Table of Contents

Research Articles

1 Global Analysis of Security and Trust Perceptions in Web Design for E-Commerce
S. Srinivasan, Texas A&M International University, USA
Robert Barker, University of Louisville, USA

14 Designing a Secure Cloud Architecture: The SeCA Model
Thijs Baars, Utrecht University, The Netherlands
Marco Spruit, Utrecht University, The Netherlands

33 Performance and Scalability Assessment for Non-Certificate-Based Public Key Management in VANETs
Pei-Yuan Shen, Queensland University of Technology, Australia
Maolin Tang, Queensland University of Technology, Australia
Vicky Liu, Queensland University of Technology, Australia
William Caelli, Queensland University of Technology, Australia

57 Towards Usable Application-Oriented Access Controls: Qualitative Results from a Usability Study of SELinux, AppArmor and FBAC-LSM
Z. Cliffe Schreuders, Leeds Metropolitan University, UK
Tanya Jane McGill, Murdoch University, Australia
Christian Payne, Murdoch University, Australia
Towards Usable Application-Oriented Access Controls: Qualitative Results from a Usability Study of SELinux, AppArmor and FBAC-LSM

Z. Cliffe Schreuders, Leeds Metropolitan University, UK
Tanya Jane McGill, Murdoch University, Australia
Christian Payne, Murdoch University, Australia

ABSTRACT

A number of security mechanisms are available for improving the security of systems by restricting the actions of individual programs to activities that are authorised. However, configuring these systems to enforce end users’ own security goals is often beyond their expertise. Little research has investigated the usability issues associated with application-oriented access controls. This paper presents the results of a qualitative analysis of user perceptions of the usability of three application-oriented security systems: SELinux, AppArmor, and FBAC-LSM. Qualitative analysis identified a number of factors that affect the usability of application-restriction mechanisms. These themes are used to compare the usability of the three systems studied, and it is proposed that these factors can be used to inform the design of new systems and development of existing ones. Changes to the three security systems are also proposed to address or mitigate specific usability issues that were identified.

Keywords: AppArmor, Application-Oriented Access Controls, FBAC-LSM, Functionality-Based Application Confinement (FBAC), HCI Security (HCISec), Sandboxing, Selinux, Usability

INTRODUCTION

End users face serious security risks related to processes maliciously misusing users’ authority. One of the largest threats to end users is flaws in applications such as PDF readers, media players, web browsers and email clients (Dhamankar, Dausin, Eisenbarth, & King, 2009). These vulnerabilities can inadvertently allow remote attackers to subvert the behaviour of programs in order to carry out malicious actions. Trojan horses, where malware poses as legitimate software and carries out malicious activities, are also a significant threat.

DOI: 10.4018/jisp.2012010104
Linux, like most operating systems, typically allows applications to act with all the authority of a user. The Linux discretionary access control (DAC) mechanism authorises processes to run with the full authority of the associated user, regardless of the trustworthiness of programs. In the current threat climate this approach is inadequate as a sole access control measure; basing security decisions on the identity of the user does not protect against processes which act maliciously due to software vulnerabilities or malware.

The Linux Security Module (LSM) framework provides a means for security extensions to be incorporated into the Linux kernel (Wright, Cowan, Smalley, Morris, & Kroah-Hartman, 2002). Many of the LSMs that have been developed can address threats posed by malicious code, by restricting specific processes to authorised actions. Examples of LSMs that can place restrictions on the activities of processes include SELinux (Smalley, Vance, & Salamon, 2001), AppArmor (previously known as SubDomain) (Cowan et al., 2000), TOMOYO (Harada, Horie, & Tanaka, 2004), and SMACK (Schaufler, 2008). However, as is typical for this class of security mechanism (DeWitt & Kuljis, 2006), these systems face usability challenges that can limit the practical benefit to end users.

A new model, known as the functionality-based application confinement (FBAC) model, has been designed to meet end user usability goals (Schreuders & Payne, 2008a). The model incorporates policy abstractions, known as functionalities, that can model the privileges authorised to processes based on the high level features applications provide (Schreuders & Payne, 2008b). A Linux implementation of the FBAC model has been developed, known as FBAC-LSM (FBAC-LSM is free open source software available at: http://schreuders.org/FBAC-LSM). The implementation also leverages automation techniques, which the FBAC model is naturally suited to.

A study has been conducted to compare the usability of three different approaches to application restrictions: FBAC-LSM, and two of the most widely deployed Linux security extensions, SELinux and AppArmor (Schreuders, McGill, & Payne, 2011). The results showed that the functionality-based mechanism enabled end users to effectively control the privileges of their applications with far greater success than the widely used alternatives. In particular, policies created using FBAC-LSM were more likely to be enforced and exhibited significantly lower risk exposure, while not interfering with the ability of the application to perform its intended task (Schreuders et al., 2011). In order to further explore and understand the reasons for the usability differences between the three security systems, this paper presents the results of qualitative analysis of the participant feedback for each of the security systems. The qualitative analysis identified a number of emergent themes in participants’ comments. These themes indicate a number of factors that affect the usability of application-restriction mechanisms, and are likely to be responsible for the usability differences between the security systems studied. These results are then discussed and used to compare the usability of the three systems studied. The paper also proposes changes to all three systems to address or mitigate specific usability issues that were identified throughout the study.

BACKGROUND

To reduce the threat of the problems posed by the limitations of user-oriented access control, application confinement has become an active area of research (Berman, Bourassa, & Selberg, 1995; Cowan et al., 2000; Goldberg, Wagner, Thomas, & Brewer, 1996; Hallyn & Kearns, 2000; Harada et al., 2004; Ott, 2002; Provos, 2002; Smalley et al., 2001). Application-oriented access controls provide restrictions based on the identity of the application or process rather than just the identity of the user. Application confinement can limit the ability of applications to access resources outside of those they require to perform legitimately, thus restricting the damage malware or exploited vulnerabilities can cause, and can limit software to actions that
are those expected by whoever configures the security policy: end users, administrators, or software developers.

Despite the fact that usability has long been acknowledged as an important aspect in the design of security systems (Saltzer & Schroeder, 1975), usability received little attention in the literature until the importance of applying human-computer interaction (HCI) techniques to the field of computer security was emphasised by studies that demonstrated that poorly designed security user interfaces resulted in degraded protection (Hitchings, 1995; Zurko & Simon, 1996). Although awareness of the importance of usability in security design has improved (Cranor & Garfinkel, 2005), and the literature now contains numerous publications related to computer security usability, very little research has investigated or addressed the usability issues associated with application-oriented access controls.

Research has explored usability within the wider field of access control and policy specification. A number of publications have identified general problems with existing user-oriented schemes (Motiee, Hawkey, & Beznosov, 2010). The problem of usability and policy complexity within access controls has been acknowledged a number of times, and methods for improving the usability of policy specification have been explored (Brodie, Karat, & Karat, 2006; Cao & Iverson, 2006; Johnson, Karat, Karat, & Grueneberg, 2010a; Johnson, Karat, Karat, & Grueneberg, 2010b; Karat, Karat, Brodie, & Feng, 2005; Reeder et al., 2008; Reeder, Karat, Karat, & Brodie, 2007; Zurko, Simon, & Sanfilippo, 1999; Zurko & Simon, 1996). The following are examples of policy authoring techniques developed by usable security researchers to overcome usability and policy complexity problems: the Adage (Zurko et al., 1999) and MAP (Zurko & Simon, 1996) projects were designed to provide usable RBAC for distributed organisations, SPARCLE is a natural language policy management tool that guides users through the task of specifying policy (Brodie et al., 2006; Karat et al., 2005), Expandable Grids is a graphical method of managing and viewing policy using a hierarchical matrix (Reeder et al., 2008), and Intentional Access Management produces user-oriented DAC ACL policy rules based on low-level access goals of end users (Cao & Iverson, 2006). Recently Johnson et al. (2010a, 2010b) have proposed techniques for improving the usability of guided natural language policy specification using policy templates. Findings of studies such as these have added weight to the idea that abstraction improves the usability of access controls and eases policy specification.

However, little research has explored the usability of application-oriented access controls. The isolation-based Apiary scheme and the data-centric FileMonster scheme have been the subjects of limited usability studies (Potter & Nieh, 2010; Schmid, Hill, & Ghosh, 2002). A simple user study, with 24 participants, was conducted that evaluated the ability of users to use applications in the Apiary environment (Potter & Nieh, 2010). The use of the Apiary desktop was compared with Xfce, a lightweight Linux desktop environment. Usability evaluation was simply measured by time-on-task and participants were “asked to rate their perceived ease of use of each environment”. It is not clear what tool they used to collect this data. Participants were also asked some other questions “including, would the Apiary environment be an environment they could imagine using full time and would it be an environment they would prefer to use full time if it would keep their desktop secure”. Results were reported as affirmative, although no inferential statistics were employed. The FileMonster paper reports a simple study that measured the number of times the tool required user interaction (Schmid et al., 2002).

A study by DeWitt and Kuljis (2006) assessed the usability of the Polaris security mechanism (Stiegler, Karp, Yee, Close, & Miller, 2006), an application-oriented access control system for Windows designed with usability in mind. The Polaris study involved 10 participants utilising the security system to carry out a number of tasks. Like the usability
study described in this paper, their success at the tasks was evaluated and perceived usability was measured. After using Polaris to attempt a number of tasks, participants on average rated the system 44.2 out of 100 using the System Usability Scale (SUS) (Brooke, 1996), and the study concluded that further work was necessary to improve the usability of Polaris.

**METHODOLOGY**

The data analysed in this paper was collected as part of a broader study investigating the security and usability of the three security systems. Only those aspects of the project relating to user perceptions of the usability of the three systems are included in this paper. The study employed a within-subjects design, where participants used all three of the systems studied: SELinux, AppArmor, and FBAC-LSM. Participants were recruited from the ICT student body of an Australian university, a Linux users group, and an information security association. The names of the systems to be used were not revealed prior to participation. Forty six people participated; however, seven left early and their results were not considered during analysis.

Each participant attempted to use each of the systems, in a randomly allocated order, to restrict the actions of two applications as defined by separate task scenarios. The scenarios describe two realistic threats: a web browser which regularly processes untrusted data, and a game that was downloaded from an unauthenticated website. The web browser used was Opera, and the game was KSirtet (a Tetris clone), which had been modified to act as a simulation of a Trojan horse. Participants were encouraged to spend a maximum of approximately one hour on each system, with a total maximum experiment time, including feedback, of four hours.

Each participant was allocated VMWare virtual machine (VM) images for each of the security systems. Due to differences in the extent that the security extensions were available for different Linux distributions, distributions were selected based on which had the highest level of integration for each security system. The SELinux environment and toolset available was provided using the Fedora 11 Linux distribution, and OpenSuse 11.1 was used for AppArmor. FBAC-LSM was deployed using its development environment, OpenSuse 10.3. The VMs were configured to be visually alike.

Information about the study and each of the security systems was provided via pre-recorded video files to ensure consistent dissemination of information. Participants were also provided with handouts including a welcome page with system use information and task scenarios, a Filesystem Hierarchy Standard (FHS) reference (the complete FHS v2.3 was also available as a PDF file), and a Linux command reference. Participants were instructed not to search the Internet for information about any of the security systems used.

Participants initially watched an introductory video, then filled in a pre-experiment questionnaire, and watched a Linux filesystem video (which gave an overview of the FHS). Then they used each of the three systems. For each security mechanism a video and handout gave an overview of the system and a demonstration of how to use the system. Each explanation covered the same level of detail: policy components, how policy is represented, states that policies can be in (either enforced or not), overview of the steps to confine an application, and a list of helpful commands. Graphical interfaces were favoured over command line tools for demonstrations. After watching the video participants attempted to confine the two applications. Once they had finished with each system, participants completed a post-task questionnaire. After using all three systems they filled in a post-experiment questionnaire, and then were taken to another room for a debriefing session where they were asked to discuss what they thought of each of the three systems.

A pilot study was conducted with four participants. The pilot group completed an additional questionnaire, in which they all indicated that they did not notice anything potentially biasing in the videos, presentations or handouts. The pilot group did identify some technical
problems (such as VM networking issues which were due to software incompatibilities), which were all resolved before the main study.

Qualitative data was collected from open-ended questions in the questionnaires, note-taking while participants used the security systems, and during debriefing sessions. The post-task questionnaire included three written response questions; these requested positive and negative comments about the system they had just used, and asked how the security system could be improved. Each of these questions had space for three responses.

The debriefing sessions included six open-ended questions, asking participants to discuss what they thought of each of the three systems, and then whether they understood the decisions they were making with each system. Participants were asked follow up questions as necessary for clarification. This was followed by the opportunity to ask participants some more specific questions about issues that seemed relevant or contentious based on previous feedback. For example, participants were asked whether they found the AppArmor severity level helpful, and whether they thought FBAC-LSM presented too much information at a time. Further details of the methodology can be obtained from Schreuders, McGill, and Payne (2011).

In order to explore the causes of difference in usability between application-restriction systems, emergent themes were identified in the post-task questionnaire comments. Theme identification is a technique used in qualitative research to provide an insight into the issues that are under study by extracting meaning and developing explanations of phenomena. The survey responses of negative and positive comments for each of the security systems were analysed using a combination of qualitative techniques. Open coding, also known as latent coding, a form of inductive data analysis, was used to code comments based on categorising the underlying meaning of the text into central tendencies. These themes were induced, that is, they emerged, from the participants comments. As is recommended in the literature (Ryan & Bernard, 2003), the complete text of all the comments was read a number of times. Following this preparation, pawing was used to label common themes, by marking up and colour coding the responses. Cutting and sorting (digitally, on a spreadsheet) was used to organise the responses to assist the process. Unmarked comments were then analysed to attempt to identify whether they related to one of the previously identified themes, or formed new categories. In most cases the categorisation of comments into themes was an intuitive abstraction of the manifest content, to a form of latent content. For example, the comment “The program was a hassle to navigate between as the layout of the interface was larger than the screen and could not be scaled down” was assigned the “interface criticism” label. Comments related to general ease of use were noted and are discussed, but were too broad to be categorised into a theme, since wherever possible comments were categorised into more specific categories, many of which were related to specific aspects of ease of use. The presentation of themes in the following sections includes descriptions of sub-themes: the types of comments that make up themes.

All of the qualitative data that was collected, including notes from debriefing interviews and those taken during the experiment, were considered when developing suggestions for improving the usability of the security systems. Each of the major issues that were identified, as well as the constructive suggestions from participants, were formulated as suggestions for improving these security systems.

RESULTS

Participant Demographics

Participants’ ages ranged from 18 to 67 with an average age of 31. The vast majority were male, with only five female participants. As can be seen in Table 1, the majority of participants evaluated themselves as possessing above average computer skill; however the range of perceptions of computer security expertise and Linux expertise was much wider.
SELinux and Usability

SELinux implements a combination of access control models such as role-based access control (RBAC), domain and type enforcement (DTE), and multilevel security (MLS), to provide mandatory controls for Linux. DTE forms the basis of application restrictions, where rules define how processes within particular domains can access resources labelled with specific types. A number of user-space tools to configure SELinux are available, and are available on Linux distributions, such as Fedora, that provide support for SELinux. Although some management tasks can be achieved using GUI tools (such as the SELinux Policy Generator tool), most require the use of command line tools. Although configuration by end users is not the primary goal for SELinux, it is widely deployed and effectively in direct ‘competition’ with the alternatives, since only one LSM can be installed on a Linux system at a time.

A total of 70 negative comments, 50 positive comments, and 51 suggestions for SELinux were collected from the responses to the post-task questionnaire. Of the three systems SELinux received the most criticism. The themes that emerged from these indicate some of the reasons for SELinux’s usability problems.

Eight themes emerged from the negative comments. They are listed here in descending order of frequency:

- Criticism of interface (18 comments)

  Participants reported a number of problems with the interface for SELinux. Although some graphical policy tools exist (and were used as far as possible in the study), in order to create policy users are required to use command line tools. However, as stated by a participant “command line interface reduces the usability [for] some people with less knowledge”. Another participant stated, “You need to run a script to compile the policy. Everything should be done using the GUI”. Even when using the tools provided to automate the generation of rules, users have to manually vet the rules generated by audit2allow. The following comment highlighted this, “no GUI for fine-grained editing of the .te files”. Three participants commented on the fact that the policy generator GUI did not fit on the display, and it could not be resized. Other criticisms of the interface included the lack of wizards to guide the processes, and the fact that, while SETroubleshoot identified policy problems, it did not allow users to use the tool to make the suggested policy changes.

- Expertise required (14 comments)

  In addition to the difficulty of using the command line (as described above), participants commented on the expertise required to use SELinux. In order to use the graphical tools to generate the initial skeleton policy, users “… require intimate knowledge of Linux, e.g., port numbers, dbus …”. A participant who found this initial part comfortable stated “While the first stage is easy, the following stages require knowledge and use of scripts and macros that would take significant time to learn”. Participants also reported that SELinux “Requires very in-depth knowledge of [the] Linux file structure”, and “Required technical understanding/knowledge about the software/security system to be usable”.

<table>
<thead>
<tr>
<th>Expertise</th>
<th>Mean</th>
<th>Std dev</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skill with computers</td>
<td>5.82</td>
<td>0.90</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>Knowledge of computer security</td>
<td>4.47</td>
<td>1.20</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>Frequency of Linux use</td>
<td>4.24</td>
<td>2.32</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>Knowledge of Linux</td>
<td>3.53</td>
<td>1.89</td>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 1. Participant self assessment

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
• Unclear or confused about behaviour (9 comments)

Some responses indicated participants were confused by SELinux or its behaviour. For example, one participant said “Allowed http access when no ports were specified in initial setup”; this seems to be due to a participant not remembering that policies are created in permissive mode. Although explained in the SELinux explanation video, the software did not inform users that new policies that are created are not automatically in effect.

• Time and number of steps involved (6 comments)

As one response stated “It’s very time consuming, the amount of work that has to be done is enormous”. Also the number of separate commands and steps were listed as negative comments.

• Complexity of the system (6 comments)

Related to the expertise required, some responses highlighted the complexity of SELinux and the terminology used.

• Unsure of success (5 comments)

Some participants were not sure if the policy they created would provide security, or whether they had successfully completed the tasks. Comments included: “It wasn’t clear if the security policy was going to work”, “Didn’t feel like I secured anything”, “No confidence that setup is successful in providing security”.

• Output/policy hard to understand (4 comments)

The format of the SELinux policy, logs (AVCs), and tools designed to make these easier for users (SETroubleshoot and audit2why) were criticised as being hard to understand. The output from these tools is at times extremely abstruse. Comments included: “Even with [audit2why] it couldn’t succinctly express what the problem was”, “The information provided when there was a security error was hard to understand.”

• Bugs (3 comments)

A bug was identified where specifying the ports in the policy generator tool would result in a “Too many values to unpack” error message. Also SELinux silently ignored some denied actions that needed to be audited in order to create rules that were necessary for Opera to run. Neither of these faults prevented policies from being created; however they did introduce complications.

• Other comments deemed too general to contribute to a specific theme

In addition to these themes, four participants commented generally that SELinux was difficult to use.

Six themes emerged from the positive comments for SELinux:

• Good interface (13 comments)

Two participants stated that they favoured the command line interface. Others referred to the graphical interface that is used in the first half of the task, stating that it is well labelled and easy to read. One participant stated that “It’s integrated nicely in the operating system”. Another stated that there were “not too many options.”

• Good auditing and alerts (6 comments)

The auditing messages and alerts were commended. SELinux was the only system (by default) to include runtime notifications of security events as they happened. Some participants found the SETroubleshoot tool helpful. As stated by participants, it “gives great informative
messages when it blocks something”, and “had tools to help you locate issues and attempt to fix them”. One participant found audit2why and audit2allow “good for summarising otherwise unintelligible log file output.”

- Ease of editing and updating (3 comments)

Some participants thought that it was easy to update and edit policy.

- Comprehensive protection (3 comments)

Some participants commended SELinux for being fully featured.

- Other noteworthy comments, including those deemed too general to contribute to a specific theme

Ten comments were that SELinux was generally easy or straightforward: for example, “Easy to use.” However, most of the evidence, both qualitative and quantitative does not support the idea that configuring SELinux is straightforward. Some of these comments explicitly refer to the initial stage, where the GUI tools were available. One separate comment stated “it did not get overly technical”, another that it didn’t “require deep knowledge.” Two comments implied that SELinux would be good for someone with further expertise. Other single comments included that it seemed secure, and was quick.

**AppArmor and Usability**

AppArmor has been proposed as an easier to use alternative to SELinux (Suse, 2011). AppArmor profiles define lists of resources each program is allowed to access. AppArmor abstractions group related low-level privileges. User-space tools to configure AppArmor are available, including graphical tools that are available on Linux distributions such as openSUSE. These graphical tools can be used to create and manage policy, including a ‘learning mode’ used to create application profiles based on the actions a program attempted previously. AppArmor policies can be long and detailed, reflecting the underlying complexity of the confined applications and the various platform layers these depend on.

Participants provided a total of 69 negative comments, 65 positive comments, and 54 suggestions for AppArmor by means of the post-task questionnaire. In light of this data, some usability issues have been identified. From the negative comments for AppArmor, 7 themes were identified. They are listed here in descending order of frequency:

- Expertise required (17 comments)

The AppArmor tools display almost every low-level resource each application utilises, and this exposes the complexity of applications and the underlying platform to end users. Most participants did not possess the expertise necessary to correctly Yet the rules that were generated. For example, one participant said “Requires some knowledge of what each action is and does in order to ensure correct configuration.”

- Criticism of interface (14 comments)

Some responses criticised aspects of the interface, such as the inability to “navigate backwards while allowing/denying things”, and the lack of a progress bar.

- Too many decisions to make (7 comments)

The large number of security decisions the user needs to make to confuse an application led many participants to click ‘allow’ without considering the security implications of each rule. For example, comments included: “Hundreds of file access decisions”, “Lots of allow/deny clicks. By the end I was just clicking allow without thinking / reading.”

- Criticism of help provided (6 comments)
Related to interface criticisms, some responses identified problems participants had with the help provided by the AppArmor tools. Comments included: “Not enough explanation as to the modes”, “User help is poor - not able to define buttons use when selecting ‘help’ in the GUI”.

- Time taken (6 comments)

Related to the number of decisions made, some responses criticised the amount of time it takes to create AppArmor profiles.

- Unclear or confused about behaviour (6 comments)

Some responses indicated participants were confused by AppArmor or its behaviour. For example, “Had to resort to command line as couldn’t figure out GUI - maybe better after used command GUI first time as realised what to do”, “Seems to only prevent access to individual files instead of denying unsafe actions.”

- Unsure of success (4 comments)

Some participants expressed that they were “not sure how secure it was by the end of the process.”

- Other noteworthy comments, including those too general to contribute to a specific theme

Two participants commented generally that AppArmor was difficult to use. Two comments indicated that it was difficult to create a restrictive policy. For example, “I thought it was too difficult to build a ‘deny all’ policy (for an untrusted app) then allow permissions subsequently.” One participant indicated that configuration was complex, which was categorised as relating to the theme of complexity in order to facilitate comparison between systems. Another participant stated that AppArmor caused the program being confined to crash, since they did not give the program sufficient privileges.

Analysis of the positive comments for AppArmor indicated these four primary themes:

- Good interface (24 comments)

AppArmor’s interface was commended for being “well laid out”, and “efficient”. Two comments mentioned that it was “well integrated into OS”. Three participants indicated that they appreciated the guided wizard-driven approach. Other features mentioned include the globbing feature, and the ability to view rules before they are applied.

- Intuitive/easy to understand (11 comments)

Some participants thought that AppArmor had “simple concepts, [and that] the general configuration and profiles are simple to grasp.” Some comments also indicated that the learning mode was a concept that was easy to understand.

- Automation (5 comments)

Five participants appreciated the ability of the learning mode to automate the task of specifying policy.

- Seemed secure (5 comments)

Some comments indicated that AppArmor “seems very secure in managing program access”, and that they could see that it “does block things.”

- Other noteworthy comments, including those too general to contribute to a specific theme

Ten comments were that AppArmor was generally “easy to use”. Two comments indicated that it was easy to edit or update policies. Other positive comments mentioned the lower
expertise required, the abstractions provided, its comprehensiveness, and the policy format.

**Functionality-Based Application Confinement (FBAC) and Usability**

FBAC-LSM implements new techniques for restricting applications. The access control model is known as Functionality-Based Application Confinement (FBAC) (Schreuders & Payne, 2008a). Policy abstractions known as functionalities are used to represent the authority for applications to carry out features they provide. Functionalities are hierarchical (which allows them to be defined in terms of other encapsulated functionalities), and are parameterised (which allows functionalities to be reused to enforce abstract and configurable goals). These features allow the FBAC model to restrict applications based on high-level security goals – restricting them to the features they should perform – while restricting them to the finely-grained privileges they require.

FBAC-LSM also employs a number of policy automation techniques which leverage correlations between application policy requirements (the FBAC functionalities applications require) and attributes of the applications and files on a Linux system. For example, the libraries applications use, and the information used to sort applications into menus used by desktop environments (such as KDE and Gnome), are related to the features they provide. These techniques automate and guide policy generation by suggesting functionalities and automating the specification of parameter values. Using these techniques, complete policies can usually be created for applications a priori; that is, without having to run the programs before or during policy specification. FBAC-LSM also contains a learning mode which can be used in the event that a specified policy does not allow a program to perform correctly.

The post-task questionnaire collected 46 negative comments, 84 positive comments, and 44 suggestions for FBAC-LSM. Eight themes were identified from the negative comments about FBAC-LSM. They are listed here in descending order of frequency:

- Criticism of interface (11 comments)
  Criticisms of the graphical interface included having “too many options”, being “a bit arcane in its terminology”, and having popup dialogs that could not be suppressed.

- Time and number of steps involved (9 comments)
  The automation process was relatively quite slow, which was criticised. Also there were “lots of steps.”

- Expertise required (7 comments)
  FBAC-LSM asked users to review all the steps that were automated. In order to fully understand the automation of parameter arguments participants “require a deep understanding of Linux file system”. As one participant put it, a “naïve user won’t be able to use software”. One participant commented: “Requires the individual to specify what the program does, not something everyone would know.”

- Unclear or confused about behaviour (4 comments)
  Some responses indicated participants were confused by FBAC-LSM or its behaviour. For example, “Not entirely sure how the program works. I know the basics but not the specifics.”

- Unsure of success (4 comments)
  Four responses indicated participants were unsure how successfully they had configured FBAC-LSM. For example, a participant commented “I’m not certain how secure the profile really is.”

---

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.
• Bugs (4 comments)

Although functional, due to its prototype status at the time of the study FBAC-LSM was unstable. Many participants experienced FBAC-LSM crash. Participants were simply informed that the crash was not their fault, that they would not lose the policies they had created, and that they should restart the VM.

• Silent denials / auditing and alerts (2 comments)

FBAC-LSM silently denies any action that is not authorised (notification is via dmesg). A graphical notification interface is in development.

• Name (2 comments)

FBAC-LSM’s name was criticised.

• Help provided (2 comments)

Two participants indicated that they would prefer further “context-sensitive help.”

Analysis of the positive comments for FBAC-LSM identified these seven themes:

• Good interface (25 comments)

Positive comments regarding FBAC-LSM’s interface relate to: the use of wizards, the use of categories for organising functionalities, and that the interface is “intuitive and explanatory.” Four positive comments were made regarding the help and guidance provided within the interface.

• Automation (19 comments)

A number of participants found the automation of a priori policy specification useful. As one participant described it, FBAC-LSM “supplies default configurations and populates suggestions on what could be good settings based on previous options chosen.” Another participant noted, “the software was very intuitive, it auto-recognised the type of program it was”. Three participants also found the learning mode useful, and noted that it included “learning ability if profile didn’t satisfy requirements.”

• Useful policy abstractions (6 comments)

Some participants made positive comments regarding the use of functionalities to assign authority. Comments included “Liked the pre-configured profile templates, i.e., Internet, IRC, etc.”, and “Nice organisation of the programs to be confined in categories depending on their functionalities”.

• Seems secure (4 comments)

For example, one participant stated that FBAC-LSM “seemed to be effective in providing adequate access for users while being secure”.

• Intuitive / easy to understand (3 comments)

As one participant stated, FBAC-LSM has a “very easy to understand model for permissions and privileges”.

• Comprehensive protection (3 comments)

Some responses commented that the level of protection provided “appears to be very comprehensive”.

• Ease of editing and updating (3 comments)

Some responses commended the ease of changing existing policy. For example one participant stated that it was “easy to update policies.”

• Other noteworthy comments, including those too general to contribute to a specific theme
Fifteen comments commended FBAC-LSM for being generally easy. The “ability to test the application without running it, eg test if can write to a file” was commended by two participants. Also FBAC-LSM was described as “quick”.

**DISCUSSION**

Usability has long been acknowledged as an important aspect in the design of security systems (Saltzer & Schroeder, 1975; Zurko & Simon, 1996). However, very little research has investigated usability issues associated with application-oriented access controls (Alexander & Jasna, 2006), or identified the security issues that arise in this field. The qualitative analysis of the comments provided by participants identified a number of emergent themes. These themes give an indication of the existing problems and strengths of the systems studied, serve as a means of comparison between systems, and identify a number of issues that can affect the usability of application-restriction mechanisms and inform both the design of new systems and the development of existing ones.

**Negative Factors Affecting Usability**

Qualitative analysis identified 11 themes in the negative comments across the three systems, which correspond to factors that can be the cause of decreased usability in application-oriented access controls. Table 2 lists these factors and shows how many comments about each system corresponded with each factor. The two most prominent issues were interface design, and expertise required. The amount of time taken, and the confusion and uncertainty of users were also common themes, which are related to the two main issues and should be considered when assessing the interface and expertise required for application-oriented access controls.

Previous HCI security (HCISec) research has demonstrated that interface problems can impact the protections provided by security systems (Alma & Tygar, 1999). The research described in this paper research supports those findings, and demonstrates that interface design impacts the usability of application-restriction schemes.

The user interface for SELinux was particularly criticised, in part due to the lack of graphical tools. AppArmor has a more complete set of graphical tools compared to SELinux, but still requires a high-level of expertise to utilise

---

**Table 2. Themes in negative comments**

<table>
<thead>
<tr>
<th>Theme</th>
<th>SELinux</th>
<th>AppArmor</th>
<th>FBAC-LSM</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poor interface design</td>
<td>18</td>
<td>14</td>
<td>11</td>
<td>43</td>
</tr>
<tr>
<td>Level of expertise required</td>
<td>14</td>
<td>17</td>
<td>7</td>
<td>38</td>
</tr>
<tr>
<td>Time taken and steps required</td>
<td>6</td>
<td>6</td>
<td>9</td>
<td>21</td>
</tr>
<tr>
<td>Unclear behaviour</td>
<td>9</td>
<td>6</td>
<td>4</td>
<td>19</td>
</tr>
<tr>
<td>Clarity of success</td>
<td>5</td>
<td>4</td>
<td>4</td>
<td>13</td>
</tr>
<tr>
<td>Help and guidance provided</td>
<td>0</td>
<td>6</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>Complexity</td>
<td>6</td>
<td>1</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>Bugs</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>Number of decisions required</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>Output format</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Auditing and alerts</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
correctly. However, the graphical interface provided for AppArmor is relatively simple, and since interface design appears to be a significant aspect of the perceived usability, this helps explain the fact that AppArmor was preferred by some participants. The FBAC-LSM interface was criticised for the amount of information presented to users. A few negative comments also focused on the large number of steps in the FBAC-LSM wizard, and the help provided. However, the FBAC-LSM interface received the least amount of criticism.

The nature of the problem of defining rules that allow programs to run and perform their legitimate tasks while also restricting programs from acting maliciously results in complex policies. The task of creating and configuring these complex policies can require advanced expertise. This is particularly the case with SELinux, which requires high levels of expertise in understanding the complex constructs that policy is composed of, in addition to the expertise necessary to use the command line tools. Due to its graphical interface AppArmor requires less expertise to operate, but still requires advanced expertise to be used meaningfully, since users need to decide which resources the program should be allowed to access.

A number of participants who rated themselves as experts in Linux and/or IT security (such as a Linux system administrator and a security professional) were not successful at utilising either AppArmor or SELinux to confine malicious code (Schreuders et al., 2011). For example, some “expert” participants believed that they had correctly confined the applications but had actually left the Trojan simulation free to perform numerous malicious actions, including compromising the security of their personal data and obtaining system or user-level compromise. This illustrates that users do not typically have the expertise to vet the low-level actions of applications. This observation has substantial ramifications for the usability of the approaches taken by many application-oriented access control mechanisms including AppArmor and SELinux, and their ability to use learning modes to confine malware. This also further emphasises the importance of the expertise required, and its effect on usability and security.

FBAC-LSM incorporates techniques to abstract policy into high-level goals and to automate tasks. The result of these features is that users are not required to vet every low-level action that applications perform, thus reducing the expertise required while still allowing users to enforce their specific security goals. The results reported in this paper support the theory that reducing the expertise requirement can improve usability, as expertise required is a stronger theme with the data collected regarding SELinux and AppArmor than with FBAC-LSM.

The other negative themes identified were more specific to particular systems. Related to the expertise required, SELinux is a particularly complex system, and the output from the tools exposes the complexity of the policy language and constructs to users. Although the interface is simple, AppArmor exposes users to the low level complexity of the resource requirements of applications, requiring users to vet nearly every action performed; this results in a very large number of security decisions for the user to make and a high level of expertise is required to vet the rules generated. Also, related to the interface, the in-program help provided by AppArmor was criticised. Many of the negative comments for FBAC-LSM are related to its prototype status at the time of the study: it was slow, contained a number of bugs, and lacked an interactive graphical notification tool.

In addition to the main common themes described at the start of this section, the following themes affected the usability of the systems studied and should be considered in future research and design: complexity of the system, format of output, auditing and alerts, and bugs.

Positive Factors Affecting Usability

Analysis of the positive comments collected identified nine themes. These correspond to factors that can have a positive impact on the usability of application-oriented access controls.
Table 3 lists these factors and indicates how many comments about each system corresponded with each factor. Again, a large number of comments related to the general quality of the interface. Automation of policy specification, the intuitiveness of the scheme, alerts, and the abstractions used also appear to be significant factors that can positively affect the usability of application-oriented access controls.

Some of the positive interface attributes that were commonly mentioned include: wizards and step-by-step guidance, simplicity of the interface, integration with the rest of the operating system, logical grouping of concepts and widgets, multiple approaches available for performing tasks, and policy review features. SELinux, AppArmor, and FBAC-LSM all employ some form of wizard interface for creating application policies. SELinux provides a wizard for the initial stage. Some participants liked this tool, but many were not satisfied by the requirement to also use command line tools. AppArmor provides a wizard for creating policies using the learning mode. FBAC-LSM also provides a wizard for an a priori policy specification, and also a learning mode tool to add rules to existing policies. Participants believed that FBAC-LSM provided the most guidance in the form of on-screen help. AppArmor had the simplest interface, in terms of the amount of details displayed at a time (although, as discussed, this was displayed for too many decisions). SELinux and AppArmor were naturally better integrated into the operating system, since these are included in the Linux distributions used, and were incorporated into the administration tools for the distribution, although this is not an intrinsic advantage of either system. For example, SUSE Yast includes an AppArmor panel with launchers for the AppArmor configuration tools. The fact that AppArmor and FBAC-LSM include the ability to review the policies that are created were also described as positives. FBAC-LSM included extensive review and query capabilities that allow the permissions to be tested without running the application.

Automation was a common theme in praise for FBAC-LSM. FBAC-LSM’s automation techniques leverage the policy abstractions used by the scheme to create policies an a priori (Schreuders, Payne, & McGill, 2011). The effect of this is that a lower level of expertise is required, and fewer decisions need to be made by end users, compared to the learning mode method of automation provided by AppArmor and SELinux. The learning mode scheme of AppArmor was commended by some participants. The intuitiveness theme, mainly attributed to AppArmor, can in part be explained by the simplicity of the notion of learning rules based on what an application does. FBAC-LSM also included a learning mode, for the situation that policies created an a priori required extra rules. Comments such as “Had learning ability if profile didn’t satisfy requirements” indicated that

Table 3. Themes in positive comments

<table>
<thead>
<tr>
<th>Theme</th>
<th>SELinux</th>
<th>AppArmor</th>
<th>FBAC-LSM</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good interface</td>
<td>13</td>
<td>24</td>
<td>25</td>
<td>62</td>
</tr>
<tr>
<td>Automation</td>
<td>0</td>
<td>5</td>
<td>19</td>
<td>24</td>
</tr>
<tr>
<td>Intuitive</td>
<td>0</td>
<td>11</td>
<td>3</td>
<td>14</td>
</tr>
<tr>
<td>Seemed secure</td>
<td>0</td>
<td>5</td>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>Editing and updating</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>Comprehensive</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>Abstractions</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Alerts</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>
those participants understood that the FBAC-LSM learning mode can be used if policies that are created do not provide all the privileges required by the application to function.

A fundamental aspect of the FBAC model is the policy abstraction (functionalities) that assigns privileges based on the features that applications provide. As described, FBAC-LSM’s automation techniques leverage these abstractions. A number of participants also explicitly identified these abstractions as positive aspects of FBAC-LSM (shown in Table 3 as the abstractions theme), other comments described the FBAC-LSM permission model as intuitive. Some mention the primitive abstractions used by AppArmor: for example, “KDE”. In contrast, the complexity of the SELinux model was criticised. These results support the theory that the type of policy abstraction implemented has a significant impact on the usability of application-oriented access control schemes, and that functionalities can provide usability benefits.

Other factors, such as how secure the system seems and how easy it is to update and change existing policies, can also have an effect on perceived usability. The alerts and notifications employed also can have an effect. SELinux systems, by default, typically come with user-space notification and auditing tools, which were praised by some participants. AppArmor notification tools exist, but are typically disabled by default. FBAC-LSM did not include notification tools, due to its prototype status.

**USABILITY IMPROVEMENT SUGGESTIONS**

Based on all the qualitative information collected, this section proposes changes to SE-Linux, AppArmor and FBAC-LSM to address or mitigate a number of the issues identified. Due to fundamental design properties of these systems, some of these suggestions may be challenging to implement. However, they are included for completeness and to aid in the design of new solutions to application-oriented access controls.

**Suggestions for Improving SELinux**

Currently no graphical tool has been developed to step users through the entire task of confining applications using the standard SELinux primitives. The system-config-selinux tool could be used to browse an overview of the policies installed and to edit the SELinux configuration, and polgengui, the SELinux Policy Generation tool, could be used to create a barebones template policy module. However, no graphical interface was available to complete policy development or to compile and apply the created policy. Some participants reported that they were uncomfortable with the amount of command-line interaction required, and some participants simply refused to attempt the console portion of the task. Therefore it is proposed that tools are developed for SELinux that:

- Cover the whole process of creating, editing, compiling and enabling policies; and
- Step users through the complete process using a wizard style interface.

The tools used to create rules based on access denials do not currently step users through the process of vetting the generated rules. During the usability study it was observed that most participants did not review the rules that they added to policy modules. Therefore, since SELinux relies on the successful vetting of these rules it is suggested that the graphical interface:

- Facilitate the vetting of learned rules.

It is also suggested that a graphical tool should:

- Provide information about the practical impact of rules. For example, a list of all the permissions afforded to a program, and
the ability to query whether an application is authorised to access particular resources.

Based on participant feedback it is also recommended that:

- Graphical tools provide further documentation and hints; and
- The ability should be included to take action from the setroubleshoot tool to make the policy changes it suggests.

Rules are represented in a format that was reported by a number of participants as hard to understand. Thus, it is suggested that:

- The policy language is simplified to make rules easier to understand.

This is particularly important while no GUI tools are available to assist in the vetting of rules. Other alternative approaches such as intermediary policy languages (such as the common intermediary language (cil) and simplified policy description language (spdl) (Nakamura, Sameshima, & Tabata, 2009) and easier to understand alternative views of policy may help alleviate policy abstruseness. It is also recommended that:

- The feedback from all selinux tools be improved. This should include the scripts used to compile policy modules. Improving the output from audit2why and setroubleshoot is particularly important since the format of the avc denial logs cannot easily be read by humans. Perhaps include a simplified interface for less technical users.

A number of flaws were noted that should be corrected:

- The polgengui tool did not inform users that the created policies start in permissive mode.
- The polgengui tool contained a bug that occasionally reports that there were “too many values to unpack” when port numbers are specified.
- The size of the polgengui tool’s window did not fit on low resolution displays and could not be resized.
- Default policies should provide actual confinement: the default policy for ksirtet did not.
- AVC denial logs were sent to one of two separate log files (dmesg and audit.log). This has been acknowledged as a bug.
- The failure to log relevant denials, as was the case with Opera, should be investigated.

Suggestions for Improving AppArmor

Most users do not appear to have the expertise required to perform vetting using the current AppArmor interface. Since the level of security AppArmor provides depends on the successful vetting of rules, the following changes are proposed.

Participants were divided over the usefulness of the severity levels that are displayed as a guide for users during vetting. A few pointed out that, for most resources, the severity level was ‘unknown’. Many found the severity levels to be ambiguous. Therefore these improvements are proposed:

- Clarify severity levels using colour coding; and
- Clarify the scale used by displaying what the maximum and minimum levels are.

It is suggested that apparmor should also provide more useful information about resources and executables, such as including descriptions of:

- The purpose of each resource;
- The risks of granting access to certain resources; and
- The recommend actions.
This could be implemented by extending the current resource database that is used to calculate severity levels, or by assigning this information directly to files using extended attributes.

Based on participant feedback it is also suggested that AppArmor should:

- Provide more help describing the on-screen elements. For example, clarify the meaning of access types such as “mrw”;
- Provide an optional simplified interface for less experienced users;
- Allow users to navigate back to change their mind about previous rules they have set;
- Give an indication of progress through the process of vetting rules;
- Optionally, provide rules in list format for vetting;
- Allow users to skip denials and deal with them later rather than only having the option to allow or deny;
- Allow users to edit profiles as text from the apparmor control panel;
- Allow users to use the graphical update tool to only update specific policies, rather than updating all the policies that have denied access to resources; and
- Improve the interface integration between the various graphical tools, such as the new application profile wizard and the update wizard.

Also, where possible, parts of the process could be automated. For example:

- Automatically suggest globing where appropriate.

Some participants inadvertently left profiles in complaining mode. Therefore:

- The enforcing state should be made more obvious to users.

Participants found it hard to create policies incrementally due to the potentially very large number of resources used. Therefore it is recommended to:

- Have predefined templates that can be used as a starting point to develop profiles.

Suggestions for Improving FBAC-LSM

Although FBAC-LSM demonstrated significant usability advantages, there is room for future improvement. A number of participants provided criticism of the graphical policy manager tool that should be addressed in future development.

The FBAC-LSM policy manager required users to review each step that was automated. Some participants would have preferred:

- Options to automatically accept the automated suggestions; or
- The ability to review the entire automated configuration at once rather than on a step by step basis.

Some participants would have preferred a simplified interface, while others appreciated the detail provided. Therefore, it is suggested that:

- The policy manager should provide advanced and simplified modes to cater to the differing expertise of users.

Also, where possible:

- The terminology used by the graphical tools should be simplified;
- Slightly larger fonts should be used where appropriate; and
- Less information should be displayed at a time.

When FBAC-LSM prevents an application from performing actions, access is silently denied. There are situations where users should be notified. Therefore:
• A notification tool should be developed that can inform users when actions are denied, and enable users to take appropriate actions. It is suggested that not every denied action needs to be reported to the user, but when a denied action is likely to be malicious or the access is required for the program to function the tool should alert the user.

The automation features are slow to calculate suggestions. Therefore:

• Functionality suggestions and parameter value automation should be optimised in order to perform automation faster.

Other relevant suggestions provided by participants include:

• The GUI should provide more information about the security implications of functionalities;
• The mechanism for reloading and saving policies should be improved;
• The GUI should allow applications to be sorted alphabetically.

CONCLUSION

Major differences in usability have been identified between three different approaches to application-restrictions: SELinux, AppArmor and FBAC-LSM (Schreuders et al., 2011). This paper presents the results of a qualitative analysis of user comments to explore the reasons for the usability differences between the three security systems. These results provide an insight into the causes of usability issues associated with application-oriented access controls.

A number of themes were identified in survey feedback from participants. These themes indicate factors that can affect the usability of application-restriction mechanisms. The factors were used to compare the three systems and explain how their different approaches to application confinement lead to differences in usability. Qualitative analysis supported the theory that techniques employed by FBAC-LSM, such as policy abstraction and automation, have a positive effect on usability. These results demonstrate that the FBAC model’s policy abstractions, and the automation techniques developed for FBAC-LSM, result in usability benefits.

A number of other factors that affect the usability of application-confinement schemes were also identified. It is proposed that the factors that were identified can be used to inform the design of new systems and development of existing ones.

REFERENCES


Cliffe Z. Schreuders recently completed his PhD at Murdoch University in Western Australia. He is a Senior Lecturer at Leeds Metropolitan University. His research interests include computer security, application-oriented access controls and sandboxing, usable security, and the methods and effects of free culture and open source software development.

Tanya Jane McGill is an Associate Professor in Information Technology at Murdoch University in Western Australia. She has a PhD from Murdoch University. Her major research interests include e-learning, ICT education and information security. Her work has appeared in various journals including Computers & Education, Decision Support Systems, Journal of Computer Assisted Learning, Behaviour and Information Technology and Journal of Information Privacy.

Christian Payne is a Lecturer in Computer Science at Murdoch University in Perth, Western Australia. He holds a PhD, also from Murdoch University, and his research interests include operating system security models, applied cryptography, risk modelling and novel access control architectures.

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.