Trusted Computing and Provenance: Better Together

John Lyle and Andrew Martin
Oxford University Computing Laboratory

Abstract

It is widely realised that provenance systems can benefit from greater awareness of security principles and the use of security technology. In this paper, we argue that Trusted Computing, a hardware-based method for establishing platform integrity, is not only useful, but immediately applicable. We demonstrate how existing Trusted Computing mechanisms can be used for provenance, and identify the remarkable similarity and overlap between the two research areas. This is accomplished through presenting architectural ideas for a trusted provenance system, and by comparing the respective requirements and capabilities of trusted systems and provenance systems.

1 Introduction

Provenance information is essential for maintaining the integrity of scientific results, particularly those that are difficult to reconstruct independently. Through it we can verify the origins of primary source data, how it has subsequently been processed, and create a complete set of instructions as to how to recreate the final results. Many systems exist to support the collection of provenance information in e-Science [2], providing some of the following functionality [37]:

- Find 'the sources of faulty, anomalous processing outputs.' [2]
- Allow judgement of data quality [24]
- Support the replication of results
- Maintain the correct attribution of data
- Augment results with additional experimental context
- Enhance trust in scientific results [9]

We postulate that as such provenance information becomes more widely available, and more reliance is placed upon it, there will be an increasing need for strong guarantees of its accuracy: in information security terms, many will be concerned with provenance *integrity* and therefore with tamper-proofing the systems which record and process such information.

Meanwhile, research in Trusted Computing is attempting to provide trustworthy platforms, where the integrity of data storage and program execution can be assessed (and enforced) remotely. The aim is to provide security and assurance despite the presence of malicious software. Longstanding research and development efforts mean that implementations using a mix of hardware and software mechanisms are available today, with the hardware components quickly becoming ubiquitous in commonplace computing platforms. In this paper we argue that these new security technologies provide many of the features that provenance systems require. And because they have security as a primary design goal, can be used to implement *trustworthy* provenance systems with little additional effort or modification.

The rest of this paper is structured as follows. In Section 2 we provide a brief overview of provenance and discuss Trusted Computing in more detail. We then discuss the need for trusted provenance in Section 3. Following this, we outline a provenance architecture built on Trusted Computing technology and standards, highlighting how existing software and specifications can be used, what modifications would be necessary, and the advantages of doing so. In Section 5 we show that several research problems (and proposed solutions) are common to both Trusted Computing and provenance, and that both areas can benefit from collaboration with each other. We then mention some of the remaining challenges in implementing trustworthy provenance, and finally Section 7 presents our conclusions.

2 Background

2.1 Provenance

'Provenance' or 'lineage' generally refers to information that 'helps determine the derivation history of a data product, starting from its original sources' [37]. In other words, a record of where data came from and how it has been processed. This is particularly applicable to e-Science, as the quality of experimental data is important. Indeed, Moreau et al. [24] state that:

> 'In an ideal world, e-science end users would be able to reproduce their results by replaying previous computations, understand why two seemingly identical runs with the same inputs produce different results, and determine which data sets, algorithms, or services were involved in their derivation.'

The assertions made in a provenance system (*p-assertions*) can be categorised in many ways. Cheney et al [6] refer to 'how', 'why' and 'where' statements, and Vázquez-Salceda et al. [47] define 'interaction', 'relationship' and 'actor state' categories. The latter being information about the state of a participant in a workflow or process that manipulates or creates the original data.

There are several existing systems defined for recording process provenance. We skip a full review, as details can be found in survey papers [37, 2]. The First Provenance Challenge [25] is a good starting point for comparison.

Security and data integrity are becoming increasingly relevent to provenance, as researchers become more aware of the threats posed. Several proposed systems use kernel and file system-level monitoring to protect the collection of provenance information [46, 36], removing it from the user's control. Hasan et al. [14] provide a thorough analysis of threats to provenance systems, and have proposed a system using encryption and chained signatures to provide integrity protection. The authors make a good point that without a 'trusted pervasive hardware infrastructure', there will always be potential attacks. We will demonstrate in this paper that such an infrastructure can readily be provided by Trusted Computing, and therefore believe that our paper is complementary to their work. Similarly, Zhang et al. [48] use hash chains to provide *tamper-evident provenance* in databases, and tackle the issue of providing audit logs of compound objects rather than just for a linear sequence of operations. They state that the use of trusted hardware is 'impractical' due to the loosely-organised nature of provenance collection and sharing. We belive that Trusted Computing is cheap and pervasive enough to avoid these issues in many scenarios. Tan et al [39] have also listed several security requirements for provenance, discussing signatures on p-assertions in order to provide integrity guarantees, as well as accountability. Braun et al. [3] discuss the challenges of securing provenance data when it may contain sensitive or confidential information. We are more concerned with the *integrity* of provenance information in this paper.

2.2 Trusted Computing and Integrity Reporting

Trusted computing is a paradigm developed and standardized by the Trusted Computing Group [45]. It aims to enforce trustworthy behaviour of computing platforms by identifying a complete 'chain of trust', a list of all hardware and software that has been used [22]. This chain of software can then be compared to a list of known 'good' applications, unlike standard approaches - such as virus scanners - that try to recognise and eliminate 'bad' software.

The technologies proposed by the TCG are centered around the Trusted Platform Module (TPM). In a basic PC implementation, the TPM is a chip connected to the CPU. It provides isolated storage of encryption keys and 16 Platform Configuration Registers (PCRs). These PCRs can be used to hold *integrity measurements*, in the form of 20 byte SHA-1 hash values. They can only be written to in one way: through the `extend()` command. This appends the current register value to the supplied input, hashes it, and stores the result in the PCR. A PCR value therefore reflects a *list* of individual hashes. After n values have been extended into PCR_m, the contents will be:

$$PCR_m = SHA1(A_n \parallel SHA1(...SHA1(A_1 \parallel SHA1(A_0 \parallel 0))))$$

In order to work out what individual inputs have been extended to make the final PCR value, a separate 'measurement log' must be kept. This is a list of the items that have been extended, and may just be a reference to an executable file. When the log is replayed, by rehashing every entry in order, the final result should match the value in the PCR, proving that the log is an accurate and complete history of the programmes run on the platform. The first two columns in table 1 are a simplified example of a measurement log. In a real system, there would be hundreds of measurements stored in 16 or more individual PCR values.

The limited functionality offered by the TPM is ideal for recording the boot process of a platform. The idea being that, starting from the bios, every piece of code to be executed is first hashed and extended ('measured') into a PCR by the preceding piece of code. Typically PCRs 0-10 are used for this purpose. This principle is known as *measure before load* and must be followed by

all applications. If so, no program can be executed before being measured, and because the PCRs cannot be erased, this means that no program can conceal its execution from the TPM. The first item in the chain cannot be independently measured at runtime, and is referred to as the *root of trust for measurement*—we would aim for it to be immutable if possible. A platform is said to support *authenticated boot* when it follows this process as it provides a way for users to authenticate their platform's boot sequence against reference values.

In order for users to assess a remote machine, the TPM supports a feature called *remote attestation*, which allows a platform to report the integrity measurements collected during authenticated boot. When challenged, the TPM can create a *signed* copy of its PCR values. The signing key ('attestation identity key', AIK) being demonstrably linked to a trusted platform, with privacy protection where necessary. This is then given to the challenger for inspection, along with the measurement log.

The software running at the platform can be identified by matching the hash values in the log with reference data. This requires a list of *reference integrity measurements* (RIMs) contained within a Reference Manifest Database [41]. These measurements are collected from their original source: the software and hardware manufacturers.

Of course, the boot process alone may not be enough information for a remote user. Details about runtime state and configuration will also be necessary, and this is the subject of much Trusted Computing research [12]. This is discussed further in Section 4.4.

The TPM offers other functionality in addition to the platform configuration registers and attestation. It can be used to store cryptographic keys in a protected area of memory, and can provide a secure source of random numbers. When used with an external time server, it can also report the current time on the platform through an internal 'TickCounter'. The TPM can 'TickStamp' arbitrary data to assert that it was present on the platform at that exact time.

2.3 Example attestation and verification process

When a user challenges a remote platform to attest, the following steps are performed:

1. The user requests a signed copy of the remote platform's PCR values.

2. The platform must first obtain an *Attestation Identity Key* (AIK) and certificate from a trusted third party, the *Privacy CA*. This key will only be certified if the platform has a suitable TPM. The certificate is, in essence, a credential proving the validity of the TPM.

3. The remote platform's TPM performs the `TPM_Quote()` operation, using the AIK, which exports a signed hash of the requested PCRs (an 'attestation'). In table 1, this would be a signed copy of value `0x8A484`

4. This is sent to the user, along with the AIK certificate and a copy of the *measurement log* (the first two columns in table 1), which contains a list of all executables that the platform claims to have run.

5. The user checks that the attestation was signed with the key pair given on the AIK certificate, and that the certificate is itself signed by a trusted Privacy CA. This shows that a real TPM was used, so the PCR values cannot have been forged without hardware tampering.

6. The measurement log is now replayed, to check that the final hash value (the result of hashing each entry in order) matches the reported contents of the PCRs. If this is the case, then the log must be an accurate representation of the execution state of the platform. From table 1, this means re-calculating $SHA1(\texttt{0x5D27D} \| SHA1(\texttt{0xB4898} \| SHA1(\texttt{0x2D207} \| \texttt{0x00})))$ and comparing the result to `0x8A484`.

7. Having verified the log, each *entry* in it must now be checked to see if it is considered trustworthy. This will be done by comparison with a known list of reference measurements (RIMs). In our simplified example, if the user knows that `0x2D207` is the hash of a trustworthy BIOS, `0xB4898` is the hash of the Grub bootloader and `0x5D27D` is the Linux kernel, then they can be sure that only these things are running. As a result, the platform may be deemed 'secure'.

3 The Case for *Trusted* Provenance

Perhaps one of the most significant reasons for keeping provenance information is to provide assurance in the quality of scientific results [9]. This usually means protecting against unintentional error, or malfunctioning equipment. However, for high-profile science, such as climate change and pharmaceuticals, the risk of intentional, malicious intervention becomes just as important. In these situations there are threats from outside – organisations and individuals wishing to manufacture results supporting their interests, or discredit research that damages their products. Separately, and perhaps more invidiously, the user/researcher may have their own motivation

Table 1: Example authenticated boot process.

Event	Hash	PCR Value	
BIOS	0x2D207	0x8FC78	=SHA1(0x2D207 $\|\|$ 0x00)
Bootloader	0xB4898	0xE1A9F	=SHA1(0xB4898 $\|\|$ $SHA1$(0x2D207 $\|\|$ 0x00))
Kernel	0x5D27D	0x8A484	=SHA1(0x5D27D $\|\|$ $SHA1$(0xB4898 $\|\|$ $SHA1$(0x2D207 $\|\|$ 0x00)))

for falsifying data. We believe that provenance systems should be able to identify and record these threats. But to do so reliably, records must be robust and secure. They must be highly tamper-resistant, making any successful attack on their integrity infeasible. If provenance records are not protected, then they cannot provide convincing evidence of the quality of the data itself.

Clearly, the provenance of results relies upon both hardware and software—as those who encountered the Intel Pentium floating point bug [16] learnt to their cost. In a massively distributed system, such as that provided by a grid or cloud computing scenario, such concerns are all the more important—but potentially also give rise to a significant overhead in metadata management. Such systems may be located outside the user's own department, perhaps a different university or even on another continent. They are subject to the oversight of many unseen administrators, hardware changes, and software upgrades and patches.

Moreover, the scope for malicious interaction is great: too many individuals are involved, and so reliance upon informal trust relationships is infeasible. Even where all those participating are honest, there remains the possibility of viruses and trojans.

Such concerns are illustrated well by the challenges of 'public resource computing' projects such as climate*prediction*.net [38]. By distributing computational tasks to hundreds of thousands of users around the world, substantial resources can be brought to bear upon a task like climate modelling—but the results are open to fabrication, or the introduction of systematic bias. The duplication of tasks can help to reduce this risk, but at the cost of effectively reducing also the amount of computational power available. Although one may *hope* that well-managed grid resources will give more reliable results, as the value and impact of those results rises, the need for supporting evidence grows also.

The situation therefore seems quite hopeless: we are in a computing scenario in which we must place a high degree of trust in every possible processing platform, without any meaningful guarantee of trustworthiness. This leaves us open to manipulation, and we cannot consider reported provenance information any more reliable than the reported data. This is made worse by the mutable nature of software and data – it is too easy for a malicious party to alter programmes and records to create believable forgeries. We require some way to retake control, without losing the advantages of distributed processing. The simple addition of extra layers of software controls does not necessarily solve the problem, nor even raise the bar significantly, if the attacker has sufficient motivation.

Here is a role for Trusted Computing and secure hardware. Designed to provide a small, internal 'trusted third party', the Trusted Platform Module can be used to record and report the state of the computer in which it is embedded. This is designed to be immune to attack by software (which should eliminate the threat of malware) and can provide exactly the evidence we require that a computer has not been tampered with. If used for provenance, it means that any result processed with illegitimately modified software would always be recorded as such. The means for enhancing platform trust and for collecting provenance data are closely aligned, and can therefore be provided by the same mechanism. The only way to create false records would be to tamper with the hardware itself, an extremely expensive and time-consuming task, beyond the capabilities of most. Having identified that distributed scientific experiments face significant threats, and are performed in low-assurance situations, it seems essential that provenance data be further protected to retain the quality and trustworthiness of computational results.

4 Remote Attestation as a Provenance System

Integrity measurements seem immediately applicable to two requirements of provenance: identifying which results have been affected by a known software or hardware error, and for accurately reproducing results. Without knowing the exact versions of software that were used, neither of these things will always be possible. In the language of some provenance research [6], it can help answer questions about 'how' data has been modified and unambiguously identify 'where' it originated. Information about the execution state of a processing node can be considered 'actor state' information in the categories discussed by Vázquez-Salceda et al. [47].

Remote attestation and provenance systems appear to use similar techniques to solve related problems. Because of the similarities between the two fields, in this section we present a provenance architecture based en-

tirely on available Trusted Computing software and hardware. While this is not a complete system, and does not provide answers to many provenance questions, we demonstrate that certain aspects of provenance can easily be implemented in this way, with the built-in benefit of high assurance.

4.1 An attestation-based provenance architecture

We assume a service-oriented infrastructure, perhaps implemented as a grid or cloud, with a number of remote platforms performing computations (see Figure 1). Each machine has a Trusted Platform Module and, when initially added to the network, they are issued an Attestation Identity Key (AIK), signed by a certificate authority (Privacy CA). This key will be used for subsequent attestations, and uniquely identifies the platform. At this time, an administrator will record the machine's original hardware details and software measurements. Much of this process is defined in the Trusted Computing specifications [41]. The platform itself uses software that supports authenticated boot, and the TPM will therefore record all running executables, usually in the first twelve platform configuration registers. This, along with the job request and result, will be the provenance data captured by each platform.

When the platform receives a job, it does the following:

1. Measure a hash of the received job (or, if it is a web service, the incoming request) into PCR 11 of the service's TPM.
2. Execute the job
3. Hash and measure the job result (or reply message) into PCR 11.
4. Sign PCRs 0-11 with the Attestation Identity Key and send them to the provenance store, along with the measurement log. This log contains a list of all the values extended into each PCR.

The provenance store will receive regular reports from the processing platforms, consisting of attestations, measurement logs and the results of processed jobs. This information will be connected to other sources of provenance data, such as the workflow description. As described in Section 2.3, the report signatures will be verified, and the reported PCR values will be checked to make sure that they correspond to the log. If either of these steps fail, this implies a software error (or malicious intervention) and jobs should be rerun on a different machine. In either case, a copy of the attestation and log should be recorded in the provenance store.

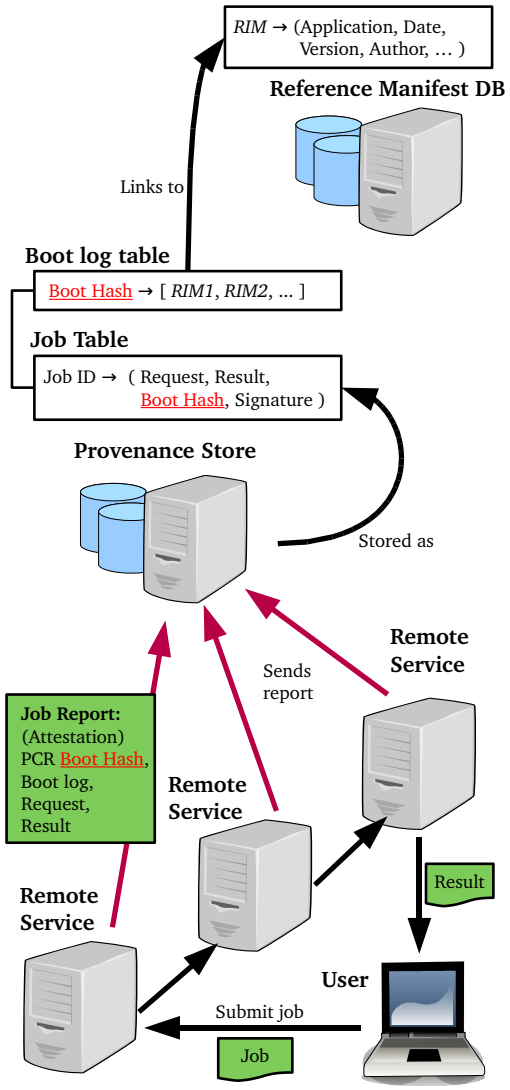

Figure 1: Diagram of an attestation-based provenance architecture. Remote services process results and attest to the provenance store, which saves and links the measurement logs to a TCG-defined Reference Manifest Database.

The contents of the measurement log for PCRs 1-10 will contain a list of every piece of software executed on the platform. This information needs to be stored for every job. However, if exactly the same software has been run as on a previous attestation, then the final PCR hash will be identical. In this case it will suffice to list all the software once, and link to it from the provenance database. This means that the majority of entries into the provenance store just consist of the attestation itself, and will therefore be extremely small - only a few 20 byte hash values. Of course, a full list of software and hashes (RIMs) will need to be maintained somewhere.

In the TCG model, this would be a Reference Manifest Database, and there are well-defined schemas for each entry, as well as protocols for keeping the database up to date and accessible [41]. Such a system is given in Figure 1. We note that compacting provenance information through hash chains has been discussed before by Hasan et al. [14] and Zhang et al. [48].

Every time a new patch is loaded onto a service machine, this will result in a new hash and therefore a new chain of trust being stored. However, the storage overhead should remain small. Based on data from 2006 to 2009, a typical web service can expect to be updated only around 22 times a year, with under 500 new hash values [21]. This is trivial amounts of data, particularly as most machines will be running near-identical sets of software.

4.2 Software and hardware details

Almost all the software and hardware required for implementing the described system is freely available today. TPMs are installed in many business notebooks and servers, and the Linux kernel now supports Authenticated Boot[1]. TCG compatible software stacks exist, including the JTSS[2] in Java and TrouSerS[3] in C++. These make writing programs that use the TPM straightforward. Reading and extending PCR value in Java, for example, requires the JTSS libraries, around ten lines of initialisation code, and then just one line to actually read or extend. Privacy CA software and an integrity-measuring Java Runtime Environment is also available on the JTSS website. The OpenPTS[4] 'Platform Trust Service' project provides the infrastructure for creating integrity measurements, collecting RIMs and connecting to an external software repository. Furthermore, web service protocols are already defined to maintain compatibility with WS- standard [26]. In fact, the only custom changes to software would involve updating the service interface (or grid middleware) to measure incoming requests and outgoing results, and to send attestations to the provenance database. This should be straightforward.

4.3 Advantages

This infrastructure immediately provides several advantages to ad-hoc reporting of platform information. Forging integrity reports is infeasible, thanks to the secure key storage provided by the TPM. Because AIKs are stored in the TPM, and cannot be disclosed, it would be extremely difficult to assert that a different machine produced the result. Because of the platform configuration registers, authenticated boot process and software support, it also should not be possible to claim that a different version of a particular piece of software was being used. And as we are measuring the input and output, we can be sure of exactly which software was used to process a particular job, and what output it produced. This means that the attested information meets the requirement given by Groth et al. [10] for high integrity p-assertions. Furthermore, attestations can potentially be created autonomously, at any point in time, a requirement defined by Groth and Moreau [11]. Thanks to the software and hardware developed, attestations must also produce a complete record of all software used, at every stage of platform boot and throughout its use. This includes firmware, drivers and shared libraries, potentially a more comprehensive (and accurate) list than other systems are capable of producing. Jobs can be time stamped, through use of the TPM's tick counter. In addition, the presence of TPM hardware is an opportunity to improve the security of credentials, as keys can remain protected from the rest of the system.

All of these benefits come merely by leveraging existing Trusted Computing techniques. This is significant because it uses a single technological base to give advantages for both short-term trust and long-term provenance.

4.4 Missing components

The system discussed can be enhanced considerably. There are some obvious missing features and functionality. The attested information gives only the *execution state* of the platform, at the level of applications run by the operating system. Nothing more fine-grained can be reported. Current implementations will also not re-measure a program, which means that the precise ordering of execution will not be preserved. Other missing information includes configuration files, environment variables, generated code, and load information (free disk space, processor utilisation, and so on). However, it would be relatively straight-forward to include all of this information, as it should only be necessary to modify middleware or adapt an existing system such as PASS [27] or Provenance Aware Condor [33]. 'Semantic Attestation' also aims to solve this problem [12].

Perhaps more importantly, integrity reports would need to be mapped to the rest of the information produced by a full provenance architecture. This includes records of *who* accessed data, what the overriding purpose of the request was, and how each individual platform was used in combination for a full process workflow. Such information must (in part) be provided by the end-user, and will need to reference the generated integrity measurements.

The above system does not provide any easy mechanism for recreating results. While it would be possible to guarantee that two results came from precisely the same

software execution, if hardware or software is changed subsequently, there is no way to re-run with the original versions. We suggest that this could be implemented through saving and restoring virtual machines. Such an approach would be compatible with the Eucalyptus cloud computing system [32].

It would also be necessary to include custom developed software in the Reference Manifest Database (RMDB), and extend the TCG schema [42] to include information about how it was built. The provenance store would also need to be modified suitably to work with a RMDB and to support searches for all results that used certain pieces of software.

5 Provenance and Trusted Computing Research: Producing the Same Solutions

Despite the lack of interaction between Trusted Computing and provenance communities, there is a great deal of overlap in current research. The security industry is looking for better methods for monitoring a platform's behaviour, a task that provenance systems already focus on: in many security contexts, *prevention* is infeasible or prohibitively expensive; *detection* is often a viable alternative. Detection of anomalies is therefore of great interest. Moreover, provenance research is looking to increase the trustworthiness and integrity of records [13, 39], a well-established problem in security. In this section we identify common areas of work, and look at the related (but perhaps unknown) literature.

5.1 Related research

Both provenance and Trusted Computing are concerned with monitoring and reporting the state of a machine used for some high-value (or high risk) function. Huh and Martin [15] look to provide more detail by intercepting and securely logging I/O requests. In the provenance domain, PASS [27] provides logging through hooking system calls, and Clifford et al. [7] provide runtime execution logging. Reilly and Naughton [33] have similar ideas, but use an extension to Condor to perform transparent logging. The work by Huh and Martin will provide a higher assurance, but the approach taken by Reilly and Naughton may result in more useful data. A common theme in this section is that Trusted Computing research currently focuses on creating comprehensive and high-integrity results, whereas provenance systems are better at extracting exactly the information considered relevent, and are not constrained by security issues.

Tracking data usage is an important functionality of a provenance system, but has also been approached many times in Trusted Computing research, with digital rights management in mind. Proposals by Nauman et al. [30] would enable 'measurement, storage and reporting of the attribute update behavior' for a data item at a remote platform. This is part of the functionality required for provenance [24], and we can imagine it being integrated with data derivation graphs. Provenance, access control and usage control have been linked before, notably by Ni et al. [31].

The construction of custom executables and their history has been approached by both areas. In earlier work, we have used integrity measurement to measure the compilation process in order to produce a trustworthy compilation certificate for an arbitrary programme [20]. This is similar to the 'Transparent Make' functionality provided by Vahdat and Anderson's TREC lineage system [46] and PASS by Muniswamy-Reddy et al. [27]. Again the goals are different, but both allow users to identify how an executable was formed and what its dependencies are. The similarities are notable, as TREC allows dependencies to be specified through Make files, and we provided the same through ANT build scripts. However, TREC is general-purpose and has many other applications, operating constantly through a kernel module intercepting system calls. This makes it more suitable for constant use. Our compilation certificates, on the other hand, are verifiable and considerably more trustworthy, but must be run independently of normal processing. Overall, there is a clear requirement for greater information about compiled software in both fields.

For full provenance information, we need to know how data is stored. The Trusted Compting Group have standards for 'Trusted Storage' [44], providing features such as disk-encryption, authentication and logging. The logging use case specified by the TCG has forensics and auditing in mind [40]. Again, this is similar to the requirements for provenance [13] - we would like the ability to go back through records and establish whether tampering or unauthorised access occurred. While we are not aware of any hardware-based related provenance research, secure and audited storage through file system and kernel support has been mentioned frequently [46, 27, 36].

Both Trusted Computing and provenance systems tend to involve modification of existing middleware. Condor has been modified by researchers in both fields [33, 29] looking to add security and lineage through monitoring and assessing individual platforms. Löhr et al. [19] proposed new modifications to grid middleware for enhanced trustworthiness, and Frew and Slaughter [8] have demonstrated provenance in the ES3 system. Similarly, Trusted Cloud Computing [35] and Provenance-Aware Cloud Computing [28] have both been discussed recently, as well as Service Oriented Architectures [1, 39]. This implies that not only are similar problems being solved, but in the same context and using the same underlying software and systems.

5.2 What provenance can gain from Trusted Computing

We have already discussed the motivation for trusted provenance, but the use of Trusted Computing has many potential advantages. Going beyond the enhanced security and assurance, Trusted Computing research typically considers a much wider range of factors that can affect system behaviour. This includes CPU architecture, use of virtualisation, protocols and software. By taking advantage of this thoroughness, provenance systems can be more comprehensive and may identify hidden factors [8] that will later be useful. Furthermore, as security problems receive greater attention and funding, it seems sensible to take advantage of the new hardware and processes that are being implemented and reuse them for provenance. Trusted Computing is led by a group of companies (such as IBM, Intel, AMD, and Microsoft [45]) with significant resources. By introducing provenance as a requirement, we believe that many of the tools being developed and used for security can become immediately useful for provenance.

5.3 What Trusted Computing can gain from provenance

Provenance research can be used by Trusted Computing researchers to enhance system security and auditing. Integrity reporting systems lack a framework for evaluation, and require a way of interpreting results. The provenance community is more aware of systems involved with semantic representation and metadata, which could be the solution to this problem. Both fields also have the problem of reporting too much data, and deciding how best to filter it. We suspect that the provenance community are further ahead in storing and querying this kind of information.

Trusted Computing suffers from another problem that is better understood in provenance: how to deal with incomplete data. Remote attestation can only give details of the current state of a platform, not historical data. Regular attestations, such as those mentioned in our proposals in Section 4 can provide a better history, but what should happen when a record is missing? How should this scenario be recorded? Provenance systems are already designed to work with incomplete information and composite data sources.

Finally, provenance seems like an excellent use of Trusted Computing, particularly as many of the criticisms of Trusted Computing are less relevent. The integrity reporting approach has been criticised as being fragile when used to make access control decisions, as any missing software in the reference database will result in the platform being denied access. However, in provenance, runtime decisions are not as important as storing a history for later use, so this fragility is less important.

6 Challenges

With all the similarities we have listed, there are still some challenges. The research directions are different: the Trusted Computing researchers are currently focused on improving security through advances in cryptographic algorithms, and isolation mechanisms. In comparison, semantic consistency and querying are perhaps more important for provenance. Furthermore, in provenance the goal is to gain extra information for later analysis, and to improve scientific results. And while security can be considered an enabler of new functionality, many still believe it to be just about *preventing* bad things from happening.

Scientific results will face different threats and attacks than many other distributed computing techniques, and a significant challenge is making sure that security does not reduce usability. In many cases, there may be a relatively low risk to the data, and this should be reflected in the security architecture. As a result, the use of Trusted Computing should be as transparent as possible, and require as little effort for users and developers of applications (often the same people). An open challenge is developing a systematic methodology for creating applications that support provenance and provide high assurance. This may involve combination of recent work on PrIMe [23] and security development lifecycles [18].

Performance is another issue that could prevent the adoption of both provenance and security technology. The TPM is a low-speed chip, and cryptographic operations (such as attestation) are relatively slow. The authenticated boot process also impacts on performance [34]. However, new versions of the TPM may be faster, using symmetric cryptography [43] and it is likely that the hardware manufacturers will be able to increase performance in the future. For the time being, there has been research looking at improving efficiency [4], and the provenance architecture we outlined would require only one attestation per submitted job and platform.

Trusted Computing also relies upon a public key infrastructure, for certifying attestation keys and identifying platforms. We have not explored the trust management issues in depth in this paper, and there will undoubtedly be issues in maintenance and implementation. Fortunately, scientific grid computing is one domain with experience in key management on a large scale, and we are optimistic in solving such problems.

7 Conclusion

Many of the concerns addressed by Trusted Computing relate to immediate and short-term policy enforcement ("shall I share this secret with that software, on that platform?"). Many provenance issues are of a more long-term nature ("where did this come from; how was it processed?"). Yet these are highly-related topics because they both rely upon the unambiguous (and tamper-proof) identification of hardware and software.

Existing Trusted Computing systems *already* provide much of the required functionality, and in a way that provides high assurance and makes forgery infeasible. Furthermore, with the introduction of security systems using Trusted Computing, the hardware becoming available, and software libraries and OS infrastructure, the basic capabilities for collecting highly-assured provenance data are being built. We have identified several places in which the two research areas overlap, such as logging, monitoring, compilation history and secure storage.

We have argued that there is a natural synergy between the two areas of research, an overlap in both the goals and the technologies for achieving them, and a strong prospect for combining the two to give rise to *trusted provenance*.

8 Acknowledgments

John Lyle's work is funded by a studentship from QinetiQ and the EPSRC. Andrew Martin is grateful for helpful conversations on the topics described here with Luc Moreau, James Cheney, and the participants in the workshop on *Provenance in Secure and Advanced Computer Systems* at the Edinburgh e-Science Institute. We also thank the reviewers for their useful advice and feedback.

References

[1] BERTHOLD, A., ALAM, M., BREU, R., HAFNER, M., PRETSCHNER, A., SEIFERT, J.-P., AND ZHANG, X. A technical architecture for enforcing usage control requirements in service-oriented architectures. In *SWS '07: Proceedings of the 2007 ACM workshop on Secure web services* (New York, NY, USA, 2007), ACM, pp. 18–25.

[2] BOSE, R., AND FREW, J. Lineage retrieval for scientific data processing: a survey. *ACM Comput. Surv. 37*, 1 (2005), 1–28.

[3] BRAUN, U., SHINNAR, A., AND SELTZER, M. Securing provenance. In *HOTSEC'08: Proceedings of the 3rd conference on Hot topics in security* (Berkeley, CA, USA, 2008), USENIX Association, pp. 1–5.

[4] CHAOWEN, C., RONGYU, H., HUI, X., AND GUOYU, X. A High Efficiency Protocol for Reporting Integrity Measurements. In *Intelligent Systems Design and Applications, 2008. ISDA '08. Eighth International Conference on* (Nov. 2008), vol. 2, pp. 358–362.

[5] CHENEY, J., Ed. *First Workshop on the Theory and Practice of Provenance, February 23, 2009, San Francisco, CA, USA, Proceedings* (2009), USENIX.

[6] CHENEY, J., CHITICARIU, L., AND TAN, W.-C. Provenance in Databases: Why, How, and Where. *Found. Trends databases 1*, 4 (2009), 379–474.

[7] CLIFFORD, B., FOSTER, I., VOECKLER, J.-S., WILDE, M., AND ZHAO, Y. Tracking provenance in a virtual data grid. *Concurrency and Computation: Practice and Experience 20*, 5 (2008), 565–575.

[8] FREW, J., AND SLAUGHTER, P. ES3: A Demonstration of Transparent Provenance for Scientific Computation. In *IPAW* (2008), J. Freire, D. Koop, and L. Moreau, Eds., vol. 5272 of *Lecture Notes in Computer Science*, Springer, pp. 200–207.

[9] GIL, Y., DEELMAN, E., ELLISMAN, M., FAHRINGER, T., FOX, G., GANNON, D., GOBLE, C., LIVNY, M., MOREAU, L., AND MYERS, J. Examining the Challenges of Scientific Workflows. *IEEE Computer 40*, 12 (Dec. 2007), 26–34.

[10] GROTH, P., JIANG, S., MILES, S., MUNROE, S., TAN, V., TSASAKOU, S., AND MOREAU, L. An Architecture for Provenance Systems. Tech. Rep. 13216, University of Southampton, November 2006.

[11] GROTH, P., AND MOREAU, L. Recording Process Documentation for Provenance. *Parallel and Distributed Systems, IEEE Transactions on 20*, 9 (Sept. 2009), 1246–1259.

[12] HALDAR, V., CHANDRA, D., AND FRANZ, M. Semantic remote attestation: a virtual machine directed approach to trusted computing. In *VM'04: Proceedings of the 3rd conference on Virtual Machine Research And Technology Symposium* (Berkeley, CA, USA, 2004), USENIX Association, pp. 3–3.

[13] HASAN, R., SION, R., AND WINSLETT, M. Introducing secure provenance: problems and challenges. In *StorageSS '07: Proceedings of the 2007 ACM workshop on Storage security and survivability* (New York, NY, USA, 2007), ACM, pp. 13–18.

[14] HASAN, R., SION, R., AND WINSLETT, M. The case of the fake picasso: preventing history forgery with secure provenance. In *FAST '09: Proccedings of the 7th conference on File and storage technologies* (Berkeley, CA, USA, 2009), USENIX Association, pp. 1–14.

[15] HUH, J. H., AND MARTIN, A. Trusted Logging for Grid Computing. In *Trusted Infrastructure Technologies Conference, 2008. APTC '08. Third Asia-Pacific* (Wuhan, China, Oct. 2008), IEEE, pp. 30–42.

[16] INTEL. Statistical analysis of floating point flaw: Intel white paper. `http://www.intel.com/support/processors/pentium/fdiv/wp/`, November 1994.

[17] JONKER, W., AND PETKOVIC, M., Eds. *Secure Data Management, 6th VLDB Workshop, SDM 2009, Lyon, France, August 28, 2009. Proceedings* (2009), vol. 5776 of *Lecture Notes in Computer Science*, Springer.

[18] LIPNER, S. The Trustworthy Computing Security Development Lifecycle. In *ACSAC '04: Proceedings of the 20th Annual Computer Security Applications Conference* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 2–13.

[19] LÖHR, H., RAMASAMY, H. G. V., SADEGHI, A.-R., SCHULZ, S., SCHUNTER, M., AND STÜBLE, C. Enhancing Grid Security Using Trusted Virtualization. In *Autonomic and Trusted Computing*, Lecture Notes In Computer Science. Springer Berlin / Heidelberg, Hong Kong, China, August 2007, pp. 372–384.

[20] LYLE, J. Trustable Remote Verification of Web Services. In *TRUST* (Oxford, UK, 2009), L. Chen, C. J. Mitchell, and A. Martin, Eds., vol. 5471 of *Lecture Notes in Computer Science*, pp. 153–168.

[21] LYLE, J., AND MARTIN, A. On the Feasibility of Remote Attestation for Web Services. In *Computational Science and Engineering, 2009. CSE '09. International Conference on* (Aug. 2009), vol. 3, pp. 283–288.

[22] MARTIN, A. The Ten Page Introduction to Trusted Computing. Tech. Rep. RR-08-11, OUCL, December 2008.

[23] MILES, S., GROTH, P., MUNROE, S., AND MOREAU, L. PrIMe: A Methodology for Developing Provenance-Aware Applications. *ACM Transactions on Software Engineering and Methodology* (June 2009).

[24] MOREAU, L., GROTH, P., MILES, S., VAZQUEZ-SALCEDA, J., IBBOTSON, J., JIANG, S., MUNROE, S., RANA, O., SCHREIBER, A., TAN, V., AND VARGA, L. The provenance of electronic data. *Commun. ACM 51*, 4 (2008), 52–58.

[25] MOREAU, L., LUDSCHER, B., ALTINTAS, I., BARGA, R. S., BOWERS, S., CALLAHAN, S., JR., G. C., CLIFFORD, B., COHEN, S., COHEN-BOULAKIA, S., DAVIDSON, S., DEELMAN, E., DIGIAMPIETRI, L., FOSTER, I., FREIRE, J., FREW, J., FUTRELLE, J., GIBSON, T., GIL, Y., GOBLE, C., GOLBECK, J., GROTH, P., HOLLAND, D. A., JIANG, S., KIM, J., KOOP, D., KRENEK, A., MCPHILLIPS, T., MEHTA, G., MILES, S., METZGER, D., MUNROE, S., MYERS, J., PLALE, B., PODHORSZKI, N., RATNAKAR, V., SANTOS, E., SCHEIDEGGER, C., SCHUCHARDT, K., SELTZER, M., SIMMHAN, Y. L., SILVA, C., SLAUGHTER, P., STEPHAN, E., STEVENS, R., TURI, D., VO, H., WILDE, M., ZHAO, J., AND ZHAO, Y. Special Issue: The First Provenance Challenge. *Concurrency and Computation: Practice and Experience 20*, 5 (2008), 409–418.

[26] MUNETOH, S., NAKAMURA, M., YOSHIHAMA, S., AND KUDO, M. Integrity Management Infrastructure for Trusted Computing. *IEICE Trans Inf Syst E91-D*, 5 (2008), 1242–1251.

[27] MUNISWAMY-REDDY, K.-K., HOLLAND, D. A., BRAUN, U., AND SELTZER, M. Provenance-aware storage systems. In *ATEC '06: Proceedings of the annual conference on USENIX '06 Annual Technical Conference* (Berkeley, CA, USA, 2006), USENIX Association, pp. 4–4.

[28] MUNISWAMY-REDDY, K.-K., MACKO, P., AND SELTZER, M. I. Making a Cloud Provenance-Aware. In Cheney [5].

[29] NAMILUKO, C. Trusted Infrastructure for the Campus Grid. Master's thesis, University of Oxford, Wolfson College, Oxford, September 2008.

[30] NAUMAN, M., ALAM, M., ZHANG, X., AND ALI, T. Remote Attestation of Attribute Updates and Information Flows in a UCON System. In *TRUST* (2009), L. Chen, C. J. Mitchell, and A. Martin, Eds., vol. 5471 of *Lecture Notes in Computer Science*, Springer, pp. 63–80.

[31] NI, Q., XU, S., BERTINO, E., SANDHU, R. S., AND HAN, W. An Access Control Language for a General Provenance Model. In Jonker and Petkovic [17], pp. 68–88.

[32] NURMI, D., WOLSKI, R., GRZEGORCZYK, C., OBERTELLI, G., SOMAN, S., YOUSEFF, L., AND ZAGORODNOV, D. 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* (Washington, DC, USA, 2009), IEEE Computer Society, pp. 124–131.

[33] REILLY, C. F., AND NAUGHTON, J. F. Transparently Gathering Provenance with Provenance Aware Condor. In Cheney [5].

[34] SAILER, R., ZHANG, X., JAEGER, T., AND VAN DOORN, L. Design and Implementation of a TCG-based Integrity Measurement Architecture. In *USENIX Security Symposium* (2004), pp. 223–238.

[35] SANTOS, N., GUMMADI, K. P., AND RODRIGUES, R. Towards Trusted Cloud Computing. In *HotCloud09: Proceedings of the Workshop on Hot Topics In Cloud Computing* (2009).

[36] SAR, C., AND CAO, P. Lineage file system. Online at `http://theory.stanford.edu/~cao/lineage`.

[37] SIMMHAN, Y. L., PLALE, B., AND GANNON, D. A survey of data provenance in e-science. *SIGMOD Rec. 34*, 3 (2005), 31–36.

[38] STAINFORTH, D., MARTIN, A., SIMPSON, A., CHRISTENSEN, C., KETTLEBOROUGH, J., AINA, T., AND ALLEN, M. Security principles for public-resource modeling research. In *WETICE '04: Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 319–324.

[39] TAN, V., GROTH, P. T., MILES, S., JIANG, S., MUNROE, S., TSASAKOU, S., AND MOREAU, L. Security Issues in a SOA-Based Provenance System. In *IPAW* (2006), L. Moreau and I. T. Foster, Eds., vol. 4145 of *Lecture Notes in Computer Science*, Springer, pp. 203–211.

[40] THIBADEAU, R. Trusted Computing for Disk Drives and Other Peripherals. *Security & Privacy, IEEE 4*, 5 (Sept.-Oct. 2006), 26–33.

[41] 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`, November 2006.

[42] TRUSTED COMPUTING GROUP. TCG Schema. `http://www.trustedcomputinggroup.org/resources/infrastructure_work_group_reference_manifest_rm_schema_specification_version_10`, November 2006.

[43] TRUSTED COMPUTING GROUP. Summary Of Features Under Consideration For The Next Generation Of TPM. `http://www.trustedcomputinggroup.org/resources/summary_of_features_under_consideration_for_the_next_generation_of_tpm`, 2009.

[44] TRUSTED COMPUTING GROUP. TCG Storage Architecture Core Specification. `http://www.trustedcomputinggroup.org/resources/tcg_storage_architecture_core_specification`, April 2009.

[45] TRUSTED COMPUTING GROUP. Trusted Computing Group Home Page. `https://www.trustedcomputinggroup.org/home`, 2009.

[46] VAHDAT, A., AND ANDERSON, T. Transparent result caching. In *ATEC '98: Proceedings of the annual conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 1998), USENIX Association, pp. 3–3.

[47] VÁZQUEZ-SALCEDA, J., ALVAREZ, S., KIFOR, T., VARGA, L. Z., MILES, S., MOREAU, L., AND WILLMOTT, S. *In R. Annicchiarico, U. Cortés, C. Urdiales (eds.) Agent Technology and E-Health*. Whitestein Series in Software Agent Technologies and Autonomic Computing. Birkhäuser Verlag AG, Switzerland, Dec. 2007, ch. EU PROVENANCE Project: An Open Provenance Architecture for Distributed Applications.

[48] ZHANG, J., CHAPMAN, A., AND LEFEVRE, K. Do you know where your data's been? - tamper-evident database provenance. In Jonker and Petkovic [17], pp. 17–32.

Notes

[1] `http://linux-ima.sourceforge.net/`
[2] `http://trustedjava.sourceforge.net`
[3] `http://trousers.sourceforge.net/`
[4] `http://sourceforge.jp/projects/openpts/`