

Application Whitelists In Virtual Organisations

Jun Ho Huh, John Lyle, Cornelius Namiluko, Andrew Martin

Oxford University Computing Laboratory
Parks Road, Oxford, OX1 3QD, UK

{Jun.Ho.Huh, John.Lyle, Cornelius.Namiluko, Andrew.Martin} @comlab.ox.ac.uk

Abstract

Many distributed batch systems, such as computational grids, require a level of *integrity protection* to guarantee the proper execution of a job or workflow. One way of achieving this, implicit in many trusted computing proposals, is to use application whitelisting to prevent unknown and untrusted applications from being executed on remote services. However, this approach has significant shortcomings across multiple administrative domains, as conflicts between locally-managed whitelists will result in many useful services *appearing* untrustworthy to users. This has the potential to limit availability and prevent trusted distributed systems from ever being successfully deployed.

We propose a set of requirements for a system which will manage these conflicts, and provide a mechanism for updating application whitelists that will increase service availability and trustworthiness. We also suggest and specify a set of components, including a centralised *configuration manager*, which will meet these requirements.

Key words: configuration management, trusted computing, trusted grid, virtual organisations, whitelisting

1. Introduction

Integrity reporting is a method for establishing trust in a computing platform by identifying and assessing the software it is running. Researchers have made several proposals to enhance the security and trustworthiness of grid [1] and service oriented architectures [2] by using this technique. These often involve checking that the service providers are running software that can be identified in a *whitelist*, a repository of ‘known good’ software configurations. This would allow them to only use computers which they *know* are running fully patched software distributions, for example. However, details on how this whitelist would be managed are rarely considered, and a common solution is to pass on the responsibility to a third party. We identify several problems with this approach, particularly when considering a virtual organisation which spans *multiple administrative domains*, each with their own security requirements. We strongly believe that managing platform configuration changes is an essential part of trusted computing, and indeed the overall problem of securing distributed systems.

The most significant problem we identify is that of conflicting domain whitelists, which can occur when domains use different versions of software, or apply different patches. This can have an adverse affect on the availability and security of the network. We do a thorough functional analysis of trustworthy distributed systems proposed by other researchers, and identify

a number of additional components which can help to mitigate this issue. Our overall approach is to mandate *inter-domain communication* so that domains inform each other when an application update or vulnerability is found, and policy or whitelist entries must be changed.

The main contributions of this paper are (1) the identification of a potential flaw in whitelist-based distributed platform authentication, (2) a set of functional, interoperability and security requirements for a system that would solve these problems, and (3) a proposed set of new components for implementing this system. In addition, we believe that we are the first to establish a consensus within the literature of the components which make up a trusted grid system.

1.1. Paper Structure

This paper is structured as follows. In Section 2 we introduce fundamentals of virtual organisations and trusted computing. We review the literature on trusted distributed systems in Section 3, to identify a set of common components that most existing trusted computing proposals use. In Section 4 we go through several use cases to establish additional requirements needed for whitelist management. New components are proposed in Section 5 to meet these requirements. We then perform a security review and show how certain key threats can be mitigated in Section 6. In Section 7 we integrate our approach with two other grid systems. Finally, in Section 8 we conclude.

2. Background and Existing Work

2.1. What Is A Virtual Organisation?

A wide range of research is conducted, archived, and reported in the digital economy. Its influence has grown over the years to include various disciplines from science through to finance and industrial engineering. In consequence, different types of distributed systems have been deployed to facilitate the collection, modeling and analysis of the dispersed data; or the sharing of the computational resources. Such systems allow multiple organisations to come together as one unit for the purpose of commonly identified goals. The fact that this is happening on the internet characterises it as a *virtual organisation* [3].

A problem arises, however, when the models or data contain sensitive information or have commercial value. They then become lucrative targets for attack, and may be copied or modified by malicious parties. This is particularly true in many of the scientific disciplines where researchers — who care about the confidentiality of their privileged data or the integrity of the collected results — are reluctant to exploit the full benefits of distributed computing. Submitting a highly privileged job to the distributed resources requires prior knowledge of the security standards of all of the target systems — only those running with acceptable security configurations and patch levels should be selected to execute the job. However, this still remains as a security gap between the job owners requirements and current technological capabilities, and serves as a barrier to up-take of existing systems.

Recently, many researchers have discussed the use of Trusted Computing (see below) to bridge this gap. For example, as a possible solution to the ‘malicious host’ problem, Cooper and Martin [4] describe a grid architecture that ensures protected job execution environments. Trusted Computing allows job owners to verify that this environment is in place before submitting their jobs. Some of the key ideas in this field of work are identified in Section 3.1.

2.2. Trusted Computing

Faced with the prospect of modern PCs (and other devices) having so much software that their behaviour is unpredictable and easily subverted, the Trusted Computing Group (TCG) [5] has developed a series of technologies based around a Trusted Platform Module (TPM) — a hardware chip usually embedded in the motherboard — which helps to provide two novel capabilities [6]: a cryptographically strong identity and reporting mechanism for the platform, and a means to *measure* the software loaded during the platform's boot process. This includes the BIOS, bootloader, OS and applications. Further details of the TPM's functionality is defined in the TCG main specification [7].

Measurements are taken by calculating a cryptographic hash of binaries before they are executed. Hashes are stored in Platform Configuration Registers (PCRs) in the TPM. They can only be modified through special TPM ordinals, and the PCRs are never directly written to; rather, measurements can only be *extended* by an entity. The *extend* command takes the existing value in the register, appends the new measurement, and stores a hash of the result. This is to ensure that no other entity can modify or overwrite a measured value. In a trusted computing platform, every executable piece of code in the *authenticated boot* process will be measured and is responsible for measuring the next piece, creating a sequential list. This creates the notion of *transitive trust*, and provides a way for a relying party to trust a large group of entities from a single root. Any malicious piece of code (e.g. rootkit) executed during the boot process will be recorded and identified.

2.2.1. Sealed Storage

Trusted computing provides the means to *seal* (encrypt) data so that it will only successfully decrypt when the platform measurements are in a particular state. The seal process encrypts data using a *storage key*, a key held internally by the TPM, and binds it to a specified PCR value. The purpose of sealing is to prevent any unauthorised software from attempting to unseal the package. The TPM enforces two key restrictions upon decrypting the sealed package:

- The package is only available on the TPM that created it.
- The package is only decrypted if the platform's current PCR values equal those that were specified at the time the package was sealed.

The implication is that the external data only becomes available to an application when the correct value (an acceptable configuration) is in the specified PCR.

2.2.2. Remote Attestation

Remote attestation allows an application to find out the software state of a remote platform. This involves the remote platform's TPM creating a certificate of its current PCR values signed with a TPM-generated Attestation Identity Key (AIK). This certificate is sent to the application, which will validate the signature and the PCR contents. The TPM signs the specified PCRs with the private half of the AIK (held within the TPM), and returns the digitally signed quote. The external application validates the signature by using the public half of the AIK, and validates the AIK by looking at the AIK credential, a certificate issued by a trusted Certificate Authority (a 'Privacy CA') which states the platform has a valid TPM. The PCR log entries are then compared against a list of known good values (a whitelist) to check if the reported PCRs represent an acceptable configuration.

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

However, attestation does not necessarily give any indication of a platform's security state but rather its execution state [8]. Sadeghi and Stubble [9] introduce the notion of 'property-based attestation' in order to simplify the process. In property-based attestation, platforms provide a list of guaranteed security properties, rather than just a binary integrity measurement. This can be implemented in a number of ways, but generally relies upon at least one party being able to match the PCR values to security properties, and then issuing certificates to this end. Property-based attestation is a level of indirection which can take some of the burden from the attestation requester.

Further, the TPM has not been designed to detect modifications of measured software in the intervening time between measurement and attestation. In consequence, the runtime state of the platform may not be reported correctly in attestation [10]. Successful in-memory attacks (e.g. exploiting buffer overflows) will allow adversaries to modify the behaviour of the platform and subvert their security controls. To guard against this style of attack, measured software needs to be isolated and protected from less trustworthy components.

2.3. Virtualization

Virtualization is a key technology used in many trusted computing solutions to provide strong isolation for the TPM-measured software. It allows a single physical host to share computing resources between multiple operating systems [11]. Each operating system runs in a *virtual machine* of its own, where it is made to believe that it has dedicated access to the hardware.

A thin layer of software called 'virtual machine monitor' operates on top of the hardware to isolate these virtual machines and mediate all access to the physical hardware and peripherals. A virtual machine runs on a set of virtual devices that are accessed through virtual device drivers. Typically, a highly privileged monitor virtual machine is created at boot time and serves to manage other virtual machines.

Many researchers have studied the benefits of isolating grid jobs and trusted software in their own virtual machines [12; 13]. A malicious job is prevented from compromising or modifying the host platform. It is also much more difficult for one job to influence another, or to read its results. This means that many untrusted jobs can be run together on one platform with a high degree of confidence that their individual confidentiality and integrity will not be affected.

2.4. Commercial Application Whitelisting Systems

Application whitelisting is a technique gathering momentum in commercial security systems. Most implement additional access controls within the operating system to stop unauthorised programs from running. Products from companies such as CoreTrace[14], SolidCore[15] and Bit9 all use application whitelists to create a safer working environment. Bit9 also claim to manage a 'Global Software Registry'[16], containing over 6 billion entries.

2.5. Security Reporting Standards

Several standards for expressing evidential information about a system's security state already exist. The Open Vulnerability Assessment Language (OVAL) [17] [18] is a combination of

a language and content repositories that promote open and public availability of security content and the standardisation of security information transfer. Its framework and vocabulary is based on three XML Schemas, including a System Characteristics schema for representing system information, a Definition schema for expressing a specific machine state, and a Results schema for reporting the results of an assessment. It includes a repository [19] of OVAL tests that are written and scrutinised by the OVAL community. To assess a system, either an existing test is selected or new one written and run using an OVAL compatible tool, producing results that conform to the Results schema.

The XCCDF[20] (Extensible Configuration Checklist Description Format) is designed ‘to provide a uniform foundation for expression of security checklists, benchmarks, and other configuration guidance’[20]. It defines a structured collection of security configuration rules for some set of target systems and a data model and format for storing results of benchmark compliance testing.

The Trusted Computing Infrastructure Working Group [21] has also produced a set of schemas that help with integrity management within a trusted infrastructure. One such schema, the *Security Qualities Schema*, defines a structure for expressing the claims about the security that can be provided by a system. This schema is mainly used by producers of a potential whitelist entry, to state the level of the Common Criteria [22], the NIST FIPS [23] or the ISO 9000 [24] to which the entry has been evaluated.

The Common Results Format (CRF) [25] is an international standard that enhances interoperability by providing a format for exchanging security assessment results. Using publicly available naming schemes such as Common Vulnerabilities and Exposures (CVE) [26], the CRF is capable of reporting the patch levels of a system, its policy compliance, vulnerabilities, configurations and software weaknesses. In addition, the CRF is capable of including standardised, detailed checking logic and low level results as well as references to result data from XCCDF and OVAL assessment. It also defines a set of metadata that defines the creation of the document, including the software that produced the document and timestamps related to its creation.

The Security Content Automation Protocol (SCAP) [27] is an effort by the NIST (National Institute of Standards and Technology) to standardise formatting and nomenclature for communicating security information about software flaws and security configurations, together with any reference data. This standard defines a protocol composed of six components:

1. Common Configuration Enumeration — for expressing system security issues
2. Common Platform Enumeration — for expressing product names and versions
3. Common Vulnerabilities and Exposures — for expressing security-related software flaws
4. Common Vulnerability Scoring System — for measuring severity of software flaws
5. XCCDF — for specifying checklists and reporting checklist results
6. OVAL — for specifying low-level testing procedures used by checklists

Its use includes patch installation verification, security configuration verification, vulnerability scoring and intrusion detection. In Section 5.2 we explore how these standards could be used to define a common message format for sharing whitelist information between different domains.

3. Trusted Distributed Systems: Establishing a Consensus View

In this section we review existing work on trustworthy grid and service oriented architectures. We do this for two reasons, firstly, to identify similarities between systems in order to create a

reasonable common abstract model. This is useful for later discussion, and necessary as these ideas have yet to be a real system. Secondly, to demonstrate that existing research has shortcomings in the area of whitelist management, to motivate our work and draw out requirements for a solution. Finally, we include a brief overview of other related work.

3.1. Identifying Key Components of an Emergent Consensus Architecture

Although there are many small differences, most research on trusted grid architectures has the following components.

Grid Job Submission Work is submitted in the form of jobs, created by a user and run on a participant ('grid node') platform [28]. These jobs are sent either to distribute a complex computation or to query a data source at the node. The emphasis in a *trusted* grid is on job confidentiality [29] and/or integrity [4].

Integrity-based Access Control The fundamental concept of these systems is that the trustworthiness of a platform is defined in terms of an attestation of software and hardware. The common problem being solved is that of a 'malicious host' [4], where one participant either steals job secrets or alters results in a way that violates the job owner's security requirements. For our common model, we also consider attestation in terms of properties and low-level hashes.

Attestation Tokens Each participant in a grid has a credential called an *attestation token* [28; 30; 1; 31] which will contain identity information and the public-half of a TPM key. The private half will be bound to the platform's TPM and certain PCR values. Job submission and service requests can be encrypted using the public key, guaranteeing that the node will only be able to process the job if it is in the correct configuration. This is referred to as a 'sealed-key' approach [28].

Central Management Several authors consider the use of a central server to manage and distribute tokens [1; 30]. We continue with this notion, particularly as many real grids [32] have such a feature for reasons such as policy enforcement and authentication. However, it is generally the case that in a virtual organisation spanning many domains there will be a federation of these central servers (perhaps one per domain) [31].

Property-based Attestation As discussed in Section 2.2.2, standard (or 'binary') attestation lacks useful semantic information, making attested PCR values difficult to interpret. It is for these reasons that many trusted architectures [33] use property-based attestation (or suggest its use [34]) in order to simplify trust decisions based on platform configurations. The task of translating hashes to properties is often given to a third party [2].

Job Delegation Both Yau et al. [1] and Löhr et al. [28] make delegation a key feature of their systems. This allows the recipient of a grid job to pass it on to other trustworthy nodes.

Minimised Trusted Computing Base Researchers are keen to point out that the overall trustworthiness of a system is largely dependent on the size and complexity of the Trusted Computing Base (TCB) [35]. This is because a larger TCB is more likely to contain security bugs, which would allow an intruder to bypass any access controls and gain direct access to what is running on a platform. As a result, large, monolithic operating systems are often avoided [36; 37].

Job Isolation A common method of protecting nodes from being damaged by a submitted job is to isolate the execution of the job from the platform itself. This can be done through sandboxing or, more often, hardware or software virtualization [38; 31; 33] (see Section 2.3 for details).

3.2. Missing Components: Whitelist and Policy Management

Much of the literature cited above proposes novel architectures for exploiting the benefits of Trusted Computing in distributed systems. The use of attestation is suggested to make sure that a grid node is in the correct configuration before being trusted with a job. However, there are a number of management issues when it comes to maintaining a list of correct configurations in a large, cross-domain system. These have not been considered in detail.

Several proposed systems assume that the job owner, an individual host on the system, is capable of managing a list of known trustworthy values [28; 29]. We believe that this is unrealistic, because any list of values will require constant modification and update. The average user will be unable (and unwilling) to support this. Often, therefore, the problem is passed on to a trusted third party [2; 33], but this only centralises the task, and does not provide any insight as to how such a third party would operate. Sometimes multiple trusted third parties are used [39], all of whom issue certificates for different applications. This seems sensible, but increases the burden on the user, as they must know about many certificate authorities.

Having a central authority — perhaps the domain administrator — maintain the whitelist appears to be a reasonable solution. They could be responsible for collecting data from a variety of sources and even doing their own testing. This is described by the TCG as an aggregation service [21] and is suggested or implied in a number of projects [1; 30; 38; 40]. Indeed, Munetoh et al. [41] propose that data is collected from a package management system and combined with a vulnerability database, in order to rate systems according to known weaknesses. Sailer et al. [42] use a package management database to make sure remote users are keeping their platform at an appropriate patch level. The database is updated whenever patches are issued, so that old versions are removed from the whitelist and new versions added.

However, we believe that the central authority solution raises new problems when applied to a system spanning multiple administrative domains. Different institutions will have different selections of software, with some custom applications and scripts. While the administrators of one domain may have a thorough list of all programs commonly available to their users, they are unlikely to know about all the software running in another domain. An example illustrates our point. Imagine a grid consisting of several universities, each of which is managed by their own IT department. If a job is submitted from Oxford University, with a confidentiality requirement, then the expectation is that this will be enforced via the Oxford Universities' whitelist of acceptable configurations. However, if this job arrives at Cambridge University, then the machines are likely to be running several unknown programs. As the authenticated boot mechanism (see Section 2.2) relies on being able to identify *every* executable, this is obviously a problem and the machine cannot be trusted. Furthermore, even if some level of homogeneity is established, if Cambridge University IT department is more diligent in updating and revoking software patches than Oxford, then the same problem will arise. The two potential outcomes are both unacceptable: either no jobs can be distributed because no trust can be established, or integrity reporting is abandoned altogether.

To avoid this scenario, we suggest in the following sections a number of steps to facilitate *reference integrity measurement sharing* between domains. We assume that each domain will have

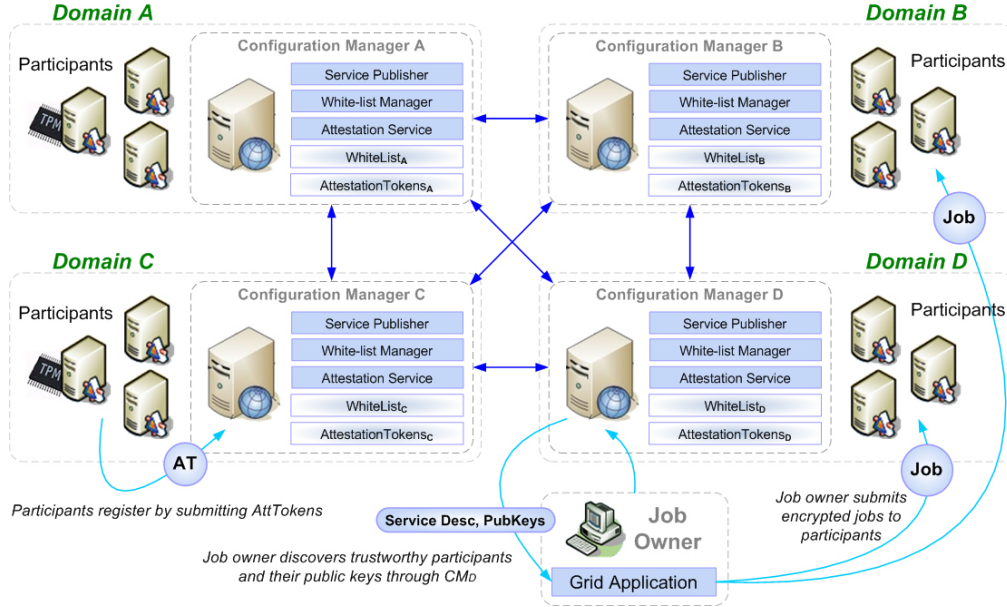


Figure 1: Abstract View of A Trusted Grid with Centrally Managed Whitelists

a dynamic list of trusted software, and will want to inform each other whenever a new update occurs. However, this relies on more functionality, including interoperability and shared semantics between domains, and a notion of *justification* when updating or modifying an executable.

3.3. Abstract View

In Figure 1 we show an overview of how a trusted grid system would behave when application whitelists are managed centrally. The features to note are the central authority, which we refer to in future sections as a *Configuration Manager*, the use of attestation tokens, and the service discovery process. In the next section, we will specify requirements which extend the central authority to deal with whitelist management issues.

4. Requirements

Building on the existing structure of trusted distributed systems, we analyse common use cases that should expose additional requirements. We assume the existence of a central authority, the *Configuration Manager (CM)*. There will be one CM per domain, each of which will hold a whitelist of trustworthy platform configurations.

4.1. Use Cases

4.1.1. Joining the Virtual Organisation

A participant wishes to join *Domain A* and advertise their resources or services in the virtual organisation. In the first step, a request is sent from the participant's platform to the externally facing *service publisher* in *CM_A*. This request consists of an attestation token (see Section 3.1)

and a service request. The service publisher will verify the trustworthiness of the participant platform: its PCR values are checked against *Whitelist_A*. If the platform is trustworthy, the token is added to the internal token repository and the participant is notified. Otherwise, a failure report describing the reasons for unsuccessful registration will be forwarded to the participant.

4.1.2. Job Submission

A researcher in *Domain A*, who is concerned about the integrity and confidentiality of their job, wishes to submit it to only the trustworthy participants.

The researcher sends a request to the service publisher (in *CM_A*) asking for service descriptions of trustworthy participants. *CM_A* requests attestation tokens from other CMs. This request will contain the service requirements, specifying the required application and hardware capabilities. The recipient CMs will filter their list of tokens, selecting the relevant ones, and return them to *CM_A*. *CM_A* iterates through each token and verifies the trustworthiness of the advertised configurations by comparing the PCR values against the local *Whitelist_A*. Only those with acceptable configurations will be selected and merged with tokens from *Domain A*. The service descriptions and the public keys are then sent to the user. These public keys will be used to establish secure sessions with the selected participants. The job secrets can be encrypted with session keys and safely distributed over the network.

4.1.3. Vulnerability Discovery and Patching

The configuration manager of *Domain A*, *CM_A*, receives information about a vulnerability, perhaps from the vendor, a public forum or through internal testing. Upon receiving this information, *CM_A* evaluates the severity of the vulnerability and changes the affected whitelist entry. This may involve removing the entry, or giving it a ‘grace period’ and scheduling its removal later.

Next, *CM_A* informs other domain configuration managers, *CM_B* and *CM_C* in this case. The two CMs perform their own analysis and decide the severity of the vulnerability. Taking into consideration the objectives of the domain and any policies governing its operation, each domain decides on how to update their whitelist.

At this point, the domains will have different states regarding similar entries in their whitelists. Some will allow the services affected by the vulnerability to continue while others will stop using those services. Each domain may wait for an official patch to be announced, or develop their own, depending on the nature of the vulnerability. When *CM_A* receives a patch, it verifies that the patch fixes the vulnerability and does not cause other problems. It then instructs all the participants within its domain to patch their systems. In parallel, it updates the local whitelist with an entry for the new patch, and removes any old entries. *CM_A* will have old copies of attestation tokens for participants, so must inform local participants (affected by the new patch) to re-register.

This update has now effectively cut-off any unpatched participants from other domains from being used within *Domain A*. This is because *CM_A* will filter out these participants when checking their configurations against the updated whitelist. This also means that other domains will not be able to use the services offered in *Domain A* because they receive configuration tokens for patched *Domain A* participants — they cannot verify these tokens since their whitelist is not updated. To fix this, *CM_A* informs *CM_B* and *CM_C* about the patch. Upon receiving the patch information, *CM_B* and *CM_C* verify the patch, update their whitelists and force the participants in the respective domains to re-register.

4.2. Functional Requirements

Having identified some of the use cases pertaining to operations of the Configuration Manager, we now consider the functional requirements of the CM and the participants:

1. All participants should be discovered through the domain CM, as only it can decide which are trustworthy. Tokens should never be shared directly.
2. The CM must only collect local attestation tokens from participants that it considers trustworthy.
3. The domain CM must be able collect tokens from other CMs.
4. For each request, the domain CM must check all collected token against its whitelist and policies.
5. Each CM should have an interface for system administrators to modify the whitelist.
6. Upon whitelist or policy update, the CM should inform other domain CMs of the changes. There may be exceptions to this rule if no other domain will be affected.
7. Domains should be as synchronised as possible with each other, with regard to whitelists and the state of participant platforms. The greater the synchronisation, the more services will be available to users.

4.3. Interoperability Requirements

Because each domain is likely to use different software and tools, it will be important to standardise communication. Service descriptions and attestation tokens should have the same format, as well as communication about vulnerabilities and policy changes. The following requirements focus on this last issue:

8. Vulnerability and policy update reports should have a common syntax and semantics.
9. Every CM should be able to interpret these reports and present them to domain administrators.
10. Every CM should be able to generate reports.
11. As much information as possible should be shared between domains, to increase the speed at which administrators can decide about patches. Reports should therefore allow extension where necessary.

4.4. Whitelist and Policy Structure Requirements

The entries in each domain whitelist will be used to verify attestation tokens. Entries must be represented in a way that makes token verification possible. Furthermore, metadata about where the entry came from, and why it has been added will be useful when reporting updates to other domains. These issues are captured by the following requirements

12. Every entry should conform to the TCG Core Integrity Manifest Schema [21] which contains a Reference Integrity Manifest (RIM) for validating attestation tokens.
13. As an extension, every entry must include any information about how it was generated, what tools were used (e.g. compilers) and the environment in which it was produced.
14. All entries will be linked to a policy, which will be applied when validating an integrity report.
15. Each policy should link to a *validation report*, which contains a list of tests which validate (or invalidate) the entry, and the reasons for the policy. This might include a URL to a vulnerability report, for example.
16. Each validation report should be as thorough as possible, so that when it is passed on to another domain, they do not have to repeat tests.

5. New Components

This section informally specifies the additional components needed to meet the above requirements. This includes the Configuration Manager and its sub-components, and the data structures needed to satisfy the interoperability requirements.

5.1. The Configuration Manager

In Section 3 we described an abstract view of the modified trusted grid system that relies on the Configuration Manager for managing ‘known good’ platform configurations (see Figure 1). The CM is ultimately responsible for deciding which platform configurations are considered trustworthy, and for controlling domain membership.

5.1.1. Domain Membership

To become part of the local domain, a participant registers with its CM by submitting an attestation token. The token content is shown below. It includes the attestation identity key (AIK) for the platform, along with a credential issued by the certificate authority ($\{\text{cred}(AIK)\}_{CA}$). This AIK is used to sign a credential for the public key (K_{pub}), which states that the key has been sealed to a PCR value that corresponds to a trustworthy authenticated boot process. The full description of the authenticated boot process (and all applications that are part of it) is given in the *Measurement Log*. In addition, a service description is included, signed by the private half of the sealed public key, demonstrating that users of this service should use this key when submitting jobs. We assume that the attestation token will be sent to the CM in an authenticated transport session, and have not included any timestamp or nonce.

$$AT = (\text{Measurement Log}, AIK, \{\text{cred}(AIK)\}_{CA}, K_{pub}, \{\text{cred}(K_{pub})\}_{AIK}, \{\text{Description}\}_{K_{priv}})$$

The authenticity of the sealed key can be checked by validating the AIK signature. The CM can then verify the trustworthiness of the platform by comparing the *Measurement Log* to the local whitelist of acceptable configurations for this service type. If the platform is trustworthy, its attestation token is added to the CM’s repository. This satisfies requirement 2.

5.1.2. Structure of the Configuration Manager

The CM performs a range of security and whitelist management functions:

- An internal *attestation service* is responsible for performing all attestation related functions to ensure that only trustworthy participants become available to the users. It will fulfil all the Whitelist and Policy Structure Requirements as well as access control functional requirements (4, 2).
- An external *service publisher* provides the necessary APIs for both the users and the participants to make requests to the CM: the users will submit service discovery requests; and the participants will submit requests to register and advertise their services/resources through the CM. This aims to satisfy requirements 1 and 3 and will use the attestation service.
- An external *whitelist manager* allows the domain administrators to efficiently update and share the whitelist entries. This is responsible for meeting functional requirements 5, 6, 7 as well as all interoperability requirements.

To conform to existing standards, we imagine that the CM would be implemented as a WS-ServiceGroup [43]. Each participant would then be a member of this CM's group, and have a ServiceGroupEntry that associates them. An entry would also contain service information by which the participation in the CM is advertised. The membership constraints would be simple, requiring only a valid token (representing trustworthy configurations), identity and description of available services. These tokens would be categorised and selected according to the types of services they advertise. We assume that there is a public key infrastructure available to verify the participants' identity.

5.1.3. *One Configuration Manager Per Domain*

We have assumed that each administrative domain will have its own CM. This is realistic for a number of reasons. Firstly, this is how many existing virtual organisations work, as each domain wishes to keep control of its data and services. Furthermore, administrators of each domain are in the best position to assess and manage their participants. There is also an issue of scale, as one CM is unlikely to be able to deal with all requests.

5.1.4. *Security Concerns*

It is not hard to imagine instances where participants change their platform configurations after joining the CM, for perfectly honest or malicious reasons. Nor is it hard to imagine scenarios where the whitelist is not kept up to date and untrustworthy participants manage to register with the CM. In both cases the CM could send service descriptions of untrustworthy participants to the user. The CM is being relied upon to manage the whitelist in a secure and timely manner, and perform attestation correctly. Therefore, the users should be provided with mechanisms to verify the identity and current state of the CM, and know that its security functions are working properly. One of the objectives of our work is to identify potential security threats and risks that might prevent users from establishing trust with a central authority like the CM, and suggest possible mitigation techniques (see Section 6).

5.1.5. *Mapping to TCG Platform Configuration Components*

In the TCG model of runtime attestation [21], integrity reports from the target platform are checked by a *Verifier*, which has access to a *Policy Database*, *Configuration Management Database* and *Reference Manifest Database*. These databases hold 'known good' configurations for platforms, known as *Reference Integrity Measurements* (RIMs) and are queried with information from the incoming integrity report. Should the report contain unknown applications, or applications running in an unexpected configuration, then the verifier tells the relying party not to trust the target platform. An overview of this process is shown in Figure 2.

In our proposed enhancements, these functions are all performed by the CM. In addition, it needs to communicate changes to these databases to other CMs, and deal with incoming change requests. Furthermore, we believe that the amount of metadata stored about each reference measurement should be extended, to include more details about how an application has been tested. The rest of this section describes how existing formats and standards (see Section 2.5) can be used to perform these functions, as well as those mentioned in previous sections. Because we are not describing an existing implementation, the proposals should be considered as a *guideline* for one approach to solving this problem, using existing formats and languages wherever possible.

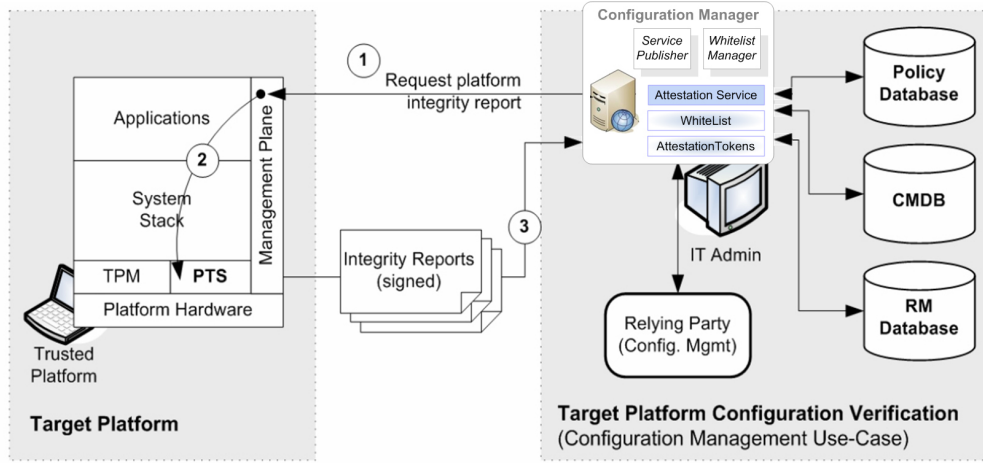


Figure 2: Using the CM as a TCG Configuration Verifier (Modified Figure 22 from [21])

5.2. Inter-domain Communication: Standards and Formats

As it has already been pointed out, multiple CMs could introduce availability issues depending on the level of inconsistency between their whitelists. For example, if CM_A is more active in inspecting software vulnerabilities and updating the whitelist entries than other domains, attestation tokens collected from CM_B , CM_C , CM_D are likely to be classified as untrustworthy by CM_A and their services will not be advertised to the users in *Domain A*.

In order to minimise the level of inconsistency, the whitelist manager (in the CM) needs to support functions that would enable efficient discovery and sharing of whitelist changes. Domains must therefore communicate vulnerabilities and whitelist entry updates using a common format and semantics. This section suggests what the content of whitelist entries would be, how entry update messages would be structured, and how they should be processed.

5.2.1. Entry Validation Information

Every whitelist entry, where possible, should be associated with validation information including known vulnerabilities and test reports. It may also include details of bugs which this version of an entry claims to have fixed. This information will support the entry's inclusion on the whitelist as well as provide information that helps CMs and domain administrators decide what policy to attach to the entry.

In a cross-domain environment, it is inevitable that different domains will use different tools to validate their whitelist entries. Each domain will specify the checks performed using their own algorithms. It is also likely that each tool will generate reports using a different format. To meet requirements 8 and 11 and support interoperability, all of these details must be expressed in a way that can be understood by other domains. Any of the formats described in Section 2.5 may be used to achieve this. However due to SCAP's ability to include OVAL and XCCDF coupled with its automation capabilities, we suggest the use of SCAP-compliant tools, many of which exist [44][45][46], for validation.

Finally, when the CM is verifying the participants, it uses a policy database to decide how to respond to a recognised platform configuration. This may well be a simple whitelist, where

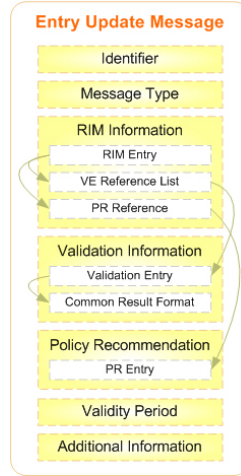


Figure 3: Entry Update Message Format

a ‘trusted’ response is returned only if all entries are found in the current database of RIMs. However, a more complicated policy might include time-based requirements, dependencies, and signature schemes. The TCG intentionally does not define its own policy language, but we expect a language such as XACML [47] would be used.

5.2.2. Entry Update Message Format

While certain entries of the whitelist will remain static, others will change as vulnerabilities are discovered, patches released or new functionality added. These changes will necessitate communication among the CMs to allow continued provision or request for resources for participants. Each domain may adopt a different internal data representation such as validation information or whitelist entry policies. To enable interoperability a common messaging format is essential. Our entry update message format aims to provide a complete set of information that enables recipients to decide what course of action to take.

The message format is shown in Figure 3. A description of the content of each field is given below:

Identifier will be composed of the CMs identity, time-stamp and message number.

Message Type specifies whether the message is about a new software release, vulnerability discovery, patch release, general information, domain policy change or participant update. This allows domain administrators to selectively act on messages of their interest. For example, one domain may only be interested in knowing when other domains update their participants.

RIM Information is a link to the relevant reference integrity measurement, defined by the TCG, which includes fingerprint values for each executable file.

Validation Information consists of entries that will be useful in validating one or more of the included RIMs. Each RIM refers to a list of relevant validation entries using the VE

(Validation Entry) Reference List. Each validation entry references a CRF (Common Result Format) document that includes the OVAL and/or XCCDF results from the SCAP-compliant tool.

Policy Recommendation contains the recommended policy to apply to this RIM when verifying platforms. This might be a simple ‘allow’, or be more complicated, including dependency information on other entries (e.g. ‘allow this version of MatLab when running under Windows’). We suggest that this field might be an XACML policy.

Grace Period specifies how long domains have until the originator CM begins enforcing their entry update. It may give a short interval to allow other administrators to assess and respond to this update message, and perform any patches they believe necessary. At the end of this period, any RIMs that have been invalidated by this update will cease to be considered acceptable by the original CM.

Administration Information includes other supporting messages from the domain administrators.

Message Metadata represents meta data related to the message including the creation time and the identity of the system administrator responsible for it.

5.3. *Whitelist Manager*

The *whitelist manager* component of the CM is responsible for dealing with entry updates as well as receiving updates from other domains. It also communicates with the service publisher within the CM to indicate any changes or updates required on the participants. This is specified in requirements 10 and 9.

When a new update is required, information is gathered from the reference measurement, policy and validation databases. In addition, the administrator should be able to add descriptions and other information. Once collected, the message is formulated according to the *Entry Update Message Format* specified above. This will satisfy requirements 11 and 16.

When a new update is received, the source of a change request must be checked (is it a valid CM?). This update is then presented to the domain administrator, flagging all existing entries in the local whitelist that will be affected. It is then the responsibility of the administrator to perform any necessary actions. However, the whitelist manager must make it easy for the metadata about the request to remain attached to any new entries.

5.4. *Further Implementation*

Given that no real ‘trusted grid’ system has been deployed yet, it was beyond the scope of this work to implement the entire grid system (i.e. all of the components identified in the Abstract View, Figure 1) as well as the Configuration Manager and its sub-components. Instead, we focused on establishing a sound implementation guideline for software developers, in case the need for implementing a trusted infrastructure arises in the future.

6. Security Review

With respect to the earlier use cases (see Section 4.1), a threat analysis was conducted using the Integrating Requirements and Information Security (IRIS) framework [48] (see Appendix

A). We identified valuable assets, stakeholders, and threats to those assets and their risks. Risks were rated using the likelihood and severity tables within IEC 61508 (see Figure 8).

From these, we identify the key security requirements and suggest a number of possible countermeasures — based on the trusted computing capabilities — to protect those assets from the threats. We assume the domain administrators are trusted, and hence the insider threats are not considered in depth in this review.

6.1. Sealed Storage

An attestation token consists of a public key credential for which the private half is bound to the platform's TPM and particular software state (see Section 5.1.1). The private key is strongly protected by the TPM and can only be made available via the TPM that sealed it. This 'sealed-key' approach is attractive from the security point of view, since secret data can be encrypted using the public key and safely distributed over unprotected communication channels. Those trusted by the data owner will be capable of decrypting the secret data only if the platform configurations have not been modified.

Requirement 1 — job secrets should be confidentiality and integrity protected upon job submission.

In response to a service-discovery request, the user receives service descriptions and public keys of trustworthy participants. The jobs can be encrypted using a symmetric session key (created locally at the user platform), which in turn, can be encrypted using the public key of the target participant (6, 7, Figure 4). Since the private half is protected by the target participant's TPM, this would be sufficient to ensure the confidentiality and integrity of any sensitive models or data being transferred as part of the job. The purpose of using a symmetric key is to improve the overall performance of cryptographic operations.

Requirement 2 — any changes made to the participant's platform configurations, after being registered with the CM, should be detected.

Sealing also enables detection of any modifications made to platform configurations after being registered with the CM. Since any secret data (e.g. a session key) sent from the CM would have been encrypted with the platform's advertised public key, if the platform has been modified to reflect an untrustworthy configuration, the private half will not be accessible to decrypt the data. The communication will break at this point and the CM will be notified. Once modification is detected, the platform's attestation token will be removed from CM's token repository, and the participant will be asked to register again.

6.2. Minimising the Trusted Computing Base and the Attack Surface

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

Requirement 3 — the attack surface of the CM software should be minimised.

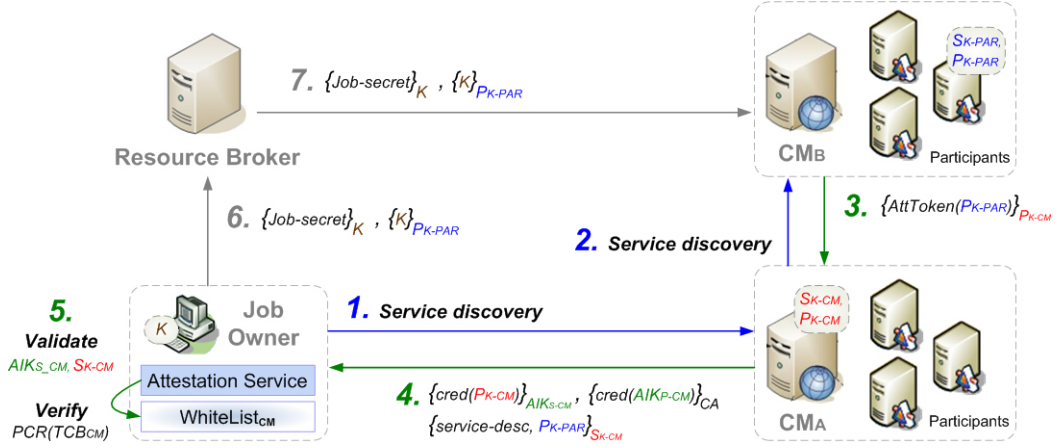


Figure 4: Job Submission Workflow

It would therefore make sense for the CM software, especially the externally facing services, to be small and simple to minimise the chance of it containing any security vulnerability. Formal methods can be used to design and implement these services with a high degree of assurance. For example, FADES (Formal Analysis and Design approach for Engineering Security) [49] integrates KAOS (Knowledge Acquisition in autoMated Specifications) with the B specification to generate security design specifications. A security requirements model built with KAOS is transformed into equivalent one in B, which is then refined to generate design specifications conforming to the original requirements. These procedures help software developers to preserve security properties and detect vulnerabilities early during requirements. As a defence-in-depth measure, external services can be isolated in a separate compartment (e.g. using virtualization) to limit the impact of any vulnerability being exploited.

Any modification made to the services themselves (e.g. by insiders) will be caught when the system is rebooted as the platform configurations (stored in the PCRs) would change; the private key sealed to this PCR value will not be accessible to decrypt any incoming secret data.

6.3. Privacy-Preserving Remote Attestation

Requirement 4 — verification mechanisms for the attestation token should identify reported configurations that are different to the actual platform configurations.

Remote attestation (see Section 2.2.2) can be used to ensure that the reported configurations match the actual platform configurations measured at the time of authenticated boot. When the CM receives the attestation token, its attestation service will first check the signature on the AIK credential, $\{cred(AIK)\}_{CA}$, to verify that the *AIK* has been generated from a valid TPM. The *AIK* is used to validate the signature on the public key credential, $\{cred(K_{pub})\}_{AIK}$, to verify that reported PCR values represent the actual values stored in the TPM.

Requirement 5 — participants' platform configurations (Measurement Log and PCR values) should not be revealed to the users or adversaries.

One of the drawbacks of remote attestation is that the privacy of the attesting platform (their software configurations and properties recorded in the *Measurement Log*) is disclosed to the remote verifier. Property-based attestation (see Section 2.2.2) has been suggested as a possible way of avoiding this disclosure, although, platform properties themselves may also reveal private and security-related information about the platform. The main concern here is that these configurations or properties, if made available to the public, could reveal security weaknesses of the platform which might be exploited by an attacker.

We believe our approach solves some of these privacy concerns by shifting all of the remote attestation functions to a smaller number of trusted CMs, and effectively making them the only components required to see the platform configurations. All token transfer operations are encrypted with the recipient CM’s public key to prevent traffic sniffing and man-in-the-middle type of attacks (3, Figure 4). Further, the CM’s response message to a service-discovery request does not include the *Measurement Log* or PCR values — only the service descriptions, public keys, and the CM’s own attestation token (used for CM verification) are returned to the user (4, Figure 4). Therefore the participants’ platform configurations are never disclosed.

6.4. Verification of the Trustworthiness of the CM

Privacy-preserving attestation relies on the correct operations of the CMs. The only way the user is going to trust the service descriptions, without looking at the platform configurations themselves, is by verifying the identity and trustworthiness of the CM.

Requirement 6 — the users should be able to verify the identity and trustworthiness of the CM and the service descriptions.

First, we assume there is a public key infrastructure available to verify the CM’s identity. Second, the security properties mentioned in Section 6.2 should be sufficient for the user to establish trust with the CM, provided that a reliable attestation mechanism is in place. Here, we describe how the state of the CM’s Trusted Computing Base could be verified by the user.

Much like the participant’s token, the CM’s attestation token — sent to the user as part of the response message — contains sufficient information for the users to verify the security configurations the CM. This includes a public key credential that identifies the private half as being sealed to its $PCR(TCB_{CM})$, and a *Measurement Log* of the authenticated boot process. The public key and service description pairs (of trustworthy participants) are signed inside the CM’s TPM using the sealed private key.

We imagine that a small, relatively static whitelist for the CM will be managed locally by the user platform. When the message arrives, the trustworthiness of the CM can be verified by comparing the *Measurement Log* to the $Whitelist_{CM}$ (5, Figure 4). If the CM is trustworthy, the signature on the service descriptions can be checked using the CM’s public key: a valid signature proves that the service descriptions are from a CM configured with the reported TCB_{CM} . Once all of these security checks pass, the user may trust the participants listed in the service descriptions and submit jobs to them.

6.5. Job Isolation with Hardware Virtualization

We identified a possible threat of a user submitting a malicious job (e.g. containing a virus) to compromise the participant’s machine (see Table A.2). From the participant’s perspective, the code is untrusted but running on their trusted host, and they need to be convinced that the code will not cause harm to their system.

Requirement 7 — the participants’ systems should be protected from rogue jobs.

As a possible mitigation technique, we suggest the use of hardware virtualization to isolate jobs from the rest of the platform. Numerous design efforts have been made to enable hardware virtualization. In such designs the virtual machine monitor ensures that all memory is cleared before being reallocated, and each virtual machine has its own dedicated memory and disk space. Both Intel and AMD processors now provide native hardware support for full, efficient virtualization [50].

The majority of current grid middleware solutions, including the Globus Toolkit [51], rely on operating systems’ access control mechanisms and memory protection to manage isolation between user accounts. However, millions of lines of code running in a mainstream operating system must be trusted to enforce these policies correctly; a single bug in any one of the privileged components might be enough for an attacker to hijack it, elevate its privileges, and take control of the host. Operating system level isolation techniques like sandboxing [13] are also vulnerable in this sense.

Virtualization, on the other hand, is capable of providing a much stronger isolation through the relatively smaller virtual machine monitor and monitor virtual machine (see Section 2.3). Malicious software (through privilege escalation) would have to compromise both components — which are designed to resist such attacks — in order to break the isolation [52].

6.6. Secure Communication Between the CMs

One of the advantages of the *entry update message* is that it provides sufficient information for the domain administrators to make decisions without going through much further verification. A problem arises, however, when an adversary accesses this message and interprets the state of the entries described. Validation information could reveal security vulnerabilities such that could be exploited by attackers.

Requirement 8 — the confidentiality of the entry update messages needs to be protected; each CM should be able to verify the integrity and authenticity of incoming messages.

To prevent this, the CMs’ attestation tokens can be exchanged (and verified) in advance and used to establish a secure public key infrastructure between the CMs. A message can be signed with the originator CM’s private key and encrypted with the recipient CM’s public key (for which the private half is sealed to its TCB). In particular, *PolicyRecommendations*, *ValidityPeriod*, *AdministrationInformation*, and the validation results documents should always be encrypted. Some attributes, however, do not expose any state information and can be left out from encryption: *RIMInformation* and *ValidationInformation*, for example, only describe the entity and references to the validation attached to it. All of the attributes need to be signed to protect the message integrity.

Upon receiving the message, the *whitelist manager* will decrypt it using the sealed private key and validate the signature using the sender’s public key. A valid signature verifies the authenticity and integrity of the message.

7. Example Integration Into Existing Systems and Observations

In this section, we consider the Trusted Grid Architecture [28] and the UK National Grid Service [53] and illustrate how our approach can be integrated into their work with minimal effort. We further discuss the observations made from these example integrations.

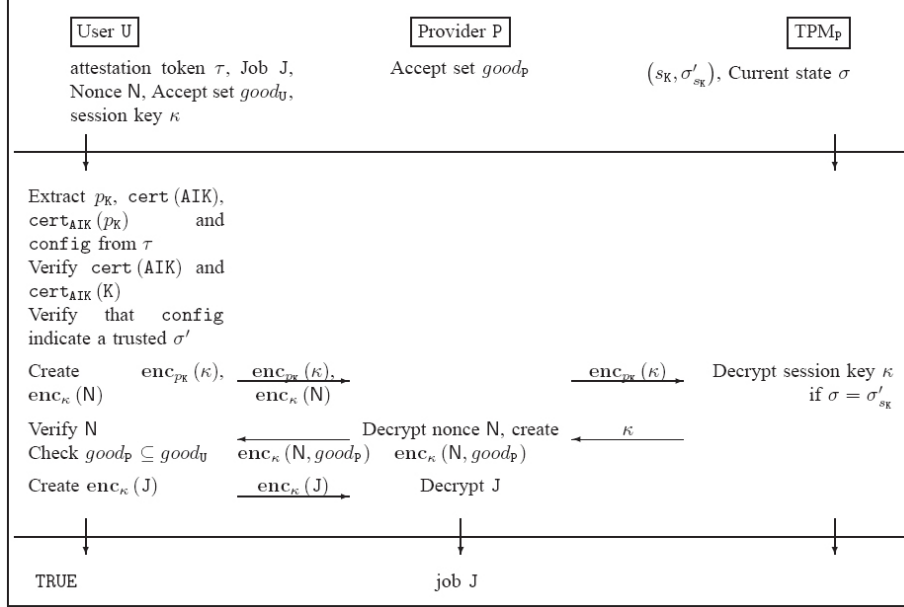


Figure 5: “Submission Protocol”, Figure 3 from [28]

7.1. Modifying the Submission Protocol in the Trusted Grid Architecture

In the Trusted Grid Architecture (TGA) *users* collect attestation tokens of service providers and verify their platform configurations against a set of known-good values ($good_U$). TGA uses its own job submission protocol (see Figure 5) which allows the user to check two important properties about the service provider:

1. Is the provider’s platform in the same state as advertised in their attestation token?
2. Does the provider’s list of known-good values ($good_P$) — which specifies all of the acceptable states of other providers the original provider could select upon delegating the job — satisfy the condition $good_P \subseteq good_U$?

If both of these conditions are satisfied, the user submits the job to the provider knowing that the job will only be *delegated* to other service providers whose states (platform configurations) also satisfy $good_U$.

The main concern with their approach, however, is that the burden of managing the whitelists ($good_U$, $good_P$) rests on the users and the service providers. In Section 3.2, we identified some of the consistency and scalability implications this might have in large-scale distributed systems. We now show how our Configuration Manager can be integrated into TGA to reduce this burden.

In the modified protocol (see Figure 6), the CM operates in between the user and the service provider, and manages both $good_U$ and $good_P$ (i.e. the whitelists). The CM, on behalf of the user, carries out all the necessary steps for attestation and selects trustworthy providers. This step also involves checking that all of the states specified in $good_P$ are trustworthy — this is possible since the CM has a complete list of all trustworthy states. The public key and service

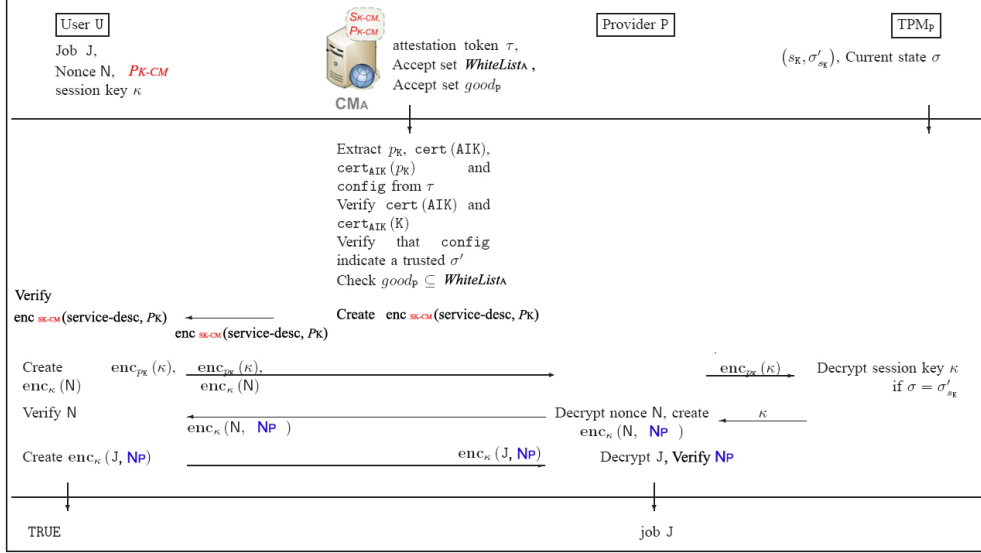


Figure 6: Integrating the CM into the “Submission Protocol” (Modified Figure 3 from [28])

description pair, signed with the CM’s private key, is sent to the user. The user no longer has access to the attestation token, and this prevents the participant’s platform configurations from being disclosed to the user. A session key is exchanged with the provider in the same way as the original protocol. The returning message is slightly different as it no longer includes $good_p$ since the state verification process is now handled by the CM; instead, the user receives the provider’s nonce (N_p).

7.2. Integration into the National Grid Service

The UK National Grid Service (NGS) is an academic research grid composed of computing and storage infrastructures from multiple institutions across the country. The aim of the NGS is to provide a robust, reliable and trusted service using ‘open standards’ based access to the full range of computation and data based research facilities required by UK researchers [53].

The NGS includes four core sites and five partner sites, each of which contributes to the provision of computational and storage nodes. The nodes from the core sites use an identical software stack and a similar filesystem, allowing the users to transparently access the resources within the grid; the partner sites, on the other hand, provide a more heterogeneous environment. Each site uses an information service, known as the Grid Resource Information Service (GRIS), to publish their static information as well as dynamic information about the service or resource availability. GLUE Information Schema [54] is used for publishing such information. The Berkeley Database Information Index (BDII) [55] (from the Enabling Grids for E-science (EGEE) project) is used to collect information from all sites. The collected information is then made available to the users using the Lightweight Directory Access Protocol (LDAP) [56].

We imagine that our CM could be deployed at each site, along side the GRIS, to publish security sensitive, whitelist entry update information (see Figure 7). Our *entry update message format* can be used, as an extension to the GLUE schema, to facilitate a standardised and unambiguous

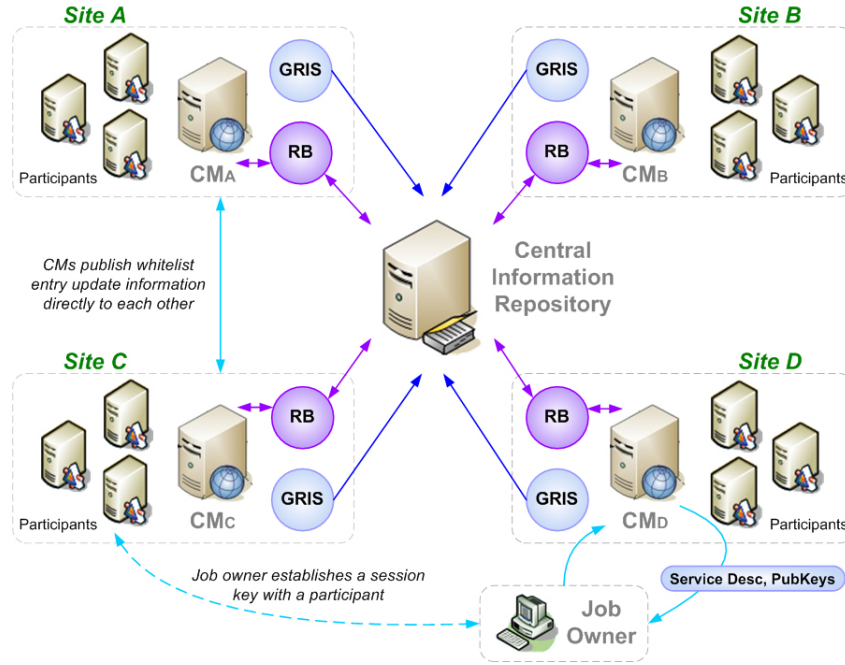


Figure 7: An Example Integration with the National Grid Service

exchange of entry update information between the sites. To prevent various validation results and recommended policies from being disclosed to the public, these information will be sent directly to the recipient CMs (and not through the central repository), using a secure public key infrastructure as suggested in Section 6.6. No modifications will be required for the operations of existing GRIS.

Consider a job submission scenario. A researcher, who wishes to run their job in the NGS, first submits a job description to the local CM. The CM uses the resource broker to discover resource availability in the NGS, and downloads attestation tokens for those with available resources. It then selects trustworthy resource providers, and sends their public key and service description pairs to the researcher. The researcher establishes a secure session with a particular resource provider (using the approach described in Section 6.1) and submits an encrypted job. The researcher may delegate the management of the session key using components such as MyProxy [57].

7.3. Observations

Perhaps the most significant overhead of our system is the cost of upgrading existing sites to support the new infrastructure. This involves installing the Configuration Manager and its various sub components at each site, and standardising the communication mechanisms between them. While this is a large change, the advantage of our infrastructure is that legacy services like the Grid Resource Information Service can still be used without modification. Hence, the user can decide — depending on the level of security required for their jobs — when to use the CM for discovering trustworthy participants.

Despite the security enhancements, the use of the CM and its functions will increase the number of user interactions necessary for submitting a job. Instead of sending a job directly to the resource broker, the user needs to request for service descriptions and public keys from the CM, and establish a secure session with the selected resource or service provider. These extra steps could affect the overall usability and performance of the job submission process. Both of these measures could be improved, however, by delegating the job submission role to the CM and trusting it to submit jobs on behalf. This would be ideal in computational grid systems where the users might not care about where their job travels as long as their sensitive data and results are protected.

Finally, the number of update messages may become significant in a large grid system. Should this be a problem, a central message repository such as the EGEE project's CIC portal [58] could be used as an alternative.

8. Conclusion and Future Work

We have argued that future trusted distributed systems will struggle to implement whitelist-based platform configuration controls unless additional communication is established between administrative domains. This point has been demonstrated through a review of the literature, as well as examples from existing grid systems. We propose a new set of requirements that must be met for this problem to be overcome, and a set of new components which would fulfil them. It is our hope that any future trusted grid system will be able to use our analysis to avoid potential availability and interoperability problems.

In addition, we have thoroughly analysed the security implications, highlighted where the risks lie, and suggested a number of ways to mitigate security concerns. In particular, our system is capable of minimising the exposure of platform configuration details, by delegating to our new component, the Configuration Manager.

In future work, we would like to expand the problem to deal with untrustworthy job submitters, and look at how application whitelisting might solve this problem. We believe that with the rise of cloud computing, where tasks take the form of complete virtual machines, the potential for malicious (or incorrect) jobs is greatly increased.

Acknowledgements

This work is partly funded by studentships from QinetiQ and the EPSRC, as well as a Rhodes Scholarship. We thank Steven Young, Matteo Turilli and David Wallom for their insights on the National Grid Service.

References

- [1] P.-W. Yau, A. Tomlinson, S. Balfe, E. Gallery, Securing grid workflows with trusted computing, in: M. Bubak, G. D. van Albada, J. Dongarra, P. M. A. Sloot (Eds.), ICCS (3), Vol. 5103 of Lecture Notes in Computer Science, Springer, 2008, pp. 510–519.
URL http://dx.doi.org/10.1007/978-3-540-69389-5_58
- [2] A. Nagarajan, V. Varadharajan, M. Hitchens, Trust management for trusted computing platforms in web services, in: STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing, ACM, New York, NY, USA, 2007, pp. 58–62. doi:<http://doi.acm.org/10.1145/1314354.1314369>.
- [3] N. Bessis, Grid Technology for Maximizing Collaborative Decision Management and Support: Advancing Effective Virtual Organizations, Information Science Reference, 2009, Ch. Preface, pp. xvii–xviii.

- [4] A. Cooper, A. Martin, Towards a secure, tamper-proof grid platform, Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on 1 (2006) 8. doi:10.1109/CCGRID.2006.103.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1630843
- [5] Trusted Computing Group, Trusted computing group website (October 2006).
URL <https://www.trustedcomputinggroup.org>
- [6] D. Grawrock, The Intel Safer Computing Initiative, Intel Press, 2006, Ch. 1–2, pp. 3–31.
- [7] Trusted Computing Group, Tpm main specification version 1.2, TCG Published (2003).
URL http://www.trustedcomputinggroup.org/resources/tpm_main_specification
- [8] J. A. Poritz, Trust[ed — in] computing, signed code and the heat death of the internet, in: SAC '06: Proceedings of the 2006 ACM symposium on Applied computing, ACM Press, New York, NY, USA, 2006, pp. 1855–1859. doi:<http://doi.acm.org/10.1145/1141277.1141716>.
- [9] A.-R. Sadeghi, C. Stübke, Property-based attestation for computing platforms: Caring about properties, not mechanisms, in: NSPW '04: Proceedings of the 2004 workshop on New security paradigms, ACM Press, New York, NY, USA, 2004, pp. 67–77.
URL <http://doi.acm.org/10.1145/1065907.1066038>
- [10] E. Shi, A. Perrig, L. van Doorn, Bind: A fine-grained attestation service for secure distributed systems, in: In Proceedings of the 2005 IEEE Symposium on Security and Privacy, IEEE Computer Society, 2005, pp. 154–168.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1425065
- [11] X. Inc., Xen: Enterprise grade open source virtualization (whitepaper), Xen Source (2005).
URL http://xen.xensource.com/files/xensource_wp2.pdf
- [12] R. J. Figueiredo, P. A. Dinda, J. A. B. Fortes, A case for grid computing on virtual machines, in: 23rd IEEE International Conference on Distributed Computing Systems (ICDCS'03), IEEE Computer Society, 2003, pp. 550–559.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1203506
- [13] K. Keahey, K. Doering, I. Foster, From sandbox to playground: Dynamic virtual environments in the grid, in: GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing, IEEE Computer Society, Washington, DC, USA, 2004, pp. 34–42. doi:<http://dx.doi.org/10.1109/GRID.2004.32>.
- [14] CoreTrace: Application Whitelisting For Enterprise Endpoint Control (Website) (August 2009).
URL <http://www.coretrace.com/>
- [15] Solidcore (Website) (August 2009).
URL <http://www.solidcore.com/>
- [16] Bit9 global software registry (website) (2009).
URL <http://www.bit9.com/products/gsr.php>
- [17] The MITRE Corporation, An introduction to the oval language, version 5.0 (2006).
URL http://oval.mitre.org/oval/documents/docs-06/an_introduction_to_the_oval_language.pdf
- [18] R. R. Matthew Wojcik, Tiffany Bergeron, Introduction to oval: A new language to determine the presence of software vulnerabilities (November 2003).
URL http://www.robergewriting.com/documents/OVALIntroductoryWhitePaper-November2003_000.pdf
- [19] The MITRE Corporation., About the oval repository (2009).
URL <http://oval.mitre.org/repository/about/index.html>
- [20] S. D. Q. Neal Ziring, Specification for the extensible configuration checklist description format (xccdf) version 1.1.4 (January 2008).
URL <http://www.csrc.nist.gov/publications/nistir/ir7275r3/NISTIR-7275r3pdf.zip>
- [21] Trusted Computing Group, TCG Infrastructure Working Group Architecture Part II - Integrity Management (November 2006).
URL http://www.trustedcomputinggroup.org/resources/infrastructure_work_group_architecture_part_ii_integrity_management_version_10
- [22] The common criteria for information technology security evaluation website (2009).
URL <http://www.commoncriteriaportal.org/>
- [23] NIST, Fips publications (2009).
URL <http://csrc.nist.gov/publications/PubsFIPS.html>
- [24] Iso 9000 standards website (2009).
URL <http://www.iso.org/iso/home.htm>
- [25] MITRE, Common result format specification 0.3 draft (September 2007).
URL <http://makingsecuritymeasurable.mitre.org/crf/downloads/crf-specification-0.3.pdf>
- [26] R. A. Martin, Managing vulnerabilities in networked systems, Computer 34 (11) (2001) 32–38. doi:<http://doi.ieeecomputersociety.org/10.1109/2.963441>.

- [27] M. Barrett, C. Johnson, P. Mell, S. Quinn, K. Scarfone, Guide to adopting and using the security content automation protocol (scap) (draft) (May 2009).
URL <http://csrc.nist.gov/publications/drafts/800-117/draft-sp800-117.pdf>
- [28] H. Löhr, H. V. Ramasamy, A.-R. Sadeghi, S. Schulz, M. Schunter, C. Stübke, Enhancing grid security using trusted virtualization, in: *Autonomic and Trusted Computing, Lecture Notes In Computer Science*, Springer Berlin / Heidelberg, 2007, pp. 372–384.
URL <http://www.springerlink.com/content/n49081t428459335>
- [29] F. Yan, W. Qiang, Z.-D. Shen, C. Chen, H. Zhang, D. Zou, Daonity: An experience on enhancing grid security by trusted computing technology, in: L. T. Yang, H. Jin, J. Ma, T. Ungerer (Eds.), *ATC*, Vol. 4158 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 227–235.
URL http://dx.doi.org/10.1007/11839569_22
- [30] J. H. Huh, J. Lyle, Trustworthy log reconciliation for distributed virtual organisations, in: L. Chen, C. J. Mitchell, A. Martin (Eds.), *TRUST, Lecture Notes in Computer Science*, Springer, 2009, pp. 169–182.
URL http://dx.doi.org/10.1007/978-3-642-00587-9_11
- [31] S. K. Nair, I. Djordjevic, B. Crispo, T. Dimitrakos, Secure web service federation management using tpm virtualisation, in: *SWS '07: Proceedings of the 2007 ACM workshop on Secure web services*, ACM, New York, NY, USA, 2007, pp. 73–82. doi:<http://doi.acm.org/10.1145/1314418.1314430>.
- [32] Oxford e-Research Centre, Oxgrid description (website) (2009).
URL <http://www.oerc.ox.ac.uk/resources/oxgrid>
- [33] T. Vejda, R. Toegl, M. Pirker, T. Winkler, Towards trust services for language-based virtual machines for grid computing, in: P. Lipp, A.-R. Sadeghi, K.-M. Koch (Eds.), *TRUST*, Vol. 4968 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 48–59.
URL http://dx.doi.org/10.1007/978-3-540-68979-9_4
- [34] W. Mao, F. Yan, C. Chen, Daonity: grid security with behaviour conformity from trusted computing, in: A. Juels, G. Tsudik, S. Xu, M. Yung (Eds.), *STC*, ACM, 2006, pp. 43–46.
URL <http://doi.acm.org/10.1145/1179474.1179486>
- [35] J. Wei, L. Singaravelu, C. Pu, A secure information flow architecture for web service platforms, *IEEE Transactions on Services Computing* 1 (2) (2008) 75–87. doi:10.1109/TSC.2008.10.
URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4695818
- [36] H. Chen, J. Chen, W. Mao, F. Yan, Daonity - grid security from two levels of virtualization, *Information Security Technical Report* 12 (3) (2007) 123 – 138. doi:DOI: 10.1016/j.istr.2007.05.005.
URL <http://www.sciencedirect.com/science/article/B6VJC-4NT84XB-2/2/9a47a62ce2959e392faf39b50d37a521>
- [37] A. Böttcher, B. Kauer, H. Härtig, Trusted computing serving an anonymity service, in: *TRUST*, Vol. 4968 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 143–154.
URL http://dx.doi.org/10.1007/978-3-540-68979-9_11
- [38] D. Wang, A. min Wang, Trust maintenance toward virtual computing environment in the grid service, in: Y. Zhang, G. Yu, E. Bertino, G. Xu (Eds.), *APWeb*, Vol. 4976 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 166–177.
URL http://dx.doi.org/10.1007/978-3-540-78849-2_19
- [39] L. Chen, R. Landfermann, H. Löhr, M. Rohe, A.-R. Sadeghi, C. Stübke, A protocol for property-based attestation, in: *STC '06: Proceedings of the first ACM workshop on Scalable trusted computing*, ACM, New York, NY, USA, 2006, pp. 7–16.
URL <http://doi.acm.org/10.1145/1179474.1179479>
- [40] J. Zhan, H. Zhang, F. Yan, Building trusted sub-domain for the grid with trusted computing, in: D. Pei, M. Yung, D. Lin, C. Wu (Eds.), *Inscrypt:Third SKLOIS Conference on Information Security and Cryptology*, Vol. 4990 of *Lecture Notes in Computer Science*, Springer, Xining, China, 2007, pp. 463–471.
URL http://dx.doi.org/10.1007/978-3-540-79499-8_36
- [41] S. Munetoh, M. Nakamura, S. Yoshihama, M. Kudo, Integrity Management Infrastructure for Trusted Computing, *IEICE Transactions on Information and Systems* E91-D (5) (2008) 1242–1251.
URL <http://dx.doi.org/10.1093/ietisy/e91-d.5.1242>
- [42] R. Sailer, T. Jaeger, X. Zhang, L. van Doorn, Attestation-based policy enforcement for remote access, in: *CCS '04: Proceedings of the 11th ACM Conference on Computer and Communications Security*, ACM, New York, NY, USA, 2004, pp. 308–317.
URL <http://doi.acm.org/10.1145/1030083.1030125>
- [43] T. Maguire, D. Snelling, Web services service group 1.2 (ws-servicegroup), Tech. rep., OASIS Open (June 2004).
- [44] Gideon technologies website (2009).
URL <http://www.thegideongroup.com/>
- [45] Bigfix security configuration and vulnerability management (2009).

- URL <http://www.bigfix.com/content/security-configuration-and-vulnerability-management>
- [46] Qualysguard website (2009).
URL <http://www.qualys.com/index.php>
- [47] OASIS, eXtensible Access Control Markup Language (XACML) Version 2.0 (February 2005).
URL http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf
- [48] S. Faily, I. Fléchais, Context is Everything: a User-Centered Framework and Tool for Integrating Security and Software Engineering, in: Computer Security Applications Conference, 2009. ACSAC 2009. Annual, Submitted, IEEE Computer Society, 2009, p. to appear.
- [49] R. Hassan, S. Bohner, S. El-Kassas, M. Hinchey, Integrating formal analysis and design to preserve security properties, Hawaii International Conference on System Sciences 0 (2009) 1–10.
doi:<http://doi.ieeecomputersociety.org/10.1109/HICSS.2009.782>.
- [50] K. Adams, O. Agesen, A comparison of software and hardware techniques for x86 virtualization, in: ASPLOS-XII: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems, ACM, New York, NY, USA, 2006, pp. 2–13. doi:<http://doi.acm.org/10.1145/1168857.1168860>.
- [51] I. Foster, C. Kesselman, G. Tsudik, S. Tuecke, A security architecture for computational grids, in: CCS '98: Proceedings of the 5th ACM conference on Computer and communications security, ACM, New York, NY, USA, 1998, pp. 83–92. doi:<http://doi.acm.org/10.1145/288090.288111>.
- [52] F. Stumpf, M. Benz, M. Hermanowski, C. Eckert, An approach to a trustworthy system architecture using virtualization, in: Autonomic and Trusted Computing, Vol. 4610/2007, Springer, 2007, pp. 191–202.
URL <http://www.springerlink.com/content/t629739n16471418/>
- [53] National grid service (ngs).
URL http://www.grid-support.ac.uk/component/option,com_frontpage/Itemid,137/
- [54] S. Andreozzi, S. Burke, F. Ehm, L. Field, G. Galang, B. Konya, M. Litmaath, P. Millar, J. Navarro, GLUE Specification v. 2.0 (February 2009).
URL <http://forge.gridforum.org/sf/docman/do/downloadDocument/projects.glue-wg/docman.root.drafts.archive/doc15023>
- [55] Berkeley database information index v5.
URL <https://twiki.cern.ch/twiki/bin/view/EGEE/BDII>
- [56] J. Hodges, R. Morgan, Lightweight directory access protocol (v3): Technical specification (2002).
- [57] J. Novotny, S. Tuecke, V. Welch, An online credential repository for the grid: Myproxy, High-Performance Distributed Computing, International Symposium on 0 (2001) 0104.
doi:<http://doi.ieeecomputersociety.org/10.1109/HPDC.2001.945181>.
- [58] O. Aidel, A. Cavalli, H. Cordier, C. L'Orphelin, G. Mathieu, A. Pagano, S. Reynaud, CIC portal: a collaborative and scalable integration platform for high availability grid operations., in: GRID, IEEE, 2007, pp. 121–128.
URL <http://dblp.uni-trier.de/db/conf/grid/grid2007.html#AidelCCLMP07>
- [59] M. A. Slaymaker, E. Politou, D. J. Power, S. Lloyd, and A. C. Simpson, e-DiaMoND: risk analysis, in: UK e-Science All Hands 2004 Proceedings, 2004, pp. 937–939.
URL <http://www.allhands.org.uk/2004/proceedings/papers/54.pdf>
- [60] I. E. Commission, Iec 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems. parts 1-7, International Electrotechnical Commission, Switzerland (2005).
URL <http://www.iec.ch/zone/fsafety/>

A. Threat and Risk Analysis

Using the Integrating Requirements and Information Security (IRIS) framework [48], a threat and risk analysis was conducted on the three *use cases* (see Section 4.1). The IRIS framework has been designed to integrate security and software engineering with user centred approaches to security. It provides a detailed meta-model for integrated requirements and risk management. For these reasons, we adopted their meta-model and risk management process to identify threats and examine associated risks.

The purpose of the analysis is to study how attackers might exploit potential security holes, and in doing so, highlight unique security challenges of managing application whitelists in a distributed environment.

A.1. Valuable Assets

Six valuable assets were defined in the first step. Each asset was assessed with respect to *integrity* — prevention of unauthorised modification of data or software; *confidentiality* — prevention of unauthorised disclosure of data; and *availability* — the prevention of unauthorised withholding of an asset [59]:

Whitelist is a list of acceptable software entries — through various software testing and verification processes — that are being trusted and managed by the Configuration Manager. Both confidentiality and integrity of the individual entries as well as the totality of the list need to be strongly protected.

Reference Integrity Measurement (RIM) contains detailed information about a software entry including its hash representation, source, and other software dependencies; its integrity needs to be assured.

Entry Update Messages are whitelist entry update recommendations a CM would send to other CMs to inform about a software entry that should be added or deleted from the whitelist; again, both the confidentiality and integrity need to be assured.

Attestation Tokens contain information about the platform configurations; integrity protected configurations need to be advertised; confidentiality is also important since tokens contain configuration details of participant platforms.

Job Secret includes the user credentials, sensitive methods, models, and data, which all need to be both integrity and confidentiality protected.

CM Software refers to various services in the CM and associated policies and configuration files, as well as the system level software necessary to enable a trustworthy CM platform; both integrity and availability of these software need to be assured.

A.1.1. Major Stakeholders

Four major stakeholders were identified in the second step:

Privileged Users of the CM include system owners and administrators who have enough privileges to modify the behaviours of the CM or access the sensitive data (e.g. whitelist) inside.

End Users use the services/resources provided by the participants; they make service-discovery requests to the CM, and submit jobs to the trustworthy participants.

Participants are the service/resource providers; to become part of the trusted system, they have to register with the CM by submitting their attestation token.

Intruders are unauthorised users trying to achieve malicious objectives like 1) stealing the job secrets; 2) making unauthorised use of resources; 3) compromising user machines; and 4) corrupting the job results.

Score (TL)	Threat Likelihood	Score (VS)	Vulnerability Severity	Score	Risk Rating	Risk Scores From TL and VS				
0	Incredible	0	Negligible	1	Intolerable	TL / VS	Catastrophic	Critical	Marginal	Negligible
1	Improbable	1	Marginal	2	Undesirable	Frequent	1	1	1	2
2	Remote	2	Critical	3	Tolerable	Probable	1	1	2	3
3	Occasional	3	Catastrophic	4	Negligible	Occasional	1	2	3	3
4	Probable					Remote	2	3	3	4
5	Frequent					Improbable	3	3	4	4
						Incredible	4	4	4	4

Figure 8: IEC 61508 Tables for Threat Likelihood (TL), Vulnerability Severity (VS), and Risk Rating

A.2. Analysing Threats and Associated Risks on the Use Cases

We went through a three step process of 1) identifying vulnerabilities of the system; 2) identifying the potential sources of attacks to vulnerabilities; and 3) identifying the implications of an attack.

The IRIS framework applies the likelihood and severity tables within IEC 61508 [60], and assigns a risk rating based on these scores (see Figure 8). Mindful of the stakeholders' interests and objectives, for each asset identified above, potential security threats and their risks have been rated:

Use Case 4.1.1 - Joining the Virtual Organisation			
Vulnerability	Threat	TL/VS	Risk (Rating)
Insecure communication channel between the participant platform and the CM	Intruders might sniff the network to steal the attestation token.	3/2	Intruders could steal the token and access the participant's platform configurations (2).
An attack surface of the external service publisher	A malicious participant might perform privilege-escalation attacks on the service publisher.	2/2	The participant could gain enough privileges to freely access the whitelist and the attestation tokens (3).
Incompetent verification mechanisms in the attestation service	A malicious participant might try to bypass the verification mechanisms and lie about their platform configurations.	4/2	The participant could bypass the verification mechanisms and join the CM with untrustworthy configurations (1).
Incompetent verification mechanisms in the attestation service	A participant, after successfully join the CM, might maliciously (or even unintentionally) modify the platform configurations.	4/2	The participant, who is already part of the CM, could modify the platform configurations without the CM knowing (1).

Table 1: Threat and Risk Analysis on Use Case 4.1.1

Use Case 4.1.2 - Job Submission			
<i>Vulnerability</i>	<i>Threat</i>	<i>TL/VS</i>	<i>Risk (Rating)</i>
Incompetent verification mechanisms for the CM's identity	An intruder might pretend to be a valid CM and return service descriptions of untrustworthy participants.	2/3	Service descriptions of untrustworthy platforms could be sent to the user — platforms for which the intruder has control over (2).
Grid jobs are executed without strong isolation.	A malicious user might submit an arbitrary code/query to compromise a remote host.	3/3	Untrusted code/query could run and compromise the host (1).
A large attack surface of the complex grid middleware on the participant platform	Intruders might perform privilege-escalation attacks to steal the grid job secrets.	3/3	Intruders could gain enough privileges to steal the job secrets (1).
Incompetent software/hardware mechanisms for filtering unwanted packets	Intruders might perform denial-of-service attacks on the service publisher to make the CM unavailable for the participants.	3/3	The CM could become saturated with external requests, such that it cannot respond to legitimate requests from the participants (1).

Table 2: Threat and Risk Analysis on Use Case 4.1.2

Use Case 4.1.3 - Vulnerability Discovery and Patching			
<i>Vulnerability</i>	<i>Threat</i>	<i>TL/VS</i>	<i>Risk (Rating)</i>
Incompetent verification mechanisms for the entry updates and RIMs	Rogue administrators might recommend (through entry updates) the participants to install patches that have known vulnerabilities.	2/2	The participants will install patches with known vulnerabilities which can later be exploited (3).
Incompetent verification mechanisms for the entry updates and RIMs	Rogue administrators might send spiteful entry updates and RIMs to other CMs, recommending software that have known security vulnerabilities.	2/3	Other CMs could update their whitelists with the malicious entry (RIM) and recommend their participants to do the same (2).
Incompetent verification mechanisms for the entry updates and RIMs.	Intruders might generate spiteful entry updates and send them to the participants.	3/3	the participants could install malware in their platforms (1).
An attack surface of the external service publisher	Intruders or participants might perform privilege-escalation attacks on the service publisher to tamper with the whitelist and RIMs.	3/2	Intruders could gain enough privileges to modify the whitelist and RIMs; for example, to add an entry for an insecure software (2).
An attack surface of the external entry update manager	Intruders might perform in-memory attacks (e.g. buffer overflow).	2/3	Behaviour of the entry update manager could be altered; for example, to create an entry update for a malicious software and send it to other CMs (2).

Table 3: Threat and Risk Analysis on Use Case 4.1.3



Jun Ho Huh holds Software Engineering and International Business degrees from Auckland University. He is currently a DPhil student at the Oxford University Computing Laboratory. His research interests include trusted virtualization, trustworthy audit and logging, and security in distributed systems.



John Lyle is studying for his DPhil degree at the Oxford University Computing Laboratory. Before this, in 2006 he received an MEng degree in Computing from Imperial College London. His research interests include Trusted Computing infrastructures and Web Services.



Cornelius Namiluko is a DPhil student at Oxford University focusing on architectures for trusted infrastructure. He completed his Master of Science Degree at Oxford with a thesis titled 'Trusted Infrastructure for the Campus Grid'. Before this, he worked as a software engineer on the SmartCare project, building a Smartcard-enabled patient care system with the 'Centre for Infectious Disease Research in Zambia (CIDRZ)' and later with the 'Elizaberth Glaser Paediatric Aids Foundation'. He graduated from the University of Zambia in 2004 with a Bachelor's Degree in Computer Science.



Andrew Martin lectures to Software Professionals as part of Oxford University's Software Engineering Programme, where he directs the Master of Science course in Software and Systems Security. He has a background in formal methods, but today devotes most of his time to issues of security in distributed systems. He has been particularly interested in the grid computing paradigm, the security questions that raises, and how the technologies of trusted computing can help to address the challenges in that area.