Trusted Logging for Grid Computing

Jun Ho Huh
Oxford University Computing Laboratory
Wolfson Building, Parks Road
Oxford, OX1 3QD
jun.ho.huh@comlab.ox.ac.uk

Andrew Martin
Oxford University Computing Laboratory
Wolfson Building, Parks Road
Oxford, OX1 3QD
andrew.martin@comlab.ox.ac.uk

Abstract

The rise of many kinds of grid systems, and associated security threats, makes very necessary the provision of trustworthy services for audit and logging. However, existing solutions tend to put little emphasis on the security of logging. We present a number of use cases where the logs have security properties in their own rights, and so the logs themselves are highly privileged: hence, these need to be integrity and confidentiality protected while being generated, accessed, reconciled and analysed with distributed services spanning across multiple administrative domains. We derive a common set of secure logging requirements to address the security gaps which exist between these use cases and existing solutions.

From the requirements, we propose a novel logging architecture for the grid based on Virtual Machine (VM) isolation and trusted computing capabilities: a small number of privileged driver VMs trigger all trusted logging requests and forward them to the secure logging service running within the log security manager. The logging service verifies the integrity of the log data and the security configurations of these driver VMs (log generators) before storing the logs.

1 INTRODUCTION

An audit trail, defined as a chronological record of system activities taken by identifiable and authenticated processes [7], plays an important role in problem determination systems; it may be used for detecting security violations and flaws in applications, for forensic examination, for proof of provenance, or for scientific record-keeping. In this paper, we present a number of grid use cases that rely on trustworthy logging services, where the logs themselves are highly privileged. These use cases have in common requirements upon protection of the *integrity* and the *confidentiality* of the privileged log information, and provision of *trustworthy analysis and reconciliation* of log data in an *interoperable* logging environment. Much of the existing research into distributed logging, however, fails to realise that building trustworthy audit and logging services requires protection of all of these security and interoperability requirements.

Foster [4] emphasises that the grid vision requires protocols, as well as interfaces and policies that are not only open and general-purpose, but also *standard*; likewise, it seems that various stakeholders, such as the Open Grid Services Architecture Working Group (OGSA-WG) within the Open Grid Forum (OGF) for example, 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 in such a context [5]. The problem is that distributed logging services and logs are inconsistent, and there is a need for a common approach to reconstruct the thread of work securely.

The remainder of the paper is organised as follows. In Section 2, we provide background information on secure logging in the grid. In Section 3, we present a number of motivational grid examples. Section 4 discusses the security gaps of existing logging solutions with respect to the motivational examples and their need for more secure logging services. Then, in Section 5, we present the secure logging requirements which address these security gaps. Mindful of the requirements described in Section 5, we describe a trusted logging architecture for the grid in Section 6. In Section 7 we focus exclusively on describing the secure logging infrastructure, based on the driver virtualisation technology of Xen. Finally, in Section 8 we discuss the contribution of this paper and the remaining work.

2 SECURE LOGGING IN A GRID ENVIRONMENT

2.1 Logging Services

Logging services facilitate the communication and recording of diagnostic audit trails, and provide means to

978-0-7695-3363-6/08 $25.00 © 2008 IEEE
DOI 10.1109/APTC.2008.9

help achieve a number of security-related objectives [7]:

- *reconstruction of events* — audit trails are used to reconstruct events when a problem occurs, and damages are assessed by analysing audit trails of system activity to identify how, when and why operations have stopped
- *intrusion detection* — again, to identify malicious attempts to penetrate a system and gain unauthorised access to resources
- *problem analysis* — status of processes running in critical applications or services can be monitored with real-time auditing
- *individual accountability* — proper user behaviour is promoted by logging users' activities, and advising that users are accountable for their traceable actions
- *dynamic access control* — researchers, working with a highly sensitive data grid for instance, may discover privileged information through information flow by accumulating history of accessed data. Audit trails enable a prevention-based countermeasure to this problem by allowing a system to scrutinise access control logs and to update its access control mechanisms according to what user has already seen from previous queries.

2.2 What is the Grid?

The notion of a *virtual organisation* runs commonly through many definitions of what constitutes a grid: many disparate logical and physical entities that span multiple administrative domains are orchestrated together as a single logical entity [13].

Foster [4] 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*

We consider an abstract version of the grid in Figure 1, that captures the essence of the definitions above: it is formed by participating resource providers coming together as a virtual organisation and uniting their individual databases as a single logical resource; each node (N) consists of external and internal services (ES, IS) where the virtualisation of data sources takes place; it also has its own

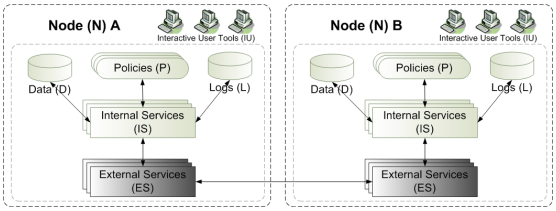

Figure 1. Abstract View of the Grid

local data (D) and logs (L), access control, security and logging policies (P) and interactive user tools (IU); standard and open external services enable communication between different nodes by managing middleware operations such as user authentication and data access. This abstract view of the grid will be used to motivate a use case described later in the paper.

The idea of grid nodes spanning *multiple administrative domains* makes the provision of trustworthy logging and audit services a difficult problem in the grid, where the services and the associated logs are stored in various formats, in different administrative domains [5, 11]: domains within a specific grid will have different standards, policies for retention and so forth. It is all *ad hoc*, which makes it extremely difficult to browse and analyse these logs.

It is not hard to imagine instances where the log data is highly sensitive, and only processed subsets may be released; nor is it hard to imagine scenarios where reconciliation of logs from different sources is needed, but neither trusts the other to see the raw data. Some of the consequent challenges of protecting integrity, confidentiality, accountability and availability of log information need to be resolved in order to securely match different log entries [19].

3 Motivational Examples

Despite the difficulties involved in managing logging services in the grid, the rise of many kinds of grid systems, and associated security threats, makes very necessary the provision of trustworthy services for audit and logging. These logs may be used for intrusion detection, for *post hoc* access control policy enforcement, for financial and business audit and due diligence, and so forth.

3.1 Healthcare Grid

An example application arises in the context of a Healthcare Grid. 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,

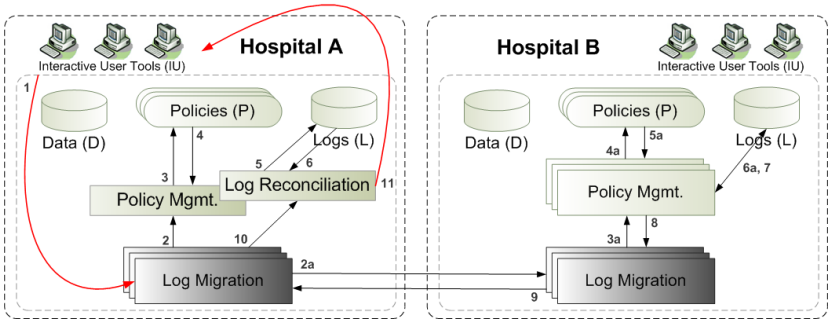

Figure 2. Use Case - Distributed Log Access in a Healthcare Grid

such as clinical trials of new treatments. 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.

For clinical data audited access will in many cases be much more appropriate than rigid access controls (since some essential accesses 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 must also be carefully logged and analysed, lest through multiple queries a researcher manages to reconstruct personal data about identified individuals [13].

These, then, are good candidate applications for the forms of secure logging services that will be described in the later sections. This paper has been motivated from concrete examples in this domain.

3.1.1 Cumulative Inference Attacks in Healthcare Grids and Distributed Audit Approach

It is not hard to imagine an undesirable scenario where a malicious researcher discovers personal information about patients from accumulating the history of accessed data. This means each site will need to record what the user has seen so far, and update its data access authorisation policies accordingly, to restrict the later access as a way of controlling such cumulative inference attacks. This is relatively easy to manage if the data is always accessed from a single node. However, it becomes much more difficult to manage in a distributed environment where an attacker could join a set of records returned from one site with other set of records that has been accessed previously from another site, and discovers sensitive information.

There is a need for a *distributed audit approach* to solve this problem where patterns of behaviour across multiple administrative domains will be detected by combining audit logs across those domains [15]. However, this approach will require each site to grant permissions to remote individuals to view its logs, which at the moment is very unlikely to happen due to the complications of negotiating trust in the grid [3]. Trustworthy log access, collection and reconciliation services designed with trusted computing capabilities may be applied in this scenario and enable users to take retrospective action in the event of undesired behaviour being detected; to allow a server to collect and reconcile distributed authorisation decisions (in the form of audit trails), and analyse which part of patient information user has already seen from other sites.

3.1.2 Use Case - Distributed Log Access in a Healthcare Grid

Given the space available, we present a use case at a high level of abstraction (refer to Figure 2), which is an essential representative of the requirements of that the secure logging should satisfy: a system developer, for the purpose of intrusion detection, wishes to query the distributed log data from a subset of hospitals that form the hospital grid. Each hospital (grid node) decides its own authorisation and restriction policies for logs.

In the first step, an auditing request is sent from an interactive user tool to an external log migration service at the local hospital A. This service then sends the request both to a local policy management service and to a number of external log migration services at other nodes. These external services then send the request to their own local policy management services which read the local log access and restriction policies, and use them to decide if the developer at hospital A is authorised to access the logs or to restrict the access to, for example, a certain set of attributes of the log data. If access is permitted, these external log migration services will return the logs to the log migration service at hospital A, which collects and sends the logs to the internal reconciliation service. The log reconciliation service processes these collected log data into useful information. This reconciled data is then returned to the system developer for

intrusion detection.

From this use case (and use cases from other domains), a threat analysis [9] has been conducted to identify the stakeholders, the assets and the threats to those assets. We do not propose to report the threats here, it would be impossible to do so in the space available, rather, we consider a selected number of key threats upon presenting the secure logging requirements in Section 5.

3.2 The Monitorability of Service-Level Agreements (SLAs)

The provision of Service-Level Agreements (SLAs) and ensuring their *monitorability* is another area that presents an instrumental example for secure logging in the grid. Skene et al [16] suggest a possibility of achieving the monitorability of SLAs with trusted computing capabilities.

Let us consider a case where the client receives no response for a service (for which they have entered into a SLA) within the agreed interval of time, and complains to the service provider that a timely response was not received. The service provider argues that no request was received, and produces a log of requests to defend its argument. There is no way for the client to find out the truth: the provider has delivered easily tampered evidence regarding this event which the client is incapable of monitoring. The problem with this type of SLA is that it is defined in terms of events that the client cannot directly monitor, and with this lack of monitorability the client must take the word of the provider with respect to the availability of the service.

This example highlights the importance of monitorability in SLAs, and hence the need for trustworthy logging and reporting services in which all stakeholders can trust another to record, report and analyse the actual transmission of requests and responses precisely. If the client is able to verify that a trustworthy logging software is running inside the service provider's platform and that the integrity of the log data is strongly protected (even from the log owners), then, the client is now capable of monitoring and placing conditions on any event, and constructing useful SLAs.

4 Relevant Work and a Gap Analysis

Having identified the key requirements of some of the grid applications that rely on secure logging, we are now in a position to present a gap analysis on existing distributed logging approaches.

The NetLogger Toolkit [18] provides client application libraries (C, Java, Python APIs) that enable one to generate the NetLogger log messages in a common format, for which it relies on the application developers to write the logging code. It heavily depends on the security mechanisms of the application to protect the integrity of the logging code running inside. An attacker could exploit the vulnerabilities of the parent application and manipulate the way logging requests are being triggered inside.

There is no attempt to protect the integrity and the confidentiality of the logs while they are transferred across the network, collected and processed at the central collection point. Distributed log access requests are processed without any authorisation policy enforcements, and hence anyone in the network can gain access to all log data stored in different sites.

Vecchio et al [20] have described a secure logging infrastructure for grid data access, that leverages and extends XACML to allow data owners to specify auditing policies, and record events of their interest. The main concern with introduction of such a flexibility in managing auditing policies is that a privileged user can delete or modify the auditing policies to manipulate the logging system. For example, the service provider could change its auditing policies so that the system only records events that would work against the client's penalty claims on a SLA.

Their auditing enforcement component sends logging information to a remote auditing service, which accepts the logs and records them in a local storage. These components are extremely weak upon trust management because there is no way for one component to verify the state of another. An attacker needs to compromise the remote auditing service, which is an ordinary .NET web service, to be able to sniff all logging information and tamper with them. Due to the inability of the auditing enforcement component to verify the state of the auditing service, it will continuously submit logging information to the compromised auditing service. Similarly, the local storage has no option but to store everything that is being forwarded from the compromised auditing service.

Quynh and Takefuji [14] have proposed a Xen-based logging solution called Xenlog which attempts to mitigate a number of security threats associated with sending the log data across the network stack, by running the logging logic and protecting the central log storage within the privileged VM called Domain-0.

A security flaw arises from the fact that the privileged Domain-0, which runs the logging service and contains the log filesystem, is always trusted to operate in a secure way, and hence there is no attempt to protect the confidentiality and the integrity upon log data transmission and storage. If a malicious grid job manages to penetrate Domain-0, their security model is undermined since it could now effortlessly access, modify or delete the logs from the filesystem, and read unencrypted log data off the shared memory. A defence-in-depth could be used to mitigate this type of attack: by encrypting and signing all log data flows and protecting the filesystem against unauthorised access.

There is no security control to verify the log generator (user domains) and the contents of the log data placed in the shared memory; this is because the logging service completely trusts the shared memory to deliver trustworthy log data. It is then, only a matter of hijacking an application inside a user domain which contains the logging code, for an attacker to compromise their security model.

Stathopoulos et al [17] have developed a framework for secure logging in public communication networks which aims to detect modification attacks against log files by generating signatures off-line and protecting the digital signatures within a trusted Regulator Authority (RA). Their framework provides a powerful mechanism for detecting unauthorised modifications to stored logs; however, it does not prevent a compromised server from fabricating the logs before being signed, or simply generating a new set of arbitrary logs.

5 Towards a Common Logging Infrastructure for the Grid

Motivational examples presented in Section 3 have in common requirements upon

- *Integrity* and accuracy of the log information — its generation, and (perhaps archival) storage; such concerns apply both to the individual logged events, and also to the totality of the logging data; the assembly or pattern of events.

- *Confidentiality* of the logged data — again, the individual log entries may contain sensitive information; the totality of the log data itself may be considerably more sensitive.

- *Trustworthy analysis* and *reconciliation* of log data — both the confidentiality and the integrity of the collected log data; upon its collection, transformation and visualisation; the totality of the reconciled results may also be very sensitive.

- *Interoperability* of the logging services, the logged data and the policies for access to these logs.

In the previous section we identified a number of security gaps from existing solutions that prevent the development of an interoperable trusted logging infrastructure which satisfies these security requirements. In this section we present the secure logging requirements at a high-level to fill these gaps, and to facilitate production and analysis of log data in the grid with strong guarantees of confidentiality and integrity.

5.1 Secure Logging Service

The most common threat on a logging service, as the gap analysis reveals, arises when an attacker compromises it, gains unauthorised access to all log data being forwarded from log generators (e.g. systems, software, middleware services), and modifies it to record fabricated data. To mitigate this type of attack, the logging service needs to be deployed independently from any parent application, on a strongly isolated compartment that provides robust memory protection. It needs to be a small and simple software designed to resist such attacks, and this should make attestation between the log generators and the logging service more feasible.

The logging service needs to be able to verify the software configurations of all log generators. Such a verification mechanism is required to protect the logging service from an attacker submitting arbitrary logs, performing denial of service attacks, and also to filter out untrustworthy logging requests from a compromised log generator.

5.2 Secure Log Storage

The security of the trusted logging architecture will only be as strong as the weakest link, and hence no matter how robust the security controls for the logging service may be, an attacker could directly penetrate the log storage to steal or to modify and delete large amount of the privileged logs, through spite or for profit. Such a threat has been realised through a security flaw in the Xen-based logging methodology [14] which completely trusts the privileged Domain-0 to always behave in a secure way, and puts no effort into safeguarding the essential security attributes of the stored logs. A malicious grid job needs to bypass the security controls of Domain-0 (VM isolation) to gain full access to the logged data and undermine their security model.

The log data need to be stored in an isolated, tamper-proof transparent storage with strong encryption, so that they are well protected against unauthorised access, modification or deletion throughout their lifetime.

5.3 Authorisation Policy Management

Just as authorisation policies for data access are enforced at each site to prevent users from gaining unauthorised access to the sensitive data or from performing inference attacks, distributed access requests for the privileged logs need to be controlled with similar authorisation policy enforcement at each site. This has been realised through the healthcare grid example, where a form of 'traffic flow' analysis may itself yield sensitive personal information about the patients, so the access logs themselves are highly privileged. Without a strong authorisation policy enforcement,

a malicious system developer could easily correlate distributed logs in the form of intrusion detection and carry out various inference attacks to discover personal information about patients.

5.4 Secure Log Migration Service

Due to the number of potential security vulnerabilities, complex grid middleware services can not be relied upon to perform trusted operations [3]. Instead the security controls required for safe log data transfer need to operate within the log migration service, to ensure that the integrity and the confidentiality of the logs are protected from the grid services compartment that manages data transfer.

There are also data flow encryption and signing requirements upon log access and transfer requests which need to be enforced in the log migration service. These are integral in preventing intruders from sniffing the privileged logs processed through the insecure grid middleware, and from launching man-in-the-middle type of attacks [9]. It is also possible for a log owner to deliver fabricated logs, as the client in the SLA use case might, in order to make false claims against the service provider. To provide a safeguard against such a threat, the log migration service needs to access the protected logs directly from the attested secure log storage, and this would be sufficient to guarantee the integrity of the logs.

5.5 Trustworthy Reconciliation Service

Motivational examples have a common set of requirements upon provision of a trustworthy reconciliation service. This would require each site to build trust with other sites, and grant permissions to view its logs. These sites (before granting permissions to access their logs) need to be assured that their privileged logs will be safeguarded from compromise throughout the whole reconciliation process.

Attestation of the reconciliation service should be sufficient to establish that robust security controls will be enforced upon *collection* and *reconciliation* of the logs — the integrity and the confidentiality of the collected logs need to be protected to prevent an intruder from stealing a large amount of these for profit.

5.6 Blind Analysis of the Logs

Imagine a scenario from the use case (refer to Section 3.1.2) where the log owners (hospitals) have agreed to share their logs for intrusion detection, but at the same time they are unwilling to let the system developer in hospital A to see the actual content of the sensitive logs, or only let part of it to be seen. This is to ensure that the developer can not perform inference attacks on the reconciled results to discover personal information about the patients.

In this scenario, log owners need to be assured that their logs will only be revealed to an extent that has been agreed, or stated in restriction policies: this requires a mechanism, possibly within the reconciliation service, to carry out a blind analysis of the collected logs so that developers only see the running application and the end results, which are just sufficient for them to analyse security problems.

6 A Trusted Logging Architecture

Based on the requirements described in the previous section we have developed a novel logging architecture for the grid (refer to Figure 3) using Virtual Machine (VM) isolation combined with trusted computing capabilities; also referred to as *trusted virtualisation*.

The Trusted Computing Group (TCG; formerly TCPA) [1] has developed a series of technologies based around a low-cost Trusted Platform Module (TPM) which helps to provide two novel capabilities [6]: a cryptographically strong identity and reporting mechanism for the platform, and a means to *measure* reliably a hash of the software loaded and run on the platform (from the BIOS upwards); such measurements are stored and retrieved from Platform Configuration Registers (PCRs) of a TPM. These provide the means to *seal* data so that it is available only to a particular platform state (a particular piece of software), and to undertake *remote attestation*: proving to a third party that (in the absence of hardware tampering) a remote device is in a particular software state; TPM-generated Attestation Identity Keys (AIKs) are used to sign PCR values and to prevent tracking of platforms. These trusted computing capabilities can be used in a virtualised environment where a physical host is segmented into strongly isolated compartments to make attestation feasible (with robust memory protection for security-critical software), and to limit the impact of any vulnerability in attested code.

In this section we consider the most important services and their security procedures as the very first step towards designing the trusted logging architecture. We do not claim that descriptions of these services and their operations are exhaustive at this stage: it is not our purpose to do so, but we do claim that these are the initial abstractions of the services of that trusted logging architecture is required to support. Our later work will entail breakdown of such abstraction into small clusters, and process of describing the syntax and semantics of each service within these clusters; it is in this spirit that we propose a generalised overview of the trusted logging services.

All log security functions are enforced by the *log security manager* VM, small code running inside *back-end driver* VMs and the *log analysis manager* VM; each of

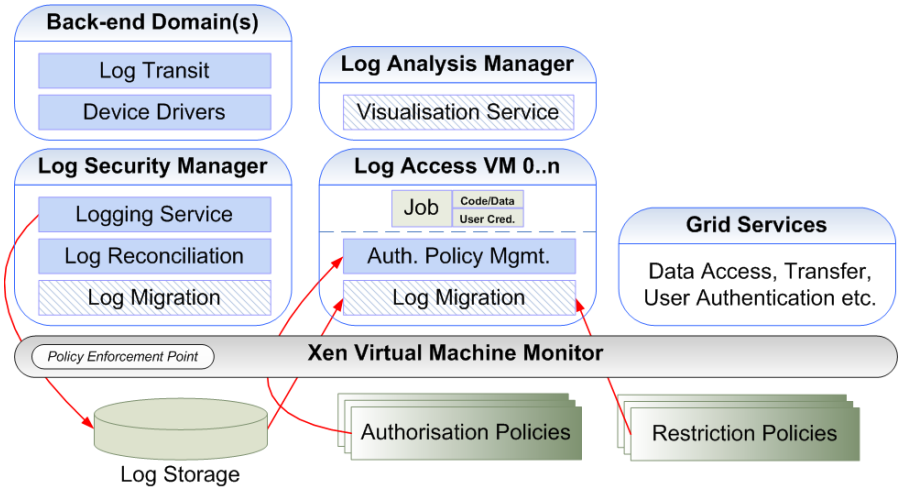

Figure 3. Abstract View of Trusted Logging Services

which has been designed to perform a small number of simple operations so that it can be compartmented with a high degree of certainty; and to minimise the number of security vulnerabilities that an attacker may try to exploit to compromise and modify these trusted services during run-time (after loading). Log owners, instead of being concerned about how logs might be compromised in complex grid middleware services can now focus on securing these simple compartments.

Attestation of these compartments, the Policy Enforcement Point (PEP) and the Virtual Machine Monitor (VMM) is sufficient for one to establish trust with a grid platform, and to be assured that its log security functions have not been subverted. The idea is to minimise any complexity involved in remote attestation with this relatively static and small trusted code-base, and hence solve the problem of managing attestation in a dynamic grid environment [3].

A small number of trusted back-end driver VMs are responsible for generating all logging requests upon use of device drivers. All other compartments must communicate with one of these privileged driver VMs to access the physical hardware. Inside these VMs, *the log transit component* collects important I/O requests and responses and submits encrypted logging requests to the log security manager. Applications and middleware services running in other compartments are no longer relied upon to generate trustworthy log data.

The log security manager VM performs a range of log security functions through the following services:

- The *logging service*, through a secure logging protocol, ensures that no adversaries can access or modify the log data transferring between the log transit components (driver VMs) and itself. It verifies the integrity of the log data before storing them in the log storage; and also ensures that the log transit and its parent driver VM is running in a trusted state, to prevent an attacker from submitting arbitrary logs and to filter out untrustworthy logging requests from a compromised driver VM.

- The *log reconciliation service* facilitates trustworthy reconciliation and transformation of the collected logs. It enables blind analysis of the logs, a key requirement identified in Section 5.6, by enforcing restriction policies on the associated logs during the reconciliation process. Consequently, the service ensures that the reconciled results only contain information for which the log owners have agreed to disclose as part of the post analysis results; log users will only have access to this processed information and not the raw data.

- The *log migration service* is an externally facing service which facilitates secure communication between VMs in one or more sites by enforcing security controls required for safe log and restriction policies transfer across the grid middleware. The external service is also deployed inside log access VMs to retrieve the logs and to send the results back securely to the remote log security manager. It protects the integrity and the confidentiality of both the logs and the restriction policies from the untrusted grid services compartment that manages data transfer.

End-user log analysis applications only have access to the externally facing *visualisation service* running inside the log analysis manager, which provides the minimal interface necessary for user applications to interactively analyse the processed log information; again, to reduce the number of

security holes that might be exploited by an attacker during run-time. It accepts the reconciled results from the log migration service (log security manager) and simply helps one to visualise them; the raw data therefore never leaves the log security manager and is completely hidden from end users.

A scheduler within the PEP executes a grid job (for collecting logs) in a per-user log access VM with trusted middleware services, security and software configurations specified by the user. The second trusted service that runs in these log access VMs is the *authorisation policy management service* which evaluates authorisation policies upon remote log access requests, and enforces limited access to the log data according to specified conditions. This prevents an attacker from gaining unauthorised access to the privileged logs or from performing inference attacks (Section 5.3).

- from the log owners' perspective, this approach ensures that the grid job verification, authorisation policy enforcements and access for logs are always governed by the trusted middleware services; logs are confidentiality protected upon migration with strong encryption; even if the job is malicious, it would have to compromise the VMM, which should be designed to resist such attacks.

- from the users' perspective, their code is running unmodified, on a trusted VM where the logs are only accessible via trusted middleware services; any arbitrary alterations or fabrications of the logs (e.g. never having run the correct code) will be detected when the user remotely attests the state of the log access VMs to verify the correctness and integrity of the logs that have been collected from multiple domains; strong separation between grid jobs and other compartments prevents a malicious VM from stealing sensitive code and data.

7 Secure Logging Infrastructure with Xen Virtual Machine Monitor

In this section we narrow down our focus on the logging service (log security manager) and the privileged back-end driver VMs: to present a secure logging infrastructure based on driver virtualisation operations of Xen, that provides strong guarantees of integrity of the log information upon its generation and storage.

7.1 The Xen Virtual Machine Monitor

First, we explore features of Xen Virtual Machine Monitor (VMM) [12] that are important in understanding the secure logging infrastructure.

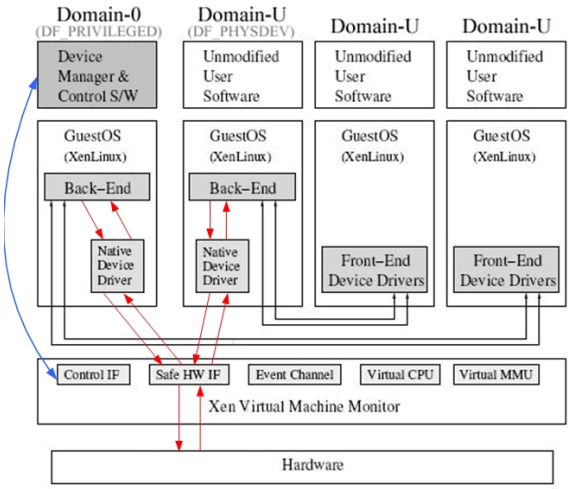

Figure 4. Xen Architecture (Adapted from [10])

Xen is a thin layer of software operating on top of the hardware to enable virtual machine abstraction that is almost the same as the underlying hardware [12]. As Figure 4 shows, the Xen VMM runs on top of the bare hardware and controls the way VMs, also known as domains in Xen, access the hardware and peripherals. A highly privileged VM, called Domain-0 is created at boot time and it serves to manage other user domains. This initial domain is responsible for hosting the application-level management software.

The *Safe Hardware Interface* (Safe HW IF) 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 VM isolation. In Xen, events replace hardware interrupts where the *Event Channel* is used to transfer asynchronous events to a VM.

The *Device Manager* runs inside Domain-0 to bootstrap the device drivers and broadcast driver availability to the OSs and export configuration and control interfaces. A *Native Device Driver* is a normal device driver which runs in a privileged VM that has access to the physical hardware. Each driver is restricted by an I/O space of the Safe HW IF so the damage a faulting device can do to others is limited. A *Front-end Device Driver* can only access a device through a back-end driver. A back-end driver receives I/O requests from a front-end driver then verifies it to make sure they are secure. Once verified, they are passed on to the real device hardware. *Back-end Drivers* usually run inside Domain-0 since they need to be in a VM which has access to the real device hardware. However, they may also run in a VM which has been given a special physical device

privilege (DF-PHYSDEV) flag. Once the kernel completes an I/O, back-end driver notifies front-end driver through an event channel that data is pending where data is transferred by use of shared memory.

7.2 Driver Virtualisation in Xen (Virtual Device Management)

In this section we explain in detail how driver virtualisations work in Xen because its operation is the key element in which the logging services are designed upon.

A domain may have two capabilities [8]: full administrative privilege (DF_PRIVILEGED) and physical device privilege (DF_PHYSDEV). DF_PRIVILEGED allows full access to memory and to all hypervisor operations which include create, inspect or destroy other domains. DF_PHYSDEV allows more restricted access. The idea is to give full administrative privilege only to Domain-0 or to a Domain-0 replacement whereas physical device privilege is intended for any back-end domain; DF_PHYSDEV alone will suffice for a back-end domain. DF_PRIVILEGED is given to Domain-0 as expected when a first domain is created. The privileged pcidev_access hypercall can then be used to configure DF_PHYSDEV in a back-end domain.

Each domain has two rings [8], one for sending requests and one for receiving responses. These two rings basically form the inter-domain communication mechanism which allow transfer of information between, for example, a client and a back-end domain. The component of the client domain involved in the communication is the front-end device driver and the component of the back-end domain involved is the back-end device driver. Both of these drivers are virtual device drivers and do not actually communicate with physical devices. They are just two ends of an inter-domain communication mechanism which allow, for example, a client to send a request 'read block 50 from device sda2 into a buffer at 0x5ac102'. The physical hardware is accessed through native device drivers running inside privileged back-end domains. These access the hardware in almost the same way as they normally do, by reading and writing to the memory mapping bits of the PCI address space and receiving interrupts.

Consider a control flow of a process running inside a client domain which reads a file [8].

1. Client process invokes read() syscall.
2. Client kernel, Virtual File System (VFS) layer invokes front-end driver for data access.
3. Front-end driver sends a message to the back-end domain with corresponding back-end driver, through I/O channel.
4. Back-end driver receives this message via I/O channel and verifies the I/O request.
5. Back-end driver forwards the request to corresponding native device driver which in turn forwards it to the physical device.
6. The physical device returns the data to native device driver.
7. Native device driver returns the data to back-end driver.
8. Back-end driver places response on I/O channel which is picked up by client's front-end driver.
9. The data is passed up to VFS layer and is returned to the client process.

7.3 Network Virtualisation Example in Xen

Xen provides each client domain with a set of virtual network interfaces to carry out all network communications [2]. For each virtual interface in a client domain a corresponding back-end interface is created in the back-end domain, which acts as the proxy for that virtual interface. The virtual and back-end interfaces communicate through an I/O channel which implements a zero-copy data transfer methodology for exchanging packets by remapping the physical page with the packet into the target domain.

All back-end network drivers in the back-end domain are connected to the native device drivers through a virtual network bridge. Only native device drivers communicate with the physical Network Interface Card (NIC) and send back responses to back-end drivers. Thus the combination of the I/O channel and virtual network bridge forms the communication path between a virtual and a physical interface.

7.4 Secure Logging Infrastructure

In Xen only Domain-0 and other driver domains with physical device privileges may access the physical hardware. As illustrated in the previous control flow example, all data access requests go through one specific back-end domain which runs both back-end and native device drivers for the physical hard-drive. Similarly, all network requests go through back-end and native device drivers running on a single back-end domain which has access to the physical NIC.

As a result of such driver virtualisation characteristics, there will always be a single point of access for all physical devices being controlled under Xen VMM, and this single access point is where the *log transit* component (that

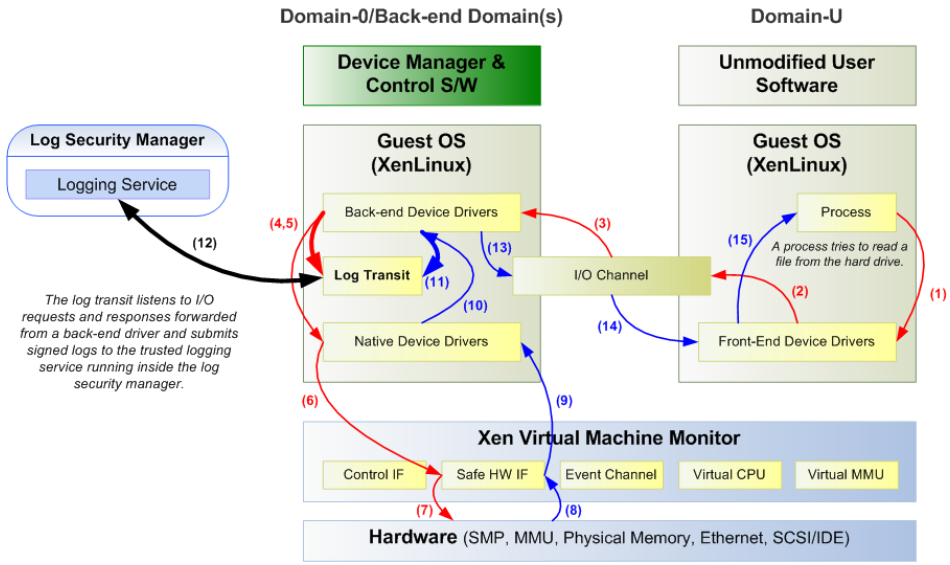

Figure 5. Secure Logging Architecture with Xen Virtual Machine Monitor

triggers logging requests) has been be deployed. The reason for this is that log generator verification becomes much more manageable from the logging service's point of view, which now only needs to attest and verify a small number of back-end domains in order to establish a complete trust chain with all log generators. This satisfies one of the secure logging service requirements identified in Section 5.1: to be able to verify all trusted log generators.

The movement of all trusted logging triggers to the back-end domains implies that only the privileged domains have control over when and how the trusted logs (which will be classified differently from logs generated via other sources) are triggered through the log transit component; as a result, the log transit components operate within a simple and safe environment offered by these privileged domains, as oppose to running inside a complex, and hence relatively insecure user domains which accommodate untrusted middleware services. This new logging paradigm solves the problems identified with NetLogger Toolkit [18], because applications (with potential security vulnerabilities) are no longer relied upon to perform trusted logging operations.

The log security manager VM is responsible for providing metrics about its current security configurations to the remote back-end domains. It generates an Attestation Token (AT) in the format below and issues the token out to all back-end domains.

$$AT = (PCR_{AIKS(LS)}, cred_{AIK(LS)}(PK), cred(AIK))$$

The token contains the PCR representation of the current state of the log security manager ($PCR_{AIKS(LS)}$), the credential of the *sealed* key ($cred_{AIK(LS)}(PK)$), and the AIK credential ($cred(AIK)$) that has been used to sign both the PCR value and the $cred_{AIK(LS)}(PK)$. A back-end domain compares this PCR value with the known-trusted value for a securely configured log security manager; if the values match, the back-end domain can be confident that the remote logging service is configured in a trusted state.

Correct verification of the $cred_{AIK(LS)}(PK)$ assures that the private key (SK) has been sealed to the reported PCR value, and it will only be available to the logging service if the log security manager is still running in the reported state. The use of the PK is explained a bit further on in this section.

We propose a preliminary secure logging infrastructure based on these concepts, where logs are triggered from the log transit component connected to a back-end driver inside a back-end domain. A back-end driver is responsible for verifying I/O requests received from a front-end driver, to ensure that they are secure. The log transit listens to these verified I/O requests and responses forwarded from a back-end driver, and submits formatted log records to the logging service via a secure communication channel (using a protocol such as SSL).

The proposed architecture (refer to Figure 5) operates in the same way as the control flow example (Section 7.2) except now the back-end driver forwards I/O requests to the log transit as well as the native device driver (4, 5, the numbering refers to Figure 5). I/O responses are also forwarded to the log transit when the back-end driver receives data from the native device driver (11). The log transit, after

receiving an I/O request and its details, waits for an I/O response to attach response and verification details to a log record.

Once all required details for a log record have been collected, a secure message is generated in the following format:

$$\{PCR_{AIKS(DomN)}, cred(AIK), LOG_{AIKS(DomN)}, N\}_{PK}$$

The message contains PCR measurement of back-end domain ($DomN$) signed with private attestation key $AIKS(DomN)$, AIK credential which contains the public attestation key $AIKP(DomN)$, log records signed with private attestation key $LOG_{AIKS(DomN)}$, and a nonce N. N ensures freshness of the message and prevents replay attacks.

The PCR value represents the state of the log transit component, back-end and native drivers running in $DomN$, and also the configuration files which prove that $DomN$ has the physical device privilege (DF_PHYSDEV); a trusted PCR value is sufficient to verify that trustworthy logs have been forwarded from a secure log generator. Both PCR measurement and log records are signed with the private attestation key of $DomN$, so that no adversaries can modify them without being detected. This approach solves the security flaws identified with Xenlog [14], because the integrity of trusted logs and the security state of all log generators are now verifiable. The entire message is encrypted with the public sealed key (PK) of the log security manager: the corresponding SK is stored in protected storage and is only accessible by the log security manager (not available to any other compartments) if it is in the trusted state indicated by AT; and this guarantees the confidentiality of the message. Furthermore, trust negotiation problems identified in existing solutions (refer to Section 4) can be addressed with this remote sealing approach.

The logging service enforces following security procedures when this message arrives (12, Figure 5):

1. Unseals the message with the SK of the log security manager.
2. It verifies the AIK credential (and $AIKP(DomN)$) with secure hardware certifying authority's public key Pca.
4. If the credential is valid, it uses $AIKP(DomN)$ to verify the signature on $PCR_{AIKS(DomN)}$ and $LOG_{AIKS(DomN)}$; correct verification indicates that PCR measurement and log records have been signed within the Virtual TPM (VTPM) using $AIKS(DomN)$.
5. Compares the PCR value with its set of trusted configurations to see if $DomN$, the log generator, is running in a trusted state.
6. If it trusts this configuration, the signed logs are stored in a secure log storage.

7.5 Evaluation

Table 1 summarises important security goals of the logging service (from Section 5.1) and shows how our secure logging infrastructure satisfies them:

Security Goals	***Secure Logging Features***
Needs to be deployed independently from any parent application.	Independent log transit components run inside small number of privileged driver domains and generate all trusted logging requests.
Needs to operate in a strongly isolated compartment with robust memory protection.	The logging service is isolated in the log security manager VM; the log transit components operate inside isolated back-end driver VMs.
Needs to be a small and simple software to make attestation feasible.	Log transit components are small pieces of code which merely forward logging requests to the logging service; the logging service simply verifies the log generators and the logs, and stores them.
Needs to verify the logs and the software configurations of all log generators.	Remote attestation between the logging service and the back-end driver domains: the logging service checks the security configurations of all remote driver domains (log transits) and verifies signatures on the logs before storing them.

Table 1. Security Goals of the Logging Service and Secure Logging Features

These security enhancements however, come at the cost of decrease in system performance: verifying a signature and carrying out a PCR quote for each log entry is likely to be very time-consuming, and it could well be detrimental to the overall grid performance. Such a tradeoff between

security and system performance can be minimised by configuring the log transit components to wait and collect a certain number of logging requests (e.g. 1000 requests) before signing the whole lot and forwarding them to the logging service; the logging service then, only verifies the signature and carries out PCR quote once for 1000 logging entries.

In this logging infrastructure we have focused more on describing *how* logging works, rather than *what* gets logged. But the *what* certainly affects usability: for example, invoking a grid service is a very different kind of event from reading a few bytes from a file; the one might occur a thousand times as often as the other. One could argue that unusual access patterns are more likely to be spotted in the high-level events than the low-level ones; and that there is a need for a mechanism to effectively manage the granularity of the logged events. Vecchio et al [20] describe a logging architecture for grid data access based on XACML where data owners specify and manage auditing policies dynamically to record events of interest to them. Our logging service could be extended to evaluate and enforce auditing policies after Step 5 (refer to security procedures described at the end of Section 7.4) using the methodology proposed in [20].

8 Conclusions and Future Work

In this paper, we have established the groundwork for building trustworthy audit and logging services in the grid. We have presented a number of use cases and carried out a security gap analysis on existing logging solutions to derive a common set of secure logging requirements: these requirements suggests the possible use of remote attestation to enforce consistent security procedures and policies in distributed logging services; and aims to facilitate production and analysis of log data with strong guarantees of confidentiality and integrity, to an evidential standard in a variety of contexts.

Although our focus has been primarily concerned with identifying the problems and the requirements, we have also proposed a preliminary logging architecture for the grid based on trusted computing technology taken together with system virtualisation. Our compartmented architecture makes attestation feasible as the grid participants only need to attest the integrity of relatively simple logging components to have strong confidence in the logs and the log access mechanisms. The driver virtualisation technology of Xen has been exploited to shift all trusted log generating code to a small number of back-end driver domains; hence, making log generator verification a much easier process for the logging service to manage in the grid; which otherwise, may have to verify hundreds of different grid applications and services that generate logs. These logs are classified differently from the logs that have been generated via untrusted routes.

8.1 Future Work

Our next step will be to develop a *trustworthy log reconciliation infrastructure* based on the secure logging requirements and the trusted compartments that have already been identified in Figure 3. This architecture will be designed to provide strong guarantees of both the integrity and the confidentiality of logs while being accessed, reconciled and analysed across multiple administrative domains. Log owners will be confident that trusted services will always govern log access and migration, and only the anonymised log information will be available to the end users; log users will be assured that their trusted code is running unmodified and collecting the classified logs from remote hosts, and be able to verify the integrity of the collected logs before carrying out post-analysis. These security enhancements have been discussed briefly in Section 6; in the next design phase, we plan to explore in details how log access VMs are managed dynamically and orchestrated together with the remote log security and log analysis managers to enable trustworthy log analysis in the grid.

In order to show the feasibility of the architecture, one or two concepts from proposed infrastructures will be carefully selected for prototype implementations: these prototypes will be our proof-of-concept to illustrate how different domains can be mutually trusted in the grid, how attestation can be achieved between heterogeneous nodes, how security policies can be enforced to VMs over distributed nodes, and so on. We believe that deploying the log transit components in driver domains and configuring them to communicate with the back-end device drivers will be one of the challenging prototype developments; this is because the low-level I/O requests forwarded to these drivers need to be interpreted and translated into useful logging information before being submitted to the logging service.

Acknowledgements

The work described is supported by a studentship from QinetiQ. Jason Crampton suggested a trusted audit and logging approach to improve the monitorability of SLAs. Andrew Simpson provided help with the healthcare grid background. John Lyle reviewed an early draft of this paper and provided constructive comments and suggestions.

The authors would also like to thank the anonymous reviewers for their careful attention and insightful comments.

References

[1] Trusted computing group backgrounder. https://www.trustedcomputinggroup.org/about/, Octo-

ber 2006.

[2] W. Z. A. Menon, A. Cox. Optimizing network virtualization in xen. Technical report, EPFL, Rice University, 2005.

[3] A. Cooper and A. Martin. Trusted delegation for grid computing. In *The Second Workshop on Advances in Trusted Computing*, 2006.

[4] I. Foster. What is the Grid? a three point checklist. *GRID Today*, 1(6), July 2002.

[5] I. Foster, H.Kishimoto, and A.Savva. The Open Grid Services Architecture, version 1.5. Specification, Open Grid Forum, 2006.

[6] D. Grawrock. *The Intel Safer Computing Initiative*, pages 3–31. Intel Press, 2006.

[7] B. Guttman and E. Roback. *Introduction to Computer Security: The NIST Handbook*. NIST Special Publication, DIANE Publishing, 1995.

[8] S. Hand. Architecture for back-end domains. http://lists.xensource.com/archives/html/xen-devel/2004-10/msg00443.html, 2004.

[9] J. H. Huh and A. Martin. Trusted logging for grid computing. Transfer dissertation, Oxford University Computing Laboratory, 2007.

[10] A. M. J. Kloster, J. Kristensen. Efficient memory sharing in xen virtual machine monitor. Master's thesis, Aalborg University, January 2006.

[11] T. Myer. Grid watch: Ggf and grid security. *developerWorks: IBM's resource for developers*, 2004.

[12] K. F. S. H. T. H. P. Barham, B. Dragovic. Xen and the art of virtualization. Technical report, University of Cambridge, Computer Laboratory, October 2003.

[13] D. J. Power, E. A. Politou, M. A. Slaymaker, and A. C. Simpson. Towards secure grid-enabled healthcare. *SOFTWARE PRACTICE AND EXPERIENCE*, 2002.

[14] N. A. Quynh and Y. 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.

[15] A. Simpson, D. Power, and M. Slaymaker. On tracker attacks in health grids. In *2006 ACM Symposium on Applied Computing*, pages 209–216, 2006.

[16] J. Skene, A. Skene, J. Crampton, and W. Emmerich. The monitorability of service-level agreements for application-service provision. In *6th International Workshop on Software and Performance*, pages 3–14, 2007.

[17] V. Stathopoulos, P. Kotzanikolaou, and E. Magkos. A framework for secure and verifiable logging in public communication networks. In *CRITIS*, volume 4347 of *Lecture Notes in Computer Science*, pages 273–284. Springer, 2006.

[18] B. Tierney and D. Gunter. Netlogger: A toolkit for distributed system performance tuning and debugging. Technical report, Lawrence Berkeley National Laboratory, December 2002.

[19] T. Van. Grid stack: Security debrief. *developerWorks: IBM's resource for developers*, 2005.

[20] D. Vecchio, W. Zhang, G. Wasson, and M. Humphrey. Flexible and secure logging of grid data access. In *7th IEEE/ACM International Conference on Grid Computing (Grid 2006), Barcelona Spain*, October 2006.