to financial and other private services websites, without exploiting any technical vulnerability.

We present two solutions to address this problem:

- 1 an administrative auditing tool which can alert administrators and users when their private data is at risk
- 2 a new View Based Access Control (VBAC) mechanism which provides stronger and yet convenient privacy support.

We also describe a proof-of-concept filesystem-based implementation and performance analysis of VBAC. Our evaluations with three well-known filesystem benchmarks show little overhead of using VBAC.

Keywords: unix privacy; private data sharing; access control; View Based Access Control; VBAC.

Reference to this paper should be made as follows: Singh, A., Liu, L. and Ahamad, M. (2008) 'Privacy analysis and enhancements for data sharing in *nix systems', *Int. J. Information and Computer Security*, Vol. 2, No. 4, pp.376–410.

Biographical notes: Aameek Singh is a Research Staff Member in the Storage Systems group at IBM Almaden Research Center. Singh graduated with a PhD from the College of Computing at Georgia Institute of Technology in 2007. His research interests are in the area of storage systems and data intensive distributed systems. His PhD dissertation proposed new access control and storage management techniques for enterprise storage-as-a-service environments. Singh has over 20 refereed publications in international journals and conferences and is a co-inventor on more than ten patent applications. He was a recipient of the IBM PhD fellowship for 2005–2006 and 2006–2007.

Ling Liu is an Associate Professor in the College of Computing at Georgia Tech. There, she directs the research programme in Distributed Data Intensive Systems Lab examining research issues and technical challenges in building distributed computing systems that can grow without limits. She has published over 200 international journal and conference articles. She is an internationally recognised expert in the areas of Database Systems, Distributed Systems, Internet Systems, and Web Services. She is currently on the editorial board of several top international journals, including *IEEE TKDE*, *IEEE Transaction on Service Computing (TSC)* and *VLDB Journal*.

Mustaque Ahamad is a Professor in the School of Computer Science at the College of Computing at Georgia Tech and Director of the Georgia Tech Information Security Center, a National Security Agency (NSA) Center of Excellence in Information Assurance Education. As Director of an interdisciplinary centre uniquely focused on usable security, Ahamad is responsible for the integration of research that empowers everyday users to better protect themselves and take charge of their online security and privacy. Ahamad is a leader in developing effective solutions to safeguard personal digital information against current cyber security threats, such as phishing, spoofing, and identity theft.

1 Introduction

Unix, Linux and its various other open-source flavours (together called *nix) are steadily growing in mainstream popularity and many enterprises now opt to set up their intranets using such *nix systems. One of the important features of these *nix systems is their ingrained multi-user support. In contrast to common single-user PC systems, these systems allow simultaneous multiple users and provide seamless mechanisms to share data between different users. For example, a user *alice* can set appropriate access ‘permissions’ on the data she wants to share with her group *students* by executing a simple `chmod` command (Linux Manual pages, 2008).

In this paper, we critically analyse the data sharing mechanisms of *nix systems. Access to shared data in these systems is dictated by the *nix *access control model*, which typically follows the original UNIX access model (Ritchie and Thompson, 1974). We aim to analyse 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 unauthorised users? As part of the analysis, we also need to look at the *convenience* of using these data sharing mechanisms. This is important since lack of convenience typically compels users to compromise, intentionally or not, their security requirements to conveniently fit the specifications of the underlying access control model. Please note that we use the phrase ‘*private sharing*’ to indicate the desire of sharing data only with a select set of authorised users.

As part of our study, we analysed how users use access control for their data sharing needs in practice. We conducted experiments at two *nix organisations of many hundred computer-literate users each. Surprisingly, we found that large amount of private data was accessible to unauthorised users. In many cases, the user’s definition of an ‘authorised 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 (complete *nix access permissions are discussed later in Section 2). This prevents other users from listing the contents of the directory, but they can `cd` into it and any user who *knows* the name of a subfile/directory can access it with appropriate permissions. The data owner *authorises* some users by explicitly giving them the names of the subfile/directory through out-of-band mechanisms like personal communication or e-mail. However, this authorisation is not the same as the system's authorisation. From the underlying system perspective, it is assumed that any user who issues the command with the right file/directory name is authorised. Thus, users who simply guess the subfile/directory 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 organisation of 825 users, over 84 GB of data was accessible, including more than 300 000 e-mails 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 behaviour 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 organisation. 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 (Carnegie Mellon Software Engineering Institute, 2008), such attacks are on a rise with 29% of the surveyed companies reporting having experienced an insider attack in the past year (Cappelli and Keeney, 2008) (it is usually believed that such attacks are much under-reported for lack of concrete evidence or fear of negative publicity (United States Secret Service and CERT Coordination Center, 2008)). Also, in complete congruence to our attack, the report finds that:

“Most incidents were not technically sophisticated or complex – that is they typically involve exploitation of non-technical vulnerabilities.”

We propose two solutions to address this problem:

- 1 A Privacy Auditing Tool that analyses the '*privacy health*' of an enterprise. Such auditing can be combined with the periodic virus scans and other security audits regularly employed by enterprises.
- 2 A new access control model which provides stronger privacy protection and yet convenient data sharing mechanisms. The View-Based Access Control (VBAC) model allows data owners' to define views of their data that can be seen by other users, while protecting other data from unauthorised users. We provide a file system implementation of VBAC and show that the overheads incurred by it are minimal.

An interesting feature of our VBAC implementation is our highly *data* oriented focus. There has been a significant amount of work on *nix security which has been primarily focused on preventing *processes* from gaining elevated privileges or sandboxing *processes* from accessing certain data (Wagner, 1999; Jaeger and Prakash, 1994; Goldberg *et al.*, 1992; Linux Manual pages, 2008). In contrast, our approach focuses on private *data* that needs to be protected. Our implementation allows administrators to use our approach only for data that is considered private – for example, user home directories and thus, does not incur any overheads for other data in the system. To summarise, this paper makes four unique *contributions*:

- 1 *Privacy analysis of *nix systems*: To the best of our knowledge, this is the first work that evaluates the privacy characteristics of real multiuser *nix installations. We analyse the *nix access control from a privacy perspective and present results on how users share their data and how much private information can be accessed by unauthorised users in real *nix installations.
- 2 *Data sharing principles*: We identify a number of useful design principles that allow for private and convenient data sharing in multi-user environments. We also analyse existing access control tools like *nix permissions model, POSIX Access Control Lists (ACLs) (Grunbacher and Nuremberg, 2008), umask (Linux Manual pages, 2008) on these principles.
- 3 *Privacy enhancements*: We present two approaches that can enhance the privacy characteristics of *nix systems. The first approach is a conservative approach that only measures the privacy health of the system and alerts users of potential threats. The second approach is more proactive and utilises VBAC to provide safer data sharing. We also describe an implementation of VBAC and its performance analysis.
- 4 *User education*: The privacy analysis and data sharing principles presented in this paper help us in devising various privacy enhancement approaches. More importantly they bring into light a massive threat to user privacy in current systems, which can be exploited overnight by a few hundred lines of code. This work thus contributes directly to increasing user awareness and education in protecting private data.

The rest of the paper is organised as follows. In Section 2, we discuss various issues that lead to privacy breaches. We also present case studies at two *nix installations which demonstrate these breaches. In Section 3, we describe a number of useful privacy principles and analyse existing tools. Section 4 gives an overview of the two privacy enhancement approaches. Section 5 presents a detailed discussion of the VBAC access control model including its design goals, implementation and performance evaluation. We describe related work in Section 6 and conclude in Section 7.

2 Data privacy: vulnerability analysis

In this section, we discuss various scenarios that lead to privacy breaches and present the results of our case studies. This analysis will provide us insights into desirable privacy and convenience characteristics. We start off with a description of *nix access control, to establish a common reference model for subsequent discussions.

2.1 *nix access control

The *nix access control primarily follows from the original UNIX access control model (Ritchie and Thompson, 1974; Ritchie, 1978). In this discretionary access model, each file system object (file, directory, links) has an associated owner who controls the access to that object. This access can be granted to three kinds of users:

- 1 owner
- 2 group
- 3 others.

The *owner* is the object owner, the *group* is a set of users to which the owner belongs (for example, a user group for students, faculty) and *others* are all other users (all users except the owner and the group). Also, note that the *owner* permissions take precedence over *group*, that is, the owner will have access as dictated by the *owner* permission bits, even though he/she also belongs to the *group*. Further, the granted access is of three types:

1 Read

For a file, this means that a user can read a file. For a directory, this means that a user can list its contents using `ls` (Linux Manual pages, 2008). For links, the permissions are for the object that are pointed-to by the link and the link's permissions itself are not used. The read permission is represented by a 'r' or a numeric value of 4.

2 Write

For a file, the write permission allows a user to write to it. For a directory, it allows a user to create, remove or rename directory contents. For links, permissions are again for the pointed-to object. The write permission is represented by a 'w' or a numeric value of 2.

3 eXecute

For a file, the execute permissions allows running the file as a program (for example, a shell script or a compiled C program). For directories, it allows users to traverse that directory and if the directory contents have appropriate permissions, the user can then access those contents. For example, to access directory `grand-child` with path `dir/child/grand-child`, both `dir` and `child` need to have execute permissions. The execute permission is represented by a 'x' or a numeric value of 1.

The permissions bits are listed in the '*owner, group, others*' order. Table 1 shows an example directory with its 3×3 access control matrix. It will be listed as `rwX r-x - -x` or in numeric terms `751` ($7 = r(4) + w(2) + x(1)$ and so on). For that directory, the owner can list its contents, create, remove, or rename its children and can traverse into that directory. The users belonging to the group of the owner can list its contents and traverse into the directory but not modify its contents. The *other* users can only traverse into the directory and not list or modify its contents.

