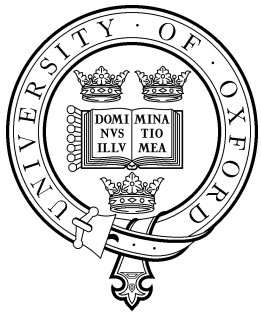
Trustworthy Logging for Virtual Organisations



Jun Ho Huh

Kellogg College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Michaelmas 2009

Acknowledgements

The author is greatly indebted to Andrew Martin for his endless support and guidance throughout the course of D.Phil. The author would like to extend his gratitude and thanks to Andrew Simpson, David Wallom, and David Power for being his transfer and confirmation examiners.

The author would like to thank John Lyle for reviewing a draft of the thesis, and providing constructive suggestions; Andrew Simpson, David Power, Mark Slaymaker, and Peter Lee for helping out with the healthcare grid examples; David Wallom, Steven Young, and Matteo Turilli for their insights on the National Grid Service; Gavin Lowe and Jackie Wang for their help with modeling the security protocol in Casper; also, Daniel James and Eric Kerfoot for their help with grammar. Not least, the author is grateful to QinetiQ for providing a studentship throughout the course.

Related Publications

1. Jun Ho Huh and Andrew Martin. Trusted Logging for Grid Computing. In *Trusted Infrastructure Technologies Conference, 2008. APTC '08. Third Asia-Pacific*, pages 30–42, Oct 2008. IEEE Computer Society.

   Chapter 4 presents a trustworthy logging system based on the contributions of the first paper. The thesis simplifies the security mechanisms and minimises the trusted computing base of the logging system discussed in this paper.

2. Jun Ho Huh and John Lyle. Trustworthy Log Reconciliation for Distributed Virtual Organisations. In *Trusted Computing*. pages 169–182, Feb 2009. Lecture Notes in Computer Science.

   The log reconciliation infrastructure presented in Chapter 5 is based on the contributions of the second paper. The thesis expands on the trusted computing ideas discussed in the paper. The first two papers also cover some of the motivational examples, use cases, and security requirements discussed in Chapter 3.

3. Jun Ho Huh and Andrew Martin. Towards a Trustable Virtual Organisation. In *IEEE International Symposium on Parallel and Distributed Processing with Applications*. pages 425–431, Aug 2009. IEEE Computer Society.

   Chapter 7 proposes trustworthy distributed systems based on the ideas developed in the third paper. More functionality is added to the 'configuration resolver' discussed in the paper.

4. Jun Ho Huh, John Lyle, Cornelius Namiluko, and Andrew Martin. Application Whitelists in Virtual Organisation. Submitted to *Future Generation Computer Systems*. Aug 2009. Elsevier.

   This paper reports on the contributions of a group research and the thesis does not cover these contributions directly. Instead, some of the proposed ideas are adapted in Chapter 7 to improve security and functionality of the 'configuration resolver'.

The material included in the thesis is, except where indicated, the author's own work.

Abstract

In order to securely monitor user or system activities and detect malicious attempts across a distributed system, provision of trustworthy audit and logging services is necessary. Existing audit-based monitoring services, however, are often prone to compromise due to the lack of guarantees of log integrity, confidentiality, and availability. This thesis presents several use cases where these properties are essential, conducts a threat analysis on these use cases, and identifies key security requirements from the threats and their risks. Then, this thesis proposes a log generation and reconciliation infrastructure in which the requirements are satisfied and threats are mitigated.

Applications usually expose a weak link in the way logs are generated and protected. In the proposed logging system, important application events are *involuntarily* recorded through a trustworthy logging component operating inside a privileged virtual machine. Virtual machine isolation makes it infeasible for applications to bypass the logging component. Trusted Computing attestation allows users to verify the logging properties of remote systems, and ensure that the collected logs are trustworthy.

Despite ongoing research in the area of usable security for distributed systems, there remains a 'trust gap' between the users' requirements and current technological capabilities. To bridge this 'trust gap', this thesis also proposes two different types of distributed systems, one applicable for a computational system and the other for a distributed data system. Central to these systems is the *configuration resolver* which maintains a list of trustworthy participants available in the virtual organisation. Users submit their jobs to the configuration resolver, knowing that their jobs will be dispatched to trustworthy participants and executed in protected environments. As a form of evaluation, this thesis suggests how these ideas could be integrated with existing systems, and highlights the potential security enhancements.

Contents

List of Figures

Chapter 1

Introduction

This chapter provides an overview of the motivation and scope of the thesis, and highlights the key contributions.

1.1 Motivation

The emergence of different types of distributed systems, and the fast spread of associated security threats (e.g. adversaries trying to steal sensitive data or models [14]) makes the provision of trustworthy, audit-based monitoring services necessary. For instance, these services could monitor and report violation of service-level agreements [111], or detect events of dubious user behaviour and take retrospective actions [110]. Existing approaches, however, are often prone to compromise due to the lack of *integrity* and *confidentiality* guarantees of log data. Not much effort has been made towards protecting and verifying these security properties upon distributed log generation, collection and reconciliation (see Section 3.4).

Meanwhile, trusted computing and virtualization have often been suggested as technologies suitable for enhancing distributed system security. Many researchers [13, 134, 57, 123] have discussed the use of *remote attestation* for discovering security configurations and establishing trust in remote platforms (see Section 2.6). One of the key motivations is to explore how trusted computing can be used to strengthen existing designs for distributed audit and logging, and to develop trustworthy monitoring services capable of new kinds of functionality hitherto impossible — such as the verification of trustworthiness of logs collected from mutually-untrusting security domains.

Trusted computing solutions, however, do not come without drawbacks. A wide range of software and hardware is required to properly manage trusted computing operations like remote attestation and authenticated boot (see Section 2.4). This has often led to criticisms that the administrative tasks involved in setting up trusted computing applications are too complicated, and usability issues as such are not being sufficiently considered [108].

There are feasibility issues too when it comes to deploying such applications in a distributed environment. Moreover, a heavy use of cryptographic operations usually degrades system performance. An effective way to evaluate whether these problems are acceptable is to construct a prototype implementation and investigate the areas of potential concern. Part of the motivation is to study security and usability issues, and provide evidence of feasibility based on prototyping work.

With the growing influence of e-Research, substantial quantities of research are being facilitated, recorded, and reported by means of distributed systems and e-Infrastructures [19]. As a result, the scope for malicious intervention continues to grow, and so do the rewards available to those able to steal the models and data. Researchers are often reluctant to exploit the full benefits of distributed computing because they fear the loss of their sensitive data, and the uncertainty of the generated results [14] (see Section 7.1). It is also a motivation of the thesis to identify the missing security components and develop potential solutions based on trusted computing capabilities.

1.2 Scope

Five distinct phases are covered in the thesis:

Requirements Analysis (see Chapter 3) For each use case scenario, a threat and risk analysis is conducted to study how attackers might exploit the security vulnerabilities of a logging system. From these, the security requirements for distributed audit and logging are identified. Existing solutions are examined with respect to these requirements and their inadequacies are analysed.

Design (see Chapters 4 and 5) This part of the thesis proposes the log generation and reconciliation infrastructure that satisfies the security requirements. Trusted computing and virtualization capabilities (in particular,

sealed storage and remote attestation) are used extensively to facilitate involuntary log generation and trustworthy reconciliation (and analysis) of distributed logs.

Prototype Implementation (see Chapter 6) A number of integral security components are selected from the log reconciliation architecture, and their prototype implementation is constructed. While doing so, the inherent security, feasibility and usability of the proposed architecture are evaluated.

Generalisation (see Chapter 7) The security components from the log reconciliation architecture are adapted and extended to solve a more generalised set of distributed system security problems. Based on these components, two different types of distributed systems are proposed to facilitate trustworthy job execution and data aggregation.

Evaluation (see Chapter 8) Security of the proposed systems are evaluated against the original requirements. Their interoperability are further evaluated through integration with the original use cases and existing grid or cloud solutions.

1.3 Novel Contributions

The central contribution of the thesis is: (1) the identification of the security requirements for distributed log generation and reconciliation, (2) the architecture design of the trustworthy log generation and reconciliation infrastructure, (3) the prototype implementation of the proposed infrastructure, and implementation guidelines for trusted computing applications, and (4) the architecture design of the trustworthy distributed systems and the central configuration verification server.

Much of the existing research has overlooked the security and interoperability issues around distributed log generation and reconciliation (see Section 3.4). In the requirements analysis phase, the key security requirements are identified and the groundwork for developing trustworthy logging system is provided. These requirements describe the essential security properties and suggest mechanisms (and services) required for protecting them.

From these requirements, a trustworthy log generation and reconciliation infrastructure is designed. It facilitates the production and analysis of log data with strong guarantees of confidentiality and integrity, to an evidential standard (acceptable for judicial use [73][1]) in a variety of contexts. A novel logging paradigm is proposed where application events are logged *involuntarily* at the system level via an isolated, integrity protected logging component. Upon installation of this infrastructure, each participant will be capable of generating and storing the log data, and proving to others that these logs are trustworthy (and accurate). Moreover, log owners will be assured that only the processed, anonymised information will be released to remote users.

The prototype implementation provides strong evidence of feasibility of the trusted computing ideas discussed in the thesis. In addition, the high-level class diagrams and implementation details provide a sound guideline for developing remote attestation and sealing applications. The prototype also uncovers some of the security and usability issues.

The thesis also proposes two different types of distributed systems — one suitable for a computational system and the other for a distributed data system. Central to these systems is the novel idea of *Configuration Resolver*, which, in both designs, is responsible for filtering trustworthy participants and ensuring that jobs are dispatched to those considered trustworthy. Combination of remote attestation and sealed key approach guarantees that jobs are processed in protected execution environments without any unauthorised interference, and returned results are integrity and confidentiality protected.

1.4 Thesis Structure

The rest of the thesis explains these contributions in detail. Chapter 2 covers the background knowledge required to understand the ideas proposed in the thesis. Concepts like distributed computing and logging as well as trusted computing and virtualization are thoroughly explained. Chapter 3 identifies a unifying set of security requirements for distributed audit and logging based on a threat analysis.

[1]This publication from NIST lists key regulations and standards (e.g. Federal Information Security Management Act of 2002, Sarbanes-Oxley Act 2002) that help define organisations' needs for secure log management.

These requirements describe the essential logging properties and pin down the security mechanisms required to protect them.

From these requirements, Chapters 4 and 5 propose a trustworthy log generation and reconciliation infrastructure. Its advantages as well as the potential areas of concern are covered through observations. Chapter 6 describes the prototype implementation constructed for the log reconciliation architecture. It provides high-level class diagrams and implementation details such as might be used as a guideline for developing trusted computing applications. Then, using the security components adapted from the previous infrastructure, Chapter 7 proposes two different types of distributed systems that satisfy a generalised set of security requirements. Again, the advantages and the remaining issues are observed.

Chapter 8 evaluates security of the proposed systems using the original requirements as a success criteria. Practicality and interoperability are evaluated through integration with the use cases and existing distributed systems. Finally, Chapter 9 summarises the key contributions of the thesis and considers potential areas of future work.

Chapter 2

Background Knowledge

This chapter introduces the concepts necessary to understand the ideas proposed in this thesis.

Sections 2.1 and 2.2 explain the fundamental concepts of logging and distributed computing. Section 2.3 discusses the distinct challenges of processing distributed logs, and Sections 2.4 and 2.5 suggest *trusted computing* and *virtualization* as potential technologies to solve these problems. Finally, Section 2.6 introduces emerging ideas in this field of work.

2.1 What is a Distributed Virtual Organisation?

The Internet has changed the way people work, research, and do business immensely. This revolution has led to market globalisation that in turn has increased market competition. Those who have survived the competitive market realised early on that the customers are usually looking for a complete package, and hence they have *collaborated* with others to provide the final product (or service) [19]. Collaboration has also been encouraged among those in pursuit of more challenging and demanding goals that involve complicated tasks.

Emerging information and communication technologies (like 'the Grid' [47]) enable such collaborative activities to take place online. Multiple organisations come together as one unit by sharing their competencies and resources for the purpose of commonly identified goals. The fact that this is happening online characterises it as a *virtual organisation* [19].

This notion of a virtual organisation runs commonly through many definitions of what constitutes a distributed (or a grid) system: 'many disparate logical and

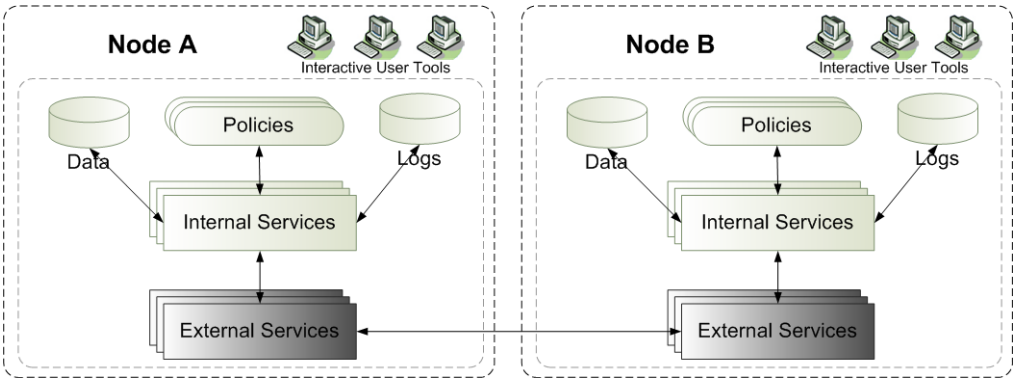


Figure 2.1: Abstract View of the Virtual Organisation

physical entities that span multiple administrative domains are orchestrated together as a single logical entity' [94]. Just as the Internet allows users to share knowledge and files, a distributed system allows organisations to share heterogeneous resources (hardware, applications, data and instrumentation) through a more informed and timely (quicker, on demand) collaborative support. The end user interacts with one large virtual computing system capable of running processes too complex for a single system.

Foster [47] defines the grid as a system that

- *integrates and coordinates resources and users from different control domains* and addresses the issues of security, policy and payment
- *...using standard, open, general-purpose protocols and interfaces...* that address fundamental issues like authentication, authorisation and resource access
- *...to deliver nontrivial qualities of service*

Distributed systems have often been used among scientists to perform their own collaborated research, although, in recent years, the focus has shifted to more interdisciplinary areas that are closer to everyday life, such as healthcare, business and engineering applications [38]. In consequence, a great deal of research [14, 13, 57, 134, 4] is being conducted to provide more reliable and secure means to interconnect resources between different organisations.

An abstract view of the virtual organisation is presented in Figure 2.1, which captures the essence of the definitions above. There are two participant nodes

coming together to form a virtual organisation, and uniting their individual databases as a single logical resource: each node consists of external and internal services responsible for virtualizing the data sources. Further, each node manages its own local data and logs, and various policies governing them. In addition, standardised external services enable communication between different nodes, managing middleware operations such as job submission and user authentication. All user interactions are handled by these external services. This abstract view is used to motivate example applications and use cases described in Sections 3.1 and 3.2, respectively.

2.2 Initial Definitions: A Log Event, Audit Log and Audit Trail

The following three definitions are consistent with the NIST Handbook: An Introduction to Computer Security [55].

Definition 2.1 *'A log event' contains diagnostic information for a single event as observed by one system or application process on a computer system.*

Definition 2.2 *'An audit log' (also referred to as just 'a log') is a record of log events generated by one particular process.*

Definition 2.3 *'An audit trail' is a chronological record of events taken by different processes; several logs may be used to create the complete audit trail devoted to a system, application or user activity.*

Any useful log event would specify the following at a minimum: when the event occurred, the associated user, the program or command used, and the result. Such log events are used to generate audit trails at both the application and operating system levels. System-level audit trails are created by combining logs generated from system processes, and capture information related to any attempt to log on (or log off), devices used, and the functions performed. Similarly, application-level trails combine application process logs and largely monitor user activities: including access, modification, and deletion of files and data. Upon modification of sensitive data, some applications may require both the 'before' and 'after' pictures to be logged.

A complete picture of an activity can be assembled by reconciling audit trails collected from both the application and system levels. For instance, user activity may be monitored by collecting audit trails from both levels, filtering events associated with the user, and sorting them in chronological order. Typically, the resulting audit trail would reveal commands initiated by the user, identification and authentication attempts, and files and resources accessed. This notion could also be extended to a distributed virtual organisation where the user's actions across multiple administrative domains would be recorded, reconciled and examined.

Different types of analysis tools have been developed [115, 130, 138] to assist administrators interpret and analyse the audit trails. These help to distill useful information from the raw data, and effectively, reduce the volume of log events. During a security review, for example, events that appear to be less critical would be removed from the audit trails. Some are designed to identify a specific order of events that indicates an unauthorised access attempt — a repeated failure of log-in attempts, for example, will be detected. Such tools are used to achieve a number of security objectives [55]:

- *reconstruction of events* — audit trails can be used to reconstruct events when a problem occurs; damages can be assessed by reviewing trails of a system activity to identify how, when and why operations have stopped.

- *intrusion detection* — audit trails can be used to identify malicious attempts to penetrate a system or to gain unauthorised access.

- *problem analysis* — status of processes running in critical applications can be monitored with real-time auditing.

- *individual accountability* — proper user behaviour is promoted by logging and monitoring user activities, and advising users that they are accountable for their traceable actions.

- *dynamic access control* — in a distributed data system, a researcher could discover sensitive information through information flow, perhaps by correlating queries against one system with queries against another; audit trails can be used to analyse what the researcher already knows from previous queries, and restrict further access to potential identifiable information.

In most cases, *applications* and *operating systems* voluntarily generate their own log events and enforce various security policies to protect them. The main problem with this approach is that any one of their security vulnerabilities could be exploited to affect the logging mechanisms and compromise the logged data. A more secure approach would involve logging important application events *involuntarily* at the system-level via a trustworthy logging component, and making it infeasible to bypass this component. This is the kind of approach explored in Chapter 4.

2.3 Processing Distributed Logs and Challenges

The following definitions are consistent with the ISO 7498-2 security standards [65].

Definition 2.4 *'Data integrity' is the "property that data has not been altered or destroyed in an unauthorised manner".*

Definition 2.5 *'Confidentiality' is the "property that information is not made available or disclosed to unauthorised entities, individuals, or processes".*

Definition 2.6 *'Availability' is the "property of being accessible and useable upon demand by an authorised entity".*

The rise of many kinds of distributed systems and associated security threats makes necessary the provision of trustworthy services for audit and logging. Such services may be used for forensic examination, for intrusion detection, for proof of provenance, for *post hoc* access control policy enforcement, for financial and business audit and due diligence, for scientific record-keeping, and so on.

These examples have in common requirements upon [121]

- *integrity* and *accuracy* of the logs — its generation, and (perhaps archival) storage; such concerns apply both to the individual log events, and also to the totality of the logs and audit trails; the assembly or pattern of events.
- *confidentiality* of the logged data — again, the individual log events may contain sensitive information; the totality of the log data itself may be considerably more sensitive.
- *availability* of the logging services and the logged data.

- trustworthy reconciliation and analysis of the logs.

In reality, many of the audit-based monitoring services are prone to compromise due to the lack of services to protect the integrity and accuracy of logs generated (and collected) from different sites. Also, because some of these logs are highly sensitive, and without the necessary privacy guarantees, neither site trusts the other to see the raw data.

Many log anonymisation techniques have been proposed to solve the latter issue [77, 103, 8]; however, adapting such techniques and assuring that these anonymisation policies will be correctly enforced at a remote site, is a whole new security issue. Moreover, anonymising the data before release is known to be a hard problem in general (see Section 7.2.2). The main problem with existing solutions is that they only provide weak protection for these security properties upon distributed log generation, access and reconciliation (see Section 3.4).

Foster [47] emphasises that the grid vision requires protocols, as well as interfaces and policies that are not only open and general-purpose, but also *standardised*. It seems that various stakeholders, such as the Open Grid Services Architecture Working Group (OGSA-WG) [90], are aware of the need to standardise the logging facilities in the grid, yet there is no direct work underway to find out what exactly needs to be standardised [60].

The idea of the nodes spanning multiple administrative domains makes the provision of audit-based controls a difficult problem — where the distributed services and associated logs are often inconsistent [60, 126, 16]. There is a need for a common approach to reconstruct the thread of work securely.

2.4 Trusted Computing

The feasibility of the audit-based monitoring services largely depends on the user's ability to verify the security state of remote logging systems, and retain control over their logs regardless of the system to which it migrates.

Faced with the prospect of modern PCs (and other devices) having so much software that their behaviour is unpredictable and easily subverted, the Trusted Computing Group (TCG) [2] has developed a series of technologies based around a Trusted Platform Module (TPM) — normally a hardware chip embedded in

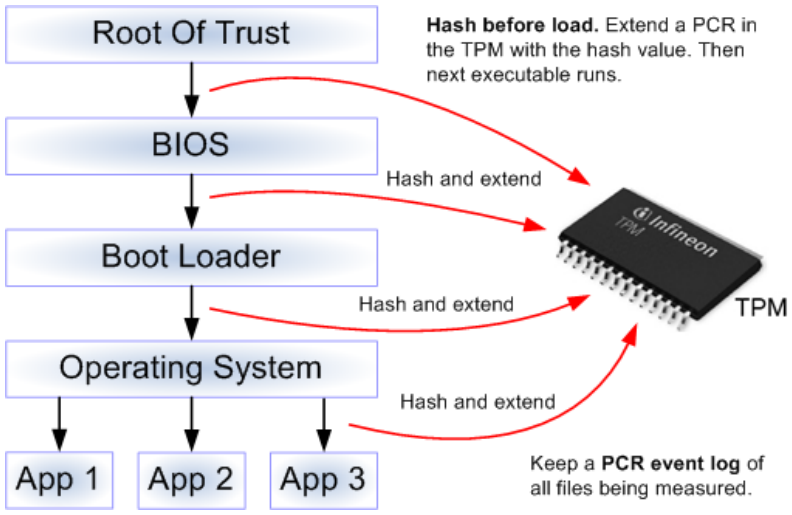


Figure 2.2: Authenticated Boot

the motherboard — which helps to provide two novel capabilities [32]: a cryptographically strong identity and reporting mechanism for the platform, and a means to *measure* the software loaded during the platform's boot process. These include, for example, the BIOS, bootloader, operating system and applications (see Figure 2.2). Further details of the TPM's functionality is defined in the TPM Main Specification [124] published by the Trusted Computing Group.

Measurements are taken by calculating a cryptographic hash of binaries before they are executed. Hashes are stored in Platform Configuration Registers (PCRs) in the TPM. They can only be modified through special TPM ordinals, and the PCRs are never directly written to; rather, measurements can only be *extended* by an entity. This is to ensure that no other entity can just modify or overwrite the measured value. A 20-byte hash of the new measurement is generated based on the PCR's current value concatenated with the new input, and a SHA-1 performed on this concatenated value.

Definition 2.7 *A PCR can be either static or dynamic [34]. A static PCR can reset only when the TPM itself resets — the PCR cannot be reset independently. Static PCRs are normally used to store the measurements. The thesis refers to a static PCR whenever a PCR is mentioned. A dynamic PCR, on the other hand, can be reset independently from the TPM, so long as the process resetting the*

PCR is under sufficient protection[1]*. This thesis refers to such dynamic PCRs as 'Resettable PCRs'.*

In a trustworthy system, every executable piece of code in the *authenticated boot* process will be measured and PCRs extended sequentially (*transitive trust*). The notion of transitive trust provides a way for a relying party to trust a large group of entities from a single root of trust: the trust is extended by measuring the next entity, storing the measurement in the TPM, and passing the control to the measured entity (see Figure 2.2). Using this extend operation has some important security implications:

- a single PCR can store the *extended result* of an unlimited number of measurements;
- the ordering property of SHA-1 provides for a different hash value when hashing two values in a different order; with this order dependency, an entity cannot pretend to run after a certain event; and
- an entity is prevented from creating a 'PCR event log' that removes the entity's own measurement from the log.

Hence, any malicious piece of code (e.g. rootkit) executed during the boot process will also be recorded and identified. A PCR event log is created during the boot process and stores all of the measured values (and a description for each) externally to the TPM. These values can be extended in software to validate the contents of the event log. The resulting hash can be compared against the reported, signed PCR value to see if the event log is correct.

2.4.1 Sealed Storage

Trusted computing provides the means to *seal* (encrypt) data so that it will only successfully decrypt when the platform measurements are in a particular state [32]. The seal process takes external data (information the TPM is going to protect) and a specified PCR value, encrypts the data internally to the TPM using a *storage key*, and creates a sealed data package (see Figure 2.3). This

[1]Only 'Locality 4' (trusted hardware) can reset a dynamic PCR — platform hardware ensures that only trusted processes have access to Locality 4 [33].

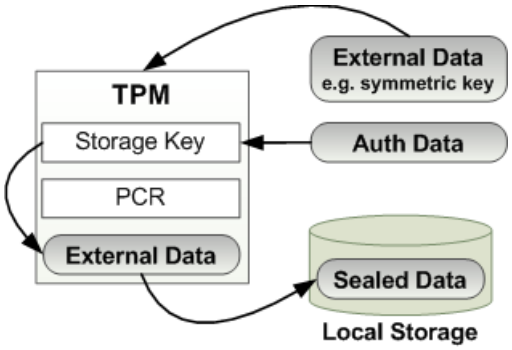


Figure 2.3: Sealed Storage

storage key binds the encrypted data to the TPM. Typically, the external data protected is a *symmetric key* that may be used to encrypt bulk data objects.

An application — responsible for keeping track of this package — sends the package back to the TPM to recover the data. A nonce, known only to an individual TPM, is also included in the package to ensure that only the TPM responsible for creating the package can unseal it.

The whole purpose of sealing is to prevent any unauthorised attempt to unseal the package. The TPM enforces two restrictions upon decrypting the sealed package:

- ensures that the package is only available on the platform bound to the TPM that created it — the TPM checks whether the nonce included in the package matches the one held internally; and
- compares the current PCR value to the specified PCR value stored in the sealed package — the operation aborts if these values do not match.

The implication is that the external data only becomes available to an application when the correct value (an acceptable platform configuration) is in the specified PCR. Chapter 5 explores different ways in which sealing can be used to protect sensitive data from compromised hosts.

2.4.2 Remote Attestation

Sealed storage provides a high degree of assurance that the data is only available if the acceptable configuration is present. But how does an external application

— that has not performed the seal operation — know that such a configuration is present in a remote platform? Trusted computing provides the means to undertake *remote attestation* [32]: proving to a third party that (in the absence of hardware tampering) a remote platform is in a particular software state.

Remote attestation involves the TPM reporting PCR value(s) that are digitally signed with TPM-generated 'Attestation Identity Keys' (AIKs), and allowing others to validate the signature and the PCR contents. The application wanting to attest its current platform configuration would call the `TPM_Quote` command specifying a set of PCR values to quote, an AIK to digitally sign the quote, and a nonce to ensure its freshness. The TPM validates the authorisation secret of the AIK, signs the specified PCRs internally with the private half of the AIK, and returns the digitally signed quote.

The external application validates the signature by using the public half of the AIK, and validates the AIK with the AIK credential — a certificate issued by a trusted Certificate Authority (a 'Privacy CA') which states the platform has a valid TPM. The PCR log entries are then compared against a list of 'known-good' values to check if the reported PCRs represent an acceptable configuration. This list is often referred to as an 'application whitelist'.

Attestation can be used on a platform that supports authenticated boot (see above) to verify that only known pieces of software are running on it. Additions or modifications to any executable will be recorded during the boot process, and noticed when log entries and PCR values are checked. With such mechanisms in place, the external application can, in theory, identify whether a remote platform has been infected with a virus or not.

2.4.3 Runtime Attestation Model

Figure 2.4 gives an overview the Trusted Computing Group's runtime attestation model [129]. In a trusted platform, the Platform Trust Services (PTS) provide the capability to select hardware and software components to be measured during the authenticated boot process. They are also responsible for computing the measurements of the selected components and the creation of an integrity report containing these measurements. The Verifier checks the incoming integrity reports using the Policy Database, Configuration Management Database (CMDB),

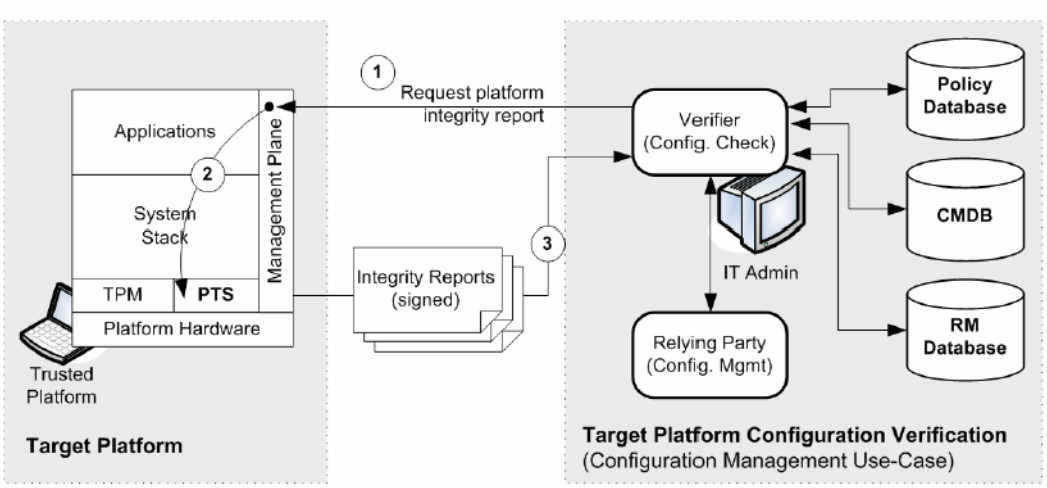


Figure 2.4: Runtime Attestation Model (Figure 22 from [129])

and Reference Manifest (RM) Database. These databases hold known-good configurations for platforms. If an attesting platform has an unknown or unexpected configuration, the Verifier informs the Relying Party not to trust this platform.

The proposed systems in Chapters 5 and 7 adapt this runtime attestation model. Each administrative domain is managed by a central Verifier (referred to as the *configuration resolver*) which checks the configurations of the participant platforms when they first register with the Verifier. Only those verified to be trustworthy become available to the Relying Party (the end users).

2.4.4 Limitations

Despite the new security primitives introduced by sealing and attestation, there are a number of issues that need to be addressed before these can be used effectively. One of the fundamental problems of attestation is tracking down the static identity for software (which is generally dynamic) to measure. After a piece of software is deployed, its configuration parameters might change, or it might be repeatedly patched and updated. Moreover, most software and operating systems they run on constantly change stored data files during execution. Such dynamic properties of software make it difficult to determine their static identities.

As a possible solution, Sailer et al. [97] have suggested measuring only the binary or relatively static parts of the platform. The idea is to measure relatively

static components that are only relevant to the correct operation of a platform. They argue that a *minimised* form of identity would remain unchanged over time. The rest of the components should have a minimal impact on the correct functioning and security of the platform.

Attesting only the identity of binary data, however, has its own drawbacks. Haldar et al. [56] state that binary attestation only proves the identity of a platform and not its *behaviour*. The user still has to interpret the behaviour of the platform. To improve usability of attestation, they have introduced 'semantic remote attestation' — a technique for verifying the remote behaviour of a platform. A trusted virtual machine reports on the high-level properties of software running on the platform: these include class hierarchies and Java Virtual Machine security policies (e.g. access controls on program variables). As a result, more reliable information becomes available to the users upon making security decisions.

In an attempt to offer a low-cost security solution, the TPM uses the main memory of a computer to run trusted (measured) applications. The trusted applications are measured when they are initially loaded into the memory, and these measurements are replayed during attestation. While the applications are running, however, in-memory attacks (such as exploiting a buffer overflow, or overwriting the program code) [109] might be performed to alter their behaviour, or steal sensitive data from the memory. The TPM has not been designed to detect such modifications in the intervening time between measurement and attestation. Hence, a runtime state of the platform might not be reported accurately through attestation. Shi et al. [41] describe this issue as the inconsistency between the 'time-of-use' and the 'time-of-attestation'.

Robust *memory protection* and *process isolation* seem the obvious solutions to this problem, yet traditional operating systems only provide weak mechanisms for both. For instance, with a small amount of effort, a malicious application could gain full access to the memory space allocated to a TPM-measured application running under the same user account. Moreover, a successful privilege escalation attack [86] — whereby a malicious user or process gains access to the administrator's account — would allow one to modify or replace the measured applications. To guard against this style of attack, measured applications and their memory

space need to be strongly isolated and protected in their own compartments. Only then can we fully exploit the benefits of trusted computing.

2.5 Virtualization

Virtualization is a key technology used in many trusted computing solutions to provide strong isolation for the trusted (TPM-measured) applications. A combinational use of these two technologies is referred to as 'trusted virtualization'. Virtualization allows a single physical host to share the computing resources between multiple operating systems [139]. Each operating system runs in a Virtual Machine (VM) of its own, where it is made to believe that it has dedicated access to the hardware. A virtual machine is also referred to as a 'compartment'.

A thin layer of software called Virtual Machine Monitor (VMM) operates on top of the hardware to isolate virtual machines and mediate all access to the physical hardware and peripherals. A virtual machine runs on a set of virtual devices that are accessed through virtual device drivers. Typically, a highly privileged *monitor virtual machine* is created at boot time and serves to manage other virtual machines. In some implementations, the monitor virtual machine *intercepts* all virtual I/O events (before they reach the virtual machine monitor) and controls the way physical hardware is accessed. This notion of interception is examined further in Section 4.2 and is used to facilitate involuntary logging of application events.

Numerous design efforts have been made to remove avoidable inter-virtual-machine communication mechanisms such as might be exploited to undermine the isolation guarantees. The aim is to make a virtual machine behave in the same way (and have the same properties) as a physically isolated machine. In such designs the virtual machine monitor ensures that all memory is cleared before being reallocated and each virtual machine has its own dedicated memory and disk space. Both Intel and AMD processors now provide hardware support for full, efficient virtualization [75, 53]. With help from these processors, the virtual machine monitor can simulate a complete hardware environment for an unmodified operating system to run and use an identical set of instructions as the host. Hardware virtualization can also speed up the execution of virtual machines by minimising the virtualization overhead.

2.5.1 Secure Isolation with Virtualization

The majority of current grid middleware solutions, including the Globus Toolkit [63], rely on operating systems' access control mechanisms to manage isolation between user accounts. For example, operating system enforced access control policies prevent malicious software (installed by a third party unknown to the host) from gaining unauthorised access to the jobs running under different user accounts. However, millions of lines of code contained in a mainstream operating system must be trusted to enforce these policies correctly [10]. A single security bug in any one of the privileged components might be enough for an attacker to hijack it, elevate its privileges, and take control of the host and the jobs running inside.

Virtualization, on the other hand, is capable of providing much stronger isolation through the relatively smaller virtual machine monitor and monitor virtual machine [59]. A malware (through privilege escalation) would have to compromise both components — which are designed to resist such attacks — in order to break the isolation [50]. In a trustworthy, virtualized system, these two components (as well as other trusted software) would be measured during the authenticated boot and their integrity would be reported through attestation. Many researchers [45, 74] have studied the benefits of separating jobs and trusted applications in their own virtual machines:

- the job owner has more flexibility in their choice of the operating system and software (i.e. the execution environment);
- job isolation prevents a rogue job from compromising the host or other jobs running in the same host;
- in-memory attacks targeted at the trusted applications are made more difficult; and
- the impact of privilege escalation attacks are limited to the isolation boundaries of a virtual machine.

2.5.2 Xen Virtual Machine Monitor

Xen is a virtual machine monitor designed for 32-bit (often called x86 or i386) and 64-bit Intel architectures [18]. It originated as a research project at the

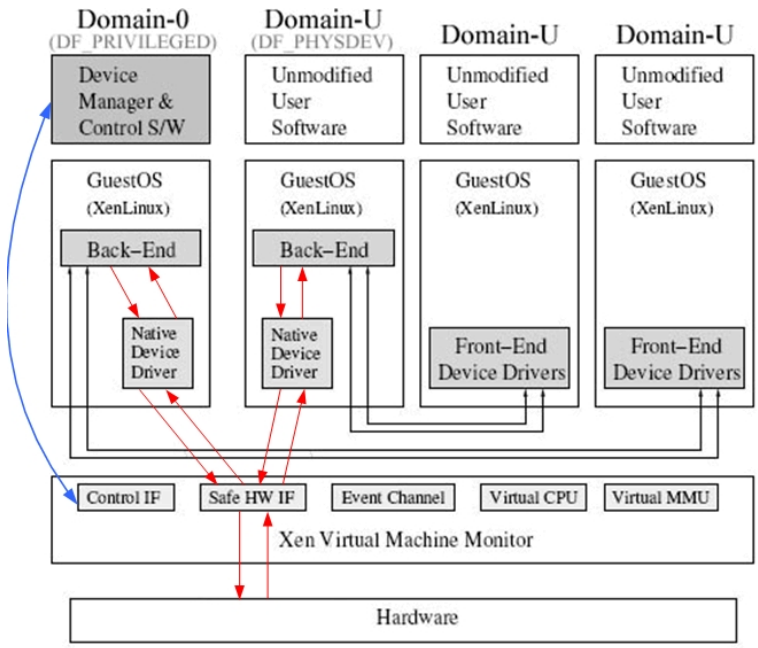


Figure 2.5: Xen Architecture (Adapted from [67])

University of Cambridge Computer Laboratory [89], and is now maintained by the Xen community as free software. Being an open source project, it provides flexible means for developers to modify its components, or add new abilities to them.

This advantage has attracted many researchers to work with Xen to isolate trusted compartments from those which are untrusted. For example, the Open Trusted Computing (OpenTC) consortium have recently implemented a 'Corporate Computing at Home' prototype [91] based on the Xen virtual machine monitor. Their prototype serves to demonstrate how trusted virtualization could be used to prepare a trusted (TPM-measured) compartment, verify its state, and access the corporate network through the trusted compartment. The systems proposed in the thesis (see Chapters 4 and 5) are also designed and implemented with Xen as the virtualization layer. The rest of the section explains the concepts of Xen that are necessary to understand how these systems work.

The Xen virtual machine monitor runs on top of the bare hardware and controls the way virtual machines (also referred to as 'domains' in Xen) access the hardware and peripherals (see Figure 2.5). A highly privileged monitor virtual

machine called *Domain-0* is created at boot time and manages other guest virtual machines. Domain-0 is responsible for hosting the application-level management software.

The *Safe Hardware Interface (Safe HW IF)* is available through the virtual machine monitor, and allows the hardware to be exposed in a safe manner. It provides a restricted environment called 'I/O space' which ensures that each device performs its work isolated from the rest of the system (a bit like virtual machine isolation). In Xen, events replace hardware interrupts. The *Event Channel* is responsible for sending asynchronous events to the virtual machines.

The *Device Manager* runs inside Domain-0 to bootstrap the device drivers and broadcast their availability to the guest operating systems, and export configuration and control interfaces. A *Native Device Driver* is a normal driver that runs in a Domain-0 (or other privileged virtual machines) with access to the physical hardware. Each driver is restricted by the I/O space of the safe hardware interface so the damage a faulting device can do to others is limited. A *Front-end Device Driver* is a virtual device driver sitting inside a guest virtual machine. A guest operating system uses the front-end to send I/O requests to the corresponding *Back-end Driver*, which is another virtual driver running inside Domain-0. The back-end checks the validity of incoming I/O requests and forwards them to the native driver to access the physical hardware. The back-ends usually run inside Domain-0 since they need access to the native drivers. Nevertheless, they may also run in a virtual machine that is given a physical device privilege (`DF-PHYSDEV`) flag.

Once the kernel completes the I/O run, the back-end uses the event channel to notify the front-end that there is pending data. The guest operating system then accesses the data using the shared memory mechanisms. The details of the shared memory mechanisms and the use of virtual drivers are covered in Section 4.2.2.

2.6 Emerging Ideas and Inadequacies

Great strides have been made in using trusted virtualization to design and construct trustworthy components for distributed systems. This section explores some of the emerging themes from this field of work and identifies their inadequacies.

2.6.1 Attestation Tokens and Sealed Key Approach

The term 'attestation token' is commonly used to describe a participant's credentials [57, 140]. Typically, it contains the participant's platform configurations and the public half of a non-migratable TPM key. The private half is bound to the platform's TPM and PCR values corresponding to its trusted computing base. Information contained in the attestation token should be sufficient for a user to verify the identity and trustworthiness of the platform.

Lohr et al. [57] combine the Perseus virtualization framework and remote attestation to create a Trusted Grid Architecture. In the Trusted Grid Architecture, users collect attestation tokens of service providers, and verify their platform configurations using a locally managed whitelist. Upon job submission, the job secret is encrypted with a service provider's public key (obtained from their attestation token), guaranteeing that only a securely configured platform will be able to access the private key and decrypt it. If the service provider's trusted computing base has changed, the private key will no longer be accessible to process the job further.

The virtualization layer is extended to include services that support secure job transfer and execution. Grid jobs are transferred across an encrypted, integrity-protected communication channel established with trusted middleware components, and their data is written to disk using a secure storage service. The attestation service uses the attestation token to verify the state of the trusted software layer prior to submitting the job data. The job data is encrypted using the public key (obtained from the token) so that only a securely configured software layer can decrypt it and execute the job.

The Trusted Grid Architecture, however, provides no mechanisms for verifying the job execution environment and the integrity of the returned results. It also fails to amply isolate the trusted components. Most of their security controls are enforced in a virtual machine that also contains a large amount of untrusted software. For example, the grid management service runs in the same compartment as the storage encryption service. This extra complexity increases the likelihood of vulnerabilities and makes attestation less meaningful. Moreover, they do not discuss how the users collect the attestation tokens and how the application whitelists are managed in a distributed environment.

Due to the lack of useful semantic (including security) information that can be conveyed through standard binary attestation (see Section 2.4.4), many (see, for example, [131, 78]) have suggested the use of *property-based attestation* [9] to add more security information. Security relevant properties of the platform are attested rather than the binary measurements of the software and hardware. In consequence, trust decisions made based on platform configurations are simplified.

2.6.2 Minimising Trusted Code

The overall trustworthiness and reliability of a participant system depends on the size and complexity of its trusted computing base — the smaller and simpler the trusted computing base is, the less probable the compromise in security would be [81]. Various methods for minimising the trusted computing base have been discussed [133, 25].

In Cooper and Martin's architecture [12], the job security manager virtual machine enforces all foundational security functions for the grid jobs. Their job security manager and virtual machine monitor form the trusted computing base within the grid platform. The digital rights management controls are protected within the job security manager (a dedicated virtual machine), providing enhanced protection for encryption keys. By dedicating the security manager to a single purpose, the complexity of the trusted code is minimised.

This type of compartmented architecture simplifies attestation since the integrity of a relatively small, simple piece of software is verified against an application whitelist. Reducing the complexity of attestation and management of whitelists is integral for developing trustworthy systems. This thesis also explores different ways of isolating log security functions in a dedicated virtual machine and minimising the trusted code.

2.6.3 Grid Middleware Isolation

Cooper and Martin [13] make a strong argument that the complex grid middleware services, which usually have a high likelihood of vulnerabilities, can not be trusted to secure users' data and credentials. For example, at least five different vulnerabilities have been found in the Globus Toolkit [48] that allow unauthorised users to compromise the middleware [13].

In their architecture, the middleware stack is isolated in an untrusted compartment of its own and is not relied upon to perform trusted operations. As a result, even if an attacker manages to compromise the middleware, they would not have gained sufficient privileges to undermine the security of a distributed system.

2.6.4 Job Isolation

The use of virtual machine isolation has been discussed many times as a solution to the 'malicious host' problem [12, 132, 131]. Typically, a job runs on a separate virtual machine of its own, where its queries or codes are executed free from unauthorised interference. The job secrets are decrypted inside the virtual machine and protected from rogue virtual machines or the host. Job isolation could also protect the host from rogue jobs [113].

Terra [123] is a virtualization architecture developed on the VMware virtualization platform [118]. VMware is modified to support encrypted and integrity protected disks. Using their trusted virtual machine monitor, existing applications can either run in a standard virtual machine, or in a 'closed-box' virtual machine that provides the functionality of running on a dedicated closed platform. The trusted virtual machine monitor protects confidentiality and integrity of the contents of the closed-box by intercepting the disk I/O requests and encrypting the disk sectors. The closed-box is strongly isolated from the rest of the platform. With hardware memory protection and secure storage mechanisms, the contents are also protected from a rogue administrator.

The authors suggest that Terra could be used to enable a secure grid platform. A closed-box would isolate the job and protect its contents from a malicious host. This closed-box would be capable of accessing its own integrity measurement by performing a system call through the trusted virtual machine monitor. The job owner would use this measurement to identify the job execution environment.

2.6.5 Trusted Execution Environment

Cooper and Martin [13] describe an architecture that aims to provide a 'trusted execution environment'. In their architecture, a grid job is encrypted and runs on an integrity protected virtual machine where it cannot be accessed from the

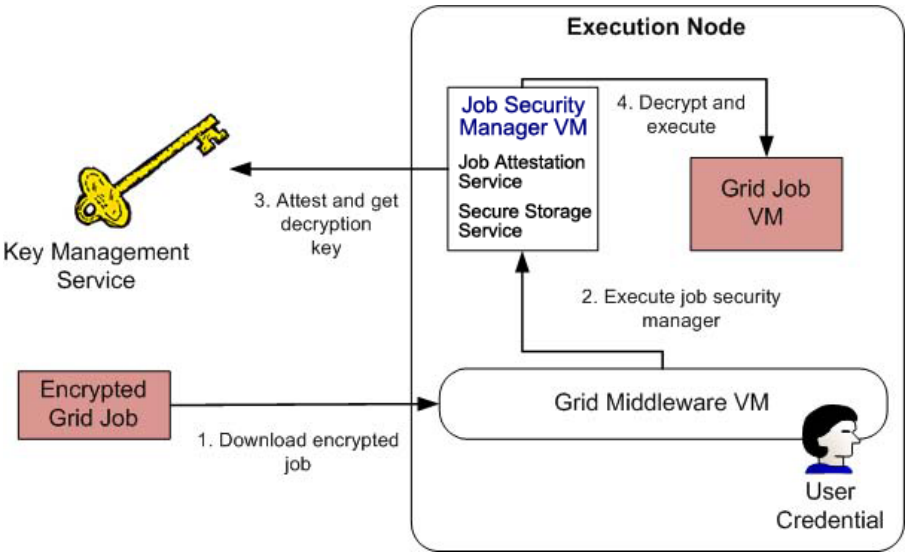


Figure 2.6: A Trusted Grid Architecture (Adapted from [15])

host platform; the data is safely decrypted inside this virtual machine during execution. Remote attestation is used to verify this environment before dispatching the job.

Their solution works by distributing a job composed of two virtual machines: the first virtual machine runs the job, and the second enforces the trusted execution environment (see Figure 2.6). This second virtual machine, referred to as the 'job security manager', isolates the security layer from the job, and allows the solution to work seamlessly with all legacy virtualization and middleware software.

One potential loophole comes from the fact that they are less concerned about the 'malicious code' problem — an untrusted code running on a participant's platform. A *job owner* specifies the virtual machine instance and its security configurations are not checked before being used. The system relies on virtualization alone to isolate rogue jobs from the host.

The type of attacks a malicious virtual machine can perform would be restricted if virtualization offers complete isolation; but no existing solution guarantees this property right now (although, it is the objective of many). For example in Xen, each virtual machine has two I/O rings, one for sending requests and one for receiving responses, and these form the inter-virtual-machine communication mechanism [18]. A rogue job could potentially hijack privileged processes and

manipulate this communication channel to perform buffer overflow attacks on the privileged virtual machines.

2.6.6 Sealed Storage

Jansen et al. [69] demonstrate how the Xen virtual machine monitor could be used to improve existing sealed storage solutions. They modify the virtual machine monitor to control access to the cryptographic keys used for sealed storage, ensuring that the keys are only accessible to the authorised virtual machines.

A similar approach is discussed in this thesis to protect the logged data from unauthorised virtual machines: the sealed storage ensures that only a securely configured logging service can access the logged data (see Section 4.2.5).

More related work is reviewed in Sections 3.4 and 7.4. Section 3.4 examines existing logging solutions in detail and discusses their inadequacies. Section 7.4 establishes an emergent consensus view based on the ideas covered here and identifies the missing pieces.

2.7 Chapter Summary

The concepts necessary to understand the rest of this thesis, including virtual organisation, secure logging and trusted virtualization have been covered in this chapter. Also, the emerging ideas for using trusted virtualization to improve distributed system security have been discussed. The next chapter will explore motivational examples extracted from a wide range of application domains that would benefit from running a trustworthy logging system; these examples will be followed by a threat and risk analysis. Based on the critical threats and their risks, a unifying set of trustworthy logging requirements will be identified.

Chapter 3

Requirements

This chapter focuses on deriving security requirements for log generation and reconciliation — issues associated with building trustworthy distributed systems (as discussed in Chapter 1) are not considered at this stage.

Section 3.1 explores a number of motivating examples to highlight the security challenges listed in Section 2.3. Driven by these examples, Section 3.2 sets out a collection of use case scenarios; for each scenario, their potential security threats and risks are analysed. From these, Section 3.3 identifies the essential security requirements as a first step toward designing a trustworthy logging system. Finally, Section 3.4 discusses the remaining gap between these requirements and existing solutions.

3.1 Motivating Examples

3.1.1 Healthcare Grids and Dynamic Access Control

An example application arises in the context of a Healthcare Grid [4]. In 'e-Health', many data grids are being constructed and interconnected, both in order to facilitate the better provision of clinical information, and also to enable the collection and analysis of data for scientific purposes, such as clinical trials of new treatments.

For clinical data audited access will in many cases be much more appropriate than rigid access controls (since some essential access may be very urgent, and hard to authorise using RBAC-style approaches). A form of 'traffic flow' analysis may itself yield much information about the patient and/or their clinical data, however, so the access logs themselves are highly privileged. Researchers' access

GP Practice (GP) T_1				
NHI	DOB	GP	$Smoke$	$Risks$
1	20/05/88	Dr. Anderson	yes	overweight
2	30/07/88	Dr. Anderson	no	allergies

Table 3.1: Data Available from the GP Practice (T_1)

Specialist Clinic (SC) T_2		
NHI	$Postcode$	$LungCancer$
1	OX2 5PS	yes
2	OX2 6QA	no

Table 3.2: Data Available from the Specialist Clinic (T_2)

must also be carefully logged and analysed, lest through multiple queries a researcher manages to reconstruct personal data that identifies an individual [94]. One possible mitigation technique would be to analyse what researchers have seen from previous queries (using audit trails), and dynamically update the access control policies to prevent potential inference attacks.

Consider the following example[1] in the context of the abstract view shown in Figure 2.1. A simplified healthcare grid consists of two nodes: a GP Practice (GP) and a Specialist Clinic (SC). A patient from the GP practice is often referred to the specialist clinic to see a specialist. It is assumed that a single table at each clinic (T_1, T_2) is made accessible to a researcher, and that the National Health Index (NHI) uniquely identifies a patient across the grid to enable the linking of data. The researcher is carrying out a study that looks at association between smoking status (T_1) and development of lung cancer (T_2) in the population of Oxfordshire. The researcher has originally been granted full access to both T_1 (at the GP practice) and T_2 (at the specialist clinic) to conduct this research. By joining the data across two clinics, the researcher could gain access to potential identifiable information about patients: for example, the researcher could find out that patient 1, born on the 20/05/88 and living in OX2 5PS who has Dr. Anderson as their GP, is a smoker and has a lung cancer.

[1]This example has been developed with help from David Power who is involved in the GIMI project [3] at Oxford, and Peter Lee who is an intern at the Auckland Hospital.

In a secure grid, as soon as the researcher finds out from querying T_2 that patient 1 has lung cancer, the researcher's access on T_1 for patient 1 would be restricted to, for example, only the NHI and $Smoke$ fields. For the GP practice to have restricted the researcher's access rights to information pertaining to patient 1 on T_1, would have required the GP practice to collect *data access logs* from the specialist clinic to build up a picture of what the researcher already knows, and to update its own access control policies. This would prevent the researcher from collecting potential identifiable information. However, in general, the specialist clinic would never give out patients' lung cancer status in the form of audit trails to an untrusted GP practice.

This type of distributed audit approach has been suggested to detect patterns of behaviour across multiple administrative domains by combining their audit trails [110]. However, a problem arises from the fact that the log owners do not trust other sites to see their privileged raw data. This approach will only work if the log owners can be assured of *confidentiality* during transit and reconciliation.

3.1.2 The Monitorability of Service-Level Agreements

The provision of service-level agreements (SLAs) and ensuring their *monitorability* is another example[2] use for trustworthy log generation and reconciliation.

A service-level agreement is a contract between customers and their service provider. It specifies the levels of various attributes of a service like its availability, performance, and associated penalties in the case of violation of the agreement [71]. Consider a case where the client receives no response for a service (for which they have entered into an agreement) within the agreed interval of time, complains to the provider that a timely response was not received, and requests financial compensation. The provider argues that no service request was received and produces an audit trail for requests in their defense. There is no way for the client to find out the truth when the provider could have delivered tampered evidence. The problem is that the service-level agreement is defined in terms of events that the client cannot directly monitor and must take the word of the provider about the service availability.

[2]Initially, Jason Crampton from Royal Holloway has suggested the possible use of trusted computing to improve monitorability of service-level agreements, recommending his own publication as a reference [111].

Skene et al. [111] suggest a way of achieving monitorability with trusted computing capabilities. This involves generating trustworthy logs and providing assurance that these logs have been used unmodified for monitoring service-level agreements. For instance, if the client is able to verify with attestation that trustworthy logging and reporting services operate at a remote site, then the client may place conditions on any event of their interest and construct more useful agreements. This approach needs to guarantee the *integrity* of all service request/response logs to an evidential standard (i.e. to a standard acceptable for judicial usages) upon distributed reconciliation and analysis. A monitoring service would then be able to generate a reliable report for the client to make claims.

Likewise, logged data often contains useful evidential information. However, the inability of a site to verify the integrity of logs collected from other sites and the lack of guarantees that their own logs are being used unmodified, make it difficult for one to adapt the usual audit-based monitoring methods.

3.1.3 Distributed Banking Services and a Rogue Trader

A bank provides a collection of services to its traders which they use to assess buying and selling opportunities for shares. The buying and selling of shares is heavily regulated with a strong need to log behaviour, and to mirror and replicate the operations. The buying behaviour is driven by a collection of data feeds; here, the input timing of the data feeds is important to the traders and to the regulators [104]. Such data feeds are usually replicated from multiple providers and cross referenced to avoid costly mistakes. The calculation of derivative values is CPU-intensive and depends on available resources; these must be logged, managed and audited.

An interesting scenario[3] emerges at peak times, when there is a need to outsource the computation and service provision because internal resources simply cannot meet the demand. For instance, if the number of trades being passed through a system suddenly increases by a large amount, a bank might not have the necessary infrastructure to cope with this spike; hence, there is an inter-

[3]This distributed banking scenario has been discussed and elaborated with Terence Harmer from Belfast e-Science Centre.

est in calling on utility resources from trusted suppliers within the regulatory framework.

Any decision, any data received, and any data sent must be logged to enable a complete picture to be assembled later. The French S Generale scandal [125, 26] is a good example that demonstrates such a need to protect the totality of logs: the bank lost approximately 4.9 billion Euros closing out positions over three days of trading. It resulted from fraudulent transactions created by a rogue trader (who went far beyond his limited authority) within the bank — but the police explained that they lack the forensic evidence to charge him with fraud.

In a more secure system, the administrator should be able to trace everything the rogue trader has done by examining the audit trails. These should also be made available to the police for an accurate investigation. The police should be assured that they have access to the complete, integrity-protected information.

3.1.4 Privacy in Public Communications Network

Privacy in communication is considered as a valuable asset among the public network providers — the mobile telephony and internet providers. For some customers, breach of privacy could lead to severe commercial losses. Regulatory authorities in many countries are responsible for auditing and regulating the security levels required for each network provider. This is to ensure that the essential security mechanisms are operational and that customers' privacy is being protected.

Despite various security measures and well-defined standards, there still remains threats like 'communication interception' and 'malicious insiders' [114]. Non-conformance with the security standards and lack of audit-based monitoring mechanisms increase the security risks.

Trustworthy audit and logging services could be used during regular security audits to ensure that certain technical measures are in place and security policies are being enforced correctly. During non-scheduled audits, if a security breach is detected, for example, audit trails could be used to determine the cause of the problem and assess the damages.

Consider an interception case for a mobile telecommunications provider in Greece [21]. As the Greek authorities and the network provider have revealed, an unknown trojan horse performed various attacks to compromise a part of the

core network of the provider. The trojan first activated a lawful interception component in the infected elements, and intercepted about 100 calls from the Greek politicians and offices, including the U.S. embassy in Athens and the Greek prime minister. While doing so, it also turned off several logging procedures to hide its presence and the traces that show the activation of the interception component. By exploiting the vulnerabilities of the core system (responsible for generating logs), the trojan managed to turn off the logging procedures and bypass the intrusion detection system. This example serves to highlight (1) the risk of merely relying on the application itself to generate security logs, and (2) the need for trusted logging execution and process isolation.

3.1.5 Post-Election Investigation

'e-voting' and post-election investigation is another area that requires a trustworthy logging system. Since elections do not always go smoothly, it is important that a voting system preserves *forensic evidence* that can be used to investigate various problems [17, 119, 136]. For this purpose, many commercial e-voting systems generate and maintain audit trails that describe system operations and voters' actions.

Unfortunately, the logging components used in current voting systems fall short in a number of aspects:

- they often record a limited amount of information, missing essential information for post investigations [6, 122];
- they provide weak protection for log integrity, making it difficult to verify whether the audit trails represent accurate and complete information [112];
- being part of the voting system, they do not provide an independent way of generating logs [100, 136]; and
- they lack protection for the voter's anonymity upon post investigations [17].

In an ideal system, all interactions between the voter and the voting machine would be recorded. Based on the integrity and totality protected audit trails, the voter's intent would be precisely reconstructed to investigate election disputes. To preserve the voter's anonymity, only the privacy protected audit trails would be made available for post-election investigations.

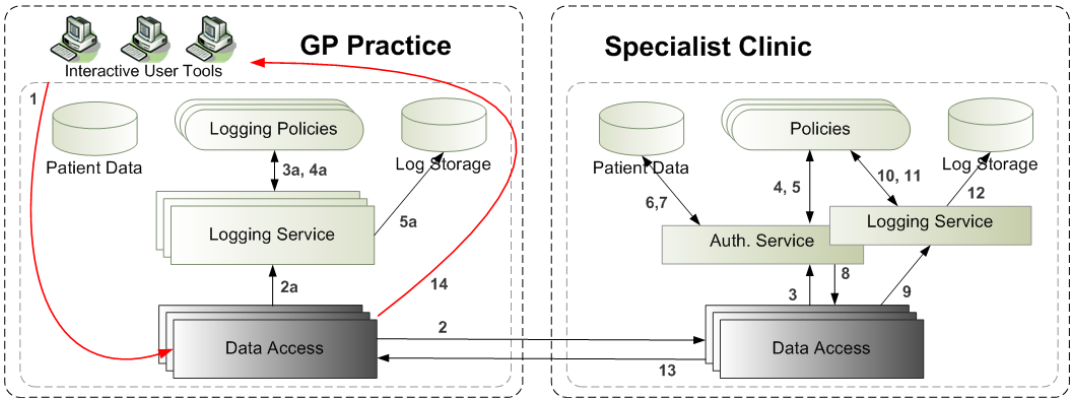


Figure 3.1: Use Case — Logging Distributed Data Access

3.2 Use Cases and Threat Analysis

Based on the abstract view of the virtual organisation (see Figure 2.1), this section presents a number of use cases to identify the potential threats likely to be faced by the examples above. These use cases are high-level representative of the integrity, confidentiality and availability requirements that logging services should satisfy.

3.2.1 Logging Distributed Data Access

The first two use cases come from the healthcare grid example (see Section 3.1.1). In the first use case (see Figure 3.1), a researcher at the GP practice wishes to query some patient data held at the specialist clinic. The specialist clinic defines its own data authorisation policies and logging policies for data access — the logging policies are used to engage logging mechanisms to record events of interest. The researcher should only see the patient information for which they have access rights, and this distributed access should be logged accurately at both clinics.

Firstly, a data access request is sent from an interactive user tool to the external data access service at the GP practice. This request is submitted to the external data access service at the specialist clinic, which forwards the request to the internal authorisation service. The request is evaluated against data authorisation policies and the authorisation decision is returned to the data access

service for logging purposes — the patient data, however, is only returned if access has been permitted. The internal logging service checks the logging policies to decide whether to log this specific data access. If the logging decision evaluates to true, the request details and authorisation decision are logged. The patient data is then sent to the data access service at the GP practice.

A threat and risk analysis has been conducted on the use case using the Integrating Requirements and Information Security (IRIS) framework [107]. The IRIS framework has been designed to integrate security and software engineering with user-centered approaches to security. It provides a detailed meta-model for integrated requirements and risk management. The IRIS meta-model and risk management process has been adopted to identify threats and examine associated risks.

The aim of the analysis is to study how attackers might exploit potential security vulnerabilities to compromise the log integrity, confidentiality and availability. The analysis also highlights the challenges of managing audit-based monitoring services.

3.2.1.1 Valuable Assets

Four valuable assets[4] are defined in the first step. Each asset is assessed with respect to integrity, confidentiality and availability (see Definitions 2.4, 2.5, and 2.6).

Log Data Both the integrity and confidentiality of the individual log events as well as the totality of the logs need to be protected; the availability of the logged data also needs to be assured upon real-time monitoring and updates.

Processed Audit Trails The logs, collected from various sites, are reconciled and processed into meaningful audit trails (see Definition 2.3) for analysis; again, the integrity and confidentiality of these audit trails need to be protected.

Audit and Logging Services These refer to various logging components and policies, as well as the system level software components necessary to enable

[4]These assets are used for identifying threats in all later use cases.

Score (TL)	Threat Likelihood
0	Incredible
1	Improbable
2	Remote
3	Occasional
4	Probable
5	Frequent

Score (VS)	Vulnerability Severity
0	Negligible
1	Marginal
2	Critical
3	Catastrophic

Score	Risk Rating
1	Intolerable
2	Undesirable
3	Tolerable
4	Negligible

	Risk Scores From TL and VS			
TL / VS	Catastrophic	Critical	Marginal	Negligible
Frequent	1	1	1	2
Probable	1	1	2	3
Occasional	1	2	3	3
Remote	2	3	3	4
Improbable	3	3	4	4
Incredible	4	4	4	4

Figure 3.2: Threat Likelihood (TL), Vulnerability Severity (VS), and Risk Rating Tables (Adapted from [107])

a trustworthy logging platform; both the integrity and availability of these software need to be assured.

Job Secrets The integrity of the log access queries and confidentiality of the user credentials need to be protected.

3.2.1.2 Major Stakeholders

In the second step, four major stakeholders[5] are identified:

Privileged Users These are system administrators who have enough privileges to modify the audit or logging behaviour of a system.

Log Owners Log owners specify what events should be logged and who should have access to the logged data.

Log Users These are end users who perform analysis on the processed audit trails.

Intruders These are unauthorised users trying to achieve a number of malicious objectives: (1) to compromise the logging services, (2) to steal or tamper with the logged data, (3) to steal the job secrets, and (4) to corrupt the returned data and end results.

[5]These stakeholders are also used in all later use cases.

3.2.1.3 Threat and Risk Analysis

A threat and risk analysis goes through a three-step process of:

1. Identifying *vulnerabilities* — vulnerabilities are weaknesses in an asset or group of assets that can be exploited by one or more threats [64].

2. Identifying *threats* — threats are potential causes of unwanted incidents or events (sources of harm to vulnerabilities); the attacker, their motive, and their target characterise threats [64].

3. Identifying *risks* — risks are combination of the probability (or likelihood) of events (e.g. attacks exploiting vulnerabilities to compromise an asset) and their consequences [64].

The IRIS framework applies the Threat Likelihood (TL) and Vulnerability Severity (VS) tables of IEC 61508 [1], and assigns a Risk Rating based on these scores (see Figure 3.2). Mindful of the stakeholders' objectives, potential threats and their risks are rated for each asset (see Tables 3.3 and 3.4):

Asset: **Log Data**			
Vulnerability	*Threat*	*TL/VS*	*Risk* (*Rating*)
Jobs are executed without isolation and input validation.	Malicious researchers might try to cover up evidence of their previous queries by submitting an arbitrary query/code.	2/2	Deletion, modification, or arbitrary insertion of the logs at the specialist clinic (3).
A large attack surface of the middleware installed on the system at the specialist clinic.	Intruders might perform privilege-escalation attacks on the middleware.	3/3	Intruders could gain sufficient privileges to steal the log data (1).

Table 3.3: Logging Distributed Data Access — Threats on Log Data

Asset: **Audit and Logging Services**			
Vulnerability	*Threat*	*TL/VS*	*Risk(Rating)*
Jobs are executed without isolation and input validation.	Malicious researchers might submit an arbitrary query/code to compromise the logging service at the specialist clinic.	2/3	Logging services could be configured to miss the data access requests (2).
An attack surface of middleware installed on the system at the specialist clinic.	Intruders might perform privilege-escalation attacks to compromise the logging service.	3/2	Intruders could gain sufficient privileges to change the logging service configurations (2).

Table 3.4: Logging Distributed Data Access — Threats on Logging Services

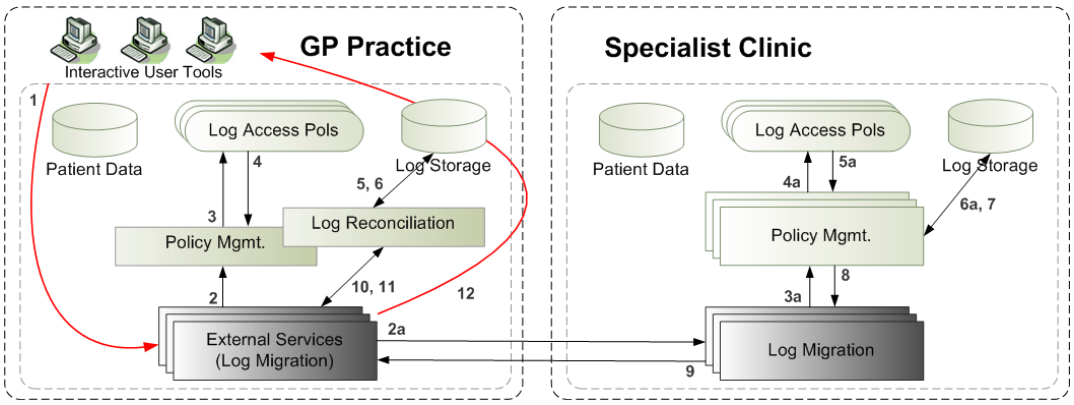


Figure 3.3: Use Case — Dynamic Access Control Policy Update

3.2.2 Dynamic Access Control Policy Update

A system administrator at the GP practice wishes to monitor the access control policies for patient data being updated (see Figure 3.3). These policies evolve around the information pertaining to data access logs collected from the specialist clinic as well as the locally stored logs. Each hospital manages their own policies governing the log access.

In the first step, a monitoring request is sent from an interactive user tool to an external service at the GP practice. Having been configured to update the policies on an hourly basis, the log migration service sends a log access request to

the external migration service at the specialist clinic. This request is forwarded to the policy management service at the specialist clinic, which reads the log access policies and checks whether the GP practice is authorised to access the logs. If access is permitted, the logs are sent to the migration service at the GP practice. The reconciliation service uses these logs to update the access control policies. The summary of the policy updates is then made available to the administrator.

3.2.2.1 Threat and Risk Analysis

This use case is instrumental for identifying threats and their risks that could potentially lead to compromise of the log confidentiality:

Asset: **Log Data**			
Vulnerability	*Threat*	*TL/VS*	*Risk*(*Rating*)
Administrators (at the GP practice) have full read access to collected logs.	Rogue administrators might freely access collected logs.	3/3	Disclosure of privileged log data (1).
Insecure communication channels.	Intruders might try to sniff the logs or redirect the traffic to a malicious machine (with man-in-the-middle type of attacks).	2/3	Unauthorised access to privileged log data (2).
Incompetent software/hardware mechanisms for filtering unwanted packets.	Intruders might perform denial-of-service attacks on the external migration service to make the specialist clinic system unavailable.	2/3	The specialist clinic system could become saturated with external requests and fail to respond to legitimate log access requests (2).

Table 3.5: Dynamic Access Control — Threats on Log Data

Asset: **Processed Audit Trails**			
Vulnerability	*Threat*	*TL/VS*	*Risk(Rating)*
Administrators (at the GP practice) have full read access to the processed audit trails.	Rogue administrators might freely access all information pertaining to the audit trails.	3/3	Unauthorised access to patient information (1).

Table 3.6: Dynamic Access Control — Threats on Processed Audit Trails

Asset: **Audit and Logging Services**			
Vulnerability	*Threat*	*TL/VS*	*Risk(Rating)*
An attack surface of the external log migration service at the GP practice.	Intruders might perform in-memory (e.g. buffer overflow) attacks to compromise the service.	2/3	Behaviour of the service could be altered; for example, to disclose the collected logs (2).
Configuration files of all services are modifiable.	Rogue administrators (at the GP practice) might change the service configurations with malicious objectives.	3/3	For instance, the log migration service could be configured to copy collected logs in an unencrypted local disk (1).

Table 3.7: Dynamic Access Control — Threats on Logging Services

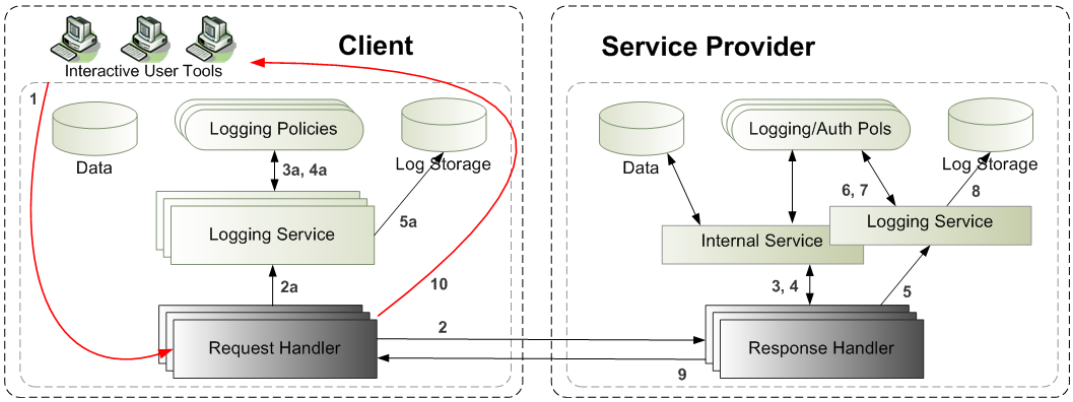


Figure 3.4: Use Case — Recording Service Requests and Responses

3.2.3 Recording Service Requests and Responses

The next two use cases are consolidated from the example on service-level agreements (see Section 3.1.2). In this scenario, the client wishes to send a request for a service (see Figure 3.4). Typically, a service would perform some processing (which may involve a use of the service provider's data), generate a response, and send it to the client. These procedures would have to comply with the conditions stated in the service-level agreements. Both the request and response details would have to be logged at all relevant points.

An interactive user tool is used to send a service request to the external request handler. The request handler submits the request to the service provider's external response handler, and, at the same time, sends the request details to the internal logging service. The service provider's response handler forwards the request to an internal service, which performs certain operations and generates a result. Both the incoming request and outgoing response details are forwarded to the logging service and recorded. The response handler then sends the results to the client's request handler, which, in turn, forwards the results to the end user. The client's logging service records the response details in association with the original request.

3.2.3.1 Threat and Risk Analysis

The following analysis demonstrates how insiders might affect the integrity of the logging services to fabricate the log events being generated:

Asset: **Audit and Logging Services**			
Vulnerability	*Threat*	*TL/VS*	*Risk(Rating)*
The logging service, in both platforms, can be modified and redeployed by administrators without the other knowing.	Rogue administrators might deploy a modified logging service that fabricates the service request or response details.	3/3	An inaccurate logging service could be deployed to record misleading information (1).
Request and response handlers, in both platforms, can be modified and redeployed by administrators.	Rogue administrators might modify the request/response handlers to submit fabricated details.	2/3	Fallacious logging requests could be submitted from these components (2).

Table 3.8: Recording Service Request/Response — Threats on Logging Services

3.2.4 Generating Cross-Domain Audit Trails

Consider a scenario where the client has not received a timely response for a service. Being convinced that this violates the service-level agreement, the client wishes to file a claim report based on the cross-domain audit trails of the requested service.

The information flow is much like the dynamic policy update use case (see Section 3.2.2). The client's log reconciliation service processes logs collected both locally and from the service provider. In consequence, a complete, chronological report is generated. This report is used to check whether the service-level agreement has been violated.

3.2.4.1 Threat and Risk Analysis

The following analysis shows: (1) how various stakeholders might compromise the log integrity to fabricate the end results, and (2) how intruders might exploit the system vulnerabilities to compromise the job secret.

Asset: **Log Data**			
Vulnerability	*Threat*	*TL/VS*	*Risk(Rating)*
Administrators have full read and write access to the logged data.	Rogue administrators, in both platforms, might modify, delete, or insert arbitrary logs to cover up evidence, or make false claims.	3/3	Fabrication of the log data (1).

Table 3.9: Cross-Domain Audit Trails — Threats on Log Data

Asset: **Job Secrets**			
Vulnerability	*Threat*	*TL/VS*	*Risk(Rating)*
An attack surface of the complex middleware.	Intruders might try to perform privilege-escalation attacks to compromise the middleware and steal the job secrets.	2/3	Compromise of middleware; unauthorised access to the job secrets (2).
An attack surface of the service provider's system.	Intruders might try to exploit security vulnerabilities of the system.	2/3	System compromise; unauthorised access to the job secrets (2).
Administrators of the service provider's system have full read and write access to the incoming jobs.	Rogue administrators might modify the log access query to miss certain data.	2/3	A modified query could be executed without the client knowing (2).

Table 3.10: Cross-Domain Audit Trails — Threats on Job Secrets

3.2.5 Monitoring Lawful Interceptions

The last use case is derived from the public communications network example (see Section 3.1.4). A system operator of a mobile telecommunication network

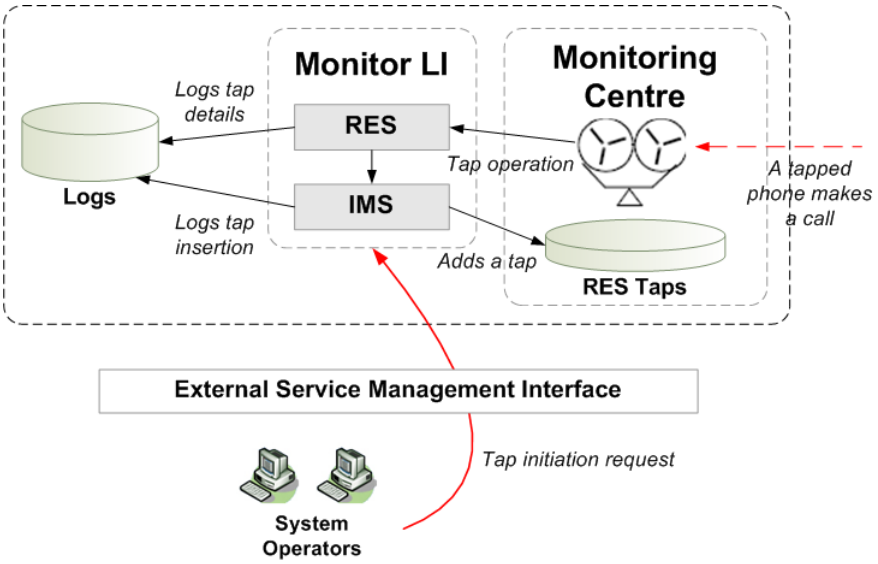


Figure 3.5: Use Case — Monitoring Lawful Interceptions

wishes to initiate tapping, and monitor calls for, a list of phone numbers by initiating a lawful interception (see Figure 3.5). The Remote-control Equipment Subsystem (RES) carries out the actual tapping operation, and the Interception Management System (IMS) initiates it by adding it to the RES database [43]. In a securely configured lawful interception system, both the RES and IMS components should log all calls being tapped. Then, these logs would be used to detect any unauthorised tap.

Firstly, the system operator sends a tap initiation request to the IMS through an external service management interface. The IMS verifies whether the operator has sufficient privileges and logs the authorisation details. If the operator is authorised, the IMS adds a tap to the RES database and logs this tap insertion. When tapped phones make (or receive) calls, the RES monitors them on switched connections, transfers the contents back to the IMS, and logs the details.

3.2.5.1 Threat and Risk Analysis

This use case serves to highlight potential problems with relying on the system itself to generate security logs and protect logged data.

Asset: **Log Data**			
Vulnerability	*Threat*	*TL/VS*	*Risk(Rating)*
An attack surface of the service management interface.	Intruders might perform privilege-escalation attacks to access the logged data.	2/3	Intruders could gain read and write access to the logged data; and for instance, hide unauthorised interception of calls (2).
Administrators have full read and write access to the logged data.	Rogue administrators might freely modify, or delete the logs to hide unauthorised interceptions.	1/3	Logs could be modified or deleted, and fail to indicate unauthorised taps (3).

Table 3.11: Monitoring Lawful Interceptions — Threats on Log Data

Asset: **Logging Services**			
Vulnerability	*Threat*	*TL/VS*	*Risk(Rating)*
An attack surface of the external service management interface.	Intruders might perform in-memory (e.g. buffer overflow) attacks to modify behaviour of the internal services.	3/3	The RES or IMS could be modified; for example, to stop logging tapping transactions (1).
Both the RES and IMS can be modified and redeployed by administrators.	Rogue administrators might modify these components to miss, or fabricate certain tapping transactions.	1/3	Inaccurate tapping details will be logged, and used later upon audit (3).

Table 3.12: Monitoring Lawful Interceptions — Threats on Logging Services

3.2.6 Summary of the Key Threats

This section identifies the common, key security threats from the analysis above and generalises them:

1. **Deletion, Modification or Insertion of the Logs** Intruders might try to cover up evidence of their malicious attempts by manipulating the logs (see Table 3.3). It is also possible for a rogue insider (with sufficient privileges) to fabricate the log data (see Table 3.9).

2. **Unauthorised Access to the Log Data** Intruders might try to steal the administrator rights to read the privileged log data (see Table 3.3). Intruders might also try to sniff the logs that are being transferred across an insecure communication channel (see Table 3.5).

3. **Modification of the Logging Service Configuration** As shown from Table 3.4, rogue (yet legitimate) users or intruders might try to modify the configuration of the logging service to miss certain transactions.

4. **Authorisation Violation** Rogue insiders (e.g. administrators) might try to disclose the privileged log data or the processed audit trails (see Table 3.5). They might also try to change the behaviour of one of the logging services to manipulate the way the logs are being generated (see Table 3.8).

5. **Denial-of-Service** Table 3.5 demonstrates that intruders might try to perform denial-of-service attacks on the log owner's external services and affect the log availability.

6. **Modification of the Logging-Related Services** Intruders or rogue insiders might try to modify the behaviour of one of the logging-related services (e.g. the log migration service) to disclose the log data (see Table 3.7). Intruders might also try to compromise the logging service and change its behaviour to miss certain transactions (see Table 3.12).

7. **Middleware Compromise** Intruders might try to compromise the complex middleware to gain unauthorised access to the job secrets as well as the accessed logs (see Table 3.10).

3.3 Audit and Logging Requirements

The use cases have in common a likely reliance upon technical measures of security aimed at substantially enhancing protection for log integrity, confidentiality and availability. This section identifies a unifying set of security requirements based on the use cases and the threats associated with each.

3.3.1 Involuntary Log Generation

In the use cases discussed above, the applications themselves are often log triggers (components that trigger logging requests) and responsible for protecting the logging logic as well as the logged data. Such applications, however, usually contain design or security flaws that adversaries could exploit to manipulate the way logging requests are being triggered. Hence, it is important to isolate the logging component and manage it *independently* from any other software. The logging component should always be available to capture application level events *involuntarily*. This is to ensure that even if the applications are modified to affect the behaviour of the log triggers, the system would still generate trustworthy logs at the system level. Threats 3, 4 and 6 (see Section 3.2.6), which discuss the possibility of the logging component being misconfigured or compromised, can be mitigated with this isolation and involuntary generation of logs.

A log user should be able to verify the integrity of these logging properties and mechanisms in a remote platform without the administrator's interference. Ideally, the logging component should be a small and simple piece of software designed to minimise the attack surface and improve the usability of attestation (see Section 2.4.4).

Despite the potential security loopholes the applications might have, some of the security critical decisions made by the applications should be captured in a trustworthy manner as possible. These should be classified differently — perhaps by labeling them with a lower trust level — from the logs generated independently from the applications.

3.3.2 Protected Log Storage

No matter how robust the logging component might be, if an intruder succeeds in obtaining administrator privileges, they can potentially access, modify or delete

the logged data directly from the disk.

As a defence-in-depth measure, the log data should be encrypted and integrity-protected on the physical storage device. Confidentiality should be guaranteed by encrypting and decrypting the log data as they are being read and written to the disk. Integrity should be implemented through the means of digital signatures and hashes: Secure Audit Web Service (SAWS) [135], for example, uses a combination of these techniques to provide a tamper-evident log storage. The cryptographic keys used in these operations should only be accessible by the authorised software.

The security mechanisms discussed here should be sufficient to mitigate threats 1 and 2 (see Section 3.2.6), which discuss the possibility of intruders or rogue insiders gaining unauthorised access and manipulating the log data. To mitigate the 'authorisation violation' threat (threat 4), one could also consider using a 'dual control' mechanism. This is typically implemented by requiring two keys, or `n` from `m` key-shares, to access sensitive log data.

3.3.3 Authorisation Policy Management

Each site should control incoming log access queries by enforcing locally managed authorisation policies. This requirement is well captured in the healthcare grid example (see Section 3.1.1), where a form of 'traffic flow' analysis might yield sensitive information about patients, and so the log access should only be granted to the authorised users. These policies should aim to restrict unauthorised access and prevent attackers from performing inference attacks. Threat 2 (see Section 3.2.6), which considers intruders gaining unauthorised access to the privileged log data, can be mitigated using this type of authorisation policy management service.

3.3.4 Log Migration Service

Due to the number of potential security vulnerabilities, complex middleware services can not be relied upon to perform trusted operations (see Section 2.6.3). Instead, the security controls required for secure job submission and log transfer should operate within a more trustable migration service. This implies data flow encryption and signing requirements upon log access and transfer. These are integral to protecting the job secrets and log data from the untrusted middleware

stack and from man-in-the-middle type of attacks [141]; these mechanisms would be responsible for mitigating threat 7 (see Section 3.2.6).

It is also possible for a log owner to deliver fabricated logs, as the service provider might do in the service-level agreement use case (see Section 3.2.4.1). To provide a safeguard against such a threat, the migration service should access the signed logs and the hash from the protected log storage through a secure channel. This should give sufficient information for an end user to verify the log integrity and authenticity.

3.3.5 Protected Execution Environment

As shown in Table 3.10, a rogue administrator could modify the log access query to miss certain logged data. There is also a possibility of an intruder compromising the host through privilege-escalation attacks, and tampering with the logged data. To mitigate such threats, the log owner's platform should provide a protected, isolated environment where the jobs can execute free from unauthorised interference. The log user should be able to verify this execution environment prior to job submission.

The job secrets, including the user credentials and the log access queries, should not be decrypted and accessed outside the protected environment. Further, these secrets should not be modified even within the protected environment. Unauthorised modification of the execution environment (for example, installing an unsigned patch) should be detected, and a modified environment should not be used for job execution.

From the log owner's perspective, this execution environment should amply isolate rogue jobs and prevent them from compromising the logs or the logging services (see Table 3.4). The log owner should specify the software pre-installed in the execution environment — only allowing those trustworthy to be used for accessing their database. Presumably such software should perform a thorough input validation before allowing the query to run.

3.3.6 Log Reconciliation Service

Before granting access to the logs, the log owners should be able to verify the security state of the requesting log reconciliation service. This verification should ensure that the logs will be used unmodified, in a protected environment.

The integrity and confidentiality of the collected logs as well as of the processed audit trails (e.g. service-level agreement violation report) should be protected. Privileged users or intruders should be prevented from stealing sensitive information or fabricating the results.

To make it difficult for insiders to gain unauthorised access, the reconciliation service should be installed on an isolated compartment and have robust memory protection. It should be a small and simple piece of code to minimise the number of security holes that might be exploited.

These security mechanisms should be sufficient to mitigate threats 2 and 4 (see Section 3.2.6), which consider intruders or rogue administrators gaining unauthorised access to the logs and audit trails being processed at a remote collection point.

3.3.7 Blind Log Analysis

Returning to the healthcare example (see Section 3.1.1), imagine that the specialist clinic has agreed to share their logs with the GP practice for dynamic access control. But at the same time they are unwilling to let the administrator at the GP practice see the actual contents of the logs; or only let part of the data be seen as a summary information. For example, "the researcher's access rights on T_1 for a patient with NHI 1, aged 20 and living in OX2 area, have been restricted to NHI and *Smoke* fields". Such anonymisation of end results ensures that the administrator cannot find out about a patient's lung cancer status, and yet still know exactly how the access control policy has been changed for the researcher.

The log owners should be assured that the sensitive information will only be revealed to an extent that has been agreed and stated in the 'log privacy policies'. This requires a mechanism, possibly within the reconciliation service, to carry out a *blind analysis* of the collected logs so that the user only sees the running application and the end results. These results should be just sufficient for the user to carry out an analysis or to know about important system updates.

3.3.8 Surviving Denial-of-Service Attacks

Denial-of-service attacks [76] could affect the availability of the logs (see Table 3.5). The vulnerability severity would be particularly high in real-time applications like the dynamic access control system which relies on timely data feeds.

To survive this type of attack, the log owner's system should be equipped with both hardware and software means (including firewalls and switches) to filter unwanted requests. The 'denial-of-service' threat, as explained in Section 3.2.6, can be mitigated by deploying these security mechanisms.

3.4 State of the Art and Gap Analysis

Having identified the security requirements, this section analyses the 'remaining gap' between these requirements and the security mechanisms used in existing solutions.

3.4.1 Log Generation

Tierney and Gunter [20] have developed a logging toolkit, known as Networked Application Logger (NetLogger), for monitoring behaviour of various elements of application-to-application communication paths in distributed systems. NetLogger provides client libraries (C, C++, Java and Python APIs), which can be used by software developers to write logging code in their software and generate application-level logs in a common format. It uses the IETF-proposed Universal Logger Message (ULM) [7] format for logging and exchanging messages. The use of a common format (which is plain ASCII text) is intended to simplify the process of reconciling potentially huge amounts of distributed log data.

In NetLogger, it is the responsibility of the developers to instrument applications to produce log events by calling methods from the NetLogger API. Typically, the developers determine security-critical points in the code (subject to logging) and what information gets logged. The main problem with this type of approach is that the developers, who may be inexperienced with security practices, could omit important details or fail to sufficiently pin down the critical points. Moreover, the process of modifying or adding new logging conditions will be inflexible — every time a logging condition needs to be added, the developers would have to change the code, recompile the software, and redeploy it.

More problems arise from relying on the application itself to protect the logging code. The danger is that attackers could exploit the attack surface of the application (e.g. by exploiting a buffer overflow vulnerability) to alter the logging behaviour. The bigger the software, the more likely it is to expose security

vulnerabilities in its attack surface [81]. Therefore, it is not a good idea to merely rely on the application itself to log security events — more trustable, independent logging components are required (see Requirement 3.3.1).

A common problem with state of the art logging services is their lack of flexibility in both the generation and the enforcement of logging conditions [35]. Typically, the logging conditions are already defined in operating system components or hard-coded by software developers. Widely used system-level logging systems like the Event Log subsystem [137] on Windows and Syslog [82] on Linux all rely on the developers to write the logging triggers. In consequence, modifying the logging conditions in these systems is difficult for normal users.

To overcome this problem, Vecchio et al. [35] have proposed a secure logging infrastructure for grid data access that leverages OASIS eXtensible Access Control Markup Language (XACML) [88] to allow data owners to specify logging policies dynamically (see Figure 3.6). Such policies allow data owners to specify conditions that, if satisfied, result in logging of details about a data access.

One of the major concerns with this approach is that a privileged user can easily alter the system's logging behaviour by changing these policies. For example, considering the threats identified in Section 3.2.3, it is not hard to imagine a rogue administrator changing the logging conditions to fabricate the service requests or response details being recorded. Unless a trustworthy monitoring service notifies the other party about the modified policies (see Requirement 3.3.1), it would be difficult to detect changes made in remote systems. Hence, this type of approach alone can not provide the security guarantees necessary for monitoring service-level agreements.

Their Auditing Decision Point constructs a logging request context to be evaluated against the logging policies (see Figure 3.6). If the auditing decision evaluates to true, the remote Auditing Service is used to record the data access. The Auditing Service is a .NET web service which simply accepts the logging requests and stores them locally. Despite their attempt to isolate the Auditing Service from other software, they do not discuss how the integrity of the Auditing Service will be protected. If attackers manage to penetrate the web server and compromise the Auditing Service, they could potentially sniff all logging requests and tamper with the details.

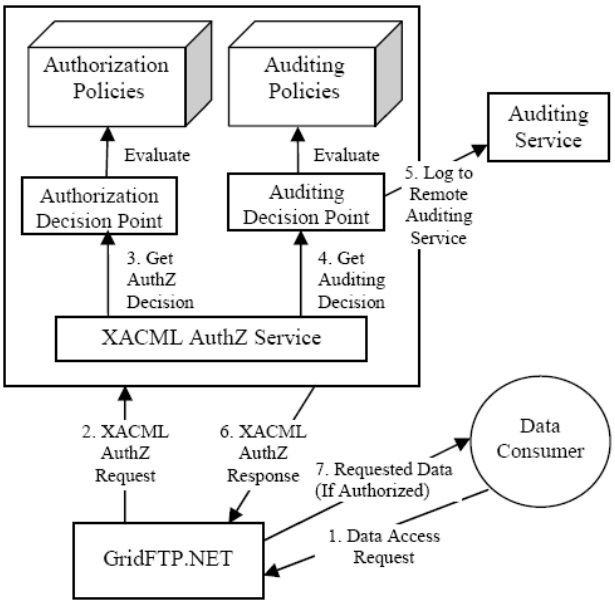


Figure 3.6: Flexible Auditing Architecture (Figure 2 from [35])

Cordero and Wagner [17] have proposed a trustworthy logging tool for post-election investigations. The tool has been designed to record all interactions between the voter and the voting machine (for example, every button pressed, or every location on the screen touched by the voter), while still preserving voter anonymity. As a result, the logs provide a complete record of everything the voter saw and did in the voting booth. The idea is to facilitate precise reconstruction and speculation of the voter's intent in case of election disputes.

They have stressed the importance of providing an *independent* way of logging the voter's actions, and suggested the use of virtual machine isolation to achieve this (see Figure 3.7). Strong isolation prevents the voting system from bypassing or manipulating the logging subsystem.

Their approach, however, still fails to achieve full independence — in fact, they state that they are not sure as to how logging mechanisms could be truly independent from the voting software. One of the reasons for this is that the developers are expected to write the logging triggers (that would call the methods from the logging subsystem) in the voting software. For example, the code that reads a touch event would call `Log.log( "touch", x, y )` to record the x and y coordinates when the voter touches the screen. Similarly, a logging trigger is

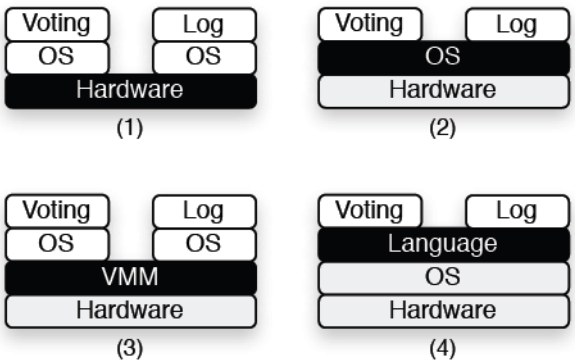


Figure 3.7: Isolation Ideas for a Logging System (Figure 2 from [17])

added to every voter-visible I/O event to capture the voter's intent.

Consequently, the surveillance of the logging behaviour is still only as good as that of the voting system. Intruders, for instance, could exploit the buffer overflow vulnerability of the voting software and turn off the logging triggers, or force them to record arbitrary values. Unless the voting system is completely free of bugs, there will always be an attack surface that could be exploited to manipulate the internal logging triggers.

To achieve stronger software independence, Garera and Rubin [119] have discussed the idea of installing the logging system in Domain-0 (the monitor virtual machine in Xen) to capture all user inputs to the voting machine directly. As a result, correct inputs (such as candidate selection) are recorded even if the voting machine is compromised. Their system has been developed at the same time as the author's own work and there is some overlap in the proposed logging ideas.

Capturing the I/O events might be sufficient for a less dynamic system like the voting machine. In systems where applications or guest operating systems also make important security decisions, however, merely recording I/O events would be insufficient to generate a comprehensive incident report (see Requirement 3.3.1). The thesis proposes a technique for capturing such security decisions in a trustable manner (see Section 4.2.6).

The threat model of their audit framework assumes those who have privileged access to the voting machine (such as the administrators) are trusted. However, as the threats in Tables 3.8 and 3.9 show, rogue administrators could potentially deploy a modified logging service to fabricate the voting events, or directly access

the log storage and modify the vote count. Such threats have also prevented the development of reliable monitoring services for service-level agreements.

To relax this kind of constraint, the thesis discusses the use of trusted computing to allow the logging system to report its integrity *without* the administrator's interference. Using this integrity reporting mechanism, the log users can verify the logging properties of a remote system and check whether the logs are trustworthy.

'Xenlog' [95] is another Xen-based logging solution which aims to mitigate threats associated with exchanging the logs across an unprotected network. The monitor virtual machine, Domain-0, has been modified to protect the logging component and central log filesystem. Typically, centralised logging services are made available through the public network (e.g. Syslog [82]). The danger is that the data packets could easily be sniffed or the traffic redirected to a malicious machine. In Xenlog, the centralised logging service and log storage are not exposed to the network; instead, the logs are forwarded internally using a shared memory mechanism between the guest virtual machines and Domain-0.

One of the concerns with Xenlog, however, is that the isolation of Domain-0 solely is relied upon to protect the log integrity and confidentiality. It does not provide a secure storage mechanism for the logs. If a rogue job manages to hijack a privileged process in Domain-0, it would then be able to freely access or tamper with the logged data. Unencrypted logging requests — being delivered through the shared memory — would also be vulnerable to this attack. A defence-in-depth technique is required to mitigate this type of attack: generating a hash and signature for each log record and encrypting the sensitive information (see Requirement 3.3.2).

Moreover, there is no security mechanism for verifying the log triggers (guest applications) and the logging requests. It is then only a matter of compromising the logging logic inside a guest application to affect the entire logging behaviour.

Stathopoulos et al. [114] have developed a framework for secure logging in public communication networks. The aim of their work is to detect modification attacks against log files by generating digital signatures offline, and protecting these signatures within a trusted 'regulator authority'. Their framework provides a powerful mechanism for detecting unauthorised modifications to the stored log data (assuming these have been generated from trusted sources). However, it

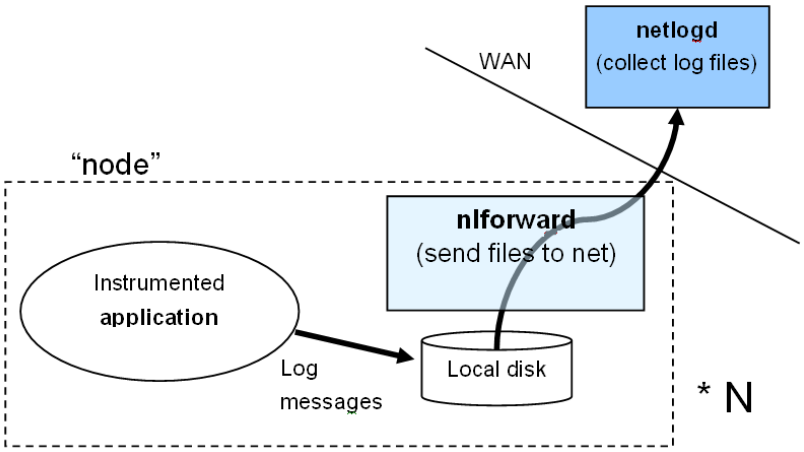


Figure 3.8: NetLogger

does not prevent a compromised server from modifying the logs before being signed, or simply inserting arbitrary logs.

3.4.2 Distributed Log Access and Reconciliation

The registry of information enables accounting and auditing in many shared grid and distributed systems. Traditional monitoring techniques (for recording resource usage, scheduling tasks, and auditing), however, have often been found to be inadequate for accounting dynamically distributed resources [68]. The main problem is that they do not accurately capture true information, and even if they do, the integrity and confidentiality of the information are not well protected. This prevents trustable and equitable access to distributed resources.

In NetLogger [20], logs (generated from distributed applications) are scanned every few seconds and forwarded to a central server (see Figure 3.8). A server daemon, called 'netlogd', collects these logs, sorts, and stores them in a file on the central server's local disk. In consequence, applications can log events in realtime to a single destination over the wide-area network. NetLogger enables flexible and interactive analysis of these centrally merged audit trails with the provision of a graphical visualisation tool. Several researchers [54, 29] have discussed the use of NetLogger to enhance monitoring and troubleshooting services for grid components.

The log integrity and confidentiality threats discussed above, however, undermine their security model: log access requests are processed without any authorisation policy enforcement (see Requirement 3.3.3), and the logs are transferred across the network unencrypted or integrity protected. Further, there is no attempt to safeguard the logs while they are being collected and processed at the central reconciliation point (see Requirement 3.3.6).

Similar security problems undermine other monitoring tools such as Accounting Processor for Event Logs (APEL) [22], which builds accounting records from system and gatekeeper logs generated by a site; and Grid Monitoring System (GMS) [85], a system that captures and stores grid job information in a relational database, and supports resource usage monitoring and visualising.

Most of the current projects (including the ones mentioned above) have been developed for computational grids, and they are primarily focused on measuring usages of the hardware (CPU, memory, disk), the jobs in execution, and the load of the system. However, there are other important events in the grid that should also be tracked: for example, who accesses a service or data and when. Service-oriented events as such are unlikely to be captured with traditional approaches.

To overcome these inadequacies, Alfonso et al. [37] have developed a Distributed General Logging Architecture for Grid Environments (DiLoS). Their architecture provides general logging facilities in service-oriented grid environments to enable tracking of the whole system. One of its application models is to facilitate accounting for resource-providing services to measure and annotate who has used which services and to bill users.

In this accounting domain, however, DiLoS does not consider the log integrity issues covered in Section 3.2.4.1. Without strong security mechanisms to protect the log integrity, their architecture cannot be relied upon to perform calculating and billing functions.

Piro et al. [102] have developed a more secure and reliable data grid accounting system based on resource usage metering and pricing. All communications are encrypted [101], but a privileged user may still configure the Home Location Register, a component that collects remote usage records for accounting, to disclose sensitive usage records and compromise confidentiality and privacy (see Requirements 3.3.6 and 3.3.7). A rogue resource owner may modify the Computing Element, which measures the exact resource usage, in order to fabricate the

usage records and prices for profit.

Despite these security weaknesses, not much research has been conducted on how trusted computing could be used to bridge the gap. The thesis discusses the use of remote attestation and the sealed key approach (see Section 2.6.1) to facilitate trustworthy log reconciliation, ensuring log integrity, confidentiality, and availability (see Chapter 5).

Chuvakin and Peterson [16] discuss some of the challenges of logging in distributed environments such as those of web services. In the simplest case, where two machines are talking to each other using a web service, events would be logged independently on both machines. The challenge is that there is no consistency between these logs generated from loosely coupled services: decisions are made locally, so what to log (logging policies), how to log (standards and formats), and how to analyse the data are likely to be managed differently by each service. Hence, a complete view of the overall architecture will not be available. Additional technologies are required for consistent aggregation and reconciliation of distributed logs.

3.5 Chapter Summary

Based on a number of motivational examples and use cases associated with each, a threat and risk analysis has been conducted in this chapter. Then, a unifying set of trustworthy audit and logging requirements, which pin down the services and mechanisms necessary to mitigate the key threats, has been identified. In the final section, the gap between these requirements and existing logging solutions has been analysed. The next chapter will describe a trustworthy logging system that has been designed with these requirements in mind.

Chapter 4

Design: Log Generation

This chapter describes a trustworthy logging system that aims to satisfy the *involuntary log generation* and *protected log storage* requirements.

Section 4.1 gives an overview of the complete log generation and reconciliation architecture and describes its trusted computing base. Section 4.2 focuses on the log generation aspects of the architecture and explains how the involuntary logging and secure log storage mechanisms are orchestrated together. Finally, Section 4.3 observes some of the remaining issues and challenges of implementing the logging system.

4.1 Architecture Overview

Mindful of our security requirements (see Section 3.3), this section proposes a trustworthy log generation and reconciliation architecture based on trusted computing capabilities (covered in Sections 2.4 and 2.5). Upon installation of this architecture, a participant in a distributed system will be capable of generating and storing logs, and proving to others that these logs are trustworthy.

Figure 4.1 shows an overview of the logging components which form the trusted computing base of the proposed architecture. All the log security functions are enforced by the 'log transit' component deployed inside the privileged monitor virtual machine (referred to as Domain-0 in Xen) and the 'log access manager' virtual machine. These are designed to perform a small number of simple operations to minimise the chance of them containing any security vulnerability.

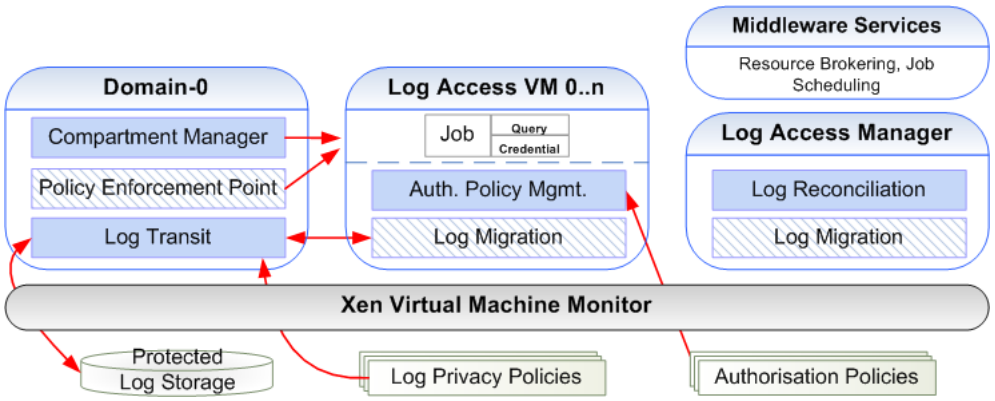


Figure 4.1: Architecture Overview

Attestation of these two components as well as the 'Policy Enforcement Point' (PEP) and the virtual machine monitor is sufficient for one to establish trust with the logging system, and know that its log security functions have not been subverted. This relatively static and small trusted computing base aims to minimise any complexity involved in performing remote attestation and improve its usability. The importance of establishing a minimised form of static identity has been covered in Sections 2.4.4 and 2.6.2.

A default setting in Xen requires all guest virtual machines to communicate with Domain-0 to access the physical hardware. Taking advantage of this, the log transit monitors all I/O requests and responses inside Domain-0, captures the events of interest, and records the use of device drivers. Such events are recorded *independently* from applications running in the guest virtual machines.

The log access manager virtual machine is responsible for securely collecting distributed logs and processing them into useful audit trails for analysis:

- The externally facing 'log migration service' enforces the security controls required for safe log transfer across the untrusted middleware stack; this external service also runs inside the per-job 'log access virtual machines' to access the logs and send them back securely to the requestor log access manager.

- The 'log reconciliation service' facilitates trustworthy reconciliation of the collected logs and processes them into meaningful audit trails; it enables *blind log analysis* by enforcing sticky log privacy policies (specified by the log

owners) during the process; in consequence, only the processed, anonymised information — for which the log owners have agreed to disclose — are released to the log users.

These two services are trusted and measured as part of the log access manager. The log migration service also provides the interface necessary for various analysis tools to specify the log access job requirements. Such end-user tools only have access to this restricted, minimal interface available through the migration service: again, in order to reduce the number of security holes which might be exploited at runtime. After log reconciliation, only the processed results are forwarded to the analysis tools. The raw data, therefore, never leaves the log access manager.

When a log access job reaches a participant platform, the policy enforcement point (inside Domain-0) first checks the security configuration of the job and the log requestor platform. If the job is trustworthy, a per-job log access virtual machine is launched to process the job. The second trustworthy service that runs in the log access virtual machine is the 'authorisation policy management service'. Before granting access to the log data, it evaluates the log requestor's access rights defined in the authorisation policy. This ensures only the authorised users access the log data and makes it difficult for attackers to perform inference attacks.

From the log owners' perspective, this approach ensures that the security operations like job authentication and authorisation policy enforcement are always governed by the services they trust. The migration service encrypts the logs upon their transfer to protect confidentiality. In order to take control of the host platform, a rogue job would have to compromise both the virtual machine monitor and Domain-0 — components which are designed to resist such attacks.

Meanwhile, the log users are guaranteed that their log access query runs unmodified in a trustworthy virtual machine, where the logs are always accessed via trusted middleware services. This should be sufficient for the users to verify the accuracy and integrity of the collected logs. Any arbitrary alterations of the job execution environment (i.e. the attested virtual machine image) will be detected by the policy enforcement point and the users will be informed. Moreover, strong isolation between the virtual machines prevents a rogue virtual machine from reading the users' sensitive data or generated results.

4.2 Trustworthy Logging System

The rest of the chapter focuses on the 'log generation' aspects of the architecture and describes a logging system based on the shared memory mechanisms of Xen. Central to the architecture is the independent *log transit* component, which sits inside Domain-0 to involuntarily intercept all I/O events and record events of interest.

4.2.1 Assumptions

This section states two assumptions about the design of the trustworthy logging system.

1. The privileged virtual machine in Xen, Domain-0, manages all of the back-end and native device drivers; all other guest virtual machines must communicate with Domain-0 to access the physical hardware.

2. There is a mechanism available that allows the system owner to specify the logging policies — these policies state what I/O events should be logged; the proposed logging component (referred to as the log transit) uses this mechanism to determine whether to log a particular event.

4.2.2 Shared Memory Operations in Xen

In addition to the Xen components discussed in Section 2.5.2, this section further explains how the shared memory is used between the guest virtual machines and Domain-0 for exchanging I/O requests and responses.

By default a 'full administrative privilege' is given to Domain-0 for creating, inspecting or destroying virtual machines, and managing back-end device drivers [117]. It is also possible to give a 'physical device privilege' (a more restricted privilege) to a back-end driver domain to delegate all the device driver operations in a separate virtual machine. However, in order to simplify the overall workflow, the default setting — whereby Domain-0 manages all device drivers — is assumed in the proposed logging system (see assumption 1, Section 4.2.1).

The Xen virtual machine monitor provides 'shared memory' for guest virtual machines to communicate with Domain-0: the I/O ring, built on top of the

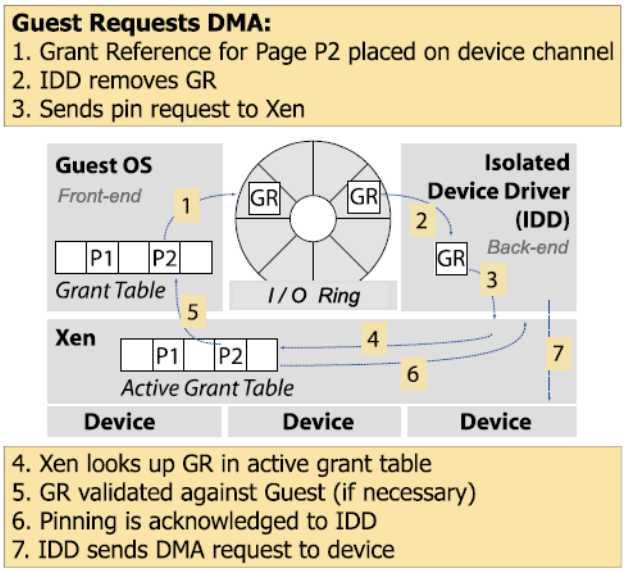


Figure 4.2: Using I/O Ring to Request a Data Transfer (Figure 2 from [49])

shared memory mechanism, is used by the front-end (guest domain) and back-end (Domain-0) device drivers to exchange I/O requests and responses [18].

Two pairs of producer-consumer indexes are used by the device drivers to update the I/O ring (see Figure 4.2). A front-end places I/O requests onto the ring, advancing a request-producer index, while a back-end removes these requests for handling, advancing an associated request-consumer index. I/O responses are queued onto the same ring, although this time with the back-end as producer and the front-end as consumer. An example I/O request generated by the front-end would look like 'read block 50 from device sda2 into a buffer at 0x5ac102'. The back-end may reorder requests or responses using a unique identifier assigned to each. The physical hardware is then accessed through the corresponding native device driver.

The 'event channel' is used by the device drivers to send asynchronous notifications of queued requests or responses [49]. Such notifications are triggered by the device drivers attached to the opposite end of the bidirectional channel (meaning each end may notify the other).

Consider a control flow of a guest process reading a file [117]:

1. The guest process invokes read() syscall.

2. The guest kernel, virtual file system layer, invokes the front-end device driver for data access.

3. The front-end places a message in the shared memory (I/O ring) and notifies the back-end device driver using the event channel.

4. The back-end picks up the message from the shared memory and validates the request.

5. The back-end translates the request and forwards it to the corresponding native device driver which in turn sends it to the physical device.

6. The physical device returns the data to the native device driver which in turn sends the data to the back-end.

7. The back-end places the I/O response on the shared memory and notifies the front-end using the event channel.

8. The front-end picks up the response and passes the data to the guest process via the virtual file system layer.

Section 4.2.4 explains how these procedures would change with the introduction of the log transit component.

4.2.3 Network Interface Card Example

Consider a network interface card usage example in Xen. Each guest virtual machine is provided with a set of virtual network interfaces for network communication [5]. For each virtual interface, a corresponding back-end interface is created in Domain-0 (or a separate driver domain if it exists) which operates as a proxy for the virtual interface.

These two interfaces exchange network packets through the shared memory mechanism. The back-end interfaces are connected to the native device driver through a virtual network bridge. The native device driver communicates with the physical network interface card and sends back responses to the back-end.

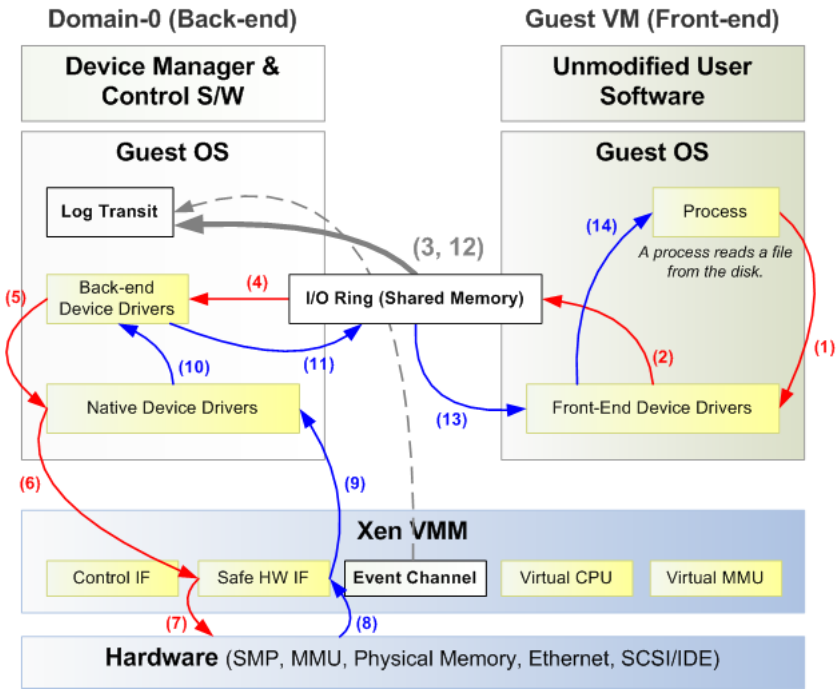


Figure 4.3: Xen-based Trustworthy Logging System

4.2.4 Involuntary Log Generation

Under the default setting in Xen, only Domain-0 has access to the physical hardware. As shown in the control flow example (see Section 4.2.2), all data access requests go through the back-end in Domain-0 before reaching the physical disk. Similarly, all network requests go through the back-end interfaces before reaching the physical network interface. In consequence, each physical device is available through a *single access point* in Domain-0.

Taking advantage of this, the *log transit* is designed to operate inside Domain-0 to *involuntarily* intercept and log I/O events of interest. As a result, only a small amount of code running in Domain-0 has control over when and how trustworthy logs (classified differently from logs generated via other sources) are generated. The logs are independently generated, solving the common problem of untrusted applications being relied upon to manage logging operations (see Section 3.4.1). The remainder of this section describes how the log transit is integrated with the shared memory mechanisms.

The event channel is configured to automatically notify the log transit when-

ever a device driver informs the opposite end about an I/O event queued in the shared memory (see Figure 4.3). As the first component to be notified, the log transit reads the queued I/O event details before the device driver. The device driver is notified when the log transit finishes reading the details. Such I/O event details — intercepted from the shared memory — are processed into a standard log format, signed, encrypted, and stored in the protected log storage.

It is assumed that a mechanism is already available for the system owner to specify *what* I/O events should be captured by the log transit (see assumption 2, Section 4.2.1). As suggested by Vecchio et al [35], languages like XACML [88] could be extended to allow the system owner to specify logging policies dynamically. After being notified about the queued I/O event, the log transit evaluates and enforces these logging policies to determine whether to log that particular event.

Figure 4.3 demonstrates how the original control flow example (see Section 4.2.2) would change with the log transit in place. When the front-end triggers the event channel to notify the back-end about an I/O request placed in the shared memory, the event channel first notifies the log transit about the request. The log transit accesses the I/O ring, and captures the request details and the front-end (subject) that has made the request (step **3** in Figure 4.3). From these, it generates a full, formatted log record:

$$LogRecord = \{\texttt{LogData}\}_{K_{enc}} \text{ , } \texttt{Type} \text{ , } \texttt{Subject} \text{ , } \texttt{Timestamp} \text{ , } \texttt{Hash} \text{ , } \{\texttt{Hash}\}_{S_K}$$

`LogData` :	records the details of the I/O request or response
K_{enc} :	the symmetric encryption key used for encrypting the `LogData`
`Type` :	describes the operation type
`Subject` :	the user or virtual machine responsible for triggering the I/O event
`Timestamp` :	the date and time at which the I/O event occurs
`Hash` :	the hash of the full log record
S_K :	the private key used for signing the `Hash`

This log record captures the details of the I/O request or response (`LogData`), operation `Type` which defines the type of the I/O event (e.g. data access, network access), `Subject` which identifies the user or virtual machine responsible

for triggering the I/O event, and `Timestamp`.

Only after this process is finished, the back-end is informed about the I/O request. Like in the original flow, the back-end then picks up the request (and removes it) from the I/O ring and communicates with the native device driver to access the physical disk.

When the data is accessed and returned via the native device driver, the back-end places the I/O response in the shared memory and triggers the event channel to inform the front-end about the queued response. Again, the event channel first informs the log transit to read the response details from the I/O ring (step **12** in Figure 4.3). Knowing the I/O response type and the recipient front-end, the log transit draws an association between the response and the original request (captured previously), and generates a log record in the format described above.

4.2.5 Secure Log Storage

The log transit holds a signing key pair generated through the TPM. The private signing key (S_K), used for signing the hash of the full log record (see above), is *sealed* to the PCR value that corresponds to the trustworthy authenticated boot process (see Section 2.4): from the BIOS upwards, the boot loader, the virtual machine monitor, and the log transit are measured during this process. The administrator defines this sealing property when the logging system is first installed.

If any one of these measured software components is tampered with by an adversary (including a rogue administrator), the signing key will no longer be accessible as the changes will be measured during the boot process and the PCR value will be altered. The current PCR value must match the measurement at the time the key was sealed for access to be authorised by the TPM. Any malicious code (such as a virus or rootkit) executed during the boot process will also be identified in this manner. Moreover, the TPM is robust against software-based attacks trying to steal the private signing key.

Integrity is implemented by storing a hash of every log record generated, and checking that the hash matches when the log record is retrieved. This hash is digitally signed using the private signing key to demonstrate the log authenticity and integrity. Both the `Hash` and its digital signature ($\{\texttt{Hash}\}_{S_K}$) are included in the log record. The public signature-validation key can be used by external

applications to verify that the collected logs have been generated from a trustworthy logging system, signed by a securely configured log transit component, and they have been *integrity protected*.

As illustrated from the healthcare grid example, the log data could contain sensitive information. Their *confidentiality* can be ensured by encrypting and decrypting the privileged log data ($\{\texttt{LogData}\}_{K_{enc}}$) using a symmetric encryption key (K_{enc}) as they are written and read from the disk. The encryption key is also sealed to the PCR value corresponding to the authenticated boot process, and only accessible by a securely configured log transit component. Again, if any of the measured software is tampered with, the key will no longer be accessible to decrypt the log data upon retrieval. This implies that all later log access operations must also go through the log transit for decryption. The next chapter explains secure log access operations in detail.

Even if an intruder successfully hijacks administrator privileges inside a guest virtual machine, the impact of further attacks — for example, trying to access or tamper with the log data directly from the disk or memory — will be limited by the isolation boundaries of the virtual machine. The log data stored inside the dedicated disk or memory space of Domain-0 will still be out of reach. Furthermore, even if an intruder manages to hijack privileged software component in Domain-0, they will still not be able to unseal the encryption key to read the private information. Hence, the logs are safeguarded from attacks involving compromising untrusted software in Domain-0 or the log transit itself. This would solve the problem identified in Xenlog (see Section 3.4.1), where a successful privilege escalation attack inside Domain-0 could lead to compromise of the logging services and the logged data.

Since the storage device drivers only read and write encrypted and integrity protected log data, they are not considered to be part of the trusted computing base. Hence, these are not measured during the authenticated boot process.

4.2.6 Application and OS Level Security Decisions

One of the drawbacks of involuntary log generation, however, is that security decisions made at the guest application and operating system level will be missed by the log transit. This is because the log transit intercepts the I/O request and response details (that require the use of device drivers), whereas the application

and operating system level security decisions are made independently of the use of device drivers. Hence, the security decisions never reach the log transit. Instead, these are likely to be logged and protected by guest applications and operating systems. Table 4.1 identifies some of the integral security decisions that will be missed.

To overcome this problem, functionality is added to the log transit for identifying the 'logging requests' triggered from the guest applications and operating systems, and processing them into the log record format suggested above. Typically, a log record consists of a subject, operation type, timestamp, and log message. The log transit searches for such data attributes and patterns from the incoming I/O disk write requests to identify the logging requests. For instance, the guest operating systems usually rely on system level logging subsystems — such as 'Syslog' [82] for Unix/Linux and 'Event Subsystem' [137] for Windows — to record security events. Being aware of the common log formats used in such systems (and their unique properties), the log transit can efficiently identify logging requests triggered from them. The integrity and confidentiality of these logs are protected by the secure log storage mechanisms described above. The user authentication decisions, data authorisation decisions, input validation results and important virtual machine level system calls are captured using this approach.

There is a weakness in this approach, however, namely that compromised applications could send arbitrary logging requests (in the form of disk write requests) to obscure analysis results. The log transit can not identify an individual application triggering logging requests. For this reason, these logs are considered to be less trustworthy than the ones generated independently from the guest applications, and are marked with a *lower trust level*. It is then up to the end user to use this trust level information appropriately upon filing audit-based reports.

4.3 Observations

This section explains how the involuntary logging system helps in meeting the security requirements identified in Section 3.3. The performance and scalability issues are also discussed.

Event Categories	*Events Captured*	*Security Decisions Missed*
Authentication	User data access attempt and response (e.g. username and password).	User authentication decision, e.g. user JohnSmith failed to log on to Outlook.
Authorisation	Authorisation policy access attempt and response.	Data authorisation decision, e.g. user JohnSmith was granted read and write access to File1.
Data access	Data access attempt and response.	– – –
Changes	System, application and privilege changes; data modification or deletion attempt and response.	– – –
Invalid input	Inputs entered by the user, e.g. the keys pressed.	Inputs classified as invalid (or malicious) by the input validator.
Program startups and terminations	Request to load and run an executable, memory allocation and access, request to terminate a process.	– – –

Table 4.1: Security Decisions Missed by the Log Transit

4.3.1 Satisfying the Requirements

The log transit sits inside the monitor virtual machine, and involuntarily intercepts the I/O events and generates log records for the events of interest. Guest applications and operating systems run inside separate virtual machines and have no way of bypassing or compromising the log transit. This satisfies the 'involuntary log generation' requirement (see Requirement 3.3.1).

Upon log storage, a hash and its signature are generated for each log record, and the log data is encrypted using a symmetric encryption key. Since both the signing key and encryption key are sealed to the secure state of the log transit, a compromised log transit will neither be able to generate valid signatures nor decrypt the logged data. External applications can verify the log integrity by comparing the hash with a computed hash and validating the signature. These features fulfill the 'protected log storage' requirement (see Requirement 3.3.2).

The involuntary logging system solves the common security problem of the developers being relied upon to instrument applications to generate logs and protect logged data (see Section 3.4.1). The log transit is isolated in the monitor virtual machine and it generates log records independently from the applications. These are then securely stored on a dedicated disk space of the monitor virtual machine.

Attacks on the logged data are made infeasible with this isolation and encryption. A rogue application, for example, would have to compromise the monitor virtual machine, the log transit, and the key secrets in order to tamper with the logged data. Moreover, the authenticated boot and sealed key mechanisms make it infeasible for a rogue administrator to change the log transit configurations without being caught.

4.3.2 Further Isolation

Authenticated boot measures the state of the log transit and its settings once at boot time. Hence, a rogue administrator could still alter the log transit settings files (such as the logging policies) at runtime to affect the behaviour of the logging system. Unless the system reboots, these changes will not be reflected on the PCR value, and remote entities will have no knowledge about it.

One way to reduce this risk is to further isolate the log transit to a virtual machine of its own, and configuring Domain-0 so that the only operations allowed on this virtual machine are 'start' and 'terminate'. No external interfaces should be made available to tweak the configurations or settings. This would ensure that once the virtual machine is launched (and measured), there is no way even for administrators to alter its behaviour. The worst they could do is terminate the virtual machine and turn off the log transit, but this should be easily detectable. These security properties could be verified with attestation.

This would also be a suitable way of reducing the size of the trusted computing base. The log transit would run in a relatively small virtual machine dedicated to itself and perform its normal operations without any change, although the event channel would have to be configured to inform this virtual machine (instead of Domain-0) whenever an I/O event occurs. A secure communication channel would have to be established between the two. Moreover, a virtual TPM [80] instance would be bound to the log transit virtual machine to secure the log

data. This virtual TPM would be used to attest and verify the security state of the log transit virtual machine.

One of the drawbacks of this approach is that it would require more changes to the Xen virtual machine monitor than the original approach, since the event channel would have to communicate with the log transit virtual machine directly. Also, setting up a virtual TPM can introduce further security and usability issues [116].

4.3.3 Performance Degradation

One of the key drivers behind the development of computational distributed systems is high performance [62]. However, the suggested use of virtualization and various cryptographic operations would necessarily incur a performance penalty [113].

Running a job inside a virtual machine requires extra information flow upon accessing the hardware. As the control flow example shows (see Section 4.2.2), each I/O request would go through a number of virtual device drivers before reaching the physical hardware; the same applies when receiving an I/O response. A recent study [93] suggests that a typical virtualized, distributed system incurs 20 percent performance penalty over native execution. With the introduction of native hardware support in all recent CPU architectures [75, 53], however, this overhead will be minimised in time to come.

Another area of concern is the overhead of the I/O event interception, log data encryption, and signing operations performed by the log transit. These cryptographic operations will further increase the time it takes for the job to access the hardware. Without a prototype, it is difficult to accurately measure the performance overhead of these operations. However, the use of these operations could be tailored to fit the security requirements of different systems. For instance, in the service-level agreement example where log confidentiality is less important (see Section 3.1.2), the log transit could be configured to skip encryption. Generating a hash and its signature would be sufficient to ensure log integrity.

Another alternative to speed up the execution of a job virtual machine is to accumulate the log records during its execution-time and encrypt them only once before the virtual machine terminates.

4.3.4 System Upgrade

Perhaps the most significant overhead is the cost of upgrading existing systems to enable involuntary log generation. This involves installing the Xen virtual machine monitor and various logging components, and configuring the event channel operations. While this is a large change, the advantage of the proposed architecture is that the guest applications can be used in their own virtual machines without modification — the logs will be automatically generated through the log transit component.

4.4 Chapter Summary

An involuntary, trustworthy logging system has been described in this chapter based on Xen and trusted virtualization. How the proposed system satisfies the 'involuntary log generation' and 'protected log storage' requirements has been explained in Section 4.3.1. The next chapter will describe a trustworthy log reconciliation infrastructure that considers the rest of the security requirements identified in Section 3.3.

Chapter 5

Design: Distributed Log Reconciliation

This chapter proposes a log reconciliation infrastructure which allows log users to: (1) verify the logging and security properties of a remote system, (2) securely access distributed logs, and (3) carry out a *blind analysis* on confidentiality and integrity protected audit trails. This infrastructure aims to satisfy the trustworthy log reconciliation requirements identified in Section 3.3.

Section 5.1 introduces a 'central information directory' which manages the participants' logging system configurations. Section 5.2 describes the operations of the log reconciliation infrastructure in detail. Section 5.3 formally verifies a log reconciliation protocol described in Section 5.2 using Casper [52]. Finally, Section 5.4 discusses some of the remaining issues and challenges of developing the proposed infrastructure.

5.1 The Configuration Resolver

When Diffie and Hellman first introduced public key cryptography, they suggested that a telephone directory could be extended to publish public keys [24]. For instance, if one wanted to find person `B`'s public key, one would look up the telephone directory, find `B`'s public key, and send `B` a message encrypted with the public key.

Drawing inspiration from this principle, a central information directory is added to the original abstract view (see Figure 2.1) to publish information about the participants and their logging system configurations. This extended abstract

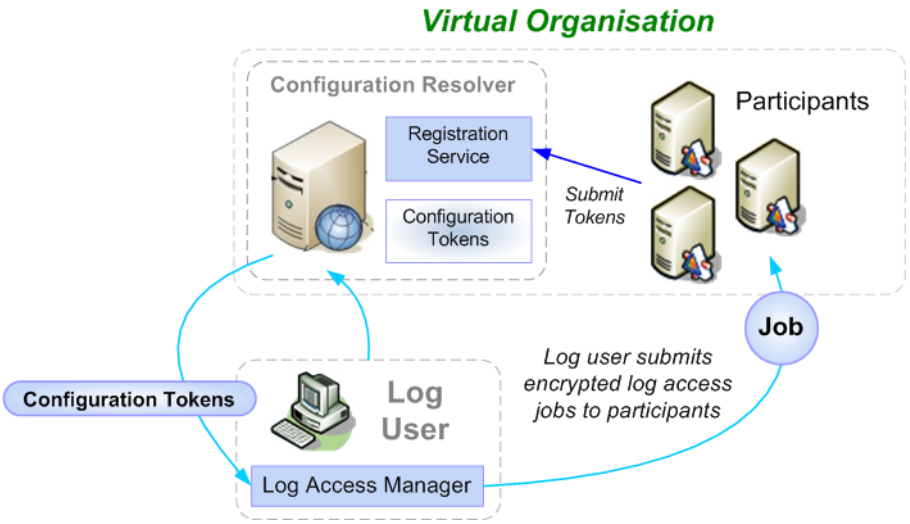


Figure 5.1: Abstract View with the Configuration Resolver

view is presented in Figure 5.1. In the following sections, this information directory is referred to as the 'configuration resolver'.

5.1.1 Assumptions

This section states assumptions about the design of the configuration resolver and the log reconciliation infrastructure.

1. The configuration token, which is submitted by a participant when registering with the configuration resolver, is sent in an authenticated transport session.

2. A public key infrastructure is available and this can be used to verify the identity of the configuration resolver, participants (log owners) and end users (log users).

3. A log owner's system supports trusted computing and virtualization; as minimum, mechanisms like authenticated boot (as described in Section 4.2.5) and remote attestation are enabled in this system.

4. An end user maintains a whitelist of all trusted logging software configurations and are capable of keeping this list up to date.

5. Similarly, a log owner maintains a whitelist of all trusted user system configurations and are capable of keeping this list up to date.

5.1.2 Participant Registration

To demonstrate the trustworthiness of the logging system installed on the platform, the participant registers with the configuration resolver by submitting a 'Configuration Token' (CT) which demonstrates the following properties:

- the logs are generated involuntarily and independently from guest applications and operating systems,
- the log integrity and totality are protected by the secure storage mechanisms, and
- the logs are reported with their integrity, totality and availability protected, and without any interference from adversaries.

These properties should be sufficient for a log user to establish trust with a remote logging system and the various monitoring services that depend on it. This notion of 'configuration-discovery' is inspired by the attestation token approaches discussed in current research (see Section 2.6.1).

Such a discovery mechanism will be necessary, for example, in economy-based distributed systems [96, 28] where service-level agreements must be carefully monitored and used to assess financial compensation when they have been violated. A client can download the configuration tokens from the resolver and verify the security configurations of the logging software (and the audit-based monitoring services) running in remote systems before constructing service-level agreements (see Section 3.1.2).

In fact, the resolver could be used in any of the distributed systems discussed in Section 3.1 for discovering the security properties of the logs generated at remote sites. For instance, in the rogue trader example (see Section 3.1.3), the system administrator could use the tokens to verify that the evidential logs collected from remote trading systems have been integrity and totality protected. Based on these trustworthy logs, they could file an accurate forensic report to the police for an investigation.

$$CT = (\ \texttt{PCR Log},\ AIK\ ,\ \{\texttt{cred}(AIK)\}_{CA}\ ,\ P_K\ ,\ \{\texttt{cred}(P_K)\}_{AIK}\ ,\ \{\texttt{Identity}\}_{S_K}\ )$$

`PCR Log` :	the list of the loaded (and measured) applications and their hash values
AIK :	the public half of the Attestation Identity Key
$\{\texttt{cred}(AIK)\}_{CA}$:	the AIK credential issued by the Privacy Certificate Authority
P_K :	the public half of the non-migratable TPM key
$\{\texttt{cred}(P_K)\}_{AIK}$:	the P_K credential signed using the private half of the AIK
$\{\texttt{Identity}\}_{S_K}$:	the participant's identity signed using the private half of the TPM key

The token content is shown above. It includes the Attestation Identity Key (AIK) for the platform, along with a credential issued by the Privacy Certificate Authority ($\{\texttt{cred}(AIK)\}_{CA}$). This AIK is used to sign a credential for the public key (P_K), which states that the key has been sealed to *two* PCR values which correspond to (1) a trustworthy authenticated boot process, and (2) log access virtual machine image files. The full description of the authenticated boot process (as described in Section 4.2.5) and the virtual machine image files is given in the `PCR Log`.

In addition, `Identity` information is included, signed by the private half of the sealed public key, demonstrating that the user should use this public key when submitting log access jobs to this participant. It is assumed that the configuration token will be sent to the resolver in an authenticated transport session, and so any timestamp or nonce has not been included (see assumption 1, Section 5.1.1).

The authenticity of the sealed key can be checked by validating the AIK signature. The trustworthiness of the participant's logging system can be verified by comparing the `PCR Log` to the local *whitelist* of acceptable configurations (see Section 2.4.2).

The first PCR value proves that the trustworthy logging system has been responsible for generating and protecting the log data, allowing the user to have high confidence in the logs. The second PCR value guarantees the software configuration of the log access virtual machine. This second value is stored in a *resettable PCR* (see Definition 2.7) because the virtual machine image is remeasured at runtime for integrity verification.

5.1.3 Functionality

The configuration resolver acts as a token directory in the proposed infrastructure and offers no other complex functionality. The burden of verifying tokens is left to the log users. This is attractive from a security perspective, as the resolver can remain an untrusted component. The worst that a malicious resolver can do is affect the availability of the infrastructure.

However, the simple resolver does increase the management overhead on each user node as they will all need the ability to check tokens. This involves maintaining a list of trustworthy software, a whitelist, and keeping a revocation list of compromised TPMs and platforms. The security of the infrastructure depends on the proper management of this information. A suitable compromise might be to devolve some of this functionality to local proxy-resolvers, which would perform the token filtering for one specific administrative domain. This keeps control local to one site, but would decrease the effort at each individual node.

To conform to existing standards, it is imagined that the resolver would be implemented as a WS-ServiceGroup [127]. Each node would then be a member of this resolver's group, and have a 'ServiceGroupEntry' that associates them. An entry would also contain identity information by which the node's participation in the resolver is advertised. The membership constraints would be simple, requiring only a valid token and identity. These tokens would be selected by the identity information.

It is assumed that there is a public mechanism available (e.g. public key infrastructure) to verify their identity (see assumption 2, Section 5.1.1). As a result, the levels of indirection introduced by the Trusted Computing Group [2] to prevent any loss of anonymity are unnecessary. Hence, the Privacy CA is not a key component of the system and is not required to protect privacy of the participants. AIKs can be created as soon as the platform is first installed, and should very rarely need updating.

5.2 Trustworthy Log Reconciliation

With the new configuration-discovery mechanisms available through the configuration resolver, the rest of the security mechanisms and their operations are described.

5.2.1 Creation and Distribution of a Log Access Job

All log user interactions are made via the external 'log migration service'. It provides the minimal interface (APIs) necessary for development of analysis tools. Such tools should be designed to allow the user to select acceptable logging system configurations (from a pre-defined whitelist), and specify the user credentials, the participant systems to collect the logs from, and the log access query (step **1** in Figure 5.2). The domain administrators, for example, would use these tools to reconstruct distributed events and identify malicious user activities or problems.

The migration service first makes a configuration-discovery request by submitting the identity information of the selected participants. In response, the resolver sends back the configuration tokens (CTs) of the requested participants (steps **2** and **3** in Figure 5.2). These tokens are used by the migration service to (1) ensure that the reported logging system configurations match the configurations measured at the time of authenticated boot, (2) verify the authenticity of the sealing property, and (3) verify the trustworthiness of the logging system and log access virtual machine configurations. For each token,

- the signature on the AIK credential ($\{\texttt{cred}(AIK)\}_{CA}$) is first checked to verify that the AIK has been generated by a valid TPM;
- the AIK is used to validate the signature on the public key credential ($\{\texttt{cred}(P_K)\}_{AIK}$) to verify that reported two PCR values represent the actual values stored in the TPM, and the private half (S_K) is sealed to these PCR values; and
- then finally, the contents of the `PCR Log` are compared to the known-good whitelist entries.

Once all of these security checks have been passed, the migration service trusts the logging and log access mechanisms available at the participant system and prepares a log access job for submission. If one of the security checks fails, however, the participant details and the reasons for failure are presented to the user. It is then the responsibility of the user to make necessary decisions.

The migration service creates a set of log access jobs for the trustworthy logging systems, each of which contains the user's credential, log access query, job description, user's nonce (N_U) and an Attestation Token (AT) (step **4** in Figure

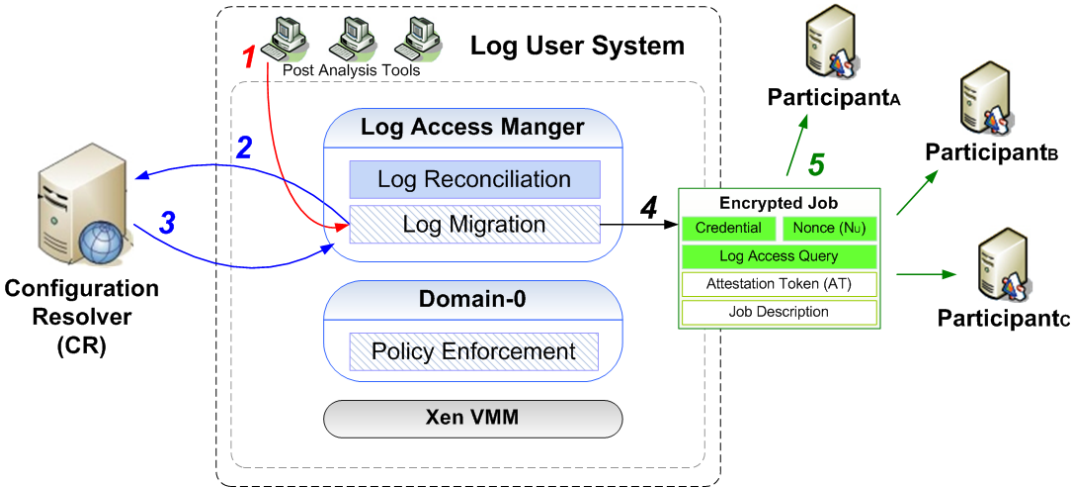


Figure 5.2: Creation and Distribution of a Job

5.2). Much like the configuration token, the attestation token contains sufficient information for the participant systems to verify the state of the user system; the only difference between the two token structures is that the identity information is not included in the attestation token:

$$AT = (\ \texttt{PCR Log},\ AIK\ ,\ \{\texttt{cred}(AIK)\}_{CA}\ ,\ P_K\ ,\ \{\texttt{cred}(P_K)\}_{AIK}\ )$$

The user's public key credential ($\{\texttt{cred}(P_K)\}_{AIK}$) identifies the private half (S_K) as being sealed to the PCR value corresponding to a trustworthy authenticated boot process of the user system: the BIOS, bootloader, virtual machine monitor, monitor virtual machine, and log access manager are measured and recorded in the PCR. These components form the trusted computing base of the user system. The sealed private key (S_K) is used to sign the log access query inside the TPM — it will only be accessible for signing if the trusted computing base has not been modified.

The job secret consists of the user's credential, log access query and its signature, and nonce. These are encrypted with the target participant's public key (obtained from the configuration token) to prevent an adversary or a compromised host from stealing secret information. The encrypted jobs are safely distributed over the public network (step **5** in Figure 5.2). The middleware stack — used mainly for resource brokering — can only read the unencrypted job de-

scription to identify the target participants and distribute the jobs to their policy enforcement points.

5.2.2 Operations of a Log Access Virtual Machine

This section explains how the log access job gets processed at one of the target participant systems. Any security processing required before becoming ready to be deployed in a *per-user* log access virtual machine is done through the 'policy enforcement point' inside Domain-0. The PCR value corresponding to the user system's authenticated boot process (available from the user's attestation token) is compared to the known good-values defined in a whitelist. This is referred to as 'integrity-based job authentication'. It verifies that the job has been created and dispatched from a securely configured log access manager (step **1** in Figure 5.3).

Upon successful attestation, the policy enforcement point measures the log access virtual machine image and configuration files, and resets the *resettable PCR* with this new measurement — the virtual machine image consists of a security patched operating system and trustworthy middleware stack ('authorisation policy management' and 'log migration services') which provides a common interface for the job to execute the query and access the logs. These processes, managed by the policy enforcement point, are trusted and have access to Locality 4 — trusted hardware that can reset a dynamic PCR.

The policy enforcement point then attempts to unseal the private key (bound to the trusted computing base and log access virtual machine image) in order to decrypt the job secret. This key will only be accessible if the participant system is still running with the trustworthy logging configurations *and* the virtual machine image has not been changed. This is intended to guarantee that only a trustworthy virtual machine has access to the decrypted job secret. The signature of the log access query is then verified using the user's public key: a valid signature proves that the query originates from a trustworthy user system and the encrypted secret correlates with the attestation token.

If all of these security checks pass, the 'compartment manager' launches a trustworthy virtual machine using the verified image and deploys the decrypted job to make use of the middleware stack (step **2** in Figure 5.3). Before allowing the log access query to run, the authorisation policy management service checks

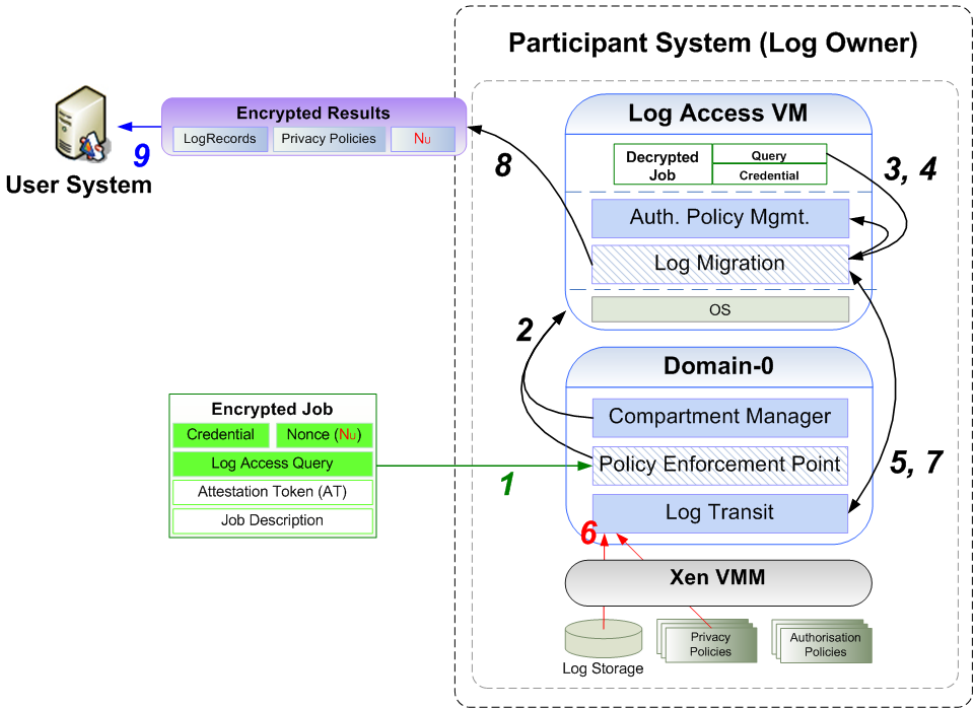


Figure 5.3: Operations of a Log Access Virtual Machine

whether the user (log requestor) is authorised to run the query on the participant system (steps **3** and **4** in Figure 5.3). The job is processed further if the conditions stated in the authorisation policy are satisfied.

The log migration service checks the query for any attempt to exploit vulnerabilities in the database layer (e.g. SQL injection) before making a request to the log transit component (in Domain-0) to run the query (step **5** in Figure 5.3). The log transit decrypts the $\{\texttt{LogData}\}_{K_{enc}}$ before returning the *LogRecords* to the migration service (see Section 4.2.4 for log record details). Each *LogRecord* contains its `Hash` and digital signature which can be used by the user to verify the log authenticity and integrity. Since the decryption key (K_{enc}) is sealed to the trustworthy authenticated boot process, a compromised log transit (or any other software) in Domain-0 will not be able to decrypt and read the $\{\texttt{LogData}\}_{K_{enc}}$. This also prevents a rogue administrator from modifying the log transit configurations to tamper with the `LogData` before returning them.

During this process, a log `Privacy Policy` is also selected to protect the privacy of the log data (step **6** in Figure 5.3). Such a policy, specified by the log

owner, states what part of the requested `LogData` is allowed to be disclosed. For instance, in the dynamic access control example (see Section 3.1.1), the `Privacy Policy` will restrict disclosure of the *LungCancer* status in table T_2. Existing log anonymisation techniques such as FLAIM [8] could be used to specify these policies. The idea is to sanitise the sensitive data while pertaining sufficient information for analysis.

A rogue virtual machine situated between the log access manager and the log transit might try to modify the *LogRecords* or read the decrypted `LogData`. Verifying the log integrity is not an issue since any modification would be detected when the *LogRecord* is compared against the stored `Hash` and the signature is validated. However, this would not stop the rogue virtual machine from constantly modifying the *LogRecord* to reduce the log availability. Real-time monitoring applications that rely on timely data feeds would be affected most by this type of attack. The log confidentiality could be compromised if the rogue virtual machine manages to read the `LogData` and transfer them over the network.

To prevent these attacks, the virtual machine monitor creates an exclusive and secure communication channel between the two virtual machines using its shared memory mechanisms (see Section 4.2.2). It ensures that no other virtual machine on the platform has access to the shared memory region used to transfer the *LogRecords*. The two virtual machines notify each other about the queued log requests and responses via the event channel.

The migration service generates a secure result message (step **8** in Figure 5.3) containing the *LogRecords*, log `Privacy Policy`, and user's nonce (N_U):

$$Result = \{LogRecords\ ,\ \texttt{Privacy Policy}\ ,\ N_U\}_{K_{session}}$$

The result message is encrypted using a symmetric session key ($K_{session}$, created locally at the participant system), which, in turn, is encrypted using the user's public key (P_K) obtained from the attestation token. Since the private half is protected by the user's TPM, this is sufficient to ensure the confidentiality and integrity of the *LogRecords* being transferred over the network. Moreover, a compromised user system will not be able to access the private key — sealed to the trusted computing base of the user system — to decrypt the session key

and see the log records. The purpose of using a symmetric key is to improve the overall performance of the cryptographic operations.

The user's nonce (N_U) is sufficient to verify that the *Result* message has been generated from a trustworthy virtual machine and an unmodified log access query has been executed.

5.2.3 Reconciliation of Collected Logs

The result message and the encrypted session key arrives at the policy enforcement point of the user system (step **1** in Figure 5.4). First, it decrypts the session key using the sealed private key and uses the session key to decrypt the *Result* message. The decrypted message is then forwarded to the log migration service which compares the returned nonce (N_U) with the original nonce (step **2** in Figure 5.4). A matching value verifies the integrity of the job execution environment.

The migration service also verifies the authenticity and integrity of each *LogRecord* by (1) validating its signature ($\{\texttt{Hash}\}_{S_K}$) and (2) comparing the `Hash` with a computed hash of the *LogRecord*. A valid signature proves that the returned *LogRecord* has been generated involuntarily through a securely configured log transit component; a matching hash guarantees that the *LogRecord* has been integrity protected.

The internal reconciliation service reconciles the logs collected from the selected participants and processes them into meaningful audit trails (steps **3** and **4** in Figure 5.4). During this process, the log `Privacy Policy` is enforced to hide the private and potential identifiable information, while still releasing enough information for the user to carry out useful analysis. Attestation of the user system's log access manager (step **1** back in Figure 5.3) is sufficient to know that these policies will be enforced correctly. Virtual machine isolation and its robust memory protection prevent an attacker from accessing the memory space of the log access manager and reading the raw `LogData`.

More functionality could be added to the reconciliation service depending on the application requirements: for example, the dynamic access control system (described in Section 3.2.2) would benefit from having a function that automatically updates the access control policies using the reconciled audit trails.

Only the anonymised audit trails are returned to the original analysis tool (step **5** in Figure 5.4). This satisfies the *blind log analysis* requirement (see Re-

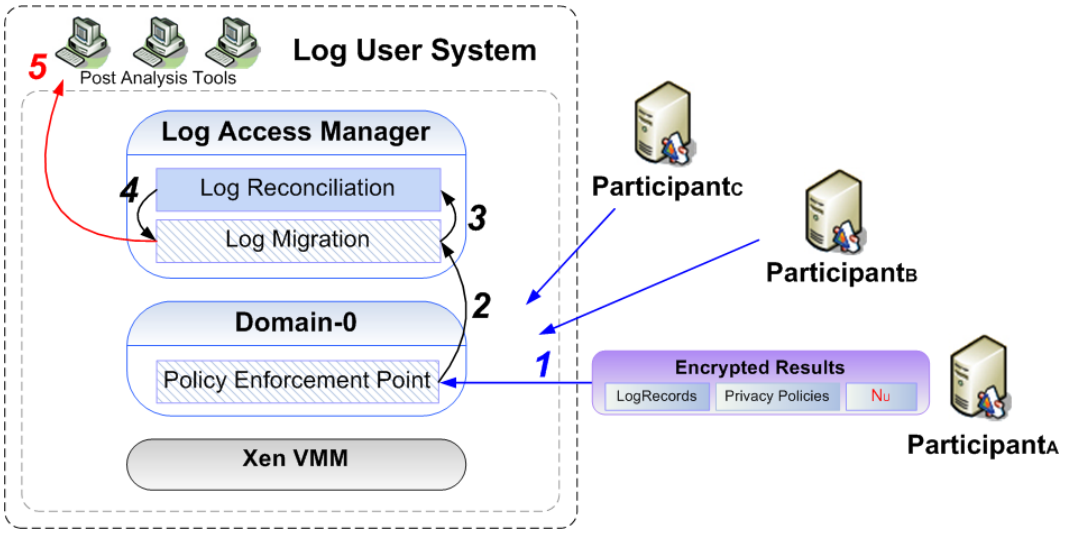


Figure 5.4: Reconciliation of Collected Logs

quirement 3.3.7). In the dynamic access control example, a summary of the policy updates would be generated from the anonymised audit trails. The administrator would only see this summary of how the access control policies have been updated for different users. Virtual machine policy attestation [92], for example, could be used on the log access manager to guarantee that the audit trails will not be exported to an unauthorised device.

5.3 Formal Verification of the Security Protocol

Casper [52] is a security protocol modeling application developed at Oxford University Computing Laboratory which takes a formal description (Casper script) of a protocol and its requirements (*specifications*) in a simple, abstract language, and generates a corresponding Communicating Sequential Processes (CSP) model [23]. This CSP model can be checked with the Failures/Divergences Refinement (FDR) tool [46] to identify possible attacks or to verify that no such attacks exist. Basically, the generated model is tested against the specifications representing desired security properties — FDR explicitly enumerates and explores the state space of the model to check whether insecure traces (sequences of messages) can occur. Most commonly checked properties in Casper are *secrecy* and *authenticity*.

This section presents a security protocol modeled in Casper, which captures

the essential log access and reconciliation transactions described above. The specifications describe the desired security properties of the protocol. These are formally verified using FDR to demonstrate that the confidentiality of the job secrets and logs are protected in the protocol.

5.3.1 Assumptions

In order to simplify the protocol description, some of the components and their transactions that are sufficiently obvious are not considered in this protocol. These are captured as assumptions instead:

1. The `Log User (u)` already knows about the identify of the `Log Owner (o)`. This information is used to fetch the `PCRQuote(LogOwner)` from the `Configuration Resolver (r)`.

2. A Privacy CA is available to validate the AIKs and generate AIK certificates.

3. The Privacy CA's certificate and the AIK certificate is contained in the `PCRQuote` for authenticity checking.

4. A middleware stack is responsible for forwarding the job dispatched from the `Log User (u)` to the `Log Owner (o)`. However, the middleware is not modeled here since an `Intruder (Ivo)` can perform all the attacks that a rogue middleware is capable of performing.

5.3.2 Free Variables and Processes

```
#Free Variables
u, o, r :  Agent
nu :  Nonce
q :  LogAccessQuery
log :  LogsAndPoliciesAccessed
kuo :  SessionKey
cu :  Credentials
Creds :  Agent -> Credentials
pcrqo, pcrqu :  PCRQuote
```

```
PCRQ : Agent -> PCRQuote
pko, pku :  PublicKey
PK : Agent -> PublicKey
SK : Agent -> SealedSecretKey
InverseKeys = (kuo, kuo), (PK, SK)
```

The type of the variables and functions used in the protocol description are defined first. Free variables `u, o, r`, all of which are `Agents`, are instantiated with the actual variables `LogUser, LogOwner, ConfigurationResolver`, respectively. The full description of the protocol can be found in Appendix A.

In the protocol description, the variable `nu` is taken to be of type `Nonce` and represents the `LogUser`'s nonce. `PCRQ` function returns the `PCRQuote` of each `Agent`. For instance, `PCRQ(o)` is the PCR quote of the `LogOwner` and it contains the platform configuration of the `LogOwner` and validation data. Functions `PK` and `SK` return the public key and sealed private key of an `Agent` respectively. For instance, `PK(o)` represents the `LogOwner`'s public key, and `SK(o)` represents the corresponding private key that has been sealed to the PCR value reported in `PCRQ(o)`. Note that this sealing property is not really captured in the model — Casper has not been designed to model such a property.

```
#Processes
CONFIGURATIONRESOLVER(r,o) knows PCRQ, PK
LOGUSER(u,o,r,q,nu) knows SK(u), Creds(u), PK, PCRQ
LOGOWNER(o,log,kuo) knows SK(o), PK, PCRQ, Creds
```

Each `Agent` running in the protocol is represented by a CSP process. The names of the CSP processes representing the agents are `CONFIGURATIONRESOLVER, LOGUSER,` and `LOGOWNER`. These give names to the roles played by each `Agent` and define the variables and functions the agent in question is expected to know at the beginning of the protocol run. For example, the `LogUser` is expected to know their own identity (`u`), the nonce (`nu`), the public key (`PK`) and PCR Quote (`PCRQ`) functions, and their own sealed private key (`SK(u)`) and credential (`Creds(u)`).

5.3.3 Protocol Description

```
#Protocol Description
0.  u -> r :  o
1.  r -> u :  PCRQ(o) % pcrqo, PK(o) % pko
[ pcrqo == PCRQ(LogOwner) and pko == PK(LogOwner) ]
2.  u -> o :  {Creds(u) % cu, q, {q}{SK(u)}, nu}{pko % PK(o)}, PCRQ(u) % pcrqu,
PK(u) % pku
[ cu == Creds(LogUser) and pcrqu == PCRQ(LogUser) and pku == PK(LogUser) ]
3.    -> o :  u, cu % Creds(u), pcrqu % PCRQ(u)
4.  o -> u :  {log, nu}{kuo}, {o, u, kuo}{pku % PK(u)}
```

The protocol description shows the sequence of messages in the protocol. In message 0, the `LogUser` (`u`) sends the identity of the `LogOwner` (`o`) to the `Resolver` (`r`). Then in message 1, `r` responds by sending back `o`'s PCR quote (`PCRQ(o)`) and public key (`PK(o)`). The first two messages represent the process of obtaining the log owner's configuration token from the resolver.

After receiving the quote, `u` validates the received data against the actual values expected (`pcrqo == PCRQ(LogOwner) and pko == PK(LogOwner)`). Since it is not really possible to model remote attestation in Casper, this validation is only a close approximation of the process of comparing the reported PCR value with the known good configurations.

In message 2, `u` submits a job containing its own credentials (`Creds(u)`), the log access query (`q`) and signature (`{q}{SK(u)}`), and the nonce (`nu`). These represent the job secret and are encrypted using the validated public key of `o` (`PK(o)`). The job also contains `u`'s PCR quote (`PCRQ(u)`) and public key (`PK(u)`), which represent `u`'s attestation token. When this message arrives, `o` validates the user's credential and attestation token against the actual values expected (`cu == Creds(LogUser) and pcrqu == PCRQ(LogUser) and pku == PK(LogUser)`). Again, this is only a close approximation of user authentication and attestation.

In message 3, `o` discovers the user's identity (`u`) knowing that the user's credential, PCR quote, and public key are all valid. Finally, in message 4, `o` sends back the accessed logs and privacy policies (`log`) and the nonce (`nu`) encrypted

with a symmetric session key (kuo). This session key is encrypted using the user's public key (PK(u)).

5.3.4 Specifications

```
#Specification
StrongSecret(u, Creds(u), [o])
StrongSecret(u, q, [o])
StrongSecret(u, nu, [o])
StrongSecret(o, log, [u])
StrongSecret(o, kuo, [u])
Agreement(u, o, [q,nu])
Agreement(o, u, [q,log,kuo,nu])
```

Specifications define the security requirements of the protocol. Specifications starting with StrongSecret require that in any complete or incomplete runs, certain data should be secret. For instance, in the first specification (StrongSecret(u, Creds(u), [o])), u can expect the value of the variable Creds(u) to be a secret while sharing it with o. This specification would fail if u can take part in a run (complete or not) where the role o is not taken by the intruder, but the intruder learns the value Creds(u). Similarly, the next four specifications require the log access query (q), user's nonce (nu), the accessed logs (log), and the session key (kuo) to remain secret.

Agreement specifies the authentication property. For instance, the second Agreement(o, u, [q,log,kuo,nu]) specifies that o is correctly authenticated to u, and the agents agree upon q, log, kuo and nu. If u completes a protocol run with o, then o has previously been running the protocol, apparently with u, and both agents agreed on the roles they took and the values of the variables q, log, kuo and nu. The first agreement specifies a similar property between u and o.

A graphical front-end for FDR, CasperFDR [52], is used to check whether these security properties are satisfied. First, CasperFDR compiles the Casper script into a CSP description and creates *refinement assertions* corresponding to the specifications. Then, it invokes FDR to verify all the assertions. These

assertions and verification results are given below:

```
Assertion SECRET_M::SECRET_SPEC [T= SECRET_M::SYSTEM_S
No attack found

Assertion SECRET_M::SEQ_SECRET_SPEC [T= SECRET_M::SYSTEM_S_SEQ
No attack found

Assertion AUTH1_M::AuthenticateLOGUSERToLOGOWNERAgreement_q_nu
[T= AUTH1_M::SYSTEM_1
No attack found

Assertion AUTH2_M::AuthenticateLOGOWNERToLOGUSERAgreement_q_log_kuo_nu
[T= AUTH2_M::SYSTEM_2
No attack found
```

The first two assertions correspond to the secrecy specifications; the third assertion corresponds to authentication of the `LogUser` to the `LogOwner`; the fourth assertion corresponds to authentication of the `LogOwner` to the `LogUser`. The verification results show that 'no attack is found' and all of the security properties are satisfied.

5.3.5 Iterative Modeling Process

Even though the verification results show that the final version of the protocol is free from attacks, a number of possible attacks have been found (and fixed) during an iterative modeling process.

For instance, in one of the earlier versions of the protocol, the signature of the log access query (`{q}{SK(u)}`) was not included as part of the job secret in message 2 (see above). In consequence, the log owner could not verify whether the job secret correlated with the log user's public key. This meant the intruder could swap the log user's PCR quote and public key with their own, and read the log data encrypted with their public key. After observing the possible attack traces through FDR, the signature of the query was added to the job secret to solve this problem.

This iterative process has been useful for identifying potential loopholes as such, and refining the protocol to minimise the number of loopholes. The security procedures described in Section 5.2 are based on the final (attack-free) version of the log reconciliation protocol.

5.4 Observations

This section discusses how the proposed infrastructure fulfills the security requirements presented in Section 3.3. Potential scalability and performance issues are also discussed.

5.4.1 Satisfying the Requirements

A number of security mechanisms work together to satisfy the 'authorisation policy management', 'log migration service', and 'protected execution environment' requirements (see Requirements 3.3.3, 3.3.4 and 3.3.5). Remote attestation and runtime verification of the virtual machine are used to verify the logging system properties, and securely collect the logs from the verified system. The policy management and migration services form the middleware stack of the per-job virtual machine, providing functions for user credential validation and secure log access.

Before job submission, the log user verifies the security state of these services using the configuration token, and relies on the sealed key mechanism to guarantee the verified execution environment. The log access virtual machine image is remeasured at runtime to ensure that the private key (required to process the job further) is only accessible if the state of the services has not changed. Further, strong job isolation ensures that the log access query runs free from any unauthorised interference.

Before allowing the job to run, the policy enforcement point checks the security configuration of the user's log reconciliation service. Attestation of the reconciliation service is sufficient to know that the log privacy policies will be enforced correctly upon reconciliation, and only the processed, anonymised results will be released to the end user. These mechanisms are responsible for meeting the 'log reconciliation service' and 'blind log analysis' requirements (see Requirements 3.3.6 and 3.3.7).

As the gap analysis shows (see Section 3.4.2), existing approaches often lack mechanisms for verifying the log integrity and confidentiality. With the proposed infrastructure, the user can verify the log integrity by: (1) verifying the logging properties of a remote system that generated the logs, and (2) checking the hash and validating the signature of each log record.

The log owner, before granting access, verifies the security configurations of the user's log access manager. A securely configured log access manager will provide sufficient protection for log confidentiality and privacy upon reconciliation. The log records are encrypted in a way that only a securely configured log access manager can decrypt it.

5.4.2 Configuration Token Verification

The trustworthiness of the infrastructure is dependent on the ability for each user to make the right decision about the security provided by logging systems at other participant nodes. The identity and security configurations of the logging systems are reported in the `PCR Log` contained in the configuration tokens. These values are then compared to a whitelist of acceptable software.

However, this assumes prior knowledge of all trusted logging software configurations (see assumptions 4 and 5, Section 5.1.1), which may not be the case if the virtual organisation is particularly large. Such a scalability issue is magnified when considering settings files, many of which will have the same semantic meaning but different measured values. It is difficult to assess how big a problem this is, but future work may look at using property-based attestation [9] as a potential solution.

5.4.3 Performance Degradation

Attestation involves expensive public key operations for signing the PCR values and validating the signatures. It also involves comparing the reported PCR event log with the whitelist and verifying the trustworthiness of a platform.

In the proposed system, attestation is performed twice upon job submission: the user system verifies the logging system configurations before dispatching the job, and the logging system verifies the user system configurations before allowing the query to run. The fact that the platforms are mutually attesting each other at runtime is a performance concern. The resulting overhead might be unacceptable

in realtime applications that depend on timely data feeds. For instance, in the dynamic access control system (see Section 3.1.1), this delay could prevent the access control policies from being updated in time, allowing various inference attacks to succeed.

The attestation service, however, could be configured to minimise the use of attestation. Since the trusted computing bases of both platforms are designed to be relatively static, the previous attestation results could be used again and again up to a given expiry date. A fresh attestation would be performed when the previous results expire, removing the need to attest every time a job is submitted. If the trusted computing base changes at a time before the expiry date, the sealed key mechanism would detect it and inform the verifying platform. The verifying platform would then request for the latest configuration (or attestation) token to perform a fresh attestation.

5.4.4 System Upgrade

The most significant overhead of the proposed system is the cost of upgrading existing nodes to support the new infrastructure. This involves installing the Xen virtual machine monitor and various logging and per-job virtual machines. While this is a large change, the advantage of the system is that legacy operating systems and middleware can still be used in their own virtual machines. The overall administration task is therefore not so large. Furthermore, virtualization is increasing in popularity, and it seems likely that the scalability and management advantages will persuade the participants into upgrading to a suitable system anyway.

5.4.5 Specifying the Log Privacy Policies

FLAIM [8] has been suggested as a log anonymisation technique that could be used for specifying the log privacy policies. However, FLAIM (and many other existing techniques) only provides anonymisation mechanisms suitable for basic log data structures (for example, network logs). On the other hand, the dynamic access control example (see Section 3.1.1) is concerned with relations between different log database queries and complex information flow. Hence, depending on the application requirements, it would be necessary to expand on techniques like FLAIM to handle complex log data structures upon specifying privacy policies.

5.5 Chapter Summary

A trustworthy log reconciliation infrastructure that ensures log integrity, confidentiality and availability has been described in this chapter. How this infrastructure satisfies the security requirements (identified in Section 3.3) has been explained in Section 5.4.1. The next chapter will describe in detail a prototype implementation that has been constructed based on a number of features selected from this log reconciliation infrastructure.

Chapter 6

Prototype Implementation

This chapter describes a prototype implementation of the security features selected from the log reconciliation infrastructure (see Chapter 5). One of the key motivations is to demonstrate the basic feasibility of the proposed trusted computing ideas, and uncover security and usability issues. A general implementation strategy for developing trusted computing applications is also discussed.

Section 6.1 gives an overview of the implemented features. Section 6.2 explains the implementation details using high-level class diagrams. Finally, Section 6.3 discusses the feasibility, security and usability issues raised through the development of the prototype.

6.1 Prototype Overview

Based on the security components proposed in the previous chapter, a prototype implementation has been constructed using the 'IAIK TCG Software Stack for the Java (tm) Platform' [61] (jTSS) as the software building block, and Xen [18] as the underlying virtualization technology.

6.1.1 Implemented Components and Assumptions

An abstract view of the prototype implementation is shown in Figure 6.1. This diagram gives a high-level overview of the components that have or have not been implemented, and those that have been assumed to already exist.

Green represents implemented components whereas *red* represents uncompleted ones, and *dotted grey line* represents assumed components. Considering

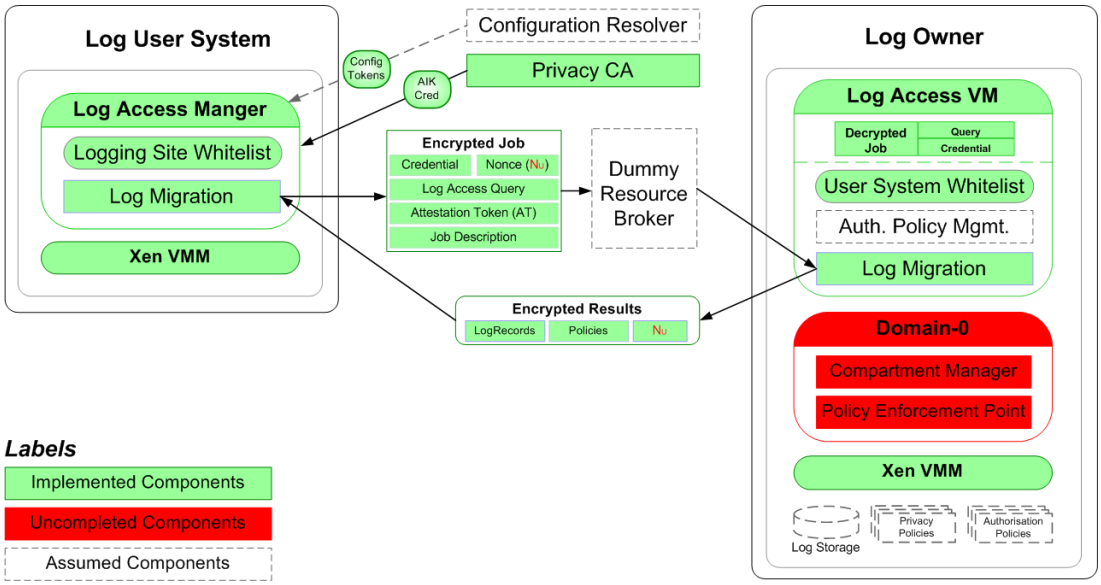


Figure 6.1: Prototype Implementation Overview

that the key motivation is to demonstrate feasibility, components that have already been implemented by others, or that are sufficiently straightforward (in terms of devising implementation strategies), have been assumed.

In the log user system, the 'log access manager' virtual machine and its 'log migration service' have been implemented successfully. The log migration service has been implemented as a web service offering secure job creation functionalities to the log user. An open source, Java EE compatible application server called 'GlassFish' (version 2.1) [120] has been used for web services development — GlassFish provides an easy and reliable environment for developing enhanced web services. A simple 'whitelist' of participant system configurations has been defined using XML. IAIK's 'Privacy CA' (version 0.2) [79] has been configured to provide the trusted computing protocols necessary for validating the Attestation Identity Key (AIK) and signing its credential ($\{\texttt{cred}(AIK)\}_{CA}$).

The 'configuration resolver' has been assumed to provide the 'configuration tokens' to the log user system. Each configuration token contains properly constructed AIK credential signed by the Privacy CA, the public key credential signed with the private AIK, and the PCR event log. Log migration service functions for performing attestation, creating a job, encrypting the job secret, and dispatching it to the participant system have all been implemented. The

intermediary resource broker service has been assumed.

In the target participant (or log owner) system, the 'log access virtual machine' and its 'log migration service' have been implemented. A whitelist of log user system configurations has been defined using XML. Security functions for performing attestation based on this whitelist and encrypting the accessed logs with the user's public key have been implemented.

The task of configuring Domain-0 to support the dynamic launch of a log access virtual machine has been left out due to the implementation challenges. Instead, this work is suggested as future work. The log database, log privacy policies as well as authorisation policies have all been assumed to already exist. The 'authorisation policy management service' has also been assumed to perform access control operations inside the log access virtual machine.

6.1.2 Selected Features

Given the time constraint, a few integral components and their security features have been selected for implementation. Each of these features is likely to be questioned about its feasibility, hence requires a proof-of-concept.

6.1.2.1 Creating and Dispatching Log Access Job

The first set of features have been selected from the workflow associated with selecting trustworthy logging systems, creating a log access job, and dispatching it to those selected (see Section 5.2.1):

1. **Describing the Whitelist** Known good logging system configurations should be listed to indicate the expected PCR index and software measurements.

2. **Selecting Trustworthy Logging Systems** Logging systems should be selected by verifying the authenticity of the reported credentials (obtained from the configuration tokens), and comparing the PCR event logs against the known good values specified in the whitelist.

3. **Creating a Job and Encrypting its Secret** For each job being created, the job secret — user credentials, nonce, log access query — should be encrypted using the public key (obtained from the configuration token) of the target participant. The unencrypted part of the job should include the

user system's own attestation token. While constructing this token, the AIK credential should be generated and signed by the Privacy CA, and the public key credentials should be signed by the private AIK. This should indicate that the private half is sealed to the PCR corresponding to the user system's trusted computing base. The log access query should be signed using this private key to allow remote systems to verify its authenticity before running it.

4. **Submitting the Job** The jobs should be dispatched to the target participant systems via a resource brokering service (an assumed component). This service should only be able to read the job description to figure out the address of the target participant system.

5. **Isolating the Trusted Computing Base** The log migration service should operate inside an isolated, log access manager virtual machine.

6.1.2.2 Operations of the Log Access Virtual Machine

The next set of features have been selected from the workflow associated with authenticating the log access job, verifying the integrity of the log access virtual machine, and accessing the logs (see Section 5.2.2).

6. **Integrity-Based Job Authentication** The authenticity of the PCR quote and sealed key should be checked first by validating the AIK signature on the public key credential (obtained from the attestation token). Then, the PCR event log should be compared to the known good values stated in the whitelist. The job should be authenticated only if it has been dispatched from a trustworthy user system.

7. **Measuring the Virtual Machine Image Files** The log access virtual machine image and configuration files should be measured at runtime, and the resettable PCR (see Definition 2.7) should be reset with this new value. Modifying any one of these files should result in a different value and prevent access to the sealed private key.

8. **Decrypting the Job Secret** The job secret should be decrypted using the sealed private key. If any one of the two PCR values change, this key should no longer be accessible for decryption.

9. **Validating the Signature** The signature of the log access query should be validated with the public key. A valid signature indicates that the query originates from a trustworthy user system and the encrypted secret correlates with the attestation token.

10. **Encrypting the Log Results** The accessed logs, privacy policies, and the user's original nonce (N_U) should be encrypted with a symmetric session key. This session key, in turn, should be encrypted with the user's public key.

6.1.2.3 Verifying the Returned Results

The last set of features have been selected from the workflow associated with decrypting the log results, and verifying the returned nonce (see Section 5.2.3):

11. **Decrypting the Log Results** The symmetric session key should be decrypted first using the sealed private key, which should only be accessible if the user system's trusted computing base has not been modified. This session key should then be used to decrypt the log results. Finally, the decrypted nonce should be compared to the original nonce. A matching value guarantees that the logs have been accessed through a trustworthy log access virtual machine.

6.2 Implementation Details

The selected features have been implemented using the technologies described in Section 6.1.1. This section provides a high-level overview of the important libraries, classes and methods, and explains how these have been orchestrated together.

6.2.1 Class Diagrams

The following two class diagrams (see Figures 6.2 and 6.3) highlight the important libraries, classes, methods and files, and describe the relationships between them. The first diagram describes the structure of the implemented services such as might be deployed in the log user system.

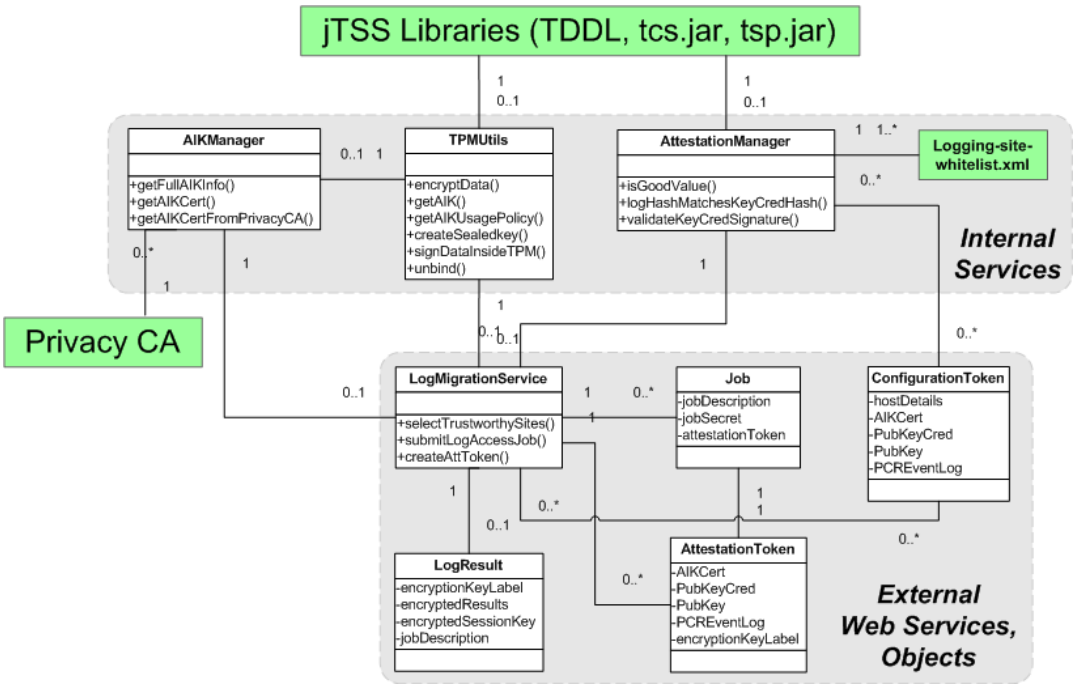


Figure 6.2: Class Diagram — Log User System

The structure can be divided into three layers. The bottom layer is the stack of jTSS libraries — including the TSS Device Driver Library (TDDL), TSS Core Services (`tcs.jar`) and TSS Service Provider (`tsp.jar`) [128] — which provide the core methods for utilising the TPM and TPM keys.

The middle layer represents the internal services which are used by the `LogMigrationService` to perform various trusted computing operations. The `TPMUtils` class provides user-friendly methods for using the TPM to perform cryptographic operations including TPM key access. The `AIKRetriever` class provides methods for retrieving an AIK certificate if one exists, or generating one through the Privacy CA. In the process of generating full AIK information, it uses `TPMUtils` to access the AIK and its usage policies from the TPM.

The `AttestationManager` class provides a complete set of methods for performing attestation. It accepts either `ConfigurationToken` or `AttestationToken`, checks the authenticity of the credentials, and compares the reported PCR event logs with the whitelist (`logging-site-whitelist.xml`).

The top layer represents the `LogMigrationService` and the data objects it uses. These objects are `ConfigurationToken`, `AttestationToken`, and `Job`. The

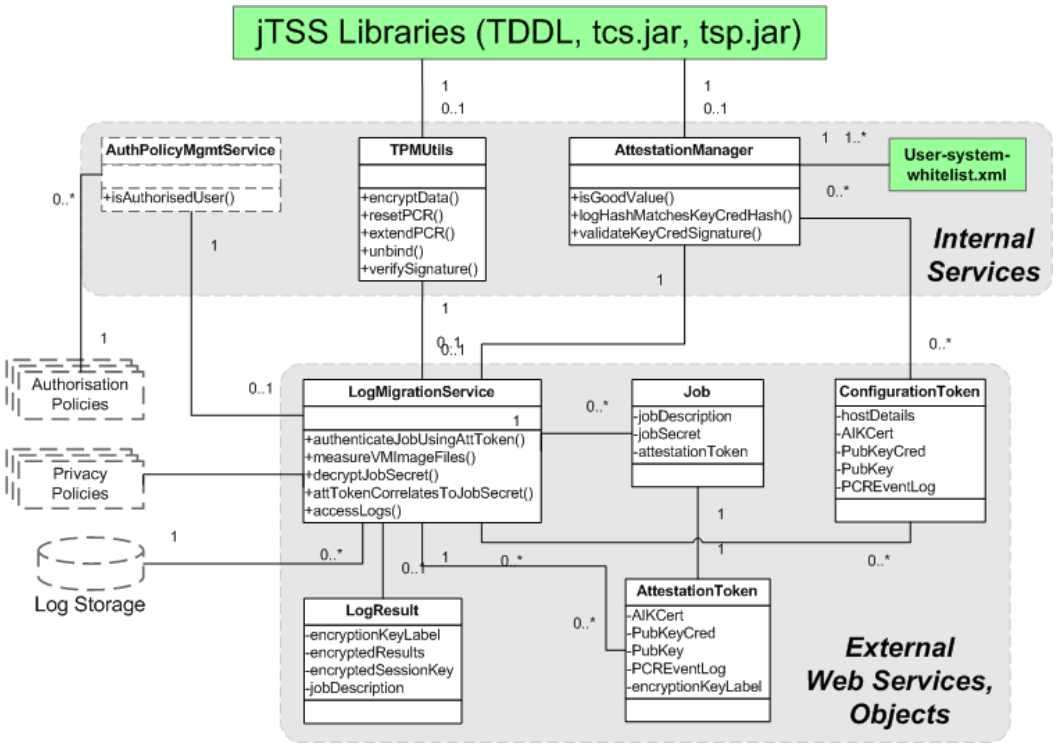


Figure 6.3: Class Diagram — Log Owner System

`LogMigrationService` provides external web methods for submitting log access jobs. It uses all of the internal services (middle layer) upon selecting trustworthy logging sites, generating log access jobs, and dispatching them.

Figure 6.3 describes the services such as might be deployed in the participant (or log owner) system. The structure is much like that of the user system — divided into three layers of jTSS libraries, internal classes, and the external web service.

The dotted grey line represents the assumed classes and data: the log database and the policies are assumed; the `AuthPolicyMgmtService` is also assumed to provide the access control methods. The internal classes provide the same type of methods as described before. The `LogMigrationService` offers web methods for external applications to submit jobs and access the logs. These methods rely on private methods for authenticating the job and generating secure log results.

6.2.2 Implemented Features

This section refers back to the selected features (see Section 6.1.2) and explains the implementation details for each. The first five features are described with reference to the first, log user system class diagram (see Figure 6.2):

6.2.2.1 Describing the Whitelist

The known good logging site and user system configurations are managed through a simple whitelist defined using XML.

Here is an example `logging-site-whitelist.xml` file:

```
<Whitelist desc='Whitelist of trustworthy logging sites'>
   <KnownGoodValue id=1 desc='Log transit runs inside Dom0'>
      <ExtendedPCR index=10 desc='Authenticated boot'>
         <ExpectedValue desc='BIOS'>a7d7da6d80c497a...</ExpectedValue>
         <ExpectedValue desc='TrustedGrub'>a760a0de...</ExpectedValue>
         <ExpectedValue desc='2.6.21.fc8xen'>e9a35e...</ExpectedValue>
         <ExpectedValue desc='Dom0-ConfigFiles'>97b...</ExpectedValue>
         <ExpectedValue desc='Dom0-fc8'>3a8aeb65abc...</ExpectedValue>
         <ExpectedValue desc='Dom0-LogTransit'>6ea8...</ExpectedValue>
      </ExtendedPCR>
      <ExtendedPCR index=16 desc='Log access VM Configs'>
         <ExpectedValue desc='ConfigFiles'>a2e57ee0...</ExpectedValue>
         <ExpectedValue desc='ROVirtualDiskImage'>0...</ExpectedValue>
      </ExtendedPCR>
   </KnownGoodValue>
   <KnownGoodValue id=2 desc='Log transit runs inside the driver VM'>
      <ExtendedPCR index=10 desc='Authenticated boot'>
         <ExpectedValue desc='BIOS'>a7d7da6d80c497a...</ExpectedValue>
         <ExpectedValue desc='TrustedGrub'>a760a0de...</ExpectedValue>
         <ExpectedValue desc='2.6.21.fc8xen'>e9a35e...</ExpectedValue>
         <ExpectedValue desc='Dom0-ConfigFiles'>97b...</ExpectedValue>
         <ExpectedValue desc='Dom0-fc8'>3a8aeb65abc...</ExpectedValue>
         <ExpectedValue desc='DriverVM-ConfigFiles'>..</ExpectedValue>
         <ExpectedValue desc='DriverVM-LogTransit'>...</ExpectedValue>
      </ExtendedPCR>
      <ExtendedPCR index=16 desc='Log access VM Configs'>
         <ExpectedValue desc='ConfigFiles'>a2e57ee0...</ExpectedValue>
         <ExpectedValue desc='ROVirtualDiskImage'>0...</ExpectedValue>
      </ExtendedPCR>
   </KnownGoodValue>
</Whitelist>
```

A whitelist may contain multiple instances of `KnownGoodValue`, each of which is assigned with a unique `id` and represents an instance of a trustworthy logging site configuration. A `KnownGoodValue` has two instances of `ExtendedPCR` since the participant's PCR quote comprises of two measurements. In the given example, PCR 10 contains a measurement corresponding to the authenticated boot process, and *resettable* PCR 16 contains a measurement corresponding to the log access virtual machine image.

Finally, each `ExtendedPCR` contains the expected hash values. The exact sequence of the authenticated boot process is also captured in the whitelist — a single difference in the measurement sequence will alter the computed hash value. The whitelist of user system configurations, managed by the participant, is defined in a similar manner.

6.2.2.2 Selecting Trustworthy Logging Systems

`LogMigrationService` (`LMS`) provides the `LMS.selectTrustworthyLoggingSites()` method, which uses `AttestationManager` (`AM`) to determine whether a configuration token represents a trustworthy logging site. Attestation involves three steps:

1. `AM.isGoodValue()` method compares the PCR event log to the known good values stated in the `logging-site-whitelist.xml`.

2. `AM.logHashMatchesKeyCredentialHash()` checks to see if the software-computed final hash value matches the actual PCR value quoted in the public key credential — this is intended to guarantee that the PCR event log truthfully represents the executables that have been measured and extended in the specified PCRs.

3. `AM.validateKeyCredentialSignature()` validates the AIK signature on the public key credential by decrypting the credential's validation data with the public AIK, and compares this decrypted value to a digest.

6.2.2.3 Creating a Job and Encrypting its Secret

`LMS.submitLogAccessJob()` creates a log access job for each of the selected participants (logging sites) and dispatches it through the resource broker. During this process, it uses `TPMUtils.encryptData()` to encrypt the job secret with the target participant's public key, and uses `LMS.createAttToken()` to create an attestation token. This method, in turn, uses `AIKManager.getFullAIKInfo()` to access the full AIK credential.

If an AIK certificate already exists, it calls `getAIKCertificate()` to use the `TPMUtils.getAIK()` and `TPMUtils.getAIKUsagePolicy()` methods to fetch the AIK and its usage policies from the local storage. If a certificate does not exist,

it calls `getAIKCertificateFromPrivacyCA()` to generate a new AIK certificate through the Privacy CA.

In a similar manner `TPMUtils.createSealedKey()` generates a *bound* key pair and seals the private half to the specified PCR. The public key credential is signed with the private AIK and returned to `LogMigrationService`. This credential contains validation information for the AIK signature and the sealing property.

`TPMUtils.signDataInsideTPM()` signs the log access query using the sealed private key, and this signature is also included as part of the job secret. This method would fail if the PCR value changes, disallowing access to the sealed key. `Job.jobDescription` contains information necessary for the middleware to forward the job correctly to the target participant, and specifies the hardware and software requirements for running the job.

6.2.2.4 Submitting the Job

A dummy resource brokering service (one of the assumed components) merely reads `Job.jobDescription` to find the target participant, and submits the job using the `LMS.submitJob()` method of the participant system.

6.2.2.5 Isolating the Trusted Computing Base

`LogMigrationService` as well as all other classes and libraries shown in Figure 6.2 are deployed inside an independent guest virtual machine. A dummy log analysis application accesses the externally-facing web service methods of `LogMigrationService` to collect logs from trustworthy logging sites.

6.2.2.6 Integrity-Based Job Authentication

The next five features are described with reference to the second participant (log owner) system class diagram (see Figure 6.3).

After the job arrives, `LMS.authenticateJobUsingAttToken()` authenticates the job based on the integrity report. It uses `AttestationManager` to determine whether the job has been dispatched from a trustworthy log user system (see Section 6.2.2.2 for details). PCR log event (obtained from the attestation token) is compared with `user-system-whitelist.xml`, and the authenticity of the PCR quote is verified during this process.

6.2.2.7 Measuring the Virtual Machine Files

Before decrypting the job secret, `LMS.measureVMImageFiles()` resets PCR 16 using `TPMUtils.resetPCR()`. Then, it measures the virtual machine files at runtime and extends PCR 16 with the new measurement using `TPMUtils.extendPCR()` method.

6.2.2.8 Decrypting the Job Secret

`LMS.decryptJobSecret()` sends `AttestationToken.encryptionKeyLabel` (the label of the public key used to encrypt the secret) to `TPMUtils.unbind()`, which uses the label to retrieve the sealed private half and decrypts the job secret. The decrypted bytes are converted into a `JobSecret` object before being returned. If either one of the two PCR values change, the private key will no longer be accessible for decryption.

6.2.2.9 Validating the Signature

`LMS.attTokenCorrelatesToJobSecret()` calls `TPMUtils.verifySignature()`, which decrypts the signed log access query with the user's public key and compares it with the original query. A valid signature implies that the attestation token correlates with the job secret, and the query has been generated from an integrity-protected user system.

Due to a number of implementation challenges and the time constraints, a feature that launches the virtual machine from the verified files at runtime has been left out. Instead, this feature is assumed to run after signature validation to deploy the job on a trustworthy virtual machine instance.

6.2.2.10 Encrypting the Log Results

`LMS.accessLogs()` calls `AuthPolicyMgmtService.isAuthorisedUser()` to check whether the user is authorised to run the specific log access query — this assumed method always returns true. Then the query is executed to access the logs and privacy policies from a dummy database.

Using `TPMUtils.encryptData()`, the accessed data and the original nonce are encrypted with a symmetric session key. This session key is encrypted with the user's public key. The final `LogResult` object contains the `encryptionKeyLabel`

(public key label), `encryptedResults` and `encryptedSessionKey`, and the original `jobDescription`.

6.2.2.11 Decrypting the Log Results

To describe the last feature, the first class diagram (see Figure 6.2) is referenced again. After the `LogResult` arrives, the encrypted data and the `encryptionKeyLabel` is passed on to the `TPMUtils.unbind()` method. This method retrieves the sealed private key with the given key label and decrypts the session key. This session key is then used to decrypt the logs, privacy policies and nonce. The returned nonce and the original nonce are converted into `HexString` before being compared. A matching value verifies the integrity of the job execution environment.

6.3 Observations

The prototype implementation has been developed over three months, containing approximately 6500[1] lines of code. This section discusses the feasibility, security and usability issues raised from the prototype work, and highlights the potential areas of concern. Based on the implementation details, a general guideline for developing remote attestation (and sealing) applications is also suggested.

6.3.1 Feasibility

Considering that the biggest motivation for this work is to demonstrate the feasibility of the proposed log reconciliation infrastructure, the results are promising: prototype implementations of the selected features have been constructed and documented successfully (see Section 6.2). These provide strong evidence of feasibility for the trusted computing ideas proposed in Chapter 5.

Several components and their features (for example, the configuration resolver) have been assumed on the grounds that their implementation strategies should be sufficiently straightforward. One exception to this is the 'runtime launch of a virtual machine' feature, which has been left out due to the implementation challenges involved.

The Open Trusted Computing consortium, in their recently developed 'Corporate Computing at Home' prototype [91], have demonstrated how the integrity

[1]This number does not include the jTSS libraries.

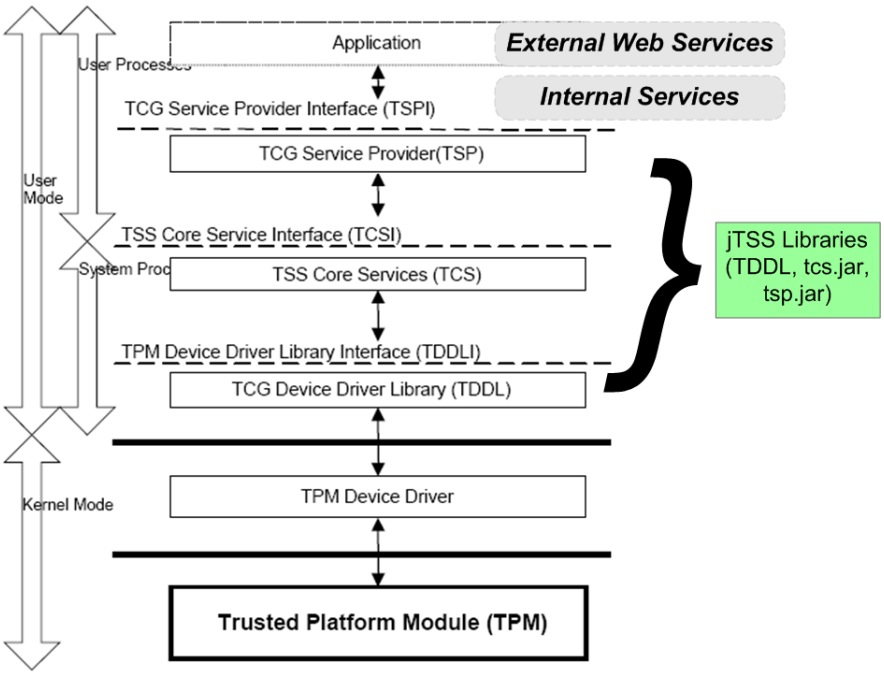


Figure 6.4: TCG Software Layering Model (Modified Figure4:i from [128])

of the corporate virtual machine could be verified and launched at runtime to guarantee a trustworthy working environment. Future work may look at adapting their implementation methods.

6.3.2 Establishing a Guideline

Another motivation for this work is to establish a guideline for developing trusted computing (in particular, remote attestation and sealing) applications. The high-level class diagrams (see Figures 6.2 and 6.3) highlight the essential classes and methods.

These diagrams suggest a three-tier structure: (1) the jTSS libraries that provide core methods for utilising the TPM, (2) the internal services that contain the business logic for performing attestation and sealing, and (3) the external web services and data objects that make use of the internal services to enable secure job submission. The implementation details give some ideas as to how these methods should be orchestrated together (see Section 6.2.2). Moreover, an example whitelist demonstrates some of the essential information that needs to

be captured and maintained in an application whitelist (see Section 6.2.2.1).

Figure 6.4 shows the Trusted Computing Group's software layering model [128] and how the three-tier structure would fit into this model. The jTSS libraries would fit into the bottom three software layers in the model: TPM Device Driver Library Interface, TSS Core Service Interface, and the TCG Service Provider Interface. The external web services would fit into the Application layer, and the internal services would fit in between the Application and TCG Service Provider Interface layers.

6.3.3 Security

Security can be analysed from the implemented features and tests. Remote attestation has been implemented to: (1) compare the PCR event log with the whitelist entries, (2) check whether the computed final hash matches the quoted PCR value, and (3) verify the authenticity of the public key credential by validating its AIK signature with the public AIK. Step (2) is responsible for checking the sequence of the authenticated boot process and verifying the integrity of the PCR event log. These procedures enable a platform configuration reporting mechanism that is integrity and authenticity protected.

After attestation, the job is encrypted using the participant's public key and dispatched via the untrusted, resource brokering service. The sealed key mechanism ensures that only a securely configured platform can decrypt the secret and process the job further. This security property has been tested by changing one of the PCR values (for which the private key is bound to) and checking whether decryption fails. The test results showed that the TPM denied further access to the private key when one of the values changed and decryption failed.

Another test involved hacking into the `Job` and `JobSecret` objects when the job arrived at the resource brokering service — the test simulated compromised middleware trying to read the job secret. The job secret, however, was only visible as an encrypted, unreadable blob data. These results also demonstrated that the log results (similarly encrypted) will be safeguarded from such attacks.

The prototype implementation has also been useful for identifying potential issues. One area of concern is the management of the key secrets (or passwords). Each time a TPM secret key is used, a key secret is used for authentication. There are many keys and cryptographic operations involved in attestation and

sealing, and it would be unrealistic to ask the user to enter the secret every time a key access is required. A more usable solution would ask for the key secret once and store it somewhere safe for later use. But who provides the safe storage for these key secrets?

Encrypting the key secret will have the same problems, and the main memory can not be trusted since it is vulnerable to in-memory attacks. In order to safeguard the key secret from such attacks, strong isolation between a trusted (TPM-measured) application and other applications is necessary. Hardware virtualization, for instance, can be used to isolate the trusted application, and protect the key secrets in a dedicated memory or disk space.

Secure management of the application whitelist is another area of concern. Since attestation relies on the whitelist having up-to-date entries, if the administrative software fails to update the entries in a secure and timely manner, the attestation service could end up verifying untrustworthy configurations as trustworthy. Hence, the whitelist entries need to be carefully updated based on the latest software vulnerability reports and patches. Their integrity also needs to be protected.

6.3.4 Usability

There are several usability issues raised from the prototyping work. One area of concern is the complexity of the administrative tasks involved in setting up this type of architecture. Some of these tasks include:

1. defining a whitelist and keeping the list up-to-date through various software testing and vulnerability discovery practices;

2. initiating the TPM, and setting up the certificates and trusted computing protocols;

3. installing Xen virtual machine monitor, configuring Domain-0 settings, and setting up several log access virtual machine images and

4. installing the three-tier architecture and configuring the policies and settings files.

Each of these requires prior knowledge of trusted computing and virtualization; even a fully trained administrator might find such tasks difficult and time consuming to complete. Future work may look at minimising the trusted computing components that need to be installed, and automating some of these tasks so that even an unexperienced administrator could easily install and manage a secure log reconciliation system.

The end users are no exception. Upon job submission, the users choose acceptable logging system configurations from looking at the whitelist entries. Hence, to a certain extent, they too have to deal with the whitelist entries and understand the concepts of sealing and remote attestation. Again, more user-friendly job submission tools which require minimal knowledge of trusted computing are needed.

6.4 Chapter Summary

The prototype implementation of the log reconciliation infrastructure has been described in this chapter, demonstrating its feasibility as well as uncovering some of the security and usability issues. Implementation guidelines for developing remote attestation and sealing applications have also been provided. In the next chapter, the key features of the reconciliation infrastructure will be adapted to describe two types of trustworthy distributed systems that allow users' jobs to be executed in protected, verifiable environments.

Chapter 7

Generalisation: Trustable Virtual Organisations

Despite substantial developments in the area of trustworthy distributed systems, there remains a gap between users' security requirements and existing trusted virtualization approaches.

This chapter aims to highlight the missing pieces and suggest possible solutions based on a generalisation approach. A number of security mechanisms from the 'log reconciliation infrastructure' (see Chapter 5) are selected and extended to solve a generalised set of security problems.

7.1 Security Challenges

A wide range of research is conducted, archived, and reported in the digital economy. Its influence has grown over the years to include various disciplines from science through to finance and industrial engineering. In consequence, different types of distributed systems have been deployed to facilitate the collection, modeling and analysis of the dispersed data, or the sharing of the computational power.

A problem arises, however, when the models or data contain sensitive information or have commercial value, and the scope for malicious attempts to steal or modify them spreads alarmingly quickly. This is particularly true in many of the scientific disciplines where researchers — who care about the confidentiality of their privileged data or the integrity of the collected results — are reluctant to exploit the full benefits of distributed computing. Some of the malicious ob-

jectives of attackers include: (1) to steal the job secrets — including the user credentials, and sensitive models or data, (2) to make unauthorised use of the distributed resources, (3) to compromise user or participant machines, and (4) to corrupt the job results.

A virtual organisation will inevitably evolve over time to accommodate new utilities and services. This dynamic nature makes it extremely difficult to bridge the 'trust gap' between the security requirements and the current technological capabilities. Submitting a highly privileged job to the distributed resources requires prior knowledge of the security standards of all of the target participant systems — only those running with acceptable security configurations and patch levels should be selected to execute the job. However, this still remains as the trust gap and serves as a barrier to up-take of existing distributed systems.

This chapter proposes two trustworthy distributed systems based on the ideas generalised from the log reconciliation infrastructure. In both systems, the 'configuration resolver' is configured to play a central role in 'discovery of trustworthy services' and 'secure job submission'. A runtime verification of the integrity of the job virtual machine guarantees the exact job execution environment.

The rest of the chapter is organised as follows. Section 7.2 discusses three well-known grid examples to highlight the challenges of forming a virtual organisation. Section 7.3 identifies the key security requirements. Section 7.4 discusses an emergent consensus view of the trusted virtualization approach, and identifies the missing components. Then, Section 7.5 describes two architectures that satisfy the requirements, suggesting possible solutions for the missing components. Finally, Section 7.6 observes the proposed architectures and discusses the remaining challenges.

7.2 Motivating Examples

The following examples serve to illustrate the common security problems of sharing computational resources or aggregating distributed data within a virtual organisation.

7.2.1 climate*prediction*.net and Condor Grids

The first example application arises with the climate*prediction*.net project [58], which serves to distribute a high quality climate model to thousands of participants around the world. It stands (or falls) on its ability to ensure the accuracy of the climate prediction methods and collected data. As a politically-charged field it could become a target for moderately sophisticated attackers to subvert the results.

This project highlights a common dual pair of problems:

- From the participant's perspective, the untrusted code runs on their trusted system; they need to be convinced that the code is not malicious, and the middleware used by the code (if any) are trustworthy.
- From the scientist's perspective, their trusted job is executed in an untrusted host without any assurance of the running environment; this host might return arbitrary or fabricated results never having run the original code, or steal their sensitive models and data.

Similar threats undermine the security of a Condor [31] system which allows relatively smaller jobs to be distributed in a Campus Grid setting. To mitigate the second problem it provides a digital certificate infrastructure for the participant to identify others. Without robust mechanisms to safeguard the keys from theft, however, this solution offers only a slight improvement over legacy architectures. Moreover, a rogue administrator might replace (or tamper with) the compute nodes with others, or subvert its security configurations to steal data and/or return contrived results.

7.2.2 Healthcare Grids

In 'e-Health', it is not hard to imagine instances where the clinical data is highly sensitive, and only the processed subsets may be released; nor is it hard to imagine scenarios where reconciliation of data from different sources is needed, but neither clinic trusts the other to see the raw data. Such data cannot normally be made available outside the healthcare trust where it is collected, except under strict ethics committee guidance, generally involving *anonymisation* of records before release [94].

Nevertheless, anonymisation reduces the amount of information available, precision of estimates and flexibility of analysis; and as a result, bias can be introduced [39]. For example[1], a researcher might be looking at association between *age*, *diet* and progression of *colon cancer*, and is aware that the risk immensely increases when one reaches the age of 50. Patient records for the first two attributes would be accessed through a GP practice and the third through a specialist clinic. The *National Health Service* (NHS) number [51] uniquely identifies a patient across the grid to enable the linking of data. In this scenario a graph plotted with anonymised age — '30-35', '35-40' ... '65-70' — is likely to miss out the important micro-trends all together; in fact, these would be better-observed with datasets closer to age 50. A supporting graph plotted with the *actual age*, say, between 45 and 55, would show these trends more clearly and improve the quality of the results.

Moreover, this distributed query would require a concrete identifier, such as the NHS number, to join patient records collected from the GP and specialist clinic. In reality, however, it is unlikely that either would give out such potential identifiable information without the necessary confidentiality guarantees. Hashing NHS number can provide some assurance but it would still be vulnerable to brute force attacks. These problems require a trustworthy application to perform *blind* reconciliation and analysis of the data from mutually-untrusting security domains: the researcher would only see this application running and the end results; the raw data should never be accessible to anyone.

7.3 Generalised Security Requirements

Mindful of the security challenges discussed in the above examples, the log reconciliation requirements (see Section 3.3) are adapted and generalised to describe the key requirements for designing a trustable distributed system:

1. **Secure Job Submission** Both the integrity and confidentiality of the job secrets should be protected upon job submission. Attackers should not be able to steal or tamper with the job secrets while being transferred via untrusted midddleware services.

[1]This example has been developed with help from David Power and Mark Slaymaker who are involved in the GIMI project [3], and Peter Lee who is an intern at the Auckland Hospital.

2. **Authorisation Policy Management** When the job arrives at the participant system, the job owner's rights should be evaluated against the authorisation policies. The job should only be processed further if the job owner is authorised to run their queries or codes on the participant system.

3. **Trustworthy Execution Environment** A trustworthy, isolated job execution environment should be provided — the jobs should be executed free from unauthorised interference (e.g. attempts to modify the data access query or the model code), and the confidentiality of the job secrets should be protected from processes running outside this environment. The user, before submitting the job, should be able to verify that the trustworthy execution environment is guaranteed at the participant system. Similarly, the participant should be ensured that only a fully verified environment is used in their system for executing the jobs.

4. **Job Isolation** The jobs should be isolated from each other and from the host. This is to prevent rogue jobs from compromising the participant system, or stealing the secrets and results of other jobs running in the same system. This should also prevent a malicious host from compromising the job integrity, confidentiality and availability.

5. **Protecting the Results** The integrity and confidentiality of the results should be protected.

6. **Digital Rights Management** In distributed data systems, unauthorised uses or modification of the sensitive data should be prohibited wherever they are processed.

7. **Blind Analysis of Data** The raw data should not be disclosed to the end user. Only the processed, anonymised results should be made accessible for analysis.

7.4 Trusted Virtualization Approach

Recently, many researchers have discussed the use of trusted computing and virtualization to fulfill some of these security requirements. In Section 2.6, existing trusted virtualization approaches were reviewed and the similarities between the

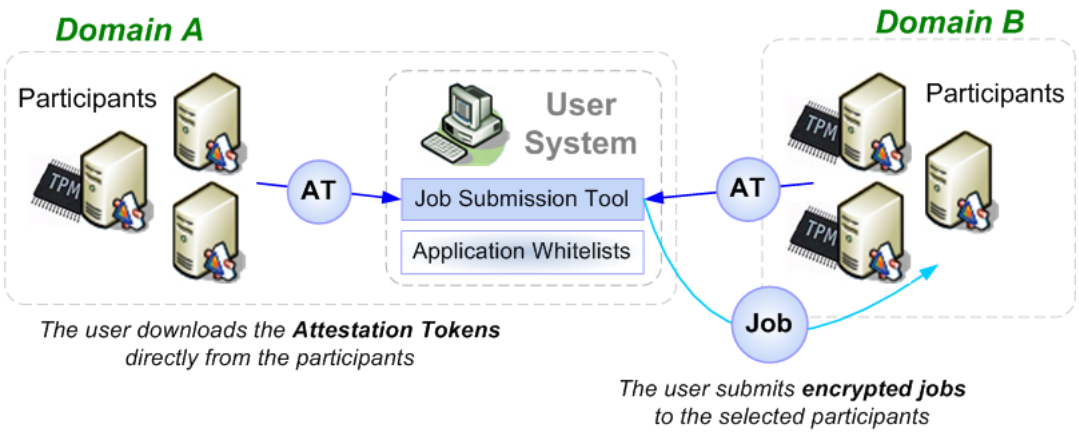


Figure 7.1: A Consensus View

emerging ideas were identified. This section establishes an 'emergent consensus view' based on these similarities (see Figure 7.1), and demonstrates its shortcomings in the areas of platform configuration management and provision of trustworthy execution environment.

7.4.1 A Consensus View

Figure 7.1 presents an emergent consensus view of the trusted virtualization ideas discussed in Section 2.6. An 'attestation token' contains a participant's platform configurations and the public half of a TPM key. The private half is sealed to the platform's trusted computing base. 'Property-based attestation' [9] allows important security properties to be measured and attested. Before job submission, the attestation token is used to verify the security configurations of remote platforms.

In a typical attestation scenario, a user would (1) obtain the attestation token of a participant system, (2) check its security configurations against a locally maintained whitelist, and (3) dispatch a job to only those known to provide a 'trustworthy execution environment'.

Between steps (2) and (3), the job would be encrypted with the participant's public key (obtained from the attestation token). Since the private half is protected by the TPM, this encrypted job would be safely distributed over the unprotected network. This is known as the 'sealed key approach'.

7.4.2 Missing Pieces and Potential Solutions

Having described the consensus view, this section identifies the missing components and suggests potential solutions.

The first missing piece is a platform configuration discovery service. In the Trusted Grid Architecture [57], the users are expected to fetch the attestation tokens directly from the participants. How the users would actually manage this process, however, is not considered in depth. Generally, it is assumed that a central service is already available for the users to discover participants' platform configurations. In consequence, various security and management issues associated with developing a 'match-making service' as such are often overlooked.

In the consensus view, the burden of performing attestation and managing the application whitelists rests with the users. This seems unrealistic in large-scale distributed systems, however, since the whitelist entries will be modified and updated constantly. An average user will not have sufficient resources to cope with these changes. Referring back to the Trusted Computing Group's runtime attestation model (see Section 2.4.3), the 'Configuration Verifier' is missing in the consensus view. Some suggest passing on the problem to a trusted third party [131, 83], but without providing much insights on how this could be implemented or managed.

The 'configuration resolver' (introduced in Chapter 5) is a good candidate for this role. It could be configured to manage the application whitelists and perform configuration verification (attestation) on behalf of the users. It would be responsible for keeping up-to-date whitelists through various vulnerability tests and data collected. This type of service is described by the Trusted Computing Group as an aggregation service and has been suggested in a number of projects [140, 132]. For instance, Sailer et al. [106] encourages the remote users to keep their systems at an acceptable patch level using a package management database. This database gets updated whenever a new patch is released, so that the new versions are added to the whitelist and old versions are removed.

From the participants' perspective, an 'integrity-based job authentication' mechanism is also missing. Only the users (job owners) are capable of verifying the participants' platform configurations and execution environments. The platform owners usually rely on a basic digital certificate infrastructure to identify

the users and authenticate the job virtual machines. This provides no assurance for the security state of the job virtual machines.

In the context of data grids — where the jobs might try to access sensitive data — the data owner should have full control over the software used for executing the query and protecting the accessed data. Virtual machine isolation can only prevent other rogue virtual machines from stealing the accessed data. If the job virtual machine itself is malicious, and tries to run malicious queries on the database, then isolation will not be enough.

Basic encryption and digital signatures are often used to protect the data once they leave the data owner's platform [70]. However, considering the number of connected nodes and the security threats associated with each, these security measures alone cannot provide the necessary confidentiality, privacy and integrity guarantees. A more reliable Digital Rights Management system is needed to allow the data owner to maintain full control over their data. The data access policies and privacy policies need to be consistently enforced throughout the distributed system (these are often referred to as 'sticky policies' [36]). The end users should only be able to access the processed, anonymised results which are just sufficient to perform the requested analysis.

Meanwhile, Australia's Commonwealth Scientific and Industrial Research Organisation (CSIRO) has developed the Privacy-Preserving Analytics (PPA) software for analysing sensitive healthcare data without compromising privacy and confidentiality [27]. Privacy-Preserving Analytics allows analysis of original raw data but modifies output delivered to the researcher to ensure that no individual unit record is disclosed, or can be deduced from the output. This is achieved by shielding any directly identifying information and deductive values that can be matched to an external database. Some of the benefits of being able to access the raw data are [39]:

- no information is lost through anonymising data prior to release and there is no need for special techniques to analyse perturbed data;
- it is relatively easier to anonymise the output than modifying a dataset when it is not known which analyses will be performed; and
- clinical decisions will be based on more reliable information and treatments can be more tailored to individuals with the likelihood of problems.

Privacy-Preserving Analytics (or any other secure analysis tools), combined with remote attestation, could provide the necessary confidentiality and privacy guarantees for the data owner to freely share raw data in the virtual organisation. For instance, attestation could verify that a trustworthy Privacy-Preserving Analytics server is responsible for performing data reconciliation and anonymising the output before releasing the results to the researcher.

7.5 Trustworthy Distributed Systems

This section proposes two types of distributed systems that aim to satisfy the generalised requirements, and bridge the gaps identified in the consensus view (see above). The 'configuration resolver' plays a central role in both systems, maintaining an up-to-date directory of trustworthy participants and handling the job distribution process.

Section 7.5.2 describes the generalised configuration resolver, and how it manages configuration verification and job distribution processes. Based on the new security primitives that make use of the resolver, Sections 7.5.3 and 7.5.4 describe a computational system and a distributed data system that are trustworthy.

7.5.1 Assumptions

This section states two assumptions about the design of the proposed distributed systems.

1. A public key infrastructure is available and this can be used to verify the identity of the configuration resolver, participants and end users.

2. A participant's system supports trusted computing and virtualization; as minimum, mechanisms like authenticated boot and remote attestation are enabled in this system.

7.5.2 Generalising the Configuration Resolver

Building on the consensus view of the trusted distributed systems, the configuration resolver is added to each administrative domain to manage the trustworthy participants' platform configurations and a whitelist of locally acceptable platform configurations (see Figure 7.2). To become part of the trusted domain, a

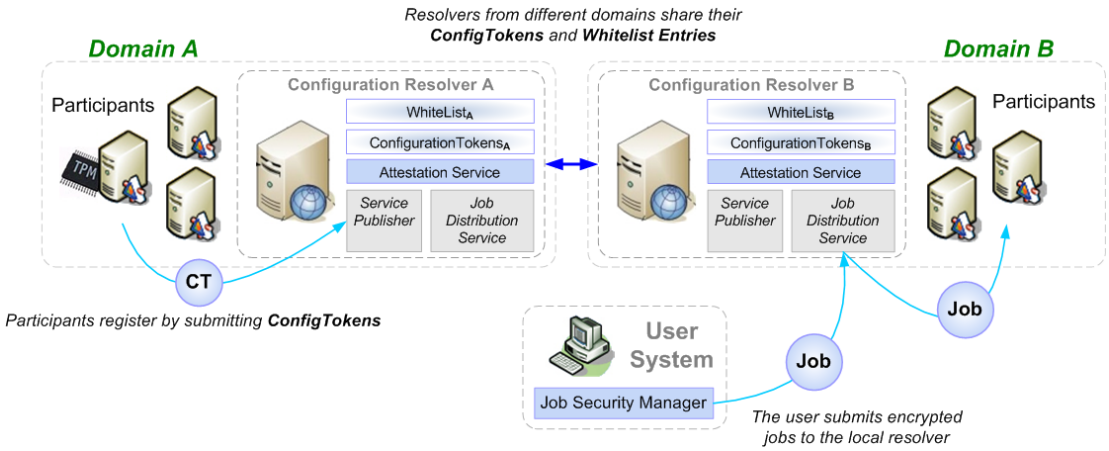


Figure 7.2: Consensus View with the Configuration Resolver

participant registers with the local configuration resolver by submitting its Configuration Token (CT).

$$CT = (\ \texttt{PCR Log},\ AIK\ ,\ \{\texttt{cred}(AIK)\}_{CA}\ ,\ P_K\ ,\ \{\texttt{cred}(P_K)\}_{AIK}\ ,\ \{\texttt{Description}\}_{S_K}\ )$$

$\texttt{PCR Log}$:	the list of the loaded applications and their hash values
AIK :	the public half of the Attestation Identity Key
$\{\texttt{cred}(AIK)\}_{CA}$:	the AIK credential issued by the Privacy Certificate Authority
P_K :	the public half of the non-migratable TPM key
$\{\texttt{cred}(P_K)\}_{AIK}$:	the P_K credential signed using the private half of the AIK
$\{\texttt{Description}\}_{S_K}$:	the service description signed using the private half of P_K

This token includes the Attestation Identity Key (AIK) and an AIK credential issued by the Certificate Authority ($\{\texttt{cred}(AIK)\}_{CA}$). A public key credential, signed by this AIK, is also included to state that the private half has been sealed to *two* PCR values which correspond to (1) a trustworthy authenticated boot process, and (2) per-job virtual machine image files (see Figure 7.3). The `PCR Log` contains the full description of the authenticated boot process and the virtual machine image files. In addition, a service `Description` is included, signed by the private half of the sealed public key, demonstrating that the users should use this public key when submitting jobs to this participant.

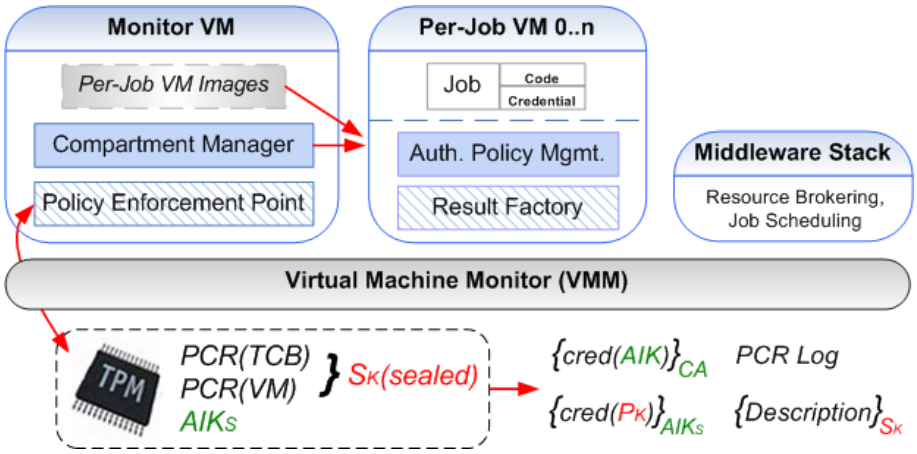


Figure 7.3: Participants' Trusted Computing Base

The resolver verifies the trustworthiness of the platform by comparing the `PCR Log` with the whitelist. If the platform is trustworthy, its configuration token is added to the resolver's token repository, ensuring that only the trustworthy participants are ever advertised through the resolver. This is different to the way the resolver operates in the log reconciliation architecture, which merely forwards the tokens to the users — there, the burden of verifying the integrity reports rests on the users.

As a minimum, the authenticated boot process will measure the BIOS, bootloader, virtual machine monitor, and privileged monitor virtual machine. Therefore, the first PCR value is sufficient to state that the platform is running in a virtualized environment and its monitor virtual machine is securely managing the per-job virtual machines (see Figure 7.3). Additionally, the second PCR value guarantees the exact software and security configurations of a per-job virtual machine (job execution environment). This second value is stored in a *resettable* PCR (see Definition 2.7) since the virtual machine image files are remeasured and verified at runtime. These security properties allow the user to have strong confidence in the correctness of the data or computational results returned from this platform.

Note, in contrast to the reviewed approaches (see Section 7.4.2), the participant controls the virtual machine instances that are allowed to be used in their platform for executing the jobs. This is responsible for meeting Requirement 3 (see Section 7.3). However, this also restricts the number of software environ-

ments that the user can choose from and will affect the overall usability of job submission.

To improve usability and flexibility, the resolver allows a participant to submit multiple configuration tokens (for a same platform), all representing the same authenticated boot process but each sealed to a different per-job virtual machine image. Such tokens could be used to offer multiple services — by configuring each software environment to provide a different service — or, offer multiple software configurations for the same service, giving the user more options to choose from.

The configuration resolver performs a range of security and platform configuration management functions through the following services (see Figure 7.2):

- An internal 'attestation service' is responsible for performing all attestation related functions to ensure that only trustworthy participants register with the resolver.
- An external 'service publisher' provides the necessary APIs for the participants to register and advertise their services through the resolver. It makes use of the attestation service.
- The users submit jobs through an external 'job distribution service', which selects the most suitable sites by looking at the service `Descriptions` and dispatches the jobs to them.
- An external 'whitelist manager' allows the domain administrators to efficiently update the whitelist entries.

Each participant becomes a member of the resolver's WS-ServiceGroup [127] and has a ServiceGroupEntry that is associated with them. An entry contains service information by which the participant's registration with the resolver is advertised. The configuration tokens are categorised and selected according to the type of services they advertise. It is assumed that there is a public key infrastructure available to verify the participant's identity (see assumption 1, Section 7.5.1).

7.5.3 Computational Distributed System

In an idealised computational distributed system, the user would not care about where their job travels to as long as their sensitive data and results are protected. It would therefore make sense for the resolver to perform trusted operations like selecting suitable sites and dispatching jobs on behalf of the Job Owner (JO). The resolver's TPM is used to measure the configurations of its external and internal services, and generate an attestation token ($AT(CR)$) to attest its security state to the users.

$$AT(CR) = (\ \texttt{PCR Log},\ AIK(CR)\ ,\ \{\texttt{cred}(AIK)\}_{CA}\ ,\ P_K(CR)\ ,\ \{\texttt{cred}(P_K)\}_{AIK}\ )$$

`PCR Log` :	the list of the loaded (and measured) applications and their hash values
$AIK(CR)$:	the public half of the resolver's Attestation Identity Key
$\{\texttt{cred}(AIK)\}_{CA}$:	the resolver's AIK credential issued by the Privacy Certificate Authority
$P_K(CR)$:	the public half of the resolver's non-migratable TPM key
$\{\texttt{cred}(P_K)\}_{AIK}$:	the resolver's $P_K(CR)$ credential signed using the private half of the AIK

Much like the tokens described previously, the resolver's public key credential ($\{\texttt{cred}(P_K)\}_{AIK}$) identifies the corresponding private key as being sealed to its trustworthy state. The `PCR Log` describes the fundamental software stack and the services that have been measured during the resolver's authenticated boot process.

In the user system, all the job security functions are enforced by the 'job security manager' virtual machine (see Figure 7.4): it is designed to perform a small number of simple security operations to minimise the attack surface. Attestation of the job security manager, monitor virtual machine and virtual machine monitor is sufficient to be assured that the job security functions have not been compromised — these components form the trusted computing base of the user system. Upon installation of this architecture, the user system will be capable of securely submitting jobs to the resolver and verifying the returned results.

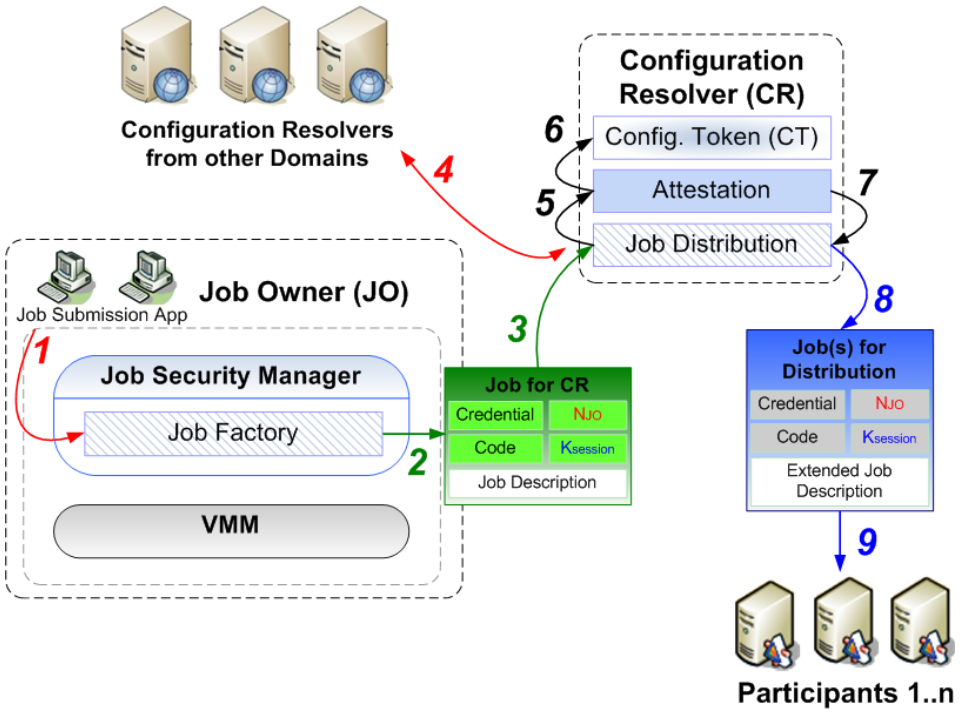


Figure 7.4: Creation and Distribution of Encrypted Job(s)

7.5.3.1 Creation and Distribution of Encrypted Job(s)

All end user interactions are made via the external 'job factory'. It provides the minimal interface (APIs) necessary for development of a job submission application. Such an application should be designed to allow the user to specify the job description (requirements), the credentials and the code to be executed.

Imagine that a scientist (the job owner in this scenario) is carrying out an experiment that aims to predict the future climate state. The scientist submits the prediction model code through their job submission application and specifies the job description (**1**, Figure 7.4). The job factory creates a secure job containing the following attributes (**2**, Figure 7.4):

$$Job = (\ \{\texttt{Credential, Code, } K_{session}\texttt{, } N_{JO}\}_{P_K(CR)}\ ,\ \texttt{Job Description}\ )$$

`Credential` :	the job owner's credential
`Code` :	the code to be executed
$K_{session}$:	the symmetric session key created by the job factory
N_{JO} :	the job owner's nonce, used for verifying the returned results
$P_K(CR)$:	the resolver's public key, used for encrypting the job secret
`Job Description` :	the description of the required applications and hardware capabilities

A symmetric session key ($K_{session}$) is included as part of the job secret; it will be used by the participant to encrypt the generated results. This session key is sealed to the PCR corresponding to the job owner system's trusted computing base, to prevent a compromised job security manager from decrypting the returned results. N_{JO} represents the job owner's nonce.

Before encrypting the job secret, the trustworthiness of the configuration resolver is verified by comparing the `PCR Log` (attained from the resolver's attestation token) against the locally managed whitelist of known good resolver configurations. If the resolver is trustworthy, the job secret — containing the user `Credential`, `Code`, session key and nonce — is encrypted with the resolver's public key for which the private half is sealed to its trusted state. This ensures that the secret is only accessible by a securely configured resolver. The job is then submitted to the resolver's job distribution service (**3**, Figure 7.4).

When the job arrives, the distribution service first attempts to decrypt the job secret with the sealed private key. It then requests configuration tokens from the resolvers managing other administrative domains (**4**, Figure 7.4). This request contains the job requirements, specifying the required application and hardware capabilities. Such information is obtained from the `Job Description`.

The recipient resolvers filter their list of tokens, selecting the relevant ones, and return them to the original resolver. The original resolver uses its internal attestation service to iterate through each token and verifies the integrity-report by comparing the PCR values against the local whitelist (**5**, **6**, **7**, Figure 7.4). Only those with acceptable configurations (for running the climate prediction model code) are selected and merged with the locally filtered tokens.

A job is recreated for each of the selected participants: during this process, the job secret is encrypted using the target participant's public key and the `Job Description` is extended with the host address (**8**, Figure 7.4). These jobs are

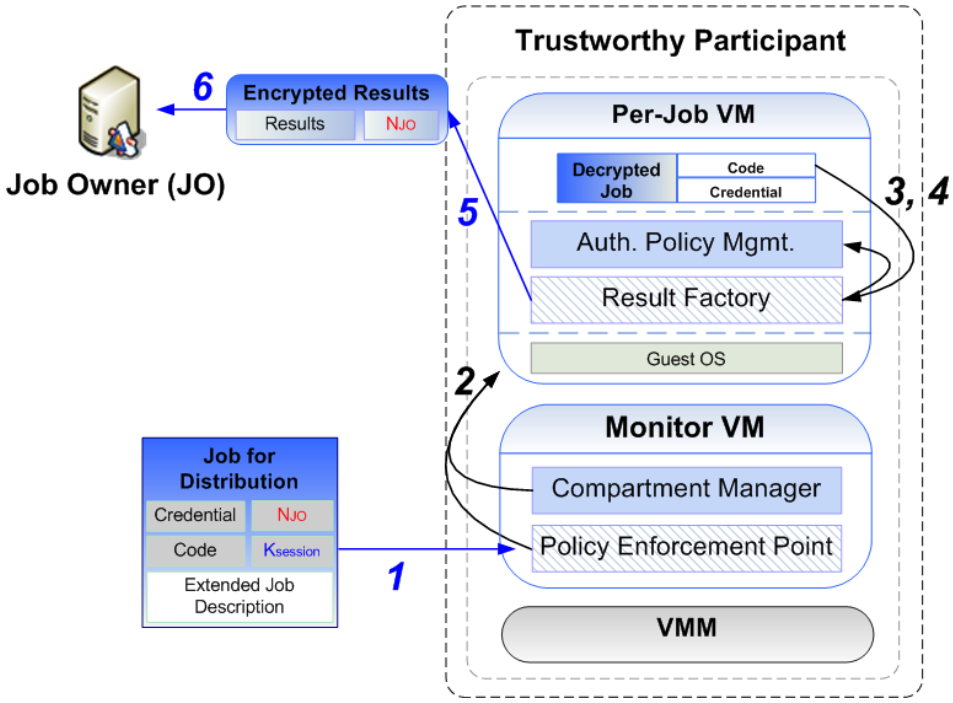


Figure 7.5: Operations of a Per-Job Virtual Machine

dispatched to the resource brokers for scheduling, which, in turn, forwards the jobs to the participants' policy enforcement points (**9**, Figure 7.4). Note, the resource brokers can only read the extended `Job Description` for identifying the target participants and scheduling the jobs.

7.5.3.2 Operations of the Trustworthy Execution Environment

Figure 7.5 demonstrates how a job gets processed at one of the participant systems. Any security processing required before becoming ready to be deployed in a per-job virtual machine is done through the policy enforcement point. It first measures the selected per-job virtual machine image (and configuration files), and resets the resettable PCR with the new value. Typically, this image consists of a security patched guest operating system and trustworthy middleware stack — the 'authorisation policy management service' and the 'result factory' (see Figure 7.5).

In order to decrypt the job secret, the policy enforcement point attempts to unseal the private key sealed to the participant's trusted computing base and the

virtual machine image. The private key will only be accessible if the platform is still running with trustworthy configurations *and* the image files have not been modified. This is intended to guarantee that only an integrity protected virtual machine has access to the job secret.

If these security checks are passed, the 'compartment manager' allocates the requested size of memory, CPU time and speed (specified in the `Job Description`), launches a virtual machine from the verified image, and deploys the decrypted job (**2**, Figure 7.5). Inside this virtual machine, the policy management service decides whether the scientist is authorised to run their prediction model in the participant platform. If the conditions are satisfied, the model is executed to simulate a probabilistic climate forecast (**3**, **4** Figure 7.5). The result factory generates a secure message containing the simulation `Results` (**5**, Figure 7.5):

$$R = \{\texttt{Results}, N_{JO}\}_{K_{session}}$$

The job owner's nonce (N_{JO}) is sufficient to verify that the results have been generated from an integrity protected virtual machine and unmodified code has been executed. The entire message is encrypted with the job owner's symmetric session key ($K_{session}$), which is protected by the job owner's TPM. This prevents attackers from stealing or tampering with the `Results`.

7.5.3.3 Collection of Results

At the job owner's system, the job factory receives the message and decrypts it using the sealed session key (**6**, Figure 7.5). Note, if the job factory has been compromised during the job execution period, the session key will no longer be accessible as the PCR value (corresponding to the trusted computing base) would have changed. Hence, a malicious job factory cannot steal the results, or return fabricated results to the original application.

The decrypted message is forwarded to the job factory which compares the returned nonce (N_{JO}) with the original. A matching value verifies the accuracy and the integrity of the results. These are then delivered to the scientist's application.

7.5.4 Distributed Data System

One of the pieces missing from the consensus view is a trustworthy, privacy-preserving analysis tool. As a potential solution, a combinational use of the 'Privacy-Preserving Analytics' software and attestation has been suggested in Section 7.4.2 to enable blind analysis of distributed data. This idea is expanded further to describe a 'Blind Analysis Server' (BAS), which allows analyses to be carried out securely via a remote server (see Figure 7.6): the user submits statistical queries by means of a job; analyses are carried out on the raw data collected from trustworthy sites, and only the processed results are delivered to the user.

The blind analysis server consists of the following components:

- The configuration resolver (see above).
- *Privacy Preserving Analysis Tool (PPAT)* — it can be any software designed to perform reconciliation on distributed raw data and run analyses on the reconciled information; it enforces the privacy policies on the processed results to protect the privacy of the sensitive data.
- *Privacy policies* — specify privacy rules governing the release of processed information; these are defined under strict ethics committee guidance to comply with legal and ethical undertakings made.

These three components form the trusted computing base of the blind analysis server. The server attests its security state through an attestation token ($AT(BAS)$): the token contains a public key credential signed by the $AIK(BAS)$ which identifies the private key as being sealed to the PCR value corresponding to its trusted computing base.

$$AT(BAS) = (\ \texttt{PCR Log},\ AIK(BAS)\ ,\ \{\texttt{cred}(AIK)\}_{CA}\ ,\ P_K(BAS)\ ,\ \{\texttt{cred}(P_K)\}_{AIK}\ )$$

The rest of the section uses the healthcare grid example (see Section 7.2.2) to explain how the security operations have changed from the computational architecture with the blind analysis server in place.

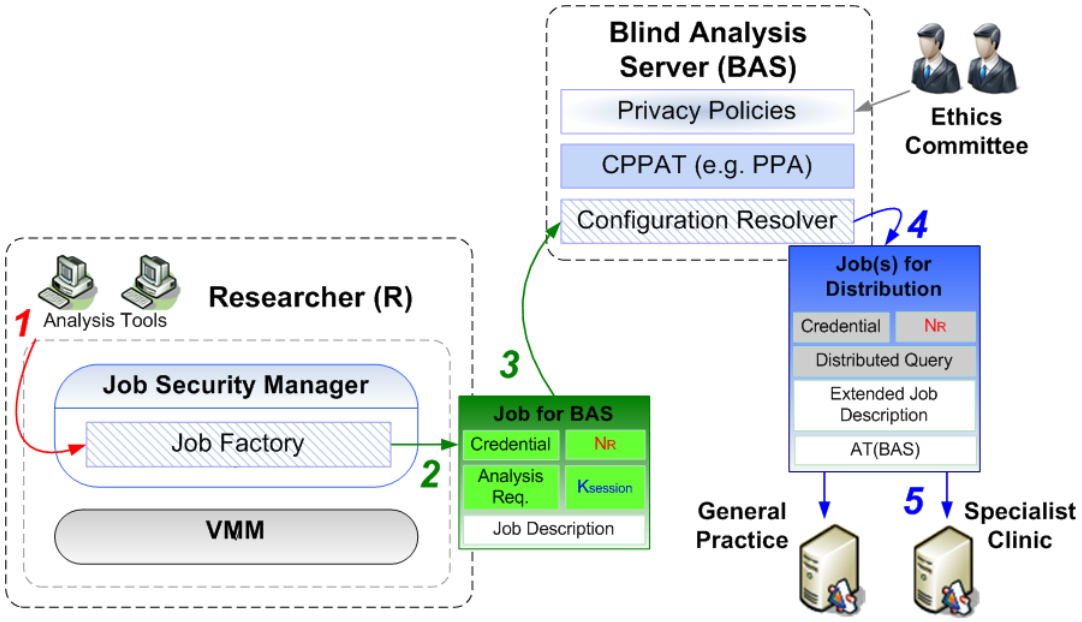


Figure 7.6: Operations of the Blind Analysis Server

7.5.4.1 Distribution of Job(s) through the Blind Analysis Server

A researcher is carrying out a study that looks at association between age (available from a GP practice) and progression of colon cancer (from a specialist clinic). This researcher specifies the analysis requirements via an external analysis tool to observe how the cancer statuses have changed for patients aged between 45 and 55 (**1**, Figure 7.6). The analysis tool should provide an appropriate interface for capturing the information required to run the analysis queries.

These user-specified analysis requirements go through the job factory, which first verifies the security state of the blind analysis server by comparing the `PCR Log` (obtained from the server's attestation token) against the known good configurations. It then creates a data access job (**2**, Figure 7.6) for which the secret is encrypted using the server's public key ($P_K(BAS)$). The analysis requirements are encrypted as part of the job secret. This job is then submitted to the configuration resolver running inside the analysis server (**3**, Figure 7.6).

In distributed data systems, the configuration resolver is configured to also manage the metadata of participants' databases. Hence, by looking at the researcher's analysis requirements, the resolver is capable of selecting relevant sites and constructing distributed database queries. Referring back to the healthcare

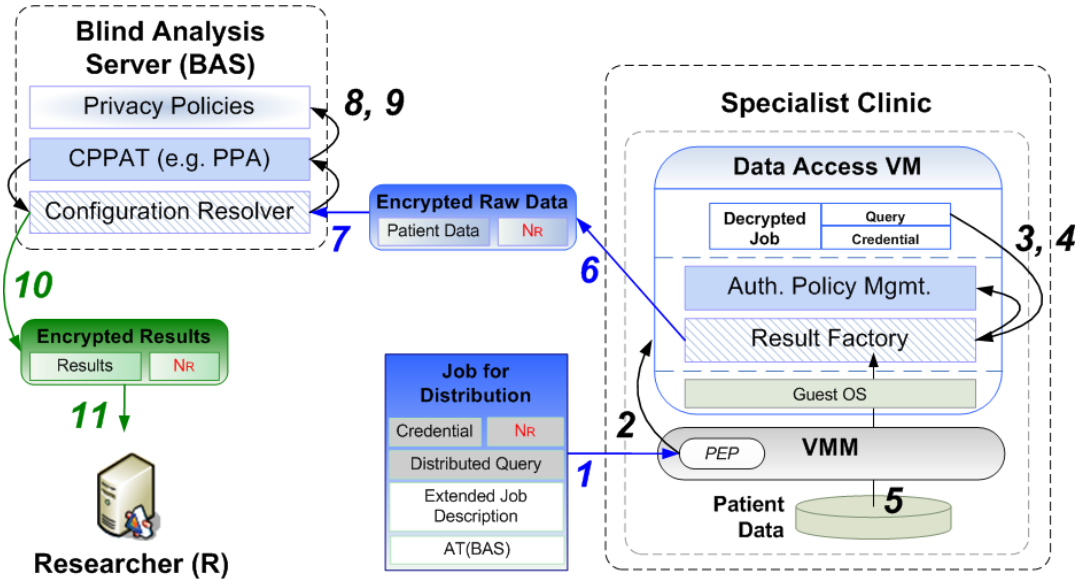


Figure 7.7: Operations of a Data Access VM

example, the resolver selects trustworthy GP and specialist clinic systems to collect the data from, constructs the distributed query, and dispatches a series of encrypted jobs to their policy enforcement points (**4**, **5**, Figure 7.6).

The unencrypted part of the job now includes the analysis server's attestation token ($AT(BAS)$) which can be used by the job recipients (data owners) to verify the trustworthiness of the server before processing the jobs. The researcher's session key, however, is omitted from this newly formed job secret since this key will only be used when the analysis server returns the final results to the researcher.

7.5.4.2 Operations of a Trustworthy Data Access Virtual Machine

Once the job arrives at the clinic, the policy enforcement point checks the security state of the analysis server using its attestation token ($AT(BAS)$) — this is how the job is authenticated at the clinic (**1**, Figure 7.7). It would detect a job dispatched from a compromised analysis server and prevent, for example, the server sending a malicious query. In order to simplify Figure 7.7, the policy enforcement point (which should be part of the monitor virtual machine) is drawn inside the virtual machine monitor.

After authentication, the per-job virtual machine image files are checked for integrity. The middleware installed on this virtual machine provides a common interface for the job to access the patient data. For instance, if implemented in Java, such services would include the Java Database Connectivity, connection string and Java virtual machine. Again, the private key — sealed to PCR values corresponding to both the trusted computing base and virtual machine files — is intended to guarantee that only a trustworthy virtual machine has access to the decrypted job secret to execute the query (**2**, Figure 7.7). The result factory checks the query for any attempt to exploit vulnerabilities in the database layer (e.g. SQL injection) before execution (**3**, **4**, **5** Figure 7.7).

A secure message containing the accessed data and the researcher's nonce (N_R) is encrypted with the data owner's symmetric session key (**6**, Figure 7.7). This session key, in turn, is encrypted using the analysis server's public key (available through the server's attestation token). Note, in contrast to the computational architecture, this result message is sent back to the analysis server and not to the researcher (**7**, Figure 7.7). The session key can only be decrypted if the server is still configured to match the original trusted computing base. Hence, a compromised server will not be able to steal the patient data.

7.5.4.3 Reconciliation of Collected Data

This result message and the encrypted session key arrives at the job distribution service of the resolver. The session key is decrypted first using the sealed private key, then the result message is decrypted using this session key. The returned nonce (N_R) is compared with the original to verify that the job has been processed (and the data has been accessed) through an integrity protected virtual machine.

Internal analysis tool (PPAT) reconciles the collected data and generates association between patients' age and colon cancer progression (**8**, **9**, Figure 7.7). During this process, the privacy policies are enforced to protect the privacy of the patient data. Attestation of the analysis server is sufficient to establish that these policies will be enforced correctly.

The final results are encrypted with the researcher's session key (obtained from the original job secret) and sent back to their job security manager (**10**, **11**, Figure 7.7). The researcher studies these anonymised results via the external

analysis tool, knowing that their integrity has been protected. This is intended to satisfy Requirements 6 and 7 (see Section 7.3).

7.6 Observations

This section explains how the proposed systems are responsible for meeting the security requirements defined in Section 7.3. The remaining whitelist management, job delegation, and security issues are also discussed.

7.6.1 Satisfying the Requirements

Job submission is a two-step process. Firstly, the job is submitted to the local configuration resolver where its secret is encrypted using the resolver's public key. The sealed key approach ensures that only a securely configured resolver can decrypt the job secret. Secondly, the resolver selects a trustworthy participant suitable for running the job; the job secret is encrypted using the public key of this selected participant and dispatched through an untrusted, public network. The private half is strongly protected by the participant's TPM. These features are responsible for meeting the 'secure job submission' requirement.

A combination of the sealed key mechanism and attestation is responsible for meeting the 'trustworthy execution environment', 'authorisation policy management', and 'job isolation' requirements. The trustworthiness of the trusted computing base and per-job virtual machine images of the participant are verified when they register with the local resolver. In this way, the resolver maintains a list of trustworthy participants.

The job is dispatched with its secret encrypted using the selected participant's public key. The private half is only accessible if neither the trusted computing base nor the virtual machine image has changed. The integrity of the virtual machine image is verified with runtime measurement of the files. These features are intended to guarantee a trustworthy execution environment that contains a securely configured authorisation policy management service. Moreover, the verification of the trusted computing base is sufficient to know that the virtual machine monitor is securely configured to provide strong isolation between the job virtual machines.

Virtual machine isolation ensures that the code is executed free from any unauthorised interference, including threats from rogue administrators to subvert the results. These results, before being sent back, are encrypted using the job owner's symmetric key which is strongly protected by the job owner's TPM. These features satisfy the 'protecting the results' requirement.

Finally, the provision of the blind analysis server aims to satisfy the 'digital rights management' and 'blind data analysis' requirements. The data owners verify the security state of the blind analysis server before allowing the query to run. Two properties checked are: (1) the state of the 'privacy preserving analysis tool' installed, and (2) the integrity of the data privacy policies. The accessed data are encrypted in a way that only a securely configured server can decrypt the data. These properties provide assurance that the integrity protected policies will be enforced correctly upon data processing, and only the anonymised results will be released to the user.

7.6.2 System Upgrades and Whitelist Management

As has been discussed before, the most significant overhead of the proposed systems is the cost of upgrading existing sites to support the new infrastructure. This involves installing the configuration resolver (or the blind analysis server) and its various sub components at each administrative domain, and standardising the communication mechanisms between them. While this is a large change, legacy resource discovery services can still be used without modification. Hence, the user can decide — depending on the level of security required for their jobs — when to use the resolver for discovering trustworthy participants. This concept is explained further through an example integration with the National Grid Service [84] in Section 8.2.2.1.

In systems spanning multiple administrative domains, different domains will likely have different software requirements and whitelist of acceptable configurations. While the administrators for one domain will be competent with the required list of software and their acceptable configurations for the local users, they will not know about all the software requirements in other domains. In consequence, multiple configuration resolvers could introduce availability issues depending on the level of inconsistency between their whitelists.

For example, if configuration resolver A is more active in inspecting software vulnerabilities and updating the whitelist entries than other domains, configuration tokens collected from configuration resolvers B, C, and D are likely to be classified as untrustworthy by resolver A, and their services will not be advertised to the users in Domain A. In order to minimise the level of inconsistency, the whitelist manager (in the resolver) needs to support functions that would enable efficient discovery and sharing of whitelist updates. The author has been engaged in a group research project [72] which explores these issues in detail, and suggests what the content of whitelist entries would be and how entry update messages would be shared.

7.6.3 Securing the Configuration Resolver

The burden of managing the tokens of trustworthy participants rests on the configuration resolver. It would also manage a revocation list of compromised TPMs and platforms. This is ideal from the perspective of the user since locally maintaining a whitelist and filtering trustworthy sites (for a large-scale distributed system) would impose too much overhead on the user. However, the resolver is now being relied upon to perform trusted operations — the user relies on the resolver to submit jobs securely to the trustworthy participants. Hence, a compromised resolver could potentially undermine the entire security model of a distributed system.

It would therefore make sense for the resolver software, especially the externally facing services, to be small and simple to minimise the chance of it containing any security vulnerability. The same idea applies to securing the blind analysis server. Formal methods can be used to design and implement these services with a high degree of assurance. For example, FADES (Formal Analysis and Design approach for Engineering Security) [99] integrates KAOS (Knowledge Acquisition in autOmated Specifications) with the B specification to generate security design specifications. A security requirements model built with KAOS is transformed into equivalent one in B, which is then refined to generate design specifications conforming to the original requirements. These procedures help software developers to preserve security properties and detect vulnerabilities early during requirements.

As a defence-in-depth measure, external services can be isolated in a separate compartment (e.g. using virtualization) to limit the impact of any vulnerability being exploited. Any modification made to the services themselves (e.g. by insiders) will be caught when the system is rebooted as the platform configurations (stored in the PCRs) would change; the private key sealed to this PCR value will not be accessible to decrypt any incoming secret data.

Assuming that a public key infrastructure is available for verifying the resolver's (or the blind analysis server's) identity, these security properties should be sufficient for the user to establish trust with the resolver (see assumption 1, 7.5.1). The trustworthiness of the configuration resolver can be verified using its attestation token (see Section 7.5.3 for details).

7.6.4 Job Delegation

In practice, the job recipient might delegate some parts of the job on to other participants — this is known as *job delegation*. In the Trusted Grid Architecture [57], the user is capable of verifying the service providers' platform configurations against a set of known good values ($good_U$). Using its job submission protocol, the user may also check to see if the service provider's list of known good values ($good_P$) — which specifies all the acceptable configurations of possible job delegatees — satisfy the condition $good_P \subseteq good_U$. If this condition is satisfied, the user submits the job to the provider knowing that the job will only be delegated to other service providers whose platform configurations also satisfy $good_U$. However, the main concern with this type of approach is that the burden of managing the whitelists ($good_P \subseteq good_U$) rests on the users and the service providers.

Although job delegation has not been considered in the proposed systems, the configuration resolver could be configured to verify the configurations of possible job delegatees before dispatching the job. Since the resolver already has access to all the trustworthy participants' platform configurations (configuration tokens), it could exchange several messages with the potential job recipient to determine whether all the possible job delegatees are also trustworthy. This would involve the job recipient sending a list of identities of the possible delegatees to the resolver, and the resolver checking to see if all of the possible delegatees are

registered. The job would only be dispatched if all of the delegatees are also trustworthy.

The advantage of this approach is that the users and service providers would not have to worry about maintaining up-to-date whitelists, or attesting and verifying the trustworthiness of the possible job delegatees.

7.6.5 Relying on the Ethics Committee

In the distributed data system, the ethics committee defines the privacy policies for different types of analyses supported by the analysis server. This seems more practical than relying on the data owners to figure out their own sticky policies when it is not known which analyses might be performed. Moreover, it would be difficult to reconcile and make sense of such policies collected from different data sources.

7.7 Chapter Summary

Two types of distributed systems have been described in this chapter based on the generalised configuration resolver, which, in both designs, is responsible for managing a token repository of trustworthy participants. The resolver ensures that jobs are distributed to only those considered trustworthy and executed in protected, verifiable environments. The next chapter will evaluate these distributed systems as well as the log generation and reconciliation infrastructure (see Chapters 4 and 5) against their original security requirements.

Chapter 8

Evaluation

The potential drawbacks of the proposed systems have been discussed previously in Sections 4.3, 5.4 and 7.6. This chapter aims to evaluate how well the systems satisfy their *security requirements*. The applicability and interoperability will then be discussed through integration with the original use cases and existing distributed systems.

Sections 8.1 evaluates security of the log generation and reconciliation infrastructure against the original security requirements and the related threats. Its interoperability is evaluated though integration with the use cases discussed earlier. Similarly, Section 8.2 evaluates security of the distributed systems against the generalised requirements. Their interoperability is evaluated through integration with existing grid and cloud systems.

8.1 Log Generation and Reconciliation

8.1.1 Success Criteria: Requirements

Eight key security requirements for trustworthy log generation and reconciliation were identified in Section 3.3. This section evaluates the security properties of the proposed system against these requirements and the relevant threats (as summarised in Section 3.2.6), and to what extent they have been fulfilled.

8.1.1.1 Involuntary Log Generation

This requirement states that the logs should be generated independently of the applications or operating systems. In the proposed system, the log transit runs

inside an isolated, privileged virtual machine to intercept I/O events of interest and generate log records involuntarily. The log transit operates independently of the guest virtual machines, ensuring that it is always available for log generation. This isolation prevents intruders or rogue insiders from misconfiguring or compromising the log transit (threats 3,4 and 6 from Section 3.2.6)

The requirement also states that the security decisions made by the applications should be captured as trustworthy a manner as possible. The log transit identifies the security decisions being logged (which are ordinary disk write requests) using common logging expressions. These are reformatted and stored using the protected storage mechanisms. Since their integrity largely depends on the state of the applications (log triggers), these are assigned with a lower trust level.

8.1.1.2 Protected Log Storage

This requirement specifies that the log records should have integrity and confidentiality protected upon storage. The logging system provides secure log storage mechanisms to safeguard the logs against unauthorised access. A hash is generated for each log record and signed with a TPM key sealed to the trusted state of the log transit and other trusted components. If the log transit or any one of the trusted components is compromised, this key will no longer be accessible for signing. The public half of the key can be used by external applications to validate the signature and compare the hash values. A valid signature and matching hash value verifies the log integrity and authenticity. This signature validation process is responsible for mitigating the 'deletion, modification, and arbitrary insertion' threat (threat 1) as discussed in Section 3.2.6.

Confidentiality is implemented by encrypting the log data with a symmetric encryption key, which is also sealed to the trusted state of the log transit. This effectively makes the log transit the only component capable of decrypting the log data. Untrusted software (including a modified log transit) running inside the privileged virtual machine will not be able to tamper with the logged data. In addition, virtual machine isolation prevents rogue virtual machines from reading the logs stored inside the dedicated disk or memory space of the privileged virtual machine. These mechanisms mitigate the 'unauthorised access' threat (threat 2).

8.1.1.3 Authorisation Policy Management

Inside the log-access virtual machine, the trusted authorisation policy management service enforces the log authorisation policies to control log access. As part of the middleware stack, its security configurations are always verified by the policy enforcement point before allowing a job to run. This is achieved by measuring the virtual machine image and verifying its integrity at runtime, before job execution. If the configuration is different, the private key (sealed to the original configuration) will not be accessible to decrypt the job secret. This guarantees that a securely configured policy management service always validates the user's access rights before executing the log access query (mitigating the 'unauthorised access' threat from Section 3.2.6).

8.1.1.4 Log Migration Service

This requirement specifies that the log migration service should be responsible for secure job submission and log transfer. In the proposed system, this externally facing service has two different roles:

1. operating inside the log access manager (log user system), it is responsible for selecting trustworthy logging systems, encrypting the job secrets, and dispatching the jobs to those selected;

2. inside the log access virtual machine (log owner's platform), it is responsible for scanning the log access query for attacks on the database, executing the query through the log transit, and sending back the encrypted log result.

The runtime verification of the log access virtual machine also checks the integrity of the log migration service. Again, if the configuration changes, the private key will not be accessible to decrypt the job secret. For both the log user and the log owner, this ensures that a trusted log migration service always controls the execution of the query and encryption of the accessed logs. The logs are encrypted with the user's public key for which the private half is protected inside the user's TPM. This prevents intruders from sniffing the logs that are transferred across the unprotected, public network (threat 2 from Section 3.2.6). The complex middleware services are no longer relied upon to perform trusted operations such as encryption, further mitigating the 'middleware compromise' threat (threat 7).

8.1.1.5 Protected Execution Environment

A configuration token, downloaded from the configuration resolver, contains the participant's public key and credential. The private half is sealed to the PCR values corresponding to the trusted computing base and the log access virtual machine image. The trustworthiness of these two reported values are verified against a whitelist, and the authenticity of the sealing property is checked by validating the public key credential. A log access job, encrypted using this public key, is then dispatched to the trustworthy logging platform. The sealed key mechanism ensures that the job is processed inside a virtual machine launched using the verified image files.

The runtime verification mechanism detects any unauthorised modification of the image files — altered files will result in a different PCR value, and the private key will no longer be accessible for decrypting the job secret. Both the user and the participant are informed when modifications are detected. As a result, the job always runs inside a protected execution environment, free from any unauthorised interference. From the participant's perspective, virtual machine isolation limits the impact of attacks performed by a rogue job.

8.1.1.6 Log Reconciliation Service

The log owners need to be assured that their logs will be used in a protected environment without modification. The log owner's policy enforcement point verifies the security configurations of the user's log access manager (including the reconciliation service) while authenticating the job. This ensures that only those dispatched from a securely configured log access manager is authenticated to access the logs and log privacy policy.

When all the log results arrive at the user system, the reconciliation service processes the collected logs into meaningful audit trails. During this process, the log privacy policies (defined by the log owners) are enforced to hide private information from the audit trails. Virtual machine isolation protects the raw data and the privacy policies from other rogue virtual machines. These mechanisms are responsible for mitigating the 'unauthorised access' and 'authorisation violation' threats from Section 3.2.6.

8.1.1.7 Blind Analysis of the Logs

The log privacy policies are included as part of the log results. Attestation of the log reconciliation service is sufficient to guarantee that the policies will be enforced correctly. Only the processed, privacy protected audit trails — still sufficient for analysis — are returned to the user. The user only sees their application communicating with the log access manager and the processed audit trails. The raw data and the privacy policies are never disclosed to the user unless the log owners allow it.

8.1.1.8 Surviving Denial-of-Service Attacks

A rogue virtual machine sitting between the log access virtual machine and the log transit could block all the log access requests and reduce availability. Realtime applications that rely on timely data feeds would suffer most from this type of attack. This threat is mitigated by establishing a dedicated communication channel between the log transit (Domain-0) and the log access virtual machine. The shared memory mechanisms of Xen are used to set up this channel. Other virtual machines will not be able to access the shared memory and interfere with the log access requests.

An intruder could also perform denial-of-service attacks on the external log migration services (see Table 4.1). The system deals with this problem by launching an independent log access virtual machine for each job, rather than relying on a single virtual machine to handle all requests. This means even if one virtual machine becomes a target for an attack, the availability of other virtual machines will not be affected. These security mechanisms are responsible for mitigating the 'denial-of-service' threat as discussed in Section 3.2.6.

8.1.2 Integration with the Use Cases

Two use cases are selected from Section 3.2 and integrated with the log generation and reconciliation infrastructure. This example integration demonstrates how the original workflow would change with the new components in place, and how these components can be used in real systems to enable trustworthy audit and logging.

8.1.2.1 Recording Service Requests and Responses

In the use case described in Section 3.2.3, the client sends a service request to the service provider expecting the response to comply with the service-level agreement. During this process, the service request and response details are logged at both the client and service provider platforms.

Imagine that the client and the service provider platforms now have the trustworthy logging system installed — implying a fully virtualized platform where the log transit intercepts I/O events and generates logs involuntarily. Legacy software would run inside guest virtual machines without any change. The user would use one of these programs to make a service request to the external request handler (running on a separate virtual machine). The request handler, in turn, would submit the request to the service provider's external response handler.

The request handler would use its virtual network interface to communicate with the corresponding back-end interface in the monitor virtual machine (see Section 4.2.3). This back-end would communicate with the physical network interface card (via the native device driver) to submit the service request to the service provider. During this process, the event channel would first inform the log transit about the service request. The log transit would read the request details from the shared memory, process the details into the standard log format, and store it through the protected log storage mechanism. Such logs would be generated independently from the guest applications or any privileged applications running inside the monitor virtual machine. If any important security decisions are being logged within the user applications, these (disk write operations) would also be identified by the log transit, and logged through the protected log storage.

When this request arrives at the service provider's platform, the response handler would forward it to an internal service (available in a guest virtual machine) to perform some processing and generate a response. While the response is being generated, the log transit would intercept all I/O transactions triggered and record events of interest. For instance, the usage of computational resources or disk spaces could be logged. In consequence, an accurate record of the full execution of the requested service would be generated.

The response handler would then send the response back to the client's original request handler, which in turn, would forward it to the user software. The service provider's network interface would be used to transmit the response, and

this transmission would be captured by the service provider's log transit. Similarly, this response would be received through the client's network interface, and the details would be logged by the client's log transit. As a result, a complete description of the service request and response would be recorded inside both the client and service provider's platforms.

8.1.2.2 Dynamic Access Control Policy Update

Section 3.2.2 describes a healthcare grid use case where the administrator at the GP practice monitors the access control policies (for patient data) being updated dynamically. Imagine that the log reconciliation infrastructure is installed in this healthcare grid, and the platform configurations of the participants (specialist clinics in this case) are available through the configuration resolver.

All user interactions with the GP system would be made through the external log migration service. An analysis tool, running on a separate virtual machine, would be available for the administrator to submit monitoring requests to the log migration service. Meanwhile, the log migration service would have been configured to automatically update the access control policies on an hourly basis, using the data access logs collected from the specialist clinic systems. All the information necessary for creating and dispatching log access jobs would have been specified during the installation stage. Such information includes the acceptable logging system configurations, user credentials, identities of the specialist clinic systems, and log access query.

The migration service would download the configuration tokens from the configuration resolver — tokens which match the identities of the selected specialist clinic systems. These tokens would be used to verify the trustworthiness of the clinic systems and their log access virtual machines. A set of log access jobs would be created for trustworthy clinic systems. Each job would contain the user credentials, log access query and its signature, job description, a nonce, and an attestation token representing the GP system. The job secret would be encrypted using the public key (obtained from the configuration token) of the target clinic system.

When the log access job arrives at the specialist clinic system, the policy enforcement point would first verify the trustworthiness of the GP system using the attestation token. It would then measure the log access virtual machine

image and reset the resettable PCR with the new measurement. Any modification detected would prevent access to the private key and the job secret. The signature of the query would then be validated to check whether the encrypted secret correlates with the attestation token.

If all of these security checks pass, a trustworthy virtual machine would be launched (using the verified image files) to process the decrypted job. The internal authorisation policy management service would first check whether the user is authorised to execute the query. If this condition is satisfied, the log migration service would execute the query through the log transit to access the log records and log privacy policy. In the example given in Section 3.1.1, this policy would restrict disclosure of the lung cancer status. Each log record would contain its hash and digital signature that can be used by the recipient migration service (GP system) to verify the log integrity and authenticity. The log migration service would generate a log result message, encrypt it with a symmetric session key (which would be encrypted using the GP system's public key), and send it back to the GP system. The GP system's original nonce would also be returned as part of this message.

The GP system's policy enforcement point would decrypt the returned session key using its sealed private key. Any modification of the trusted computing base, including the log reconciliation service, would prevent access to this private key and decryption of the message. If decryption is successful, the log records, log privacy policy and nonce would be forwarded to the log migration service. The returned nonce would be compared with the original, where a matching value would verify the integrity of the job execution environment. The migration service would then verify the authenticity and integrity of each log record by validating its signature, and comparing the hash with a computed hash of the log record.

The reconciliation service would then reconcile the logs collected from multiple specialist clinic systems, and update the access control policies according to what users have previously seen from these clinics. The log privacy policies would be enforced while generating a summary information for this policy update. The summary would describe how the policies for their patient data have been updated for different users. Only this privacy protected summary would be forwarded to the administrator for monitoring.

8.1.3 Observations

These two examples demonstrate how the proposed infrastructure could be deployed in real systems to facilitate trustworthy log generation and reconciliation. The problem identified in the original service-level agreement example (see Section 3.1.2) has been that the integrity and totality of the logs can not be guaranteed. The first integration shows how these properties could be ensured.

The logging system involuntarily captures all the essential service request and response details. The protected storage mechanisms ensure that the integrity and totality of the logs are protected. The sealed key mechanism ensures that a compromised log transit can not generate valid signatures or tamper with the logged data, further guaranteeing the log accuracy and integrity. These security enhancements would enable trustworthy reports to be filed upon violation of service-level agreements.

In the original healthcare grid example (see Section 3.1.1), the area of concern has been that the specialist clinics do not trust the GP practice to see their raw data. This prevents the development of audit-based dynamic access control mechanisms. The second integration shows how the log reconciliation infrastructure could be used to build this trust between the hospitals.

Before allowing the log access query to be executed, the specialist clinic's policy enforcement point verifies the security properties of the remote log reconciliation service. The properties checked are: (1) isolation from the rest of the platform, and (2) correct enforcement of the log privacy policies. These are sufficient to know that their logs will be confidentiality protected, and only the processed, anonymised information will be released to the administrator. Moreover, the logs and privacy policies are encrypted in a way that only a securely configured reconciliation service can ever read them. With this assurance, the specialist clinic could freely share their sensitive logs with the GP practice.

8.2 Trustworthy Distributed Systems

8.2.1 Success Criteria: Requirements

The generalised set of requirements presented in Section 7.3 is used as success criteria to evaluate security of two distributed systems proposed in Chapter 7.

8.2.1.1 Secure Job Submission

This requirement states that both the integrity and confidentiality of the job secrets should be protected. In both systems, there are two steps involved in the job submission process. Firstly, the user submits a job to the configuration resolver, encrypting it with the resolver's public key. The job factory (of the job security manager) verifies the trustworthiness of the resolver before encrypting the job secret. The resolver's private key — sealed to the resolver's trusted computing base — is protected by its TPM, so an intruder, or even a compromised resolver, would not be able to steal the job secret.

Secondly, using the job description, the resolver selects trustworthy participants that satisfy the user's service requirements. A job is created for each of those selected, and the job secret is now encrypted with the target participant's public key; the job description is extended with the host address. The encrypted job is then dispatched to the target participant via an untrusted middleware stack. Again, a rogue middleware stack, or any other intruder, would not be able to steal the job secret since the private key is protected by the participant's TPM.

8.2.1.2 Authorisation Policy Management

The runtime verification of integrity of the per-job virtual machine is intended to guarantee that a securely configured middleware stack, which includes the authorisation policy management service, correctly enforces the authorisation policies on the job. The job owner's credentials are evaluated against these policies to determine whether the job owner is authorised to run their code in the participant platform.

8.2.1.3 Trustworthy Execution Environment

The trustworthiness of the participant system and the per-job virtual machines it offers are verified when the participant first registers with the configuration resolver. This process involves the resolver's attestation service comparing the reported PCR event log with the whitelist of locally accepted platform configurations. Only those verified to be trustworthy are registered.

When a job arrives at the resolver, the job distribution service selects a participant suitable for the requested service, and dispatches the job encrypted with

the target participant's public key. The sealed key mechanism ensures that if the participant platform or the virtual machine image has been compromised, the private key will no longer be accessible to process the job further. This is intended to guarantee that a trustworthy virtual machine, that has all the required versions of software installed, executes unmodified code to produce accurate results. This is also attractive from the participant's perspective, since they can specify and verify the exact job execution environments that are allowed on their machine.

8.2.1.4 Job Isolation

This requirement specifies that the jobs should be isolated from each other as well as from the host. Attestation performed by the configuration resolver is sufficient to verify that a participant platform is running in a fully virtualized environment, and its virtual machine monitor is securely configured to provide strong isolation between the per-job virtual machines.

Then, the sealed key approach guarantees that the job is executed only if the target platform still holds these isolation properties. In a fully hardware virtualized environment, each job virtual machine would have its own dedicated memory and disk space (see Section 2.5). A rogue job would have to compromise both the virtual machine monitor and the privileged virtual machine to break this isolation.

8.2.1.5 Protecting the Results

This requires that the integrity and confidentiality of the results should be protected. In the proposed systems, virtual machine isolation guarantees that the code is executed free from any unauthorised interference, including attempts by a rogue job or administrator to subvert the results.

After executing the code, the result factory (running inside the per-job virtual machine) generates a secure result message encrypted with the job owner's symmetric session key — a key that is protected by the job owner's TPM. The job owner's nonce (obtained from the job secret) is also included as part of this message. This nonce is used to verify the integrity of the execution environment.

8.2.1.6 Digital Rights Management and Blind Data Analysis

In the distributed data system, the blind analysis server is responsible for releasing only the *processed* results to the end users and never the raw data. Through attestation, the data owner verifies the trustworthiness of the blind analysis server before executing the query and sending back their encrypted data.

Two important security properties are checked during this process: (1) the security configurations of the 'privacy preserving analysis tool' installed; and (2) the integrity of the data privacy policies. Verification of these properties is sufficient for the data owner to trust the analysis tool to enforce the privacy policies correctly upon processing the collected data. The final results are encrypted using the end user's session key that is protected by the TPM. The end user only accesses and performs analysis on this processed, anonymised data.

8.2.2 Integration with Grid and Cloud Systems

This section demonstrates how the proposed security mechanisms could be integrated with the UK National Grid Service [84] and Eucalyptus [30]. In doing so, it identifies some of the important practical and interoperability issues.

8.2.2.1 The National Grid Service

The National Grid Service [84] is a UK academic research grid, intended for production use of computational and data grid resources spanning multiple institutions across the country. The aim of the National Grid Service is to provide a reliable and trusted service using open, standards-based access to the distributed resources.

The grid consists of four core sites at Oxford, Manchester, Leeds, and STFC-AL, as well as five partner sites at Cardiff, Bristol, Lancaster, Westminister and Queens. Each site contributes to the provision of computational or data nodes. The nodes sitting on the core sites provide transparent access to the resources by using an identical middleware stack and similar filesystems, whereas the partner sites provide a more heterogeneous environment.

At each site, the Grid Resource Information Service publishes static and dynamic information about the service or resource availability using the GLUE Information Schema [105]. Information from all the sites is collected and aggregated

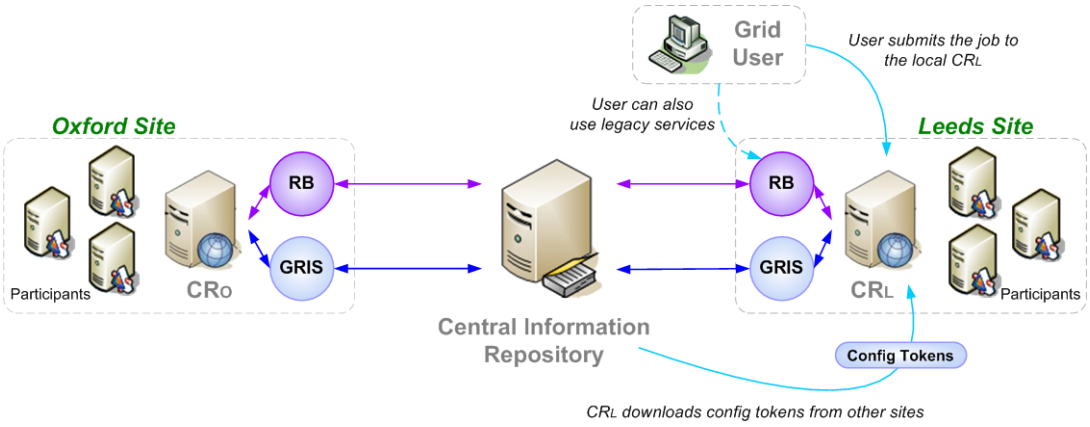


Figure 8.1: Integration with the National Grid Service

by the Berkeley Database Information Index system [40] (a central information repository), which holds information about all services and resources available in the grid. It queries the Grid Resource Information Service at each site to collect this information. The Lightweight Directory Access Protocol [66] is used for making the aggregated information available to the users.

As the first step of integration, the configuration resolver would be deployed at each site to publish a filtered list of configuration tokens (representing trustworthy participants) through the Grid Resource Information Service (see Figure 8.1). These tokens would have to be signed by the configuration resolver for authenticity and to indicate that these represent trustworthy participants.

The central information system would then query the Grid Resource Information Service at each site to collect these tokens and make them available to all the configuration resolvers. In this scenario, the Grid Resource Information Service would merely act as a proxy between the resolver and the central information system. The signatures of the tokens would be validated by the central information system before aggregating them. The resolvers would have to be authenticated at the central information system before being granted access to the aggregated tokens; verification of the resolvers' attestation tokens would be sufficient for this purpose. This integration would allow each site, through their own configuration resolver, to discover all the trustworthy nodes available across the entire grid.

Imagine a job submission scenario. A researcher, who wishes to run their job

in the National Grid Service, submits a job to the local configuration resolver. The resolver first downloads the configuration tokens (that match the service requirements) from the central information system using the Lightweight Directory Access Protocol. These tokens represent trustworthy participants available in other sites. It then iterates through each token and verifies the trustworthiness of the reported configurations by comparing the PCR values against the local whitelist. Only those with acceptable configurations will be selected and merged with the tokens from the local site. After this selection process, the resolver communicates with the local Resource Broker to discover their resource availability, and further filters the ones with free resources. Finally, the resolver encrypts the job with the selected participant's public key and dispatches it.

8.2.2.2 Eucalyptus

Eucalyptus [30] is an open-source software infrastructure for implementing cloud computing systems on clusters. It started off as a research project at the University of California, and is now maintained by Eucalyptus Systems [44]. Computational and storage infrastructures that are commonly available to academic research groups have been used to provide a framework that is modular and open to experimental studies.

In essence, Eucalyptus gives the end user a full control of the virtual machines used for executing their code. Amazon EC2's [98] SOAP and 'Query' have been emulated to provide the necessary interfaces to start, access, and terminate the job virtual machines. Currently, it supports virtual machines that run on Xen virtual machine monitor [89].

Eucalyptus consists of four high-level components, each of which has been implemented as a stand-alone web service (see Figure 8.2). The Node Controller controls the virtual machine instances that run on the host it is attached to. The Cluster Controller is a resource brokering service that communicates with the Node Controllers to gather information about their resource availability, and schedule virtual machine executions on the first Node Controller that has free resources. The Storage Controller implements Amazon's S3 interface, providing mechanisms to store and access virtual machine images and user data. The Cloud Controller provides the entry-point to the cloud system for end users and

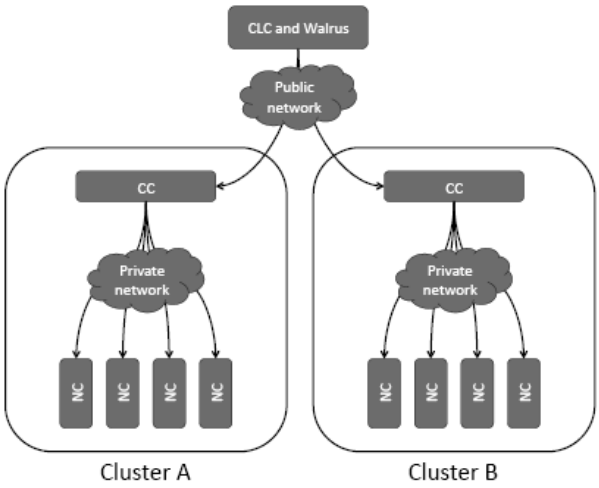


Figure 8.2: Eucalyptus Design (Figure 1 from [30])

administrators. It communicates with the Cluster Controllers to find out about the resource availability and make high-level scheduling decisions.

Imagine that Eucalyptus is used to construct a distributed, academic research system, where the resources are provided to the authenticated users at low (or zero) cost. This cloud system would then have same security problems as existing academic grid systems — the uncertainty of the security configurations of the hosts, and of the confidentiality and integrity of the results (see Section 7.1). Some of the proposed security mechanisms could be integrated into Eucalyptus to solve these problems.

In the modified design (see Figure 8.3), the configuration resolver operates inside the trusted Cloud Controller and manages a list of the acceptable known-good configurations of the hosts. It would be a simple whitelist containing trustworthy configurations of Xen virtual machine monitor, its privileged virtual machine (Domain-0), and a set of standardised security components running inside.

From the user's perspective, all other Eucalyptus components are untrusted. The user submits a job and a virtual machine instance (which is their trusted job execution environment) separately to the configuration resolver; a signed log, describing the applications installed on the virtual machine, is also submitted. The resolver first inspects the security configurations of this virtual machine by comparing the log with its security policies (possibly a virtual machine whitelist).

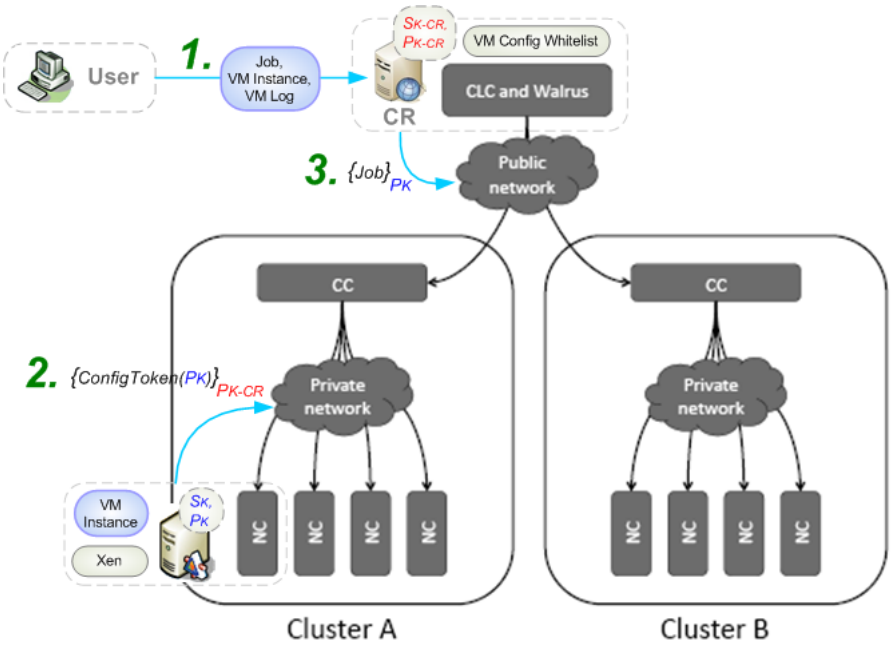


Figure 8.3: Integration with Eucalyptus (Modified Figure 1 from [30])

Suggesting implementation strategies or analysing overheads for managing such policies is beyond the scope of this example.

After the virtual machine is authenticated, the Cloud Controller makes requests to the Cluster Controllers to send an 'initiate execution environment' request to the Node Controller that has sufficient resources to host the virtual machine. The selected Node Controller forwards the virtual machine instance to Domain-0 of the host that in turn measures the virtual machine instance, and resets a resettable PCR with the new measurement. It then generates a key pair and seals the private half to the PCR values corresponding to its Xen configurations and the virtual machine instance. The public key and its credential are included as part of the the host's configuration token which is generated on the fly, and sent back to the resolver via the Cluster Controller.

The return of the configuration token indicates that the job execution environment is ready to be launched, and the selected host is expecting a job. The resolver's attestation service verifies the trustworthiness of this execution environment by comparing the PCR event log (obtained from the token) with the whitelist of secure Xen configurations and the hash of the original virtual

machine instance. If the environment is trustworthy, the resolver encrypts the job using the public key (from the token) of the selected host, and dispatches it through the public network.

The untrusted components — the Cluster Controller, Node Controller and various networks — will not be able to read the job secret since the private key is protected by the host's TPM. Moreover, if either the host's Xen configuration or the virtual machine instance has been modified, the private key will no longer be accessible to decrypt the job. If any modification is detected, the Cloud Controller will be informed, and it will search for another host with free resources. Once the job execution environment is launched successfully, the resolver informs the user about the selected host and the state of the virtual machine.

8.2.3 Observations

As has been observed several times, there would be a significant overhead involved in upgrading the participant systems to support trusted computing and virtualization. Various security virtual machines will have to be installed and the virtual machine monitor will have to be configured to manage these securely. Although this is a large change, the advantage of the discussed approach is that legacy components like the Grid Resource Information Service and the central information system can be used with only small modification.

Moreover, many existing cloud systems [11, 42], including Eucalyptus, already support virtualization and submission of job virtual machines. With the recent introduction of hardware support for virtual machine execution (see Section 2.5), it seems likely that future developments will also make use of virtualization. The administrative tasks involved in upgrading such systems would be much smaller.

Despite the security enhancements, the use of the configuration resolver will increase the number of messages being exchanged upon job submission. In the National Grid Service integration, the user submits a job to the local configuration resolver rather than to the Resource Broker. The resolver requests configuration tokens from the central information system, filters the trustworthy participants, and checks their resource availability through the Resource Broker. Once these checks are done, it encrypts the job with the selected participant's public key and submits the job on the user's behalf.

These extra message sends and cryptographic operations will affect the overall performance of job submission. However, for those wanting to submit performance critical jobs, the legacy services will still be available for use. Such jobs can be submitted directly through the local Resource Broker and skip all the trusted computing operations (see Figure 8.1). Usability will not be affected as much since the user relies on the resolver to carry out attestation and job submission.

The role of the configuration resolver is slightly different in the Eucalyptus integration. It no longer manages the token repository, rather, the token is generated on the fly by the selected host and sent back to the resolver. The resolver merely verifies the trustworthiness of the host using the token and dispatches the encrypted job if the host is trustworthy. The reason for this change is that the user submits their own virtual machine instance, and so the host's configuration token (which reflects on the state of this virtual machine) can only be generated after the host receives the virtual machine instance.

The main advantage of the discussed approach is that the user would only have to trust the Cloud Controller and the resolver running inside. The user would verify the identity and the trustworthiness of the Cloud Controller (through attestation) prior to job submission. If the Cloud Controller is trustworthy, then the user may submit their job knowing that its secret will be safeguarded from all other middleware components, and their virtual machine instance will be launched at a trustworthy host.

As a possible mechanism for authenticating the virtual machine instances, 'virtual machine level whitelisting' has been suggested. This would probably involve managing a list of acceptable baseline configurations, and performing a log based inspection on the changes made to each virtual machine. The challenge is that these virtual machines will be used for different applications, implying that different software will be installed (or removed) depending on their requirements. To keep track of these changes and detect undesirable configurations, a trustworthy logging system would need to be installed and monitor the virtual machines. The administrators should be informed about the changes that conflict with the cloud's security policies (application whitelists).

8.3 Chapter Summary

In this chapter, the proposed systems have been evaluated against their security requirements and related threats, demonstrating how well these requirements are satisfied. Their applicability and interoperability has also been discussed through example integration with the original use cases and existing cloud and grid systems. The next chapter will summarise the key contributions and conclude this thesis.

Chapter 9

Conclusion and Future Work

This chapter summarises the key contributions described within the thesis and considers potential areas of further work. Section 9.1 summarises the key contributions and their significance. Section 9.2 outlines the scope for future work. Finally, Section 9.3 concludes the thesis.

9.1 Summary of the Contributions

9.1.1 Audit and Logging Requirements

Secure management of logs in distributed systems is often considered a task of low priority. However, it becomes more critical when the logs have security properties in their own right. The thesis presents several use cases (see Section 3.2) where log integrity, confidentiality and availability are essential, and these properties need to be protected upon log generation and reconciliation. A threat and risk analysis, conducted on these use cases, shows how attackers might exploit potential security holes. The analysis also highlights the unique security challenges when managing logs in a distributed environment.

Trustworthy audit and logging requirements (see Section 3.3) identify the security properties that need to be ensured and the mechanisms required to mitigate the threats discussed earlier. These requirements also pin down the services, such as the log migration and reconciliation services, that would be required to develop trustworthy monitoring applications for distributed systems.

9.1.2 Involuntary Logging System

Operating systems and their applications are often relied upon to record security critical events and protect the logged data. The main problem with this approach is that a single bug in the application might be sufficient for an attacker to subvert the logging system and compromise the logged data (see Section 3.4.1). The thesis proposes a new 'involuntary logging' paradigm to tackle this problem.

The proposed system generates logs independently from guest applications via an isolated logging component (see Section 4.2.4). This component, referred to as the 'log transit', sits inside the privileged virtual machine to capture I/O events of interest and record them through the protected storage mechanisms. Integrity is implemented by storing a hash (and its signature) of every record and checking that the hash matches when the log record is retrieved. Confidentiality is guaranteed by encrypting and decrypting the log records as they are written and read from the disk.

The key advantage of this system is that the applications have no control over the log transit, and have no way of bypassing it. Moreover, the sealed key approach ensures that the signing key and encryption key will only be available to a securely configured log transit. Any modification of the trusted computing base or log transit will be detected, preventing a further generation of signed logs, or decryption of the logged data. The system satisfies both the 'involuntary log generation' and 'protected log storage' requirements (see Section 8.1.1).

9.1.3 Trustworthy Log Reconciliation

Many existing audit-based monitoring services are prone to compromise due to the lack of mechanisms for verifying the integrity and accuracy of the logs. In some applications the logs contain sensitive information, and without the necessary confidentiality guarantees, the log owners do not trust other sites to see their raw data. In order to bridge this security gap, the thesis proposes a log reconciliation infrastructure that would enable audit-based monitoring with strong guarantees of the log integrity and confidentiality.

Upon deployment of the infrastructure, each participant will be capable of generating and storing log data, and proving to the users that these logs are trustworthy. A directory service, the 'configuration resolver', is used to collect the configuration tokens from the participants and make them available to the

log users. The user system downloads configuration tokens to verify the trustworthiness of the logging systems installed on participant platforms; the log access jobs are dispatched to only those considered trustworthy. The job secrets are encrypted with the public keys of the selected participants. The sealed key mechanism protects the private halves by storing them on the target participants' TPMs. This ensures that the job secrets are safeguarded from untrusted middleware services.

Before granting access to the logs, the participant system verifies the security state of the user's log reconciliation service. A securely configured service would enforce the log privacy policies correctly and release only the anonymised results. Runtime verification of the log access virtual machine is intended to guarantee a strongly isolated, trustworthy job execution environment. Again, the sealed key mechanism ensures that only the integrity verified virtual machine can decrypt the job and execute the query. These security properties are sufficient for the user to know that the query has been executed without any unauthorised interference.

A matching hash value and a valid signature verify log integrity and the fact that logs have been generated by a trustworthy log transit. The reconciliation service allows the user to see the processed results for analysis, while still withholding access to privileged raw log data. This is referred to as 'blind log analysis' within the thesis. The security protocol has been formally verified using Casper [52] (see Section 5.3).

9.1.4 Implementation Strategies

The thesis also describes a prototype implementation of the features selected from the log reconciliation infrastructure (see Chapter 6). The prototype implementation provides strong evidence of feasibility for the trusted computing ideas (in particular, remote attestation and sealed key approach) proposed. Some of the key implemented features are:

1. an XML based whitelist, which demonstrates how a hierarchical structure could be used to store the known good PCR values effectively;

2. a remote attestation service, which compares the PCR event log (obtained from a configuration token) with the whitelist, and verifies the token authenticity to determine whether the platform is trustworthy;

3. a virtual machine verification mechanism, which measures the log-access virtual machine instance at runtime, and prevents access to the sealed private key (for job decryption) if the files have been modified.

The high-level class diagrams (see Section 6.2) provide a baseline for developing applications for remote attestation and sealing. These indicate that a 'three-tier structure' might be suitable. The bottom layer would comprise of the jTSS libraries that provide core TPM methods, the middle layer of internal services containing the logic for attestation and sealing, and the top layer of external web services. Figure 6.4 shows how these would fit into the Trusted Computing Group's software layering model [128]. The implementation details further indicate how these classes and methods could be orchestrated together.

9.1.5 Trustworthy Distributed Systems

In many disciplines, the models and data contain significant commercial or intellectual value. These often become targets for various attacks, usually associated with the compromise of the sensitive information or modification of the results (see Section 7.1). Such threats have discouraged researchers (in these sectors) from exploiting the full potential of distributed computing.

The thesis proposes two different types of trustworthy distributed systems (see Chapter 7) — one applicable for a computational system and the other for a distributed data system. Central to the distributed systems is the novel idea of the 'configuration resolver'. In both designs, this is responsible for filtering trustworthy participants and ensuring that the jobs are dispatched to only those considered trustworthy.

The job secrets (models and data) are encrypted with the public key of the trustworthy participant, and safely distributed over the unprotected network. The private half will only be accessible if the security configurations of the participant's trusted computing base and the virtual machine image remain unchanged. Runtime verification of the virtual machine integrity guarantees a trustworthy execution environment.

In the distributed data system, the configuration resolver operates inside the *blind analysis server*, and, together, they provide a trustworthy environment to run statistical analyses on the raw data without disclosing it to anyone. Attestation of the blind analysis server is sufficient to establish that only the processed,

anonymised results will be released to the user. The main advantages are that no information is lost through anonymisation (prior to release of data), and in consequence, analyses are carried out on more accurate datasets, producing high-quality results.

9.2 Future Work

9.2.1 Making Use of Trustworthy Logs

The thesis focuses more on describing how audit and logging works, rather than looking at what type of analysis could be performed using the logged data. As future work, it would be interesting to explore security applications that would benefit from having access to the trustworthy logs, and study how the involuntary logging mechanisms could be adapted to satisfy their requirements.

For example, cloud systems like Eucalyptus [30] could benefit from trustworthy logging. Since the cloud user is given a full control of the virtual machine instance, the user could install unverified applications or lower the security settings on the virtual machine. This is an obvious threat to the job and to the host. A reliable, runtime monitoring service could be used to monitor such events, enforce the security policies (defined by the cloud owner), and detect any undesirable changes.

Future work may look at integrating the involuntary logging system with Eucalyptus to monitor security setting changes and installation (or removal) of applications. It would report on any event that violates the security policies. Although, deciding exactly what events should be logged and how these security policies should be defined (and enforced) will be a challenging task.

9.2.2 Performance Analysis

Security benefits of trusted computing do not come without performance implications. All the cryptographic and virtual machine operations introduced by the logging system will affect the system's overall performance. As high performance is one of the key drivers behind the development of computational distributed systems, the new security mechanisms sit uneasily with these aspirations.

Despite several suggestions for improving performance (see Section 4.3.3), a more accurate assessment would be necessary to analyse the performance implications and devise enhancement strategies. Hence, future work should consider constructing a prototype implementation of the logging system, deploying it on an open cloud system, and measuring the performance overhead. A simple test would involve submitting identical jobs to two different hosts, one with the logging system installed and one without, and measuring the difference in execution time.

9.2.3 Generalising the Logging System

One of the drawbacks of the logging system is that it requires trusted virtualization and installation of several logging components. As the prototype work illustrates (see Chapter 6), trusted virtualization encompasses a wide range of software and hardware, so each participant would have to cope with a highly varied environment. Unless trusted virtualization becomes ubiquitous (which it might do in the long run), this upgrade overhead will serve as a strong barrier to uptake of the logging system.

As a possible short term solution, the logging components could run without trusted virtualization support. For instance, existing operating system kernels could be modified to intercept all I/O events before they reach the device drivers, or use special drivers, and generate logs independent to user applications.

Instead of relying on virtual machine isolation, operating system level isolation techniques like 'sandboxing' [17] could be used to isolate the jobs and user applications from the kernel. In scenarios where privilege escalation or insider attacks are less likely, sandboxing would provide sufficient isolation. It would prevent normal users (e.g. researchers) from accidentally, or even maliciously, changing the logging policies and settings. In systems where threats are less critical, this would be a suitable compromise between security and upgrading cost.

9.2.4 Trustworthy Distributed System Prototype

Despite ongoing research and prototyping efforts [57, 13, 87], no real 'trustworthy distributed system' has been constructed yet. As the first step towards developing a real system, the proposed security components could be implemented based on the ideas suggested in Chapter 6. These components could then be integrated

with existing distributed systems like the National Grid Service [84] using the approaches discussed in Section 8.2.2.1.

For instance, the central information system in the National Grid Service could be extended to collect configuration tokens from the Grid Resource Information Service at each site, and make them available to the attestation services. Participant platforms would have to support authenticated boot and provide configuration tokens upon domain registration.

While enhancing grid security, this work will also help uncover practicality, usability and performance issues, and demonstrate whether the security requirements (see Section 7.3) are fully satisfied by the proposed systems.

9.3 Conclusion

A wide range of research is conducted, archived, and reported in the digital economy. Different types of distributed systems have been deployed over the years to facilitate the collection and modeling of the dispersed data, or the sharing of the computational resources.

A problem arises, however, when the models or data have commercial value. They often become targets for attack, and may be copied or modified by malicious parties. In many scientific disciplines, the researchers — who care about confidentiality of the sensitive information, or integrity of the results — are unwilling to make full use of distributed computing due to these security issues [14]. The deployment of many kinds of distributed systems and associated threats makes provision of trustworthy audit and logging services necessary.

The thesis explores a number of use cases where the log integrity, confidentiality, and availability are essential, and proposes a log generation and reconciliation infrastructure that provides strong protection for these properties. The *involuntary logging system* generates logs independently from guest applications through a strongly isolated, integrity protected logging component. Upon installation of the complete infrastructure, each participant will be capable of generating logs and proving to others that the logs are trustworthy.

To bridge the 'trust gap' between the scientists' requirements and current technologies, the thesis proposes two types of distributed systems. Both provide the means to verify the security configurations of the participant platforms

through a central *configuration verification server*. The jobs are dispatched to only those considered trustworthy, and are guaranteed to run in protected execution environments.

Future work may look at implementing some of these security components, and integrating them with existing grid or cloud systems. As well as enhancing the system security, this work will help uncover practicality, usability and performance issues of the proposed trusted computing approaches.

Bibliography

[1] IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems. Parts 1-7. International Electrotechnical Commission, Switzerland, 1998-2005.

[2] Trusted Computing Group Backgrounder. `https://www.trustedcomputinggroup.org`, October 2006.

[3] A. C. Simpson and D. J. Power and M. A. Slaymaker and E. A. Politou. GIMI: Generic Infrastructure for Medical Informatics. In *Proceedings of the 18th IEEE Symposium on Computer-Based Medical Systems*, pages 564–566, June 2005.

[4] A. C. Simpson, D. J. Power, D. Russell, M. A. Slaymaker, G. Kouadri-Mostefaoui, X. Ma and G. Wilson. A healthcare-driven framework for facilitating the secure sharing of data across organisational boundaries. In T. Solomonides et al., editor, *HealthGrid 2008*, pages 3–12. IOS Press, 2008.

[5] A. Menon, A. Cox, W. Zwaenepoel. Optimizing Network Virtualization in Xen. In *USENIX Annual Technical Conference*, pages 15–28, Boston, MA, USA, May 2006. USENIX.

[6] A. Yasinsac, D. Wagner, M. Bishop, T. Baker, B. de Medeiros, G. Tyson, M. Shamos, and M. Burmester. Software Review and Security Analysis of the ES&S iVotronic 8.0.1.2 Voting Machine Firmware. `http://www.cs.berkeley.edu/~daw/papers/sarasota07.pdf`, February 2007.

[7] Abela, J., T. Debeaupuis. Universal Format for Logger Messages. `IETFInternetDraft,http://www.ietf.org/internet-drafts/draft-abela-ulm-05.txt`.

[8] Adam Slagell and Kiran Lakkaraju and Katherine Luo. FLAIM: A Multi-level Anonymization Framework for Computer and Network Logs. In *LISA '06: Proceedings of the 20th conference on Large Installation System Administration*, page 6, Berkeley, CA, USA, 2006. USENIX Association.

[9] Ahmad-Reza Sadeghi and Christian Stüble. Property-based Attestation for Computing Platforms: Caring About Properties, Not Mechanisms. In *NSPW '04: Proceedings of the 2004 workshop on New security paradigms*, pages 67–77, New York, NY, USA, 2004. ACM Press.

[10] Ahmad-Reza Sadeghi and Christian Stüble. Taming "Trusted Platforms" by Operating System Design. In *Information Security Applications*, **2908**, pages 1787–1801. Lecture Notes in Computer Science, 2004.

[11] Amazon Web Services. Amazon Elastic Compute Cloud (Amazon EC2). `http://aws.amazon.com/ec2/`.

[12] Andrew Cooper and Andrew Martin. Towards a Secure, Tamper-Proof Grid Platform. In *Sixth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'06)*, pages 373–380. IEEE Computer Society, May 2006.

[13] Andrew Cooper and Andrew Martin. Trusted Delegation for Grid Computing. In *The Second Workshop on Advances in Trusted Computing*, 2006.

[14] Andrew Martin and Po-Wah Yau. Grid security: Next steps. *Information Security Technical Report*, **12**(3):113–122, 2007.

[15] Andy Cooper. Presentation: Towards a Trusted Grid Architecture. `http://www.nesc.ac.uk/talks/902/Trusted_services_edinburghJul08.pdf`, July 2008.

[16] Anton Chuvakin and Gunnar Peterson. Logging in the Age of Web Services. *IEEE Security and Privacy*, **7**(3):82–85, 2009.

[17] Arel Cordero and David Wagner. Replayable voting machine audit logs. In *EVT'08: Proceedings of the conference on Electronic voting technology*, pages 1–14, Berkeley, CA, USA, 2008. USENIX Association.

[18] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*, pages 164–177, New York, NY, USA, 2003. ACM.

[19] Nik Bessis, editor. *Grid Technology for Maximizing Collaborative Decision Management and Support: Advancing Effective Virtual Organizations*, chapter Preface. Information Science Reference, May 2009.

[20] Brian Tierney and Dan Gunter. NetLogger: A Toolkit for Distributed System Performance Tuning and Debugging. In *IFIP/IEEE 8th International Symposium on Integrated Network Management*, pages 97–100, April 2003.

[21] Bruce Schneier. Schneier on security: Phone tapping in Greece. `http://www.schneier.com/blog/archives/2006/02/phone_tapping_i.html`, February 2006.

[22] Rob Byrom, Roney Cordenonsi, Linda Cornwall, Martin Craig, Abdeslem Djaoui, Alastair Duncan, and Steve Fisher. Apel: An implementation of grid accounting using r-gma. Technical report, CCLRC - Rutherford Appleton Laboratory, Queen Mary - University of London, 2005.

[23] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall International, 2004.

[24] Carl Ellison and Bruce Schneier. Ten Risks of PKI: What You're not Being Told about Public Key Infrastructure. *Computer Security Journal*, **16**(1):1–7, 2000.

[25] Haibo Chen, Jieyun Chen, Wenbo Mao, and Fei Yan. Daonity - grid security from two levels of virtualization. *Information Security Technical Report*, **12**(3):123 – 138, 2007.

[26] Chris Gourlay. Rogue trader Jerome Kerviel taken into police custody. `http://business.timesonline.co.uk/tol/business/industry_sectors/banking_and_finance/article3256630.ece`, January 2008.

[27] Christine M. O'Keefe. Privacy and the Use of Health Data - Reducing Disclosure Risk. *electronic Journal of Health Informatics*, **3**(1):e5, 2008.

[28] Chuliang Weng and Minglu Li and Xinda Lu. Grid resource management based on economic mechanisms. In *The Journal of Supercomputing*, **42**, pages 181–199. Springer Netherlands, May 2007.

[29] Daniel K. Gunter and Keith R. Jackson and David E. Konerding and Jason R. Lee and Brian L. Tierney. Essential grid workflow monitoring elements. In *The International Conference on Grid Computing and Applications*, 2005.

[30] Daniel Nurmi and Rich Wolski and Chris Grzegorczyk and Graziano Obertelli and Sunil Soman and Samia Youseff and Dmitrii Zagorodnov. The Eucalyptus Open-Source Cloud-Computing System. In *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 124–131, Washington, DC, USA, 2009. IEEE Computer Society.

[31] David C. H. Wallom and Anne E Trefethen. OxGrid, a campus grid for the University of Oxford. In *UK e-Science All Hands Meeting*, 2006.

[32] David Grawrock. *The Intel Safer Computing Initiative*, chapter 1–2, pages 3–31. Intel Press, 2006.

[33] David Grawrock. *The Intel Safer Computing Initiative*, chapter 8, page 139. Intel Press, 2006.

[34] David Grawrock. *Dynamics of a Trusted Platform*, chapter 13, page 209. Intel Press, February 2009.

[35] David Vecchio and Weide Zhang and Glenn Wasson and Marty Humphrey. Flexible and Secure Logging of Grid Data Access. In *7th IEEE/ACM International Conference on Grid Computing (Grid 2006)*, pages 80–87, Washington, DC, USA, October 2006. IEEE Computer Society.

[36] David W. Chadwick and Stijn F. Lievens. Enforcing "sticky" security policies throughout a distributed application. In *2008 workshop on Middleware security*, pages 1–6, New York, NY, USA, 2008. ACM.

[37] Carlos de Alfonso, Miguel Caballer, José V. Carrión, and Vicente Hernández. Distributed general logging architecture for grid environments. In *High Performance Computing for Computational Science - VECPAR 2006*, **4395/2007**, pages 589–600, 2007.

[38] De Roure, D. and Jennings, N.R. and Shadbolt, N.R. The Semantic Grid: Past, Present, and Future. *Proceedings of the IEEE*, **93**(3):669–681, March 2005.

[39] George T. Duncan and Robert W. Pearson. Enhancing Access to Microdata While Protecting Confidentiality. *Statistical Science*, **6**(3):219—232, 1991.

[40] EGEE Web. Berkeley database information index v5. `https://twiki.cern.ch/twiki//bin/view/EGEE/BDII`, November 2009.

[41] Elaine Shi and Adrian Perrig and Leendert van Doorn. BIND: A fine-grained attestation service for secure distributed systems. In *In Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 154–168. IEEE Computer Society, 2005.

[42] Enomaly. Enomaly — Product Overview. `http://www.enomaly.com/Product-Overview.419.0.html`, 2009.

[43] Ericsson AsiaPacificLab Australia Pty Ltd. Ericsson IMS User Manual. `http://www.quintessenz.at/doqs/000100003497/IMS_USER_MANUAL.pdf`, 2001.

[44] Eucalyptus Systems. Euclayptus Systems — About Us. `http://www.eucalyptus.com/about/story`, 2009.

[45] Renato J. Figueiredo, Peter A. Dinda, and Jose A. B. Fortes. A case for grid computing on virtual machines. In *23rd IEEE International Conference on Distributed Computing Systems (ICDCS'03)*. IEEE Computer Society, 2003.

[46] Formal Systems (Europe) Ltd. Failures-Divergences Refinement - FDR2 User Manual. http://www.fsel.com/documentation/fdr2/fdr2manual.pdf, June 2005.

[47] Ian Foster. What is the Grid? a three point checklist. *GRID Today*, **1**(6), July 2002.

[48] Foster, Ian and Kesselman, Carl and Tsudik, Gene and Tuecke, Steven. A security architecture for computational grids. In *CCS '98: Proceedings of the 5th ACM conference on Computer and communications security*, pages 83–92, New York, USA, 1998. ACM.

[49] Keir Fraser, Steven Hand, Rolf Neugebauer, Ian Pratt, Andrew Warfield, and Mark Williamson. Safe hardware access with the Xen virtual machine monitor. In *In Proceedings of the 1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS)*, 2004.

[50] Frederic Stumpf and Michael Benz and Martin Hermanowski and Claudia Eckert. An Approach to a Trustworthy System Architecture Using Virtualization. In *Autonomic and Trusted Computing*, pages 191–202. Lecture Notes in Computer Science, 2007.

[51] Richard Freeman. Medical records and public policy: the discursive (re)construction of the patient in Europe. In *Workshop 9: 'Policy, Discourse and Institutional Reform.* ECPR Joint Sessions of Workshops, April 2001.

[52] Gavin Lowe. A Compiler for the Analysis of Security Protocols. *Journal of Computer Security*, **6**(1–2):53–84, 1998.

[53] Geoffrey Strongin. Trusted computing using AMD "Pacifica" and "Presidio" secure virtual machine technology. *Information Security Technical Report*, **10**(2):120–132, 2005.

[54] Gunter, Dan and Tierney, Brian L. and Tull, Craig E. and Virmani, Vibha. On-Demand Grid Application Tuning and Debugging with the NetLogger Activation Service. In *GRID '03: Proceedings of the 4th International Workshop on Grid Computing.* IEEE Computer Society, 2003.

[55] B. Guttman and E. Roback. *Introduction to Computer Security: The NIST Handbook*, pages 211–222. NIST Special Publication, DIANE Publishing, 1995.

[56] Vivek Haldar, Deepak Chandra, and Michael Franz. Semantic remote attestation: A virtual machine directed approach to trusted computing. In *In Proceedings of the 3rd conference on Virtual Machine Research And Technology Symposium (VM '04).* USENIX Association, 2004.

[57] Hans Löhr and HariGovind V. Ramasamy and Ahmad-Reza Sadeghi. Enhancing Grid Security Using Trusted Virtualization. In *Autonomic and Trusted Computing*, pages 372–384. Lecture Notes in Computer Science, 2007.

[58] Hayley J. Fowler and Daniel Cooley and Stephan R. Sain and Milo Thurston. Detecting change in UK extreme precipitation using results from the climateprediction.net BBC climate change experiment. *Extremes*, **Online First**, February 2010.

[59] Hohmuth, Michael and Peter, Michael and Härtig, Hermann and Shapiro, Jonathan S. Reducing TCB size by using untrusted components: small kernels versus virtual-machine monitors. In *EW11: Proceedings of the 11th workshop on ACM SIGOPS European workshop*, page 22, New York, NY, USA, 2004. ACM.

[60] I. Foster and H.Kishimoto and A.Savva. The Open Grid Services Architecture, version 1.5. http://www.ogf.org/Public_Comment_Docs/Documents/Apr-2006/draft-ggf-ogsa-spec-1.5-008.pdf, 2006.

[61] IAIK. Trusted Computing for the Java(tm) Platform. `http://trustedjava.sourceforge.net/`, 2009.

[62] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*, chapter 2: Computational Grids. Morgan-Kaufman, 1999.

[63] Ian Foster and Carl Kesselman and Gene Tsudik and Steven Tuecke. A security architecture for computational grids. In *Proceedings of the 5th ACM conference on computer and communications security*, pages 83–92, New York, NY, USA, 1998. ACM.

[64] ISO/IEC. Information technology – Security techniques - Management of information and communications technology security - Part 1, ISO 13335-1:2004.

[65] ISO/IEC. Information Processing Systems - Open Systems Interconnection - Basic Reference Model - Part 2: Security Architecture, ISO 7498-2: 1989.

[66] J. Hodges and and R. Morgan. Lightweight Directory Access Protocol (v3): Technical Specification, 2002.

[67] A. Mejlholm J. Kloster, J. Kristensen. *Efficient Memory Sharing in Xen Virtual Machine Monitor*. Master's thesis, Aalborg University, January 2006.

[68] James Broberg and Srikumar Venugopal and Rajkumar Buyya. Market-oriented Grids and Utility Computing: The State-of-the-art and Future Directions. *Journal of Grid Computing*, **6**(3):255–276, September 2008.

[69] Bernhard Jansen, HariGovind V. Ramasamy, and Matthias Schunter. Flexible integrity protection and verification architecture for virtual machine monitors. In *Second Workshop on Advances in Trusted Computing*, 2006.

[70] Jesus Luna and Marios D. Dikaiakos and Theodoros Kyprianou and Angelos Bilas and Manolis Marazakis. Data Privacy considerations in Intensive Care Grids. In *Global Healthgrid: e-Science Meets Biomedical Informatics*, **138**, pages 178—187. IOS Press, 2008.

[71] Jon Maclaren Rizos and Jon Maclaren and Rizos Sakellariou and Krish T. Krishnakumar. Towards Service Level Agreement Based Scheduling on the Grid. In *Proceedings of the 2 nd European Across Grids Conference*, pages 100–102, 2004.

[72] Jun Ho Huh and John Lyle and Cornelius Namiluko and Andrew Martin. Application Whitelists in Virtual Organisations. *Future Generation Computer Systems*, 2009. Submitted.

[73] Karen Kent and Murugiah Souppaya. *Guide to Computer Security Log Management*. NIST Special Publication 800-92, September 2006.

[74] Katarzyna Keahey, Karl Doering, and Ian Foster. From sandbox to playground: Dynamic virtual environments in the grid. In *5th International Conference on Grid Computing (Grid 2004)*. IEEE Computer Society, 2004.

[75] Keith Adams and Ole Agesen. A comparison of software and hardware techniques for x86 virtualization. In *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems*, pages 2–13, New York, NY, USA, 2006. ACM.

[76] Lee Garber. Denial-of-Service Attacks Rip the Internet. *Computer*, **33**(4):12–17, 2000.

[77] Patrick Lincoln, Phillip Porras, and Vitally Shmatikov. Privacy-preserving sharing and correction of security alerts. In *13th conference on USENIX Security Symposium*, pages 17–17, 2004.

[78] Wenbo Mao, Fei Yan, and Chunrun Chen. Daonity: grid security with behaviour conformity from trusted computing. In Ari Juels, Gene Tsudik, Shouhuai Xu, and Moti Yung, editors, *STC*, pages 43—46. ACM, 2006.

[79] Martin Pirker. IAIK/OpenTC PrivacyCA documentation. `http://trustedjava.sourceforge.net/index.php?item=pca/apki`, March 2009.

[80] Melvin J. Anderson and Micha Moffie and Chris I. Dalton. Towards Trustworthy Virtualisation Environments: Xen Library OS Security Service Infrastructure. Technical report, HP Labs, April 2007.

[81] Michael Howard, Jon Pincus, and Jeannette M. Wing. *Measuring Relative Attack Surfaces*, chapter 8, pages 109–137. Springer US, December 2005.

[82] Mick Bauer. Paranoid penguin: syslog configuration. *Linux J.*, **2001**(92):10, 2001.

[83] Aarthi Nagarajan, Vijay Varadharajan, and Michael Hitchens. Trust management for trusted computing platforms in web services. In *STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing*, pages 58–62, New York, NY, USA, 2007. ACM.

[84] Neil Geddes. The National Grid Service of the UK. *e-Science and Grid Computing, International Conference on*, **0**:94, 2006.

[85] Hee-Khiang Ng, Quoc-Thuan Ho, Bu-Sung Lee, Dudy Lim, Yew-Soon Ong, and Wentong Cai. Nanyang campus inter-organization grid monitoring system. Technical report, Grid Operation and Training Center, School of Computer Engineering - Nanyang Technological University, 2005.

[86] Niels Provos and Markus Friedl and Peter Honeyman. Preventing Privilege Escalation. In *12th USENIX Security Symposium*, pages 231–242, Washington, DC, USA, 2003.

[87] Nuno Santos and Krishna P. Gummadi and Rodrigo Rodrigues. Towards Trusted Cloud. HotCloud '09 Workshop, June 2009.

[88] Oasis Access Control TC. XACML 2.0 Specification. `http://docs.oasis-open.org/xacml/2.0/`, 2005.

[89] University of Cambridge Computer Laboratory. The xen virtual machine monitor. `http://www.cl.cam.ac.uk/research/srg/netos/xen/`, 2008.

[90] Open Grid Services Architecture. Open Grid Services Architecture WG (OGSA-WG). `http://forge.gridforum.org/projects/ogsa-wg`, December 2003.

[91] Open TC. Corporate Computing at Home Instructions. `http://ftp.suse.com/pub/projects/opentc/`, July 2008.

[92] Paul England. Practical Techniques for Operating System Attestation. In *Trusted Computing - Challenges and Applications*, **4968/2008**, pages 1–13. Springer Berlin / Heidelberg, 2008.

[93] Paul Ruth and Xuxian Jiang and Dongyan Xu and Sebastien Goasguen. Virtual Distributed Environments in a Shared Infrastructure. *Computer*, **38**(5):63–69, 2005.

[94] D. J. Power, E. A. Politou, M. A. Slaymaker, and A. C. Simpson. Towards secure grid-enabled healthcare. *SOFTWARE PRACTICE AND EXPERIENCE*, 2002.

[95] Nguyen Anh Quynh and Yoshiyasu Takefuji. A central and secured logging data solution for xen virtual machine. In *24th IASTED International Multi-Conference PARALLEL AND DISTRIBUTED COMPUTING NETWORKS*, Innsbruck, Austria, February 2006.

[96] Rajkumar Buyya and David Abramson and Srikumar Venugopal. The Grid Economy. In *The IEEE 93*, **3**, pages 698–714, 2005.

[97] Reiner Sailer and Xiaolan Zhang and Trent Jaeger and Leendert van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, pages 16–16, Berkeley, CA, USA, 2004. USENIX Association.

[98] Reuven M. Lerner. At the forge: Amazon web services. *Linux J.*, **2006**(143):12, 2006.

[99] Riham Hassan and Shawn Bohner and Sherif El-Kassas and Michael Hinchey. Integrating Formal Analysis and Design to Preserve Security Properties. In *Hawaii International Conference on System Sciences*, **0**, pages 1–10, Los Alamitos, CA, USA, 2009. IEEE Computer Society.

[100] Ronald L. Rivest and John P. Wack. On the notion of "software independence" in voting systems. *Royal Society of London Philosophical Transactions Series A*, **366**:3759–3767, October 2008.

[101] Rosario M. Piro. DataGrid Accounting System - Basic concepts and current status. Workshop on e-Infrastructures, May 2005.

[102] Rosario M. Piro and Andrea Guarise and Albert Werbrouck. An Economy-based Accounting Infrastructure for the DataGrid. In *Fourth International Workshop on Grid Computing*, pages 202–204, November 2003.

[103] Ruoming Pang and Vern Paxson. A High-Level Programming Environment for Packet Trace Anonymization and Transformation. In *ACM SIGCOMM Conference*, pages 339–351, Germany, 2003.

[104] S. Ahmad and T. Taskaya-temizel and D. Cheng and L. Gillam and S. Ahmad and H. Traboulsi and J. Nankervis. Financial Information Grid - an ESRC e-Social Science Pilot. In *Proceedings of the Third UK eScience Programme All Hands Meeting*, pages 1–9, Nottingham, UK, 2004. EPSRC.

[105] S. Andreozzi, S. Burke, F. Ehm, L. Field, G. Galang, B. Konya, M. Litmaath, P. Millar, J. Navarro. GLUE Specification v. 2.0. `http://forge.gridforum.org/sf/docman/do/downloadDocument/projects.glue-wg/docman.root.drafts.archive/doc15023`, February 2009.

[106] Reiner Sailer, Trent Jaeger, Xiaolan Zhang, and Leendert van Doorn. Attestation-based policy enforcement for remote access. In *CCS '04: Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 308—317, New York, NY, USA, 2004. ACM.

[107] Shamal Faily and Ivan Fléchais. Analysing and Visualising Security and Usability in IRIS. In *Availability, Reliability and Security, 2010. ARES 10. Fifth International Conference on*, 2010. Accepted.

[108] Shane Balfe and Eimear Gallery and Chris J. Mitchell and Kenneth G. Paterson. Challenges for Trusted Computing. *IEEE Security and Privacy*, **6**(6):60–66, 2008.

[109] Shi, W. and Fryman, J.B. and Gu, G. and Lee, H.-H.S. and Zhang, Y. and Yang, J. InfoShield: a security architecture for protecting information usage in memory. In *High-Performance Computer Architecture, 2006. The Twelfth International Symposium on*, pages 222–231. IEEE Computer Society, 2006.

[110] Andrew Simpson, David Power, and Mark Slaymaker. On tracker attacks in health grids. In *2006 ACM Symposium on Applied Computing*, pages 209–216, 2006.

[111] James Skene, Allan Skene, Jason Crampton, and Wolfgang Emmerich. The monitorability of service-level agreements for application-service provision. In *6th International Workshop on Software and Performance*, pages 3–14, 2007.

[112] Srinivas Inguva and Eric Rescorla and Hovav Shacham and Dan S.Wallach. Source Code Review of the Hart InterCivic Voting System. `http://www.sos.ca.gov/elections/voting_systems/ttbr/Hart-source-public.pdf`, July 2007.

[113] Sriya Santhanam Pradheep and Sriya Santhanam and Pradheep Elango and Andrea Arpaci-dusseau and Miron Livny. Deploying Virtual Machines as Sandboxes for the Grid. In *In Second Workshop on Real, Large Distributed Systems (WORLDS 2005*, page 712, 2005.

[114] Vassilios Stathopoulos, Panayiotis Kotzanikolaou, and Emmanouil Magkos. A framework for secure and verifiable logging in public communication networks. In *CRITIS*, **4347** of *Lecture Notes in Computer Science*, pages 273–284. Springer, 2006.

[115] Stearley, J. Towards informatic analysis of syslogs. In *Cluster Computing, 2004 IEEE International Conference on*, pages 309–318. IEEE Computer Society, Sept. 2004.

[116] Stefan Berger and Ramón Cáceres and Kenneth A. Goldman and Ronald Perez and Reiner Sailer and Leendert van Doorn. vTPM: virtualizing the trusted platform module. In *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium*, pages 21–21, Berkeley, CA, USA, 2006. USENIX Association.

[117] Steven Hand. Architecture for Back-end Domains. `http://lists.xensource.com/archives/html/xen-devel/2004-10/msg00443.html`, 2004.

[118] Sugerman, Jeremy and Venkitachalam, Ganesh and Lim, Beng-Hong. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, pages 1–14, Berkeley, CA, USA, 2001. USENIX Association.

[119] Sujata Garera and Aviel D. Rubin. An independent audit framework for software dependent voting systems. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 256–265, New York, NY, USA, 2007. ACM.

[120] Sun Microsystems. SunGlassFish Enterprise Server 2.1 Application Deployment Guide. `http://docs.sun.com/app/docs/doc/820-4337`, January 2009.

[121] T. Van. Grid Stack: Security debrief (developerWorks: IBM's resource for developers). `http://www-128.ibm.com/developerworks/grid/library/gr-gridstack1/index.html`, 2005.

[122] Tadayoshi Kohno and Adam Stubblefield and Aviel D. Rubin and Dan S. Wallach. Analysis of an Electronic Voting System. *Security and Privacy, IEEE Symposium on*, **0**:27, 2004.

[123] Tal Garfinkel and Ben Pfaff and Mendel Rosenblum and Dan Boneh. Terra: A Virtual Machine-Based Platform for Trusted Computing. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP '03)*, pages 193–206, New York, NY, USA, October 2003. ACM Press.

[124] TCG Workgroup. TPM Main Specification Version 1.2. `http://www.trustedcomputinggroup.org/resources/tpm_main_specification`, 2003.

[125] The Financial Tiimes. SocGen Postmortem. `http://www.ft.com/cms/s/3/031b63e4-cbb3-11dc-97ff-000077b07658.html?nclick_check=1`, January 2008.

[126] Thomas Myer. Grid Watch: GGF and grid security. `http://www.tripledogs.com/downloads/samples/2004_GGF_and_grid_security.pdf`, 2004.

[127] Tom Maguire and David Snelling and Tim Banks. Web Services Service Group 1.2 (WS-ServiceGroup). `http://docs.oasis-open.org/wsrf/wsrf-ws_service_group-1.2-spec-os.pdf`, April 2006.

[128] Trusted Computing Group. TCG Specification Architecture Overview 1.4. `http://www.trustedcomputinggroup.org/resources/tcg_architecture_overview_version_14`, August 2007.

[129] Trusted Computing Group. TCG Infrastructure Working Group Architecture Part II - Integrity Management. `http://www.trustedcomputinggroup.org/resources/infrastructure_work_group_architecture_part_ii__integrity_management_version_10`, 2006 November.

[130] ULF-DIETRICH REIPS and STEFAN STIEGER. Scientific LogAnalyzer: A Web-based tool for analyses of server log files in psychological research. *Behavior Research Methods, Instruments and Computers*, **36**(2):304–311, 2004.

[131] Tobias Vejda, Ronald Toegl, Martin Pirker, and Thomas Winkler. Towards trust services for language-based virtual machines for grid computing. In Peter Lipp, Ahmad-Reza Sadeghi, and Klaus-Michael Koch, editors, *TRUST*, **4968** of *Lecture Notes in Computer Science*, pages 48—59. Springer, 2008.

[132] Dongbo Wang and Ai min Wang. Trust maintenance toward virtual computing environment in the grid service. In Yanchun Zhang, Ge Yu, Elisa Bertino, and Guandong Xu, editors, *APWeb*, **4976** of *Lecture Notes in Computer Science*, pages 166–177. Springer, 2008.

[133] Jinpeng Wei, L. Singaravelu, and C. Pu. A secure information flow architecture for web service platforms. *IEEE Transactions on Services Computing*, **1**(2):75–87, April-June 2008.

[134] Wenbo Mao and Andrew Martin and Hai Jin and Huanguo Zhang. Innovations for Grid Security from Trusted Computing. In *Security Protocols*, **5087/2009**, pages 132–149. Springer Berlin / Heidelberg, October 2009.

[135] Wensheng Xu and David Chadwick and Sassa Otenko. A PKI Based Secure Audit Web Server. In *Communication, Network, and Information Security*. ACTA Press, 2005.

[136] William Burr and John Kelsey and Rene Peralta and John Wack. Requiring Software Independence in VVSG 2007. `http://www.voteraction.org/files/Soft_Ind_VVSG2007-20061120.pdf`, November 2006.

[137] Windows Developer Centre. Windows Event Log. `http://msdn.microsoft.com/en-us/library/aa385780(VS.85).aspx`, 2009.

[138] Wu, K.-L. and Yu, P. S. and Ballman, A. SpeedTracer: a Web usage mining and analysis tool. *IBM Syst. J.*, **37**(1):89–105, 1997.

[139] Xen Source. Xen: Enterprise grade open source virtualization a xensource white paper. `http://xen.xensource.com/files/xensource_wp2.pdf`, 2005.

[140] Po-Wah Yau, Allan Tomlinson, Shane Balfe, and Eimear Gallery. Securing grid workflows with trusted computing. In Marian Bubak, G. Dick van Albada, Jack Dongarra, and Peter M. A. Sloot, editors, *ICCS (3)*, **5103** of *Lecture Notes in Computer Science*, pages 510—519. Springer, 2008.

[141] Yuri Demchenko and Leon Gommans and Cees de Laat and Bas Oudenaarde. Web Services and Grid Security Vulnerabilities and Threats Analysis and Model. In *GRID '05: Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, page 6, Washington, DC, USA, November 2005. IEEE Computer Society.

Appendix A

The Log Reconciliation Protocol

#Free Variables

```
u, o, r :  Agent
nu :  Nonce
q :  LogAccessQuery
log :  LogsAndPoliciesAccessed
kuo :  SessionKey
cu :  Credentials
Creds :  Agent -> Credentials
pcrqo, pcrqu :  PCRQuote
PCRQ : Agent -> PCRQuote
pko, pku :  PublicKey
PK : Agent -> PublicKey
SK : Agent -> SealedSecretKey
InverseKeys = (kuo, kuo), (PK, SK)
```

#Processes

```
CONFIGURATIONRESOLVER(r,o) knows PCRQ, PK
LOGUSER(u,o,r,q,nu) knows SK(u), Creds(u), PK, PCRQ
LOGOWNER(o,log,kuo) knows SK(o), PK, PCRQ, Creds
```

#Protocol Description

```
0.  u -> r :  o
1.  r -> u :  PCRQ(o) % pcrqo, PK(o) % pko
[ pcrqo == PCRQ(LogOwner) and pko == PK(LogOwner) ]
2.  u -> o :  {Creds(u) % cu, q, {q}{SK(u)}, nu}{pko % PK(o)}, PCRQ(u) % pcrqu,
PK(u) % pku
[ cu == Creds(LogUser) and pcrqu == PCRQ(LogUser) and pku == PK(LogUser) ]
3.    -> o :  u, cu % Creds(u), pcrqu % PCRQ(u)
4.  o -> u :  {log, nu}{kuo}, {o, u, kuo}{pku % PK(u)}
```

#Specification

```
StrongSecret(u, Creds(u), [o])
StrongSecret(u, q, [o])
StrongSecret(u, nu, [o])
StrongSecret(o, log, [u])
StrongSecret(o, kuo, [u])
```

```
Agreement(u, o, [q,nu])
Agreement(o, u, [q,log,kuo,nu])
```

#Actual variables

```
LogUser, LogOwner, ConfigurationResolver, Ivo :  Agent
Nu, Nm :  Nonce
Query :  LogAccessQuery
LogResult :  LogsAndPoliciesAccessed
Kuo :  SessionKey
InverseKeys = (Kuo, Kuo)
```

#Functions

```
symbolic PK, SK, PCRQ, Creds
```

#System

```
CONFIGURATIONRESOLVER(ConfigurationResolver, LogOwner)
LOGUSER(LogUser, LogOwner, ConfigurationResolver, Query, Nu)
LOGOWNER(LogOwner, LogResult, Kuo)
```

#Intruder Information

```
Intruder = Ivo
IntruderKnowledge = {LogUser, LogOwner, Ivo, Nm, ConfigurationResolver, PCRQ, PK,
Creds(Ivo), SK(Ivo)}
```