Securing Grid Data Using Mandatory Access Controls*

Matt Henricksen **William Caelli** **Peter Croll**

Information Security Institute,
Queensland University of Technology,
GPO Box 2434, Brisbane,
Queensland, 4001, Australia.
Email: {m.henricksen, w.caelli, p.croll}@qut.edu.au

Abstract

The main contribution of this paper is to investigate issues in using Mandatory Access Controls (MACs), namely those provided by SELinux, to secure application-level data. Particular emphasis is placed on health-care records located on the grid. The paper disccuses the importance of a trusted computing base in providing application security. It describes a secure three-tiered architecture, incorporating trusted hardware, SELinux, and application security mechanisms that are appropriate for securing sensitive application data.

1 Introduction

It is well known that security in the application space cannot exist without particular security prerequisites at levels of hardware, operating systems and any middleware sub-systems (Loscocco, Smalley, Muckelbaur, Taylor, Turner & Farrell 1998). These prerequisites include domain separation, which prevents unrelated system components and applications from interacting; protected input, which stops eavesdroppers from electrically reading data generated by peripherals or stored in memory; and the enforcement of least privilege, in which a user or process holds no more permissions than those required at the time.

These prerequisites are usually absent in commodity-level operating systems, which usually have only two distinct levels of privilege: the system administrator (or superuser) and the ordinary user level at which most applications run. The system administrator has control of the system, including the ability to execute kernel level processes and restrict the permissions of the ordinary users. The separation of privilege between administrator and ordinary user is enforced by Discretionary Access Control (DAC). In DAC, the security policy is partially written by the ordinary users, who have the capability to change the access controls on their data and program files.

By restricting the privileges of the ordinary user, the potential damage able to be caused to the overall system by the ordinary user should also be restricted. Malicious or accidental compromise of the ordinary user's account should not affect other users or the operating system itself. That is, the user's activities should occur in a sandbox comprised of the objects and resources available to that user. In a well-secured sandbox, resources are tightly controlled. A problem affecting processes within the sandbox cannot spread beyond the sandbox boundaries. For example, code executed by a process within the sandbox cannot affect code or data in another sandbox.

Problems with the effectiveness of DAC are well documented, because it does not properly support sandboxing. An example is now given in which the ordinary user runs an instance of the Apache web server that serves web-pages to visiting WWW browsers (Gayal 2003) [1]. Tightly configured DACs make sensitive data inaccessible to the browser. When a visitor requests sensitive data from the browser, the browser returns a 403 'non-accessible' error. However, in a successful attack, a malicious party launches a buffer overflow attack on the Apache process. Through the use of the process tracing tool ptrace, which controls the execution of other processes, the attacker injects malicious code into a kernel child process with system-wide access. The attacker uses the power of the kernel child process to create a new account with super-user powers, from which the sensitive data, and in fact any object located on the DAC system, are accessible. This attack succeeds even though the attack point - Apache - has only ordinary-user privileges, and has no business executing shell code at the kernel level. This attack cannot succeed in a system deploying Mandatory Access Controls (MAC) because no process in the Apache sandbox has the ability to execute in the ptrace or kernel sandboxes; the buffer overflow code injected by the attacker will be refused permission to execute in the kernel sandbox unless explicitly configured to do so. And of course, processes in the Apache sandbox have no rights to configure anything except for Apache configuration. An attacker who gains control of the Apache process has made no leeway in controlling ptrace, and therefore the ability to gain super-user privileges. This attack is summarized in Figure 1.

The security problem caused by the combination of DAC and the two-tiered system of authorization is that it is difficult to implement the principle of least privilege, which stops the chaining of attacks just described. Discretionary access controls are adequate for passive management of privacy, but are insufficient to protect systems from malicious attackers (Wright, Cowan, Morris, Smalley & Kroah-Hartman 2002). Of course, when a network is connected to the grid, there is no shortage of attackers.

In this paper, we evaluate the suitability of Mandatory Access Controls (MACs), defined and ex-

*This work was funded by ARC special initiative on e-Research: Croll, Caelli, Narashimhan, Soar, "Mechanisms for Ultra-secure Access to Large Repositories of Sensitive Data over the Grid", 2005/6, ref. SR0567386

[1] This attack is simplified. In reality, the attacker needs a local account, but details are generalized to aid clarity.

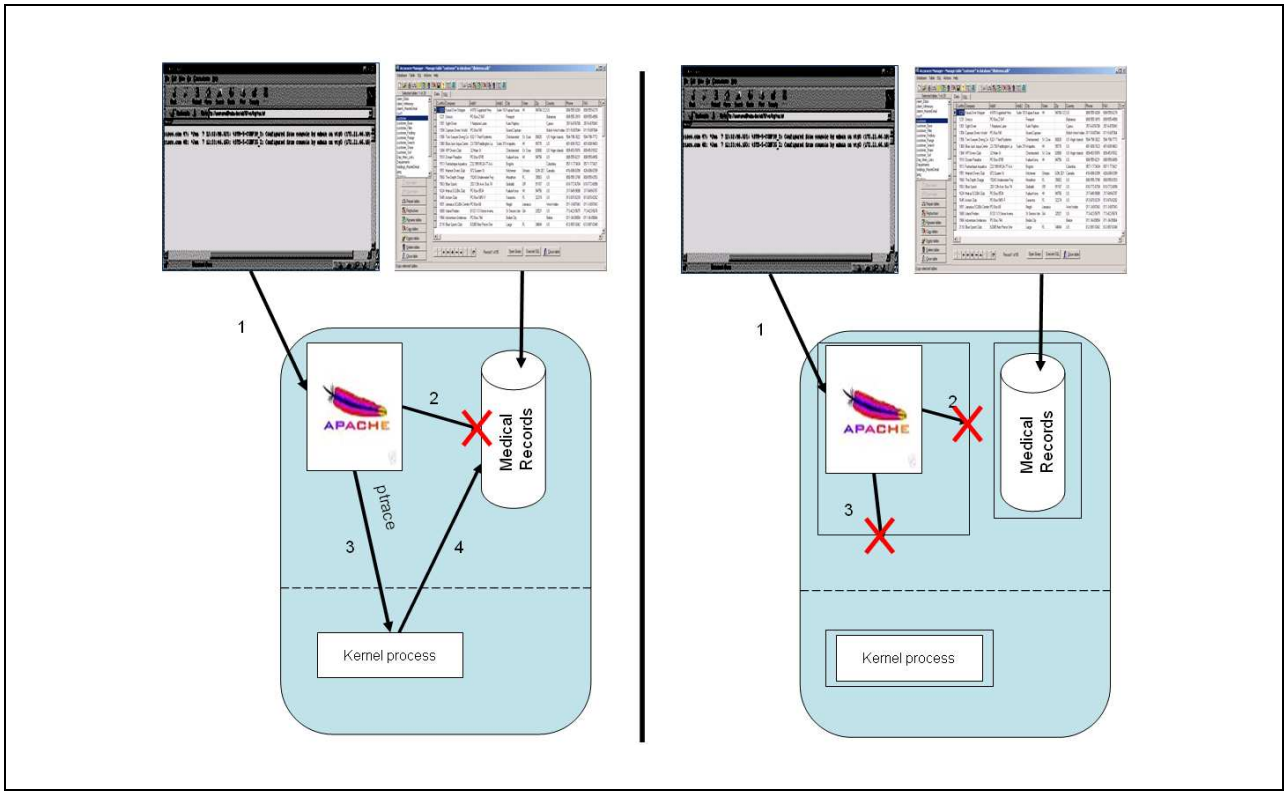

Figure 1: MAC-enforced sandboxes. In the left diagram, the attacker connects to the web server(1). DAC restrictions prevent access to medical records (2). So the attacker launches serial attacks on the web server and kernel processes using ptrace (3), gains root access, and has access to any file on the host (4). In the right diagram, the attacker connects to the sandboxed web-server (1) but is unable to access anything outside of the sandbox (2)(3). Note that in both cases, the authorized medical application can access the medical records.

plored in Section 2, particularly within the context of the e-health scenario defined in Section 3. This scenario is based on the ARC funded research that looks at the use of grid technology to facilitate access to large repositories of sensitive data where security and privacy concerns are of paramount importance. We describe suitable MAC implementations, including the Linux subsystem SELinux, in Section 4, and use these technologies to construct a three-tiered architecture in Section 5. Finally, in Section 6, we indicate future work on this project, and give some concluding remarks for others to consider when utilising SELinux for building trusted computing frameworks for use with grid architectures.

2 Mandatory Access Controls

Mandatory access controls provide for a centralized, enterprise-level security policy that is configured by a nominated and authorized security administrator. All users are equally bound by the policy, and there is no longer the concept of an all-powerful super-user as exists in DAC systems (for example, the Administrator group in Microsoft Windows, and the root user in Unix or Linux). By implementing the principle of least privilege and enforcing domain separation through sandboxes, MACs are able to provide a more robust system against attackers.

2.1 Multi Level Systems

The origin of MAC concepts originated in Multi-Level Security (MLS) models for information systems. These systems permit the control of information access by labelling subjects (processes) and objects (files, sockets, etc) with security levels or parameters. The label of a subject is assessed against the label of the object before interaction between the two is permitted. In inappropriate cases, as defined by configuration, no access will be allowed. Two famous examples of MLS models are Bell La-Padula (Bell & La-Padula 1976) and Biba (Biba 1977).

Bell La-Padula The Bell La-Padula model was developed by the U.S Department of Defence in 1976 to enforce confidentiality requirements. The model defines two mandatory access control rules. The first of these is no read-up: a subject of a given security level may not read an object with a higher security level. The second rule is no write-down: a subject of a given security level may not write to an object of lower security level.

The model supplements these rules with discretionary access control, but is widely criticized for being inflexible in non-military domains. In particular, it ignores intransitivity (if a process at level A can read at level B, and a process at level B, can read at level C, does it follow that a process at level A can read at level C?) and dynamic separation of duty (a user may be allowed to operate at level A under some conditions, but only level B under others). Like many MLS systems, Bell La-Padula cannot deal efficiently with sanitization, in which the top-secret elements of classified data are removed, and the remainder presented to lower levels of security. Bell La-Padula does not permit flexible policies. For example, in a medical context, assume that a doctor is accorded a higher level of security than is a nurse. The Bell La-Padula model prohibits the doctor from sharing a medical record encoded at his level of security with the nurse, even though an urgent need may arise to do so. Even by verbally communicating such information during emergency surgery, the doctor is violating the said security policy. On the other hand, the model does

not prohibit a new intern at the lowest security level from writing inaccurate, and possibly life-threatening information upwards to a more classified level.

Biba The Biba Model describes a type of system that ensures data integrity. Processes and objects within the Biba model are given integrity labels. Low-integrity processes cannot write to high-integrity objects. High-integrity processes cannot read objects of lower integrity. In a variation known as low watermarking, which increases the flexibility of the Biba model, a high integrity process can read lower integrity objects, but is demoted to the security level of that object until manual reconfiguration. In a further modification, some processes are classified as trusted in which case they can read objects on a lower level without any penalty.

The Biba model addresses the problem of integrity described above, in that the nurse is prohibited from writing inaccurate information upwards. However, the model still presents a severe degree of inflexibility in that the doctor cannot access the notes made by the nurse!

2.2 Role Based Access Controls

Role-Based Access Control (RBAC) is one way of alleviating configuration difficulties and complexities in the practical use of MACs, which arise from their flexibility and fine-grained level of control.

Multi-level systems directly associate users with object permissions; for example, the policy configuration may consist of statements of the form "[User] acts upon [Object]". At its simplest, RBAC (Ferraiolo & Kuhn 1992) intersperses this structure with a middle layer, that of the "role". A role allows collective rather than individual association of users and permissions of the form "[Role] acts upon [Object] where [User] is a member of [Role]". Advantages over traditional MAC technologies include: a reduced number of associations, easing configuration; a lower rate of change between roles and permissions, relative to users and permissions, making revocation of multiple permissions trivial; and increased flexibility relative to standard MLS policies - arbitrary groups of users can be assembled into roles, in which case the doctor communicating with the nurse problem is solved.

There are four basic forms of RBAC (National Institute of Standards and Technology 2003). RBAC_0 is the simple form outlined above.

RBAC_1 allows inheritance of permissions within a role hierarchy. For example, if a clinician is assigned permissions to a patient's case, then the permission is implicitly transferred to the nurse role and doctor role when they are considered to be subsets of the clinician role. Alternatively, a permission given to the doctor role is not explicilty awarded to the nurse role since they are disjunct. The implicit assignment of permissions within the hierarchy has the potential to greatly simplify policy configuration. Where there are u users, r roles, and p permissions, then RBAC_1 simplifies (in the worst case) a configuration complexity of $u \times p$ to $r \times (u + p)$ where $r \lll u$ and $r \lll p$.

RBAC_2 does not support inheritance, but implements static constraints, which can be imposed to prevent users from joining roles for which they are not qualified, or for which there is a conflict with other roles of which they are also members. It also allows dynamic constraints, which permit users to belong to multiple roles, but specify that only a subset of those roles may be active at one time. More complicated dynamic constraints may also be considered at runtime.

RBAC_3 represents the union of RBAC_1 and RBAC_2. The use of inheritance in RBAC_1 and RBAC_3 is one of the most useful features of RBAC, but unfortunately is infrequently implemented.

3 An E-health scenario

Within our research, we consider a scenario in e-health for securing application data. Many of the mechanisms used to secure health data can be generalized to the context of any sensitive data. Nevertheless, securing health data has its own set of priorities. Unauthorized disclosure of health information can have serious consequences including, aside from personal embarrassment, refusal of prospective employment, difficulties in obtaining or continuing insurance contracts and loans, and ostracisation from family and community groups (Rindflesich 1997). Once information has been disclosed, the damage cannot be undone.

In our scenario, we consider the ability of the scientific community to perform research on large repositories of sensitive data, which may be maintained in a logically and geographically distributed fashion. For example, these repositories may be accessible over a data grid . The access to these sets of data is subject to varying degrees of legal, social and ethical constraints. Individual personal health records offer significant value for medical research but the information cannot be readily accessed because of privacy legislation, lack of confidence in the security of electronic records, and the requirement to maintain end-users trust in the overall healthcare information system (Croll & Croll 2004).It is permitted in Australia to use health data for the secondary purposes described above provided that it suitably depersonalized. This generates the requirement to supply depersonalized data directly from the identifiable data source without compromising the original database.

The basic architecture for a secure mandatory access control system that provides access to a distributed system such as a data grid is shown in Figure 2. The goals of this system are: a) to provide authorized access to secure data within the grid nodes and; b) to allow dispersal across the grid nodes of depersonalized data, in which the identifying aspects of the data (such as a patients name and address) have been removed. This system is based on a Trusted Computing Base (TCB) in which MACs regulate data access through policies initiated by the organizations security architecture. This architecture is in turn derived from high level formulations based on the legislation and regulations that apply to the organization. The enhanced security of this system comes from the fact that systems managers or maintenance staff cannot change the access rights on-the-fly or, because of strict auditing, cover their tracks if they make such changes.

The protected data can be accessed via encrypted channels connecting the trusted computing nodes. The set of data that is available depends upon the access policies determined by MAC through the Policy Enforcement Server. For example, permitted depersonalized data could be decrypted by shared applications on the open access grid while highly classified data would only be available for access and processing on other trusted computing nodes.

An override function has to be accommodated whereby a clinician can get access to personal data in extraordinary circumstances, provided it is for the well being of the patient. There are times in cases of national emergency or if an identifiable patient arrives unconscious in an emergency unit that this override would be appropriate without first having to gain consent from the individual. In both situations, there

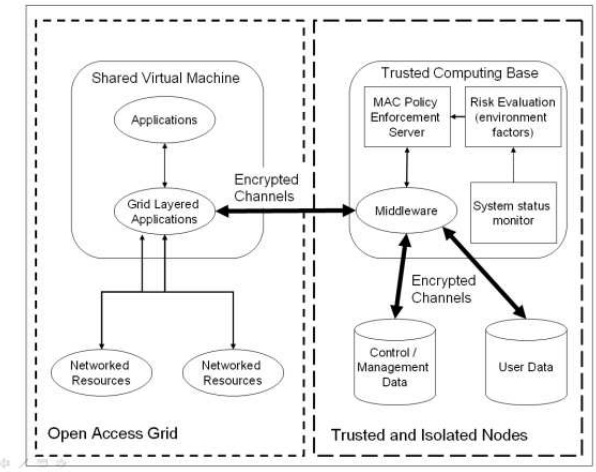

Figure 2: A Trusted Computing Base for Protecting Grid Data

would not be time to rewrite the security policies but the ability to switch to another policy set to suit the scenario would be highly beneficial. This is facilitated by the Risk Assessment Unit shown in Figure 2. Irrespective of the level of security, all underlying activities are audited.

Consequently, the requirements of this scenario with regard to MACs are: 1a) high-grained level of control; 1b) ability to securely access depersonalized data (ie. to change the security level of blocks of data); and 1c) ability to change policies in abnormal circumstances. Software systems and applications to date were not developed with such requirements in mind, but have evolved more from a corporate ownership model, where high level officials can access all the data owned by an organization. These requirements will be examined again in Section 5, in terms of the architectural solution that realizes them.

4 State of the Art in Mandatory Access Controls

The security of application data does not just depend upon the software measures taken at user level. The use of MACs to secure data involves a multilayered approach. This Section describes a selection of contemporary technologies employed at each of the levels: efforts at the hardware layer are described in Section 4.1; at the operating system kernel in Section 4.2; and at the application layer in Section 4.3

4.1 Hardware Support

Irrespective of the measures taken by a computer's operating system or application software, the security of the platform fundamentally depends upon the trustworthiness of the hardware. Off-theshelf general purpose computers, implemented using Intel x86 processors have had extensive hardware support for security for over twenty five years. These features include memory address segmentation for prevention of illicit memory access between applications, and applications and system software components; strong memory typing including separation of data, code and stack memory; and protection rings that limit CPU facilities. Recently, however, some new additional hardware components, such as the Trusted Computing Group (TCG)s Trusted Platform Module (TPM), have been created that offer protected cryptographic functions in support of overall system security. Additionally some suggested basic changes to CPU structure by Intel Corporation and others also support operating system security functionality and enforcement.

The Trusted Computing Group (TCG)'s vision of a trusted computing base is shaped by its Trusted Platform Module (TPM) (Trusted Computing Group 2006), a tamper-evident (but not tamper-resistant) chip that is added to the motherboard of secured systems. The TPMs ability to support MACs derives from its ability to store an embedded endorsement RSA key pair signed by the manufacturer. The private RSA key is never transmitted outside of the TPM, and the assurance of the trusted platform is guaranteed while the TPMs private RSA key remains undisclosed. During the boot process, the TPM generates cumulative hashes of BIOS, low level drivers and the kernel. The final hash is matched against a pre-calculated value stored elsewhere in the TPM. If the values match, the TPM releases a key that is used to decrypt crucial parts of the operating system, enabling the booting process to complete. If the hash does not match the PCR value, perhaps because the kernel code has been illicitly changed, then booting cannot continue, and the system is inaccessible; therefore the sensitive data stored on the system cannot be corrupted or distributed.

This process can be generalized to remote attestation, in which a third-party queries the software state of the machine before releasing a decryption key to protected applications. For example, we can verify in advance the identity of recipients to whom we send sanitized data or identify the secure state of the machine and its medical applications before we distribute medical records.

While the TCGs vision for a trusted-computing base is realized through the architecture-independent TPM, Intel Corporation's LaGrande CPU program is reported to involve modification to specific Intel processors and chipsets, and incorporation of TCG's TPM (version 1.2) (Intel 2003). LaGrande's most prominent security feature is domain separation through virtualization, which allows multiple operating systems and applications to run independently and without awareness of each other. The TPM provides remote attestation. In conjunction with operating- system support from such systems as Microsofts suggested Next Generation Secure Computing Base (NGSCB), LaGrande provides coarse-grained mandatory access controls. Under LaGrande, protected and unprotected software can be run simultaenously on te same trusted computing platform without interference.

4.2 Operating System Support

MACs implemented in application space can easily be bypassed without underlying support in the privileged kernel level. Consequently there are many implementations of MACs embedded within operating system kernels. Many of these implementations are purely research-based, or targeted at niche operating systems. One of the more prominent is SELinux (National Security Agency 2000), which was developed by USAs National Security Agency (NSA) and released to the public in 2000 as open source on the Linux platform.

SELinux is based upon type enforcement, which views the system as two sets of entities: one active, termed either subjects or domains; and one passive, termed objects or types. The domains and types are associated with a permission set defined within a domain definition table (DDT). The entries within the

DDT specify which processes can read, write or execute objects listed within the DDT. A separate table, termed the Domain Interaction Table (DIT), indicates transitions between domains. In a health-care context, the domain may be a medical application. The objects are network sockets that allow communication outside of the trusted computing base. As with most of the MAC systems at the operating-system level, the level of granularity does not extend further than the level of the file, necessitating further control mechanisms at the application layer.

SELinux enforces a separation between policy logic and security enforcement, which improves expressiveness. SELinux uses a policy-enforcement server (PES) that interprets the advice given by the security server (SS). The PES mandates the interaction between domains and objects such as files, directories, other processes, descriptors, sockets, etc. The separation of policy-enforcing and policy-configuration components localizes the decision point for changing policies when the circumstances that dictate those policies change, as required by the risk assessment unit of our scenario.

To determine whether a particular action is allowed between a domain and object, the PES retrieves their contexts from the context database and passes them to the SS, which returns a set of access vectors. An allow vector determines the type of operations that can be performed by the process on the object. An allowaudit vector indicates the type of auditing that should occur. Each vector contains a bitmask denoting the permissible actions. These are commensurate with standard Linux permissions, including append, create, execute, get attribute, io control, link, lock, read, rename, ulink and write, etc. The PES relays these vectors for interpretation by the Linux Operating System, which either permits the requested action or denied it. Without an appropriate allow vector, the action will automatically be denied.

SELinux uses a variant of $RBAC_2$ to associate individual users with specific types, and to disassociate individuals operating in different roles. The standard form of the domain and object contexts are [User with Role owning Type]. This allows the system to differentiate between Doctor W accessing the internet using a web-browser, and the same Doctor W using a medical application. In the former case, sensitive medical records pertaining to Doctor Ws patients should be inaccessible. So although the same user is involved, the combination of user and roles, in which he or she is involved, is not. Role hierarchies have been previously proven useful in flexibly defining record access in medical scenarios (Reid, Cheong, Henricksen & Smith 2003) but are unsupported in SELinux.

The SELinux configuration contains four default roles and three default users. The roles include 'sysadm_r' (system administrator), 'staff_r' (potential system administrator), 'system_r' (system process), and 'user_r' (non-privileged user). SELinux maintains a separate user system to the Linux account system. The default users mirror the hierarchy in standard DAC systems, and include 'root_u', 'system_u', and 'user_u'. Additional users can be defined, but this requires the policy to be recompiled, a lengthy and inconvenient process that reflects the role of the module in protecting slow-changing kernel data rather than dynamic application data. In distributed, large- scale systems, the need to recompile with the introduction of new roles and users is a serious impediment to the success of the system in protecting application data.

The flexibility provided by SELinux, at the fine-grained level of files, sockets, etc, means that configuration is a tedious task even for the hardening of the Linux kernel. The average configuration file is 50,000 lines long, although NAI Labs provides an example configuration file for standard kernels, which can be modified on a component-by-component basis. Nevertheless SELinux's high-grained flexibility and architectural modularity proves promising for securing application data.

4.3 Application-level Support

One of the more recent proposals for application-level MACs comes from the Self-Defending Objects (SDO) proposal of Holford et. al (Holford, Caelli & Rhodes 2003). In that paper, SDOs are portrayed somewhat optimistically as an extension to object-oriented programming (OOP). In reality, the descriptive phrase 'design pattern' is more apt.

In OOP programming styles, security checks are scattered throughout the source code in a pragmatic, possibly adhoc style, which Holford et. al claims leads to unchecked and potentially critical programming errors. For example, the authors claim that the following code snippet is insufficient.

```
if (SecurityClass::check_access(
        obj, credentials) == OK) {
   /** entry into this block only
        when the correct credentials
        are supplied */
   obj->accessor();
}
```

The criticism is that the compiler and object library do nothing to prevent the coder, in a fit of absent-mindedness, from calling the accessor without the enclosing security check.

Vasanta et. al (Vasanta, Holford, Caelli & Looi 2004) propose that one SDO be responsible for each security-sensitive device. Security checks protect the sensitive data encapsulated by the SDO. More specifically, the data is marked as private within the object-oriented language. The data is accessible through accessors or mutators that contain the security checks in the form of a preamble, as described in the pseudocode below.

```
class Obj {
    final T accessor(credentials) {
        /* security pre-amble */
        if (SecurityClass::check_access(
                credentials) !=OK) {
            throw access_exception;
        }
        /* standard accessor code */
        return sensitive_value;
    }
}

obj->accessor(credentials);
```

Successful authentication of the preamble sees the body of the accessor or mutator carried out; failure generates an exception, or in extreme cases calls the object destructor, which deletes the sensitive data from memory. There is no magic here.

Mandatory access occurs through the policy defined by the authenticator rather than the caller, under the assumption that the protected code is presented as a library, rather than as open source. In the latter case, then of course the code can be recompiled without the security checks.

By localizing the security checks and the associated data, Holford et. al hope for a reduction in the number security-related bugs, by virtue of the enhanced clarity of code. Thus SDOs do not provide additional security except through enhanced assurance. However, this approach leads to code tangling

whereby the purity of code, according to the principles of OOP, is lost. An object that is representing a factory is independent of an object that represents the security guard safekeeping it. Yet SDOs do not recognize this separation of duty. Furthermore, SDOs may more seriously violate the spirit of OOP, in that sensitive accessors and mutators need to be marked 'final' to avoid derived classes overriding the security checks. This at best has the potential to destroy or at least convolute the polymorphism of the respective classes, and at worst leads the security of SDOs to rest upon manual flagging of the relevant accessors and mutators. This represents only a partial solution to the problem that SDOs address: removing the error-prone component in calling security checks, wherease the 'final' qualifier necessary in SDOs is equally prone to neglect.

A single line accessor in standard OOP has the potential to become very complicated in an SDO. This problem may be solved using aspect-oriented programming paradigm (Filman 2004), but has not yet been successfully examined in this context. SDOs have the potential to become very inefficient, particularly when accessors and mutators reference multiple items of data, in which case multiple and possibly redundant checks are carried out. For example, an accessor 'C' that derives data from calling accesors 'A' and 'B' must undergo the same security check three times, despite the fact that the security context and credentials have not changed.

Furthermore, SDOs can not contain all malicious attempts by programmers to avoid security checks at the application level; for example, it is not difficult to modify the virtual function table of an SDO to bypass the check altogether. SELinux can reduce this problem by ensuring that the process that runs the application does not have write permission to the application; however, any data written by the application must in that case belong to a different sandbox.

The basic problem that SDOs address is laziness or carelessness of the programmer in calling security checks immediately prior to accessing a sensitive object. However this is fundamentally a compile-time problem that can be solved either by thorough code reviews, or by static analysis, for example by a security analog to the C 'lint' tool, or by a combination of both. Using static analysis at compile time means that sensitive accessors and mutators do not need to be declared 'final', removing the error-prone component of SDOs. Additionally, the static analysis tool adds complexity at compile-time but has no footprint at run-time, meaning that the inefficiencies of SDOs are no longer present. However, static analysis does not competely address the advantages of SDOs in a distributed environment, to which our grid-based scenarios belong. For example, the code and data may be migrated to a hostile environment in which static analysis has not been conducted. In such an environment, hardware and operating system support for SDOs must be present, even if application-level support is not. The realism of this kind of environment needs to be investigated. In any case, development of a static analysis tool to replace SDOs is left as future work. Consequently, for our grid-based prototype, we view SDOs as an interesting mechanism for localizing enforcing MACs at application level, particularly as a complementary mechanism to operating-system MACs.

5 Protecting Application Data using a Three-Tiered Architecture

In Section 3, we defined a scenario in which the securing of application data was deemed important. This involved a trusted computing base that had the dual purpose of maintaining sensitive data and releasing sanitized data onto the grid. The requirements of this scenario are MACs that possess a fine-grained level of control so as to implement the principle of least privilege. The MACs must be flexible enough to allow the export of sanitized data outside the trusted computing base. Furthermore, the policy must be able to be quickly altered under changed circumstances, such as a hacking attack upon the base, which requires the policy to be tightened, or in a pandemic, in which case the policy must become more flexible to allow appropriate medical treatment (for example, allowing the use of sanitized data in observing trends of a quickly spreading disease).

Our solution in providing a trusted computing base to protect data in this scenario spans three tiers: hardware; operating system; and application space. This architecture provides the appropriate level of security for both scenarios simultaneously, and is shown in Figure 3.

Hardware Layer The TPM proposed by the TCG forms one support for the hardware layer of the trusted computing base. It stores multiple decryption keys that are required to decrypt, respectively, the SELinux kernel, medical applications that access sensitive data, and the sanitizing software, which is responsible for allowing low-security forms of the data to be exported from the trusted computing base.

The decryption keys are released only when the trusted boot and remote attestation features deem that the security of the platform is maintained. For example, in the case that a virus undermines the security mechanisms of the operating system, it is still unable to access, manipulate or disseminate from the trusted computing base, the sensitive application data stored on the system.

We do not plan to fully implement the attestation logic within our prototype, but rather aim to use it as a proof-of-concept. Furthermore, when it comes to fruition, the Intel LaGrande CPU technology should form a part of this model through its implementation of hardware virtualization. This will increase the support given to domain separation of sandboxes in the application layer.

Operating System Layer The MACs in the operating systems layer are provided by SELinux in conjunction with the LSM framework on which is is based.

In our model, SELinux enforces domain separation at the application layer by creating sandboxes for the medical applications, including sanitization software, and their databases. The most basic sandbox in this architecture is that for the role of clinician. The security administrator can configure other sandboxes with more restricted memberships, based upon the privilege level of individual databases, or their data files. The medical application runs in the sandbox with the highest level of privilege accorded to its user.

SELinux's configuration process is not trivial, and across large systems, this can be a tiresome and error-prone administration task. In our architecture, we propose to implement extensions to the RBAC model of SELinux, to at least the level of $RBAC_2$ so that the policy of role inheritance within hierachies. This will allow our architecture to better handle the large numbers of related users within a typical health-care system.

SELinux is designed to secure operating systems which are initially configured on installation, and then only sporadically and infrequently when new key components are added to the operating system.

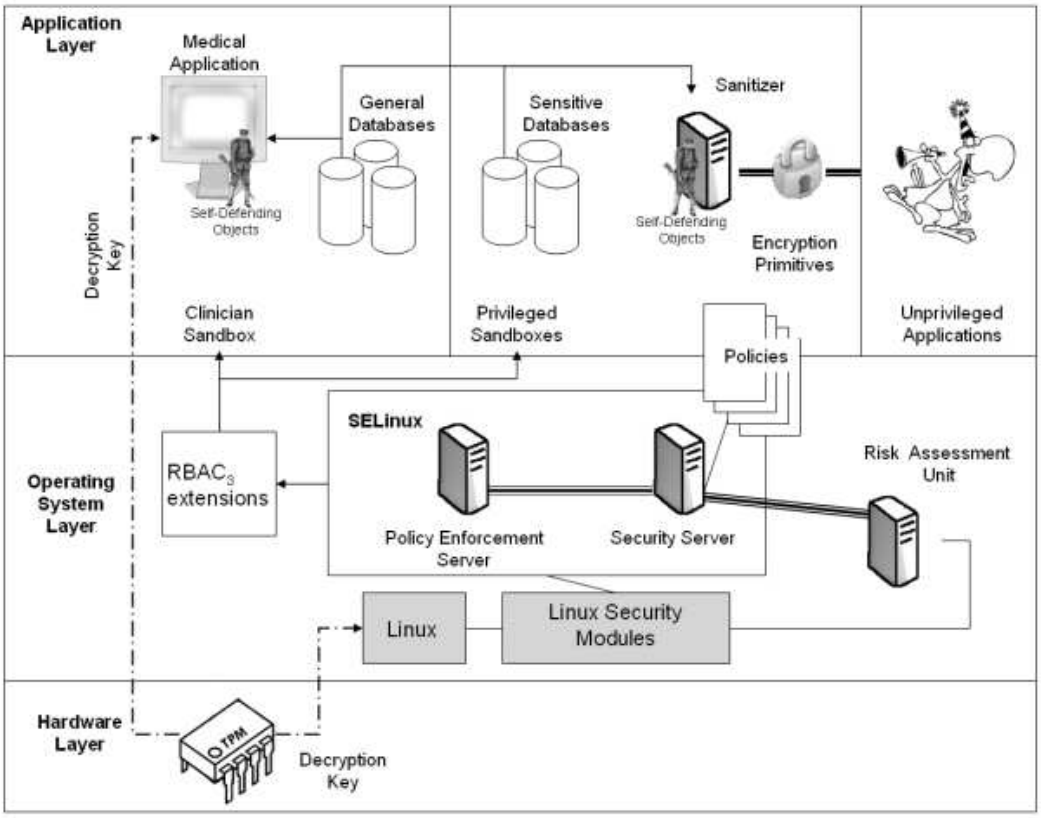

Figure 3: Protecting Application Data using Trusted Hardware in conjunction with SELinux

Adding new users or new applications requires that the SELinux policy be modified. When the policy is complete, it needs to be manually recompiled. Our scenario involves medical application data, which changes on a frequent basis. Consequently, in our architecture, the security server is modified to accept dynamic changes to policies with the minimum of disruption.

The architecture places the risk assessment unit that implements requirement 1c of Section 3 within the operating system layer. It contains sensors and logic that detect changes in environment (for example, an attempt at hacking the trusted computing base). The unit can be configured so that upon detecting these changes, it automatically forces the SELinux Security Server to change policies, or to request manual intervention from the security administrator. This reactive element of the trusted computing base enables the security policy of the sensitive data to be sensibly shaped to the environment. It is feasible because unlike many of the other technologies surveyed here, SELinux separates the logic that enforces the policy from the logic that controls its configuration.

Application Layer The application layer is divided into sandboxes by the SELinux security mechanism in the underlying operating system layer. Unprivileged applications, such as web- browsers and other software that access the internet, are not permitted to interact with the sandboxes that protect private medical information. Likewise, applications administered under the role of clinician are not able to access more specialized or sensitive databases. However, the level of granularity of SELinux is the file, whereas medical databases support records with varying privileges, which have a much finer level of granularity. Consequently, it is the responsibility of the medical application to ensure that its records are not accessed by clinicians with inappropriate levels of authority. Again, RBAC is an appropriate mechanism with which to secure this kind of data.

To ensure an equal level of security control between applications, we introduce Self- Defending Objects into the application layer. The SDOs defend all accesses to databases, based upon a common authentication object that recognizes $RBAC_2$. The authentication object uses the SELinux policy file to recognize the role of the user. However, the security labels assigned to the data records are embedded within the database files, rather than within the policy files.

A sanitization unit runs within a privileged sandbox. The security of this unit is reinforced by the use of SDOs. It packs data for export by stripping away sensitive fields, and demoting the security level of the resulting file (using the domain transitioning abilities of SELinux in the layer below). This file is encrypted prior to export. The alternative to this strategy is to implement a low- watermarking scheme within the operating system layer, such as that suggested by (Safford & Zohar 2004), but it was viewed as too complicated for this architecture.

In a large scale production system, medical data and policy configuration will be distributed. Our prototype does not encompass this element, but the research work already performed on distributing data using SDOs (Vasanta et al. 2004) means the extension of the architecture to meet this goal will be painless.

6 Conclusion and Remarks

In this paper, we surveyed mandatory access control technologies at the hardware, operating system, and application layers, with the aim of securing frequently changing application data under a health-care scenario based upon retrieving information from the

grid. The requirements of the scenario were the ability to secure data at a fine-grained level of control, and to change the policy as circumstances demanded it (for example, factors which may only be known at the application level).

In our three-tiered model, a TPM placed on the motherboard of the trusted computing base protects the upper levels of the architecture.

SELinux was deemed to be the most appropriate operating system technology because of its high level of flexibility, and the detachment of the policy enforcement from the policy configuration logic. However, the granularity of SELinux is insufficient to be able to elegantly secure application data on its own, requiring further mechanisms in the application layer. SELinux, as it stands, would be better able to support the scenario requirements if it were enhanced in a number of ways. Firstly, it should provide support for $RBAC_2$, allowing inherited permissions, simplifying SELinux's notoriously complex configuration process. Secondly, it should have a graceful mechanism for handling changes within policies, to cater for the highly dynamic environment applicable within our scenarios. Thirdly, the requirements of our scenario demand that SELinux is able to gracefully change policies when circumstances change. It is assumed that strong auditing acts as a preventative against an abuse of this function.

Future work in this area aims to determine the extent to which these changes to SELinux can support the success of self-defending objects in creating a trusted computing base for such important applications as those in the healthcare information systems area.

References

Bell, D. & La-Padula, L. (1976), Secure computing systems: unified exposition and MULTICS interpretation, Technical Report ESD-TR-75-306, MITRE Corporation, Bedford, Massachusetts.

Biba, K. J. (1977), Integrity considerations for secure computer systems, Technical Report MTR-35153, USAF Electronic Systems Division, Hanscom Air Force Base, Bedford, Massachusetts.

Croll, P. & Croll, J. (2004), 'Q.U.i.P.S a quality model for investigating risk exposure in e-health systems', *MedInfo Journal 2004* **2004**, 1023–1027.

Ferraiolo, D. & Kuhn, R. (1992), Role-based access controls, *in* '15th NIST-NCSC National Computer Security Conference', pp. 554–563.

Filman, R. E. (2004), A bibliography of aspect-oriented software development, version 1.3, Technical Report 04.05, Research Institute for Advanced Computer Science.

Gayal, S. (2003), 'SANS malware FAQ: How does the Ptrace exploit work on Linux'. Available at http://www.sans.org/resources/malwarefaq/Ptrace.php.

Holford, J., Caelli, W. & Rhodes, A. (2003), The concept of self-defending objects in the development of security aware applications, *in* 'Proceedings of the Fourth Australian Information Warfare and IT Security Conference, Adelaide, Australia', pp. 159–169.

Intel (2003), 'LaGrande Technology Architectural Overview'. Available at http://www.intel.com/technology/security/downloads/LT_Arch_Overview.pdf.

Loscocco, P., Smalley, S., Muckelbaur, P., Taylor, R., Turner, J. & Farrell, J. (1998), The inevitability of failure: The flawed assumption of security in modern computing environments, *in* 'In Proceedings of the 21st National Information Systems Security Conference, National Computer Security Center', pp. 303–314.

National Institute of Standards and Technology (2003), 'Role based access control'. Available at http://csrc.nist.gov/rbac/rbac-std-ncits.pdf.

National Security Agency (2000), 'Security-Enhanced Linux homepage'. Available at http://www.nsa.gov/selinux/.

Reid, J., Cheong, I., Henricksen, M. & Smith, J. (2003), A novel use of RBAC to protect privacy in distributed health care information systems., *in* R. Safavi-Naini & J. Seberry, eds, 'Information Security and Privacy, 8th Australasian Conference, ACISP 2003, Wollongong, Australia, July 9-11, 2003, Proceedings', Vol. 2727 of *Lecture Notes in Computer Science*, Springer, pp. 403–415.

Rindflesich, T. (1997), 'Privacy, information technology, and health care', *Communications of the ACM* **40**(80), 93–100.

Safford, D. & Zohar, M. (2004), 'A trusted linux client'. Available at http://www.research.ibm.com/gsal/tcpa/tlc.pdf.

Trusted Computing Group (2006), 'Trusted platform work group homepage'. Available at https://www.trustedcomputinggroup.org/groups/tpm/.

Vasanta, H., Holford, J., Caelli, W. & Looi, M. (2004), Architecture for securing critical infrastructures using context-aware self defending objects, *in* 'Proceedings of the Fifth Asia Pacific Industrial Engineering and Management Systems Conference'.

Wright, C., Cowan, C., Morris, J., Smalley, S. & Kroah-Hartman, G. (2002), 'Linux security module framework'. Presented at the 2002 Ottawa Linux Symposium, Ottawa, Canada.