

# Utilizing SELinux to Mandate Ultra-secure Access Control of Medical Records

Peter R Croll, Matt Henricksen, Bill Caelli and Vicky Liu

*Information Security Institute, Queensland University of Technology, Brisbane, Australia*

## Abstract

*Ongoing concerns have been raised over the effectiveness of information technology products and systems in maintaining privacy protection for sensitive data. The aim is to ensure that sensitive health information can be adequately protected yet still be accessible only to those that “need-to-know”. To achieve this and ensure sustainability over the longer term, it is advocated that an alternative, stable and secure system architecture is required. This paper considers the adoption of a model targeted at health information that provides much higher degrees of protection. A purpose built demonstrator that was developed based on enterprise-level systems software products is detailed. The long term aim is to provide a viable solution by utilizing contemporary, commercially supported operating system and allied software. The advantages and limitations in its application with a medical database are discussed. The future needs in terms of research, software development and changes in organizational policy for healthcare providers, is outlined.*

**Keywords:** *Information Security, Health Information Systems, Operating Systems, Access Control*

## Introduction

Advances in storage and communication technologies have made large repositories of data available even when they are maintained on separate systems and geographically distributed. The access to such data sets is often subject to varying degrees of legal, social and ethical constraints. Further, the data sets may not be available for open scientific research due to the sensitive and private nature of the information they contain. For example, individual personal health records offer significant value for medical and related research but such information cannot be readily accessed in a manner suitable for such research. The causes include the need to abide by national privacy legislation, the reluctance to change to electronic record format due to security fears and the requirement to maintain end-user trust in the overall healthcare information system. Even when access rights have been granted - for example the data has been sanitized by having all personal identification removed - there are still legitimate concerns over the degree of privacy that contemporary IT applications will provide. The present reality is that IT applications operate on commod-

ity level computer operating systems that do not - and cannot - provide the required level of assurance needed for scientific research to be undertaken on sensitive data. They were never designed for this purpose.

Information security mechanisms do exist to ensure sensitive information is protected and only accessible on a “need-to-know” and approval basis. It is imperative that both adversaries, such as external system “hackers” and technical/operations personnel with in-house knowledge be denied inappropriate access. The security mechanism known as Mandatory Access Control (MAC) is described below and can be used, as adapted, to enforce the necessary security and privacy processes required for handling sensitive health data. Such mechanisms have been studied and understood for over 30 years, mainly in defence related systems. However, they have not been evident in contemporary commodity level operating system or allied application level environments.

In particular, the ICT industry’s move towards development of application systems based around so-called “Service Oriented Architectures (SOA)” and “Web Services” software environments presents new security challenges in the healthcare environment. Security standards, even for these high level service structures often based around “interpreter” sub-systems such as XML, etc. and Internet/World-Wide-Web “browser” packages, are complex and only in the early stages of development and deployment. Moreover, all of these structures critically depend upon the overall security and dependability of the underlying “middleware”, operating system and computer hardware systems. An application cannot be any more secure than the underlying systems upon which it depends. The same holds true for contemporary computer grid technologies, e.g. the Globus [1]. In other words, trying to adequately secure the shared “virtual machine” environments that grid technologies currently exploit is next to impossible (i.e., far too challenging for the foreseeable future).

The primary aim of this project is to build a “Concept Technology Demonstrator”, based upon advanced cryptographic and information research and technologies (e.g., Cryptocards<sup>1</sup> and Starlight InfoSec technology<sup>2</sup>) in order to provide ultra-secure and sanitized access to the protected data sets.

---

<sup>1</sup> “Cryptographic plug-in cards”, Eracom (Safenet) Australia.

<sup>2</sup> “Starlight InfoSec Technology”, Defence Science and Technology Organisation (DSTO), Australia.

## The non-Sustainability of Current Approaches

There is a strong vested business interest by mainstream suppliers of computer operating systems and similar middleware products to perpetuate the belief that computer applications can be “*made secure from within*”, irrespective of other software or even hardware components. In other words, the correct use of their technology will ensure a sufficiently secure operating environment upon which application programs can run safely. Unfortunately, any such assumption is flawed since in reality they represent a “fortress built upon sand” [2].

Current base operating systems in the commercial arena are based around what is designated as “Discretionary Access Control” or DAC. Essentially this design allows the “owner” of any object in a system to, at their discretion (from where the term DAC comes) pass on their access rights to any other person or entity in the overall information system’s environment. In particular, DAC really does not acknowledge the essential difference between a computer “user”, a person, and the individual “processes” in the system that act on his/her behalf. Moreover, the DAC structure assumes that the user is completely familiar with and trusts any program or related software system which they cause to execute in the computer. In the current environment these assumptions, while possibly valid in the era of large mainframe computer systems with large “in-house” software development and support organisations prior to the development of global “packaged” software industry, are simply no longer true. The DAC approach, moreover, assumes that users are the formulators of their own “discretionary” policy, a situation that is no longer valid as overall information systems become subject to overriding legal, societal and enterprise policy requirements.

The applications will not be secure unless the underlying operating system and hardware have been specifically developed with security in mind. A system that is built to utilize a Mandatory Access Control (MAC) mechanism will provide levels of security relating to all aspects of the computing system, i.e., “*enforce an administrative set policy over all subjects and objects in a system, basing decisions on labels containing a variety of security-relevant information*” [3].

## Practical Implementations of MAC

There are many implementations of Mandatory Access Controls (MAC) based systems, but one of the most popular is called SELinux (Security-Enhanced Linux) [4]. It was designed and engineered by the National Security Agency, an intelligence-gathering organization belonging to the government of the United States. Released in 2000 as a patch to the Linux operating system, SELinux quickly gathered popularity within the Linux community due to its structural simplicity and the impeccable credentials of its designers. It now exists as an open-source module transparently integrated into the Linux OS kernel. Optimistically, the casual user may receive security support from SELinux without even noticing that the module is active, due to generic security configurations created by some of the recently arisen SELinux support groups.

SELinux shares two fundamental properties with many MAC systems. Firstly, the super-user concept of DAC systems (i.e. “root” or “Administrator”) is banished, so that all users of the system are controlled by the same configuration policy. The policy is written by a security administrator. If an attacker can acquire the privileges of the security administrator, then the policy can be changed to suit his or her ends. But unlike the super-user, who controls any aspect of the system, the security administrator exists only to secure the machine, and not to make use of it.

Secondly, like some other MAC systems, SELinux is based upon the concept of “type enforcement”, in which all the objects in an operating system (be they files, network sockets or processes) are labelled and classified as “domains” or “types”. The system configuration determines how domains with label  $x$  are able to access types with label  $y$ . Typically the access rights will be described, in a broad sense, as “domain  $x$  can read type  $y$ ”, “domain  $x$  can write to type  $y$ ”, “domain  $x$  can execute type  $y$ ”, combinations of these, or “domain  $x$  cannot interact with type  $y$  in any way”. This last option is the implicit default, so only positive relationships between domains and types need to be configured.

Nevertheless, the flexibility provided by SELinux tends to be its undoing, at least from the perspective of the casual user. Because there are many domains and types within even a single-user system, and because each possible positive interaction needs to be considered, configuration of access rights is a laborious and error-prone process. Add to this the fact that each system is guaranteed to be different to the one on which the prepackaged configuration was prepared, and a nightmare scenario in which SELinux denies essential accesses - such as allowing the system’s Graphical User Interface to start - becomes a common one [5].

There are well-known strategies that can help to reduce configuration complexities. One of the most popular of these is Role-Based Access Control (RBAC), which intersperses a “role  $r$ ” into the relationship between domain and type, such that, for example, if “role  $r$  can read type  $y$ ” and “if domain  $x$  is a member of role  $r$ ”, then “ $x$  can read  $y$ ”. Since there are a small number of roles relative to the number of domains and types, then the number of rules relating roles to types and domains to roles should be much fewer than the number of rules that relate domains directly to types. SELinux supports a primitive version of RBAC, yet a typical SELinux configuration file still runs to about 50,000 lines.

The RedHat company sells RedHat Linux Enterprise and sponsors the “Fedora Core” open-source software activity, both of which sport an extension of SELinux that includes “strict mode” and “targeted mode” structures. Strict mode is no different to the “vanilla” version of SELinux, but targeted mode protects only a subset of domains and types, usually those which have interaction with the external world via network sockets, etc. (that is, those objects which are most likely to be attacked by hackers). The remaining objects within the system are labelled “unconfined” and can “run amok” with only the discretionary access controls regulating their behaviour. RedHat ships SELinux in the default mode of “targeted”,

so that basic protection is afforded to the system without the mechanism becoming invasive, in turn preventing the user from being productive, and swamping RedHat support with basic administrative support requests. The flipside to this is that RedHat does not offer support to issues arising from the strict mode of SELinux. As will be seen later, this has a substantial impact on the use of SELinux to protect medical or other application data.

## Protecting Medical Application Data

The primary intention of SELinux is to protect objects embedded within the operating system, with security of application data being an afterthought. SELinux in effect partitions the operating system space into a set of “sandboxes”, protected areas between which communication is tightly regulated. The mechanism is generic, and consequently, the security administrator can create an additional series of sandboxes at the application level to protect medical and other kinds of data. For example, the administrator may configure a web-browsing sandbox that permits a web browser such as Internet Explorer or Mozilla Firefox to access the internet. In addition, the administrator may also configure a medical-related sandbox in which a medical application is permitted to access medical records. However, unless explicitly permitted, the web browser does not have access to the medical records. Neither does the medical application have the same level of exposure outside the network as the web browser. The security administrator can create arbitrary levels of complexity in the application layer by constructing sandboxes for different applications, yet the enforcement mechanism of SELinux treats them all equally and prevents unauthorized accesses. Whereas if a hacker attacked a DAC system through the network interface, and managed to acquire super-user permissions, in an SELinux scenario, the hacker would control only a single sandbox, and would need to launch additional exploits, each of which became increasingly infeasible with distance from the network interface.

An important caveat is that the “targeted” mode of Red Hat Enterprise Linux and Fedora Core does not permit application-level sandboxes, because all application processes run in the unconfined domain. Any system supporting application level security is compelled to run in strict mode, which in turn means that it is likely not to be fully supported by its commercial vendor.

## Building an SELinux Proxy

Application data tends to be much more dynamic and flexible than operating-system level data. There may be many users of an application level database, whereas the number of owners of operating system processes tends to be very small. By default, SELinux is configured for four users, including system, staff, sys-admin and ordinary users. Adding new users involves recompiling and reloading the configuration policy, as does adding new rules for interactions between domains and types. As operating-system level relationships tend to be very static, for example, changing only when new software is in-

stalled, this is not especially disadvantageous for the normal use cases for SELinux but is not well suited for creating rapidly changing sandboxes.

Our solution to this problem, which also avoids the problem of creating additional complex interactions between application and operating system level objects, is to create a proxy. The proxy runs at the application level and is secured in its own sandbox by SELinux, preventing unwanted interactions with other processes. The proxy regulates access by application-level process to protected data, using its own set of configuration files. In one sense, this solution can be viewed as nested SELinux, whereby the proxy represents a micro-instance of SELinux that deals only with application data. Operating system level processes see only a monolithic object (the proxy) representing application processes, meaning that the number of configuration rules between the two layers is linear rather than multiplicative.

The proxy deals with the added levels of interaction complexity at the application layer by using an enhanced version of RBAC, in which role permissions are inherited throughout a hierarchy. By collating roles into hierarchy, and associating the lowest member of each hierarchy with each type, this obviates the need to associate every role with every type. As an example, a vertical slice of a role hierarchy may consist of “Doctor is a subset of role Clinician” and “Surgeon is a subset of role Doctor”. Configuring the policy with “any user in the role of Clinician has access to type y” automatically covers the rules for “any user in the role of Doctor has access to type y” and “any user in the role of Surgeon has access to type y” by virtue of their membership of the family. Portions of the hierarchy can be overridden: configuring “any user in the role of Surgeon does not have access to type y” does not cause a contradiction but allows only Clinicians and Doctors access to type y.

An option in this research was to build the extended RBAC functionality natively into SELinux for which the source code is freely available. However, the benefits to operating-system level objects, which are not ordered hierarchically, are unlikely to outweigh the disadvantage in branching the SELinux source code, consequently reducing the successful uptake of this solution.

The mechanism by which the proxy works is very simple, and abstractly mirrors the SELinux mechanism. A client interacts with the proxy via a pair of Client and Server messages. For each client message received, the proxy sends exactly one server message.

The client authenticates itself to the proxy using a client message with type CREDENTIALS and with a payload containing the user, role and password that describe the client. Until the next such message is received, the proxy caches the credentials. This mimics the SELinux mechanism, which authenticates a user via a password before transitioning the user into the requested role. The proxy generally responds to credential messages by sending a dummy OK response.

The credentials are evaluated whenever the client requests access, either a read or a write, to a record in the proxy data-

base. The proxy passes the credentials, along with the record identifier and the policy to the security filter. The security filter assesses the credentials, decides whether the record can be accessed in the way intended and passes this decision to the proxy. In the case of a read request, the proxy relays the appropriate record back to the client. If the client has requested a write, then the material passed in the payload of the REQUEST\_WRITE\_FILE message is appended or overwritten to the record.

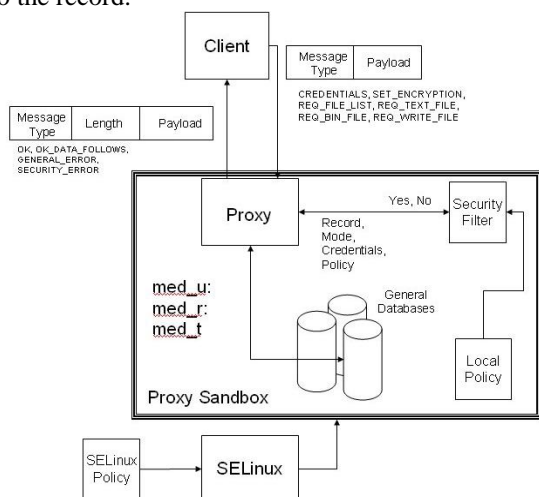


Figure 1 - Architecture of the SELinux Proxy

Whereas SELinux can protect data to the granularity of the file, the proxy has arbitrary granularity, as determined by tags exchanged between the proxy and its client. The client may wish to retrieve a single word from a database, or an entire collection of files. Our mechanism allows this with as little as a single configuration, although for more complex cases, the number of configuration rules will increase linearly in the number of database items.

There are some cases when records must be accessible even in the absence of legitimate credentials. For example, if the authorized viewer of a patient's case file is not present, but the patient requires emergency treatment, then the availability of the information is more important than its privacy. So the proxy is programmed to respond to a special role of "Emergency", in which case it moves into auditing mode, until a new set of credentials with a differing role is provided. In auditing mode, all records can be retrieved and modified, but each action is recorded and flagged for review by the security administrator. Appropriate punishment for abusing this mode can be metered out at a social level. Our prototype does not handle differential records, whereby the deltas between subsequent versions of records are stored, although this would be advantageous for malicious or accidental modification of records in auditing mode.

It is not essential for the proxy and the client to maintain an encrypted channel, since access control on the channel can be maintained by SELinux. For ease of configuration, all communication can be encrypted using commonly available algorithms such as the Advanced Encryption Standard. Our research did not consider key management issues between the client and the proxy, although the usual public key establishment protocols, such as Diffie-Hellman can be used.

To prove the effectiveness of the proxy, we developed a simple prototype of the proxy and a client, as shown in Figure 2. Auditing data for the client is shown in Figure 3. We used the proxy and client to demonstrate the security advantages of SELinux over DAC-based systems such as Windows XP. In DAC-based systems, it was relatively easy to use hacking tools such as rainbow tables [6] to break weak Windows system administrator passwords, and modify the proxy and client code to allow unauthorized and unaudited accesses. As the proxy was housed in its own sandbox under SELinux, traditional hacking tools did not provide an avenue for breaking into or changing the proxy. The issue remains that this security is present only in the unsupported "strict" mode of SELinux which is still too complex to deploy in commercial situations.

Although the proxy significantly simplifies configuration of application data, it does not address problems at the operating-system level that need to be resolved. Further research in this area needs to focus on simplifying generic SELinux configuration, to allow realistic deployment of "strict" SELinux, which supports protection of application data. This is indeed happening, as witnessed by the development of modular policy logic in Fedora Core 5, which allows the configuration to be developed and loaded in blocks relating to the processes or daemons being protected. The efficacy of this strategy has yet to be solidly determined.



Figure 2 - The Proxy Client

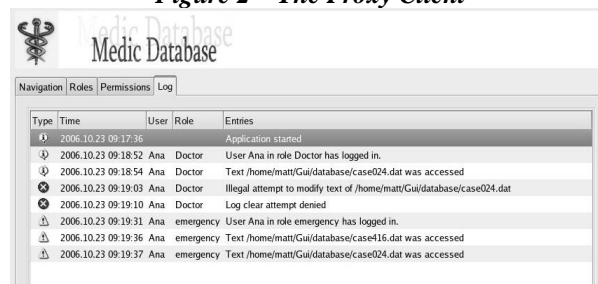


Figure 3 - Auditing data for the Proxy Client

## Conclusions

Sufficient evidence is emerging that the security requirements and obligations for the protection of sensitive health data cannot be sustained using contemporary data access control and protection mechanisms in current commercial, commodity

computer systems. “Mandatory Access Control” or MAC, incorporated into basic operating systems and allied supporting software structures, provides an alternative, strict, security policy driven approach far superior to industry standard DAC mechanisms. MAC can strengthen protection from unauthorised access to sensitive health related information from both outside and inside an organization. This provides enhanced privacy protection from staff, including knowledgeable ICT professional staff members, gaining access to such sensitive data for which they are not authorised (i.e. view, modify, copy, transmit, delete, etc.). It further provides enhanced ‘boundary’ security from outside intrusion whereby adversaries, such as hackers and spyware operatives, are unable to gain full control of an information system. In the MAC case it can be demonstrated that damage can be limited to violation of an individual user’s account [7].

This research has found that a MAC based medical data systems, although viable, still presents some key research and practical deployment challenges. In particular, the “strict” operational mode offered by SELinux may be seen as being too rigid for deploying Role-Based Access Control or RBAC structures with the required levels of flexibility needed in practical healthcare situations. Without this flexibility, system reconfiguration may be required each time a user is added or removed. This is infeasible in practice and is already the subject of a number of active research projects. It was shown with the demonstrator described in this paper that a compromise can be derived that provides an application level proxy to facilitate a secure, role-based access interface. A balance has to be struck between strict access control security and the degree of flexibility for dynamic modification of any system in the “real world”. Any approach taken should be determined from a privacy impact oriented risk assessment process. For example, such an assessment might readily determine a need for emergency over-ride capability to enable at least wide read-only access to medical/health data. Such a facility would, however, have to be subject to new audit and control requirements as well as to limitations potentially related to time periods and the location of users. In this regard an implementation that can support dynamic reconfiguration in a manageable and understandable manner may be essential. Earlier MAC systems were simply not designed for this environment where the security policy may need to be dynamic, not just in detail but also in structure.

Future needs in terms of research also involve a better understanding of the complementarity of SELinux’s concept of “type enforcement” versus more traditional security structures based around hierarchical “Multi-level Secure” or MLS schemes. In the health information area it needs to be determined whether or not such hierarchical security schemes have a place or not and, if so, to what level are modifications of the basic concepts involved necessary. Likewise, the concept of “compartmentalization”, reflected in the SELinux type enforcement system, needs to be assessed in relation to its suitability for all levels of information services needed in a nationwide health information structure. At the same time, application software development needs to become aware of the new parameters afforded by the MAC facilities and determine to what level such

applications may or may not make use of the security mechanisms and services offered, i.e. to determine the distinction between what may be labelled as “security aware” versus “security ignorant” applications. Moreover, the integration of existing software systems into this environment must be understood requiring further research into appropriate techniques for system integration in higher security environments. In turn, this places new demands on education and training as ICT professionals need to develop the skills needed to understand, utilise and manage this new environment. This indicates that necessary or desirable changes in organizational policy and management structures for healthcare providers may be also needed and, at present, full guidance to policy makers and operational management in relation to deployment of newer MAC based overall information systems do not appear to exist. This leads to the need for further research and experimental system development in the area to enable study of the economic, cultural, social and legal responses required.

## Acknowledgments

This research was supported under the Australian Research Council's Special Initiative on e-Research funding scheme (project number SR0567386). The authors would like thank RedHat (Asia Pacific) Ltd. and the E-Health Research Centre (a joint CSIRO and Queensland Health initiative) who have been highly supportive.

## References

- [1] Foster I and Kesselman C, “The Globus Project: A Status Report”, Proceedings of the Seventh Heterogeneous Computing Workshop, 1998, ISBN: 0-8186-8365-1.
- [2] Loscocco, P. A., S. D. Smalley, et al. (1998). The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments. Proc. 21st National Information Systems Security Conference, Crystal City, VA.
- [3] Loscocco, P. A. and S. D. Smalley (2001). Meeting Critical Security Objectives with Security-Enhanced Linux. Proceedings of the 2001, Linux Symposium, Ottawa.
- [4] National Security Agency. Security Enhanced Linux homepage. Available at <http://www.nsa.gov/selinux>, 2000.
- [5] Thompson, K. Does SELinux support X-Windows? <http://www.crypt.gen.nz/selinux/faq.html#CP.7> Nov. 2006.
- [6] Oeschlin, P. Making a Faster Cryptanalytic Time-Memory Trade-Off, Proceedings of Crypto 2003, pp 617-630, 2003.
- [7] M. Henriksen, W. Caelli and P R Croll “Securing Grid Data Using Mandatory Access Controls”, to appear 5th Australasian Symposium on Grid Computing and e-Research (AusGrid 2007) Ballarat, Australia, Feb. 2007.

## Address for correspondence

Prof. P.R Croll, Information Security Institute, QUT, 126 Margaret Street, Brisbane 4001, Australia. Email: [croll@qut.com](mailto:croll@qut.com)