CHAPTER 4

Attestation: Proving Trustability

In the last few chapters we have looked at the first stages in a process toward establishing trust between systems. First, the establishment of roots of trust and the measured boot components; and second, the collection of evidence throughout the measurement process. We reviewed the different roots of trust in a compute platform—namely, the RTM, RTS, and RTR—and how the measured boot process (S-RTM and D-RTM) uses the RTM to measure and store the evidence in the RTS. The next stage in this process is the presentation of this evidence through attestation protocols and appraisal of the evidence that asserts the integrity of a platform. This stage is referred to as *attestation and verification* in this book, and it is our objective for this chapter.

We introduce the concept of attestation in this chapter, along with an attestation framework that defines a logical view of the assertion layers leading to attestation of specific target entities or components. The attestation provides evidence of trust and can include any device or target system participating in the trust chain. Additionally, the chapter covers one commercial implementation of the attestation solution authored by Intel and security management independent software vendors, code-named Mt. Wilson. We provide details about the solution architecture, attestation application programming interfaces (APIs), integration of these APIs into a security management function, and workload orchestration tools for decision making. We hope application developers and security specialists will gain a solid understanding of the inner workings of attestation solutions to the level of being able to carry out integration projects and even extend the paradigm.

Attestation

Attestation is a critical component for trusted computing environments, providing an essential proof of trustability and the means for conducting audits for target computing devices. That is, attestation allows a program or platform to authenticate itself. Remote attestation is a means for a system to make reliable statements about the pre-launch and launch components in a distributed system. A remote party can then make authorization decisions based on that information. The concept of attestation is still evolving, and hence the research community has not reached a common understanding of what it means. However, here is a practical definition for the purpose of working with trusted clouds. The Trusted Computing Group (TCG) defines attestation as:

> *The process of vouching for the accuracy of information. External entities can attest to shielded locations, protected capabilities, and Roots of Trust. A platform can attest to its description of platform characteristics that affect the integrity (trustworthiness) of a platform. Both forms of attestation require reliable evidence of the attesting entity.*

There are two properties that have to be addressed to assert this trust.

1. *Measurement properties.* Includes the degree of completeness for measuring the launch and running state of the targeted device or system, and the freshness of the measurements—that is, how recent the measurements are.
2. *Attestation properties.* Includes the authenticity of the evidence to the decision process, and a measure of semantic explicitness describing the appropriateness of the evidence to the decision-making process.

These two properties help us classify the remote attestation techniques. Most of the existing remote attestation techniques can be categorized into one of the two types.

- *Static remote attestation* techniques rely on the signatures or hashes of the firmware and binaries for determining the integrity of the platform state. Static remote attestation can't be extended to measure the behavior of a platform. Furthermore, even if the hash of the boot state (static state) does not reveal any tampering, it does not follow that the run-time behavior of the application will be trustworthy.
- *Dynamic remote attestation* techniques use monitoring instead of measuring the application binary. Dynamic remote attestation techniques are relatively difficult to integrate into existing operating systems and software applications, because there is no unequivocal reference point; that is, there is no commonly agreed upon definition of what constitutes trustworthy behavior in an operating system, virtual machine monitor, or application. Benchmarks for trustworthy behavior, defined in existing remote attestation techniques, are either vague or incomplete, with only a portion of the activities performed by an application during its execution monitored. The benchmarks don't apply to virtual machine monitors because the benchmark requirements are not yet well understood.

Both static and dynamic remote attestation are relevant to virtualization and cloud computing. As described in the previous chapters, the trusted compute pool uses models that begin with the boot integrity of the platform, asserted with the static attestation techniques. Meanwhile, asserting run-time integrity needs dynamic attestation techniques. Static attestation techniques are beginning to be adopted in practical cloud computing deployments. The static techniques provide a good foundation toward reaching a trusted infrastructure. Dynamic remote attestation is complementary and brings significant value by enforcing security; hence, we can expect a strong drive for adoption. However, in order to achieve the vision and goals of a trusted infrastructure, it is an imperative to have a dynamic remote attestation facility in working order.

For context, we provide a brief overview in this chapter of remote attestation techniques discussed in the research community, including reference implementations where available. Please note that, other than Integrity Measurement Architecture, none of the schemes has seen wide adoption, if any at all.

Integrity Measurement Architecture

Integrity Measurement Architecture (IMA) is a classic static remote attestation model developed by IBM[1] for measurement and reporting of the integrity of Linux-based systems. It takes a hash of the binaries of the software code that run on any system, and compares them against known-good hashes to assert that the system is high integrity. IMA extends the trusted boot process of the TCG beyond the bootstrapping of the Linux loader, to the chain of trust from the TPM, to applications running on the system. Through extensions to the kernel of the Linux system, IMA measures the code that's loaded into memory for execution by taking a SHA-1 hash of the code prior to that execution. A measurement archive is maintained for measurements previously taken.

Integrity Measurement Architecture was the first practical implementation of a TCG-based remote attestation technique. It allows a challenger to verify a platform status by measuring the executables running on that platform. IMA forms the basis for many remote attestation techniques that followed the original implementation. The requirement for using IMA is to download a kernel patch from IBM. The prototype of IMA was implemented as a Linux Security Module on RedHat 9.0 Linux distribution and kernel version 2.6.5.

Policy Reduced Integrity Measurement Architecture

Policy Reduced Integrity Measurement Architecture (PRIMA) is a variation of IMA. According to the authors of this architecture,[2] the static code and load-time measurement cannot be used to assess the run-time behavior. This architecture introduces the concept of measured security context or label of the subject, in addition to static code. The code/data digest also includes a role field so that additional identification of subjects and objects can be done. This approach allows remote attestation to be made on the basis of secure information-flow models. The approach is rather low level and cannot be used for distributed services in an organization or the information flows that occur within the organization and in outside world. There are no known implementations in a commonly available operating system environment.

[1]See `http://researcher.watson.ibm.com/researcher/files/us-msteiner/ima.sailer_usenix_security_2004_slides.pdf`

[2]Trent Jaeger et al., "PRIMA: PolicyReduced Integrity Measurement Architecture, SACMAT2006, June 7–9, 2006, Lake Tahoe, California. ACM 1595933549/06/0006.

Semantic Remote Attestation

Semantic Remote Attestation is an attempt at creating a platform-independent remote attestation technique.[3] The core idea is that of a trusted virtual machine (TVM) capable of enforcing the requirements for those applications running within this virtual machine. The model establishes trust on the TVM and uses this trust to enforce security requirements. It attempts to measure the behavior of the code running inside a trusted virtual machine. The architecture is an incremental improvement over the original remote attestation techniques and is more flexible compared with binary attestation techniques with regard to expressiveness. This model of attestation has not been implemented, or at least published, owing to the complexity of defining and analyzing the notion of trust.

The Attestation Process

Given the discussion in the above section about the state and maturity of attestation techniques, let's look at the details of the static attestation protocol and the overall integrity measurement flow.

The integrity measurement flow describes the steps required to measure the platform integrity measurements. It includes:

- A means of generating and collecting the measurements through an RTM.
- A means of storing the measurements that is either tamper resistant or tamper evident, with a TPM for RTS and RTR.
- A means of conveying the measurements to a challenger via the attestation agents, as described in the attestation protocol below.
- A means of analyzing the measured result, and a means of asserting the trustability of the machine based on the results of that determination through a trust assessment authority or trust attestation authority (TAA).

Remote Attestation Protocol

Figure 4-1 illustrates the attestation protocol providing the means for conveying measurements to the challenger. The endpoint attesting device must have a means of measuring the BIOS firmware, low-level device drivers, operating system, virtual machine monitor components, and be able to forward those measurements to the attestation authority. The attesting device must do this while protecting the integrity, authenticity, nonrepudiation, and some cases, the confidentiality of those measurements.

[3]Vivek Haldar et al., *Semantic Remote Attestation: a Virtual Machine Directed Approach to Trusted Computing*, VM2004 Proceedings of the 3rd conference on Virtual Machine Research and Technology Symposium, vol. 3 (Berkeley, CA: USENIX Association).

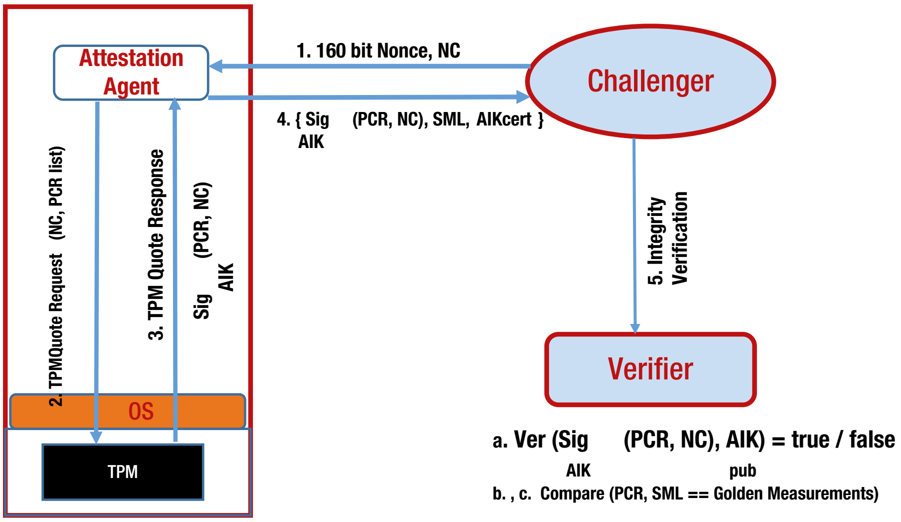

Attestation Agent
1. 160 bit Nonce, NC
Challenger
4. { Sig AIK (PCR, NC), SML, AIKcert }
2. TPMQuote Request (NC, PCR list)
3. TPM Quote Response
Sig AIK (PCR, NC)
5. Integrity Verification
Verifier
OS
TPM
a. Ver (Sig AIK (PCR, NC), AIK pub) = true / false
b. , c. Compare (PCR, SML == Golden Measurements)

Figure 4-1. *Remote attestation protocol*

Let's walk through the steps of the remote attestation protocol:

1. The challenger, at the request of a requester, creates a nonpredictable nonce (NC) and sends it to the attestation agent on the attesting node, along with the selected list of platform configuration registers (PCRs).

2. The attestation agent sends that request to the TPM as a TPMQuote request with the nonce, and the PCR list.

3. In reponse to the TPMQuote request, the TPM loads the attestation identity key from protected storage in the TPM by using the storage root key (SRK), performs a TPM Quote command, which is used to sign the selected PCRs and the provided nonce (NC) with the private key, AIKpriv. Additionally, the attesting agent retrieves the stored measurement log (SML).

4. Called the integrity response, the attesting agent sends the response consisting of the signed quote, signed nonce (NC), and the SML to the challenger. The attesting agent also delivers the AIK credential, which consists of the AIKpub that was signed by a privacy CA.

5. The challenger validates if the AIK credential was signed by a trusted privacy CA thus belonging to a genuine TPM. The challenger also verifies whether AIKpub is still valid by checking the certificate revocation list of the trusted issuing party.

6. The challenger verifies the signature of the quote and checks the freshness of the quote.

7. Based on the received stored measurement log and the PCR values, the challenger processes the SML, compares the individual module hashes that are extended to the PCRs against the known-good or golden values, and recomputes the received PCR values. If the individual values match the golden values, and if the computed values match the signed aggregate, the remote node is asserted to be in a trusted state.

This protocol is highly resistant to replay attacks, tampering, and masquerading.

How does this remote attestation protocol get implemented and manifested in an IT environment? Figure 4-2 illustrates a sample IT architecture supporting the generation, forwarding, and analysis of platform boot integrity measurements, as well as assertion of the trustability of the attestation at each decision point via a trust assertion authority, or TAA. These solutions come from a set of compatible components available from a variety of suppliers.

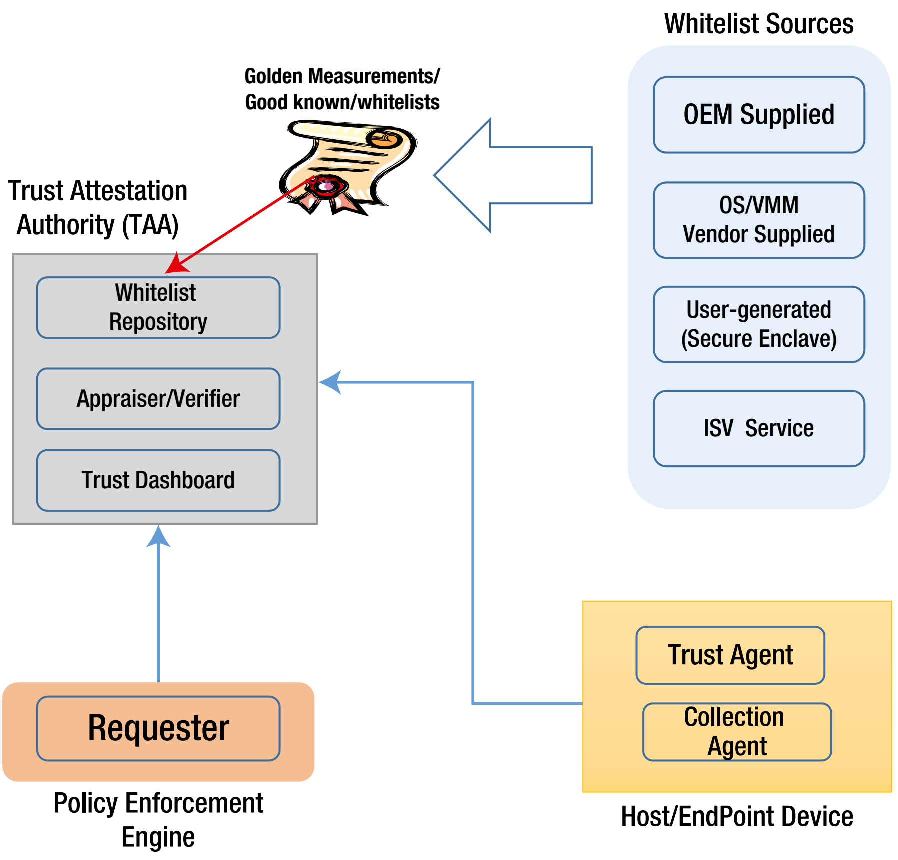


Figure 4-2. *Trust attestation authority*

Flow for Integrity Measurement

In assessing the measurements, the TAA references a set of properties. These properties represent attributes and measurements for the BIOS and the operating system and virtual machine monitors. These measurements are referred to as *golden measurements* or *whitelists*, and are:

- Provided and verified and validated through certificates by the original equipment manufacturer (OEM))
- Provided and vouched for by an ISV Service
- Collected by an authenticated administrator on first boot in an isolated or enclave type of environment

The process for carrying out the integrity measurement and verification is as follows:

1. When a new instance of a BIOS or an operating system or virtual machine monitor is made available, an initial set of trusted measurements (golden measurements) is taken on the image. These measurements are provided either through third parties such as an OEM, operating system, virtual machine monitor supplier, or through a trusted whitelist service provider to the trust authority, It may also be generated at initial provisioning by system administrators.
2. An RTM such as Intel TXT is used to take the measurement of the software components during server or device boot.
3. The measurements are stored in the TPM. A log from which the measurements can be reconstructed is stored in memory for transmission to the verifier to allow reconstruction of the measurements.
4. The TAA generates an authenticated request for measurements from the server/device, in response to an action by any requester, or the endpoint device requesting a service. This action follows the attestation protocol previously described. The trust agent receives this request and passes it to the TPM to obtain a TPMQuote for the requested PCR measurments. TPMQuote, along with the measurement log, are packaged as an integrity report, using the TCG Integrity Reporting Schema.

5. The trust agent transmits the data to the TAA's verifier. The TAA verifies the signature over the hashes by inspecting both the public key used to sign them and the signature itself, which will ensure that the nonce sent to the trust agent is the same one as the one used in the TPMQuote. It then compares those signed measurements with the golden measurements obtained earlier. There is more than a simple comparison. Depending upon the sophistication of the verifier, it can use the system measurement log (SML) to re-compute the aggregate measurements from the individual measurements, and then verify them against the golden measurements.

6. The results of the comparison, collated with other such comparisons from other machines and digitally signed, may be displayed via a user interface, such as a management console or dashboard, to the administrator or it can be provided through an API to an automated enforcement, policy engines, and orchestrators. Solutions use the results to apply, manage, enforce, and report on the trust level of the systems.

A First Commercial Attestation Implementation: The Intel Trust Attestation Platform

To provide a path toward broad use of trusted compute pools and to exemplify the vision of trusted infrastructure and cloud computing, Intel developed a remote attestation solution capable of working across a broad range of hardware and operating system and virtual machine monitor platforms: the Intel Trust Attestation Platform (TAP). The goals of the Intel Trust Attestation Platform are threefold:

- Provide a production-quality implementation of remote attestation and a trust assessment authority capable of providing verification and assessment across a broad range of devices. The Intel Trust Attestation Platform features high availability and security of the attestation platform and its interfaces.
- Provide stable and simplistic northbound and southbound application programming interfaces (APIs) for attestation information requesters, and for interfacing with different sources of integrity measurements. These are trust APIs, designed to encourage multiple interoperable attestation solutions from a variety of security-management independent software vendors. The interoperability and diversity minimize the occurrence of vendor lock-in.

- Develop the attestation platform as a true extensible and pluggable framework providing fertile ground for the deployment of innovative third-party attestation techniques and models. Initially, the solution supports a TPM-based static attestation model, and is already being extended to support dynamic attestation techniques for asserting the boot integrity of virtual machines, as well as the run-time integrity of operating systems and hypervisors.

Figure 4-3 captures the high-level architecture of the Intel Trust Attestation Platform. Consistent with the cloud approach, the Intel Trust Attestation Platform features a loosely coupled architecture with a flexible software backplane and fabric with core capabilities and services, including a set of slots to plug in various attestation blades for different types of attestation provided by Intel and third-party independent software vendors. Here are the key aspects of the architecture:

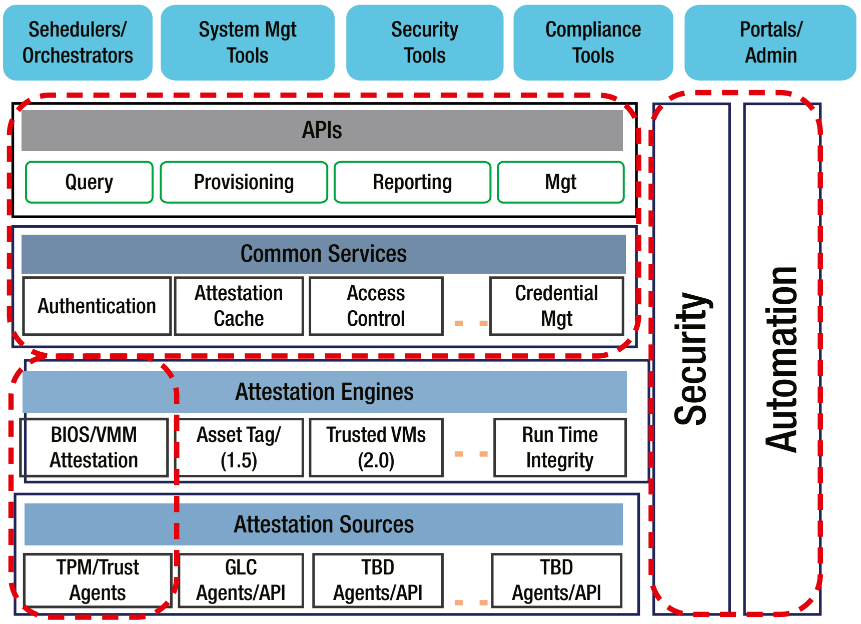


Figure 4-3. *Intel Trust Attestation Platform*

- An *API layer* acting as primary interface for:
  - Endpoint devices needing to carry out an attestation before a request for services
  - Entities requesting integrity verification for policy enforcement and visibility into the trust of the infrastructure
  - Access to compliance and security monitoring tools

- A *common services layer* for the attestation service and platform to enable authentication, authorization, and access control (AAA) for the API calls, and *a* flexible and extensible data model for the attestation platform repository accessible via APIs.
- An *attestation blade* supporting a variety of attestation types implemented as plug-ins. The attestation blade is an element of a set of pluggable components integrated into the attestation platform taking advantage of the fabric and core functionality of the platform, including interfaces, security, and common services. As shown in Figure 4-3, each blade has two distinct components:
    - A measurement and attestation agent capable of collecting measurements from an endpoint device or server.
    - A verification module that uses the attestation platform services, and provides custom verification logic for an attestation capability instance, using the northbound APIs of the attestation platform, thereby exposing an assertion function and making it available to policy enforcers and other requesting entities.

Mt. Wilson Platform

Mt. Wilson is the code name for the Intel Trust Attestation platform that has the TPM-based boot attestation functionality. It is the first attestation blade that was released as part of the attestation platform. Mt. Wilson provides a secure mechanism for customers and data center operators to attest the integrity of Intel-based systems enabled with Intel's Trusted Execution Technology (TXT) for RTM, along with third-party trusted platform modules (TPMs). The TPM stores and reports the platform measurements, including BIOS firmware and hypervisor software on servers. The architecture of the blade, described in more detail later in this chapter, is applicable to any TPM-based integrity measurement and reporting architecture.

We have assembled proof of existence working prototypes of a boot integrity attestation blade with Microsoft Windows 8, and corresponding TPM using a BIOS boot block as the RTM. We also have constructed a proof point with Citrix XenClient XT using Intel TXT on the client. A subset of the Mt. Wilson functionality has been shared with the open-source community under the name Open Attestation (OAT).

Mt. Wilson is a fast-evolving platform with new features and capabilities developed and released as the community gains experience with the technology. Here is a snapshot of key capabilities in the current Mt. Wilson solution.

Table 4-1. *Mt. Wilson Key Capabilities*

Attestation Support	PCR and module-based attestation and verification for VMware ESXi 5.1 and above, and for Xen, KVM with RHEL, SuSE, and Ubuntu Linux
APIs	REST interfaces for query, reporting, management, and provisioning functions;
	REST interfaces for whitelist definition and management
Security	Digest-style API authentication and validation using RSA keys (<signed http authorization header >)
	SAML-based API responses with signed SAML assertions
	SSL communication and mutual authentication of communication endpoints
Auditability	Secure logging of requests, responses, transactions for auditability, forensics including logging APIs, and support for CEF format for consumption into SIEM tools
Deployability	Automated installation of host trust agents and all Mt.Wilson components
	Solution validation with Hewlett Packard, Dell, Cisco hardware, etc.
Availability	Deployed as Xen/KVM/VMW, virtual machines including high availability and fault tolerance for key components for VMware
Automation and Productivity Tools	API client: utility wrapper code for API invocation and response processing
	Reference integration with OpenStack extensions to flavors, dashboard, scheduler
	Reference trust dashboard with API integration with Mt. Wilson

The rest of this chapter will provide a comprehensive view of this attestation blade, starting with the architecture and design components to support server operating systems and virtual machine monitors, followed by the core attestation related API definitions and security considerations. Sample source code examples are provided in the last section of this chapter to show how to:

- Register the servers with Mt. Wilson
- Request the trust assertions (using the trust APIs)
- Whitelist the golden measurements that are used in the appraisal and verification

Mt. Wilson Architecture

Mt. Wilson, as shown in Figure 4-4, has two main components: the *trust agent* (TA) and the *trust attestation authority* (TAA.)

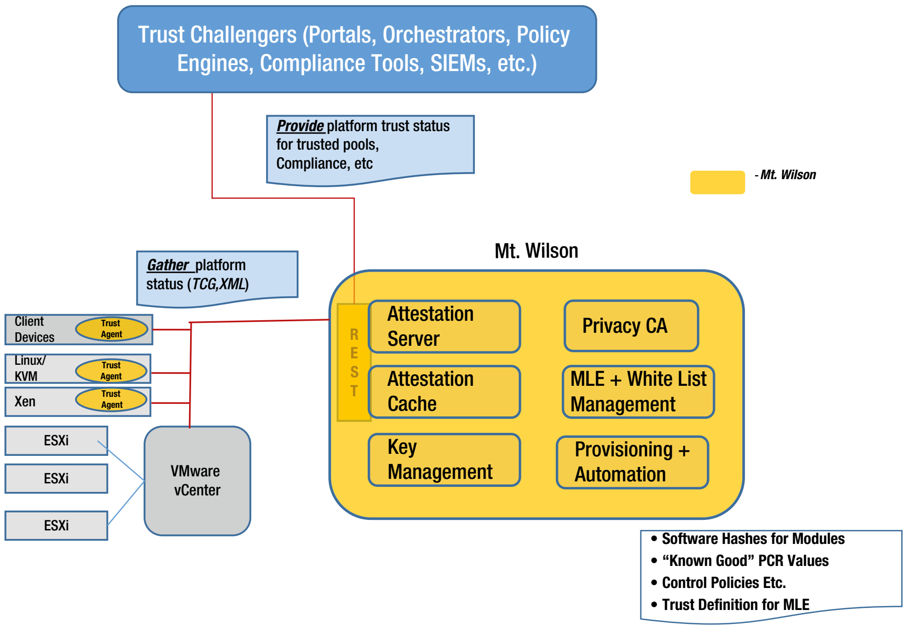

Trust Challengers (Portals, Orchestrators, Policy Engines, Compliance Tools, SIEMs, etc.)
Provide platform trust status for trusted pools, Compliance, etc
- Mt. Wilson
Mt. Wilson
Gather platform status (TCG,XML)
Client Devices
Trust Agent
Linux/ KVM
Trust Agent
Xen
Trust Agent
ESXi
ESXi
ESXi
VMware vCenter
R E S T
Attestation Server
Attestation Cache
Key Management
Privacy CA
MLE + White List Management
Provisioning + Automation
• Software Hashes for Modules
• "Known Good" PCR Values
• Control Policies Etc.
• Trust Definition for MLE

Figure 4-4. *Mt. Wilson architecture*

The trust agent runs on the device or host that is attesting with the trust attestation authority. The trust agent is the collector, and securely uploads the integrity measurements (fetched using the `TPMQuote` command) and the integrity event log from the TPM. The trust agent is not required in a VMWare environment, since vCenter provides specific APIs (called `TrustAttestationReport`) and capabilities that provide the functionality. More specifically, vCenter Agent and VMWare vCenter Server enable the necessary handshake, verification of the platform certificates, and invocation of the TPM commands, in response to any entity invoking the `TrustAttestationReport` web services API.

The trust attestation authority is the core attestation and assessor with a number of key services:

- **Attestation Server:** This is the primary service providing the APIs for the trust attestation authority. It has the function of interfacing with the attesting hosts, requesting the specific host for its measurements following the remote attestation protocol, and verifying certificates, signatures, and logs requests and responses for tracking and auditability. A key role of the attesation server is to appraise the measurements from the device/host, which involves comparing these measurements against golden measurements, whitelists, and known-good values. The whitelists are the final TPM PCR extensions for each of the PCRs of the TPM and granular SHA-1 hashes of the various loadable modules of the measured launch environment (MLE). The appraisal includes verifying the individual module hashes from the SML (event log) against the whitelists of the module hashes and recomputing the PCR values from the event log entries. The recomputed PCR value has to match the value sent from the device (which shows that the log is not compromised) and match the whitelist/known-good. In today's implementation across hypervisor and operating system vendors, there are variations in approaches to measuring the TCB. For instance, VMware has made great strides in measuring a high percentage of their TCB. Open-source operating system and hypervisor providers have, for the most part, reused the Intel reference tboot implementation, and consequently measure a small part of the TCB, mostly the kernels. As the need for trust increases in the cloud data centers, vendors have been expressing a willingness to broaden the amount of measured TCB.

- **Whitelist Management:** This service provides APIs to define the various MLEs in the environment, their attributes, policy-driven trust definition, and the whitelists for the modules or PCRs. Whitelist measurements are usually retrieved from hosts built and configured in an isolated environment/enclave, or provided by the OEM and VMV/OS vendors. The MLEs and the corresponding whitelist measurements need to be configured to specific versions of BIOS and hypervisor.

- **Host Management:** This service provides APIs to register the hosts to be attested with the system. For successful attestation, the whitelists for the BIOS and hypervisor running on the host need to be preconfigured in the Mt. Wilson system, prior to registration of the host that would attest.

- **Privacy CA:** Provides the attestation certificate for the open-source hypervisor hosts and validation of the same. The certificate authority needs to support the OCSP protocol for certificate validation. This capability is subsumed by VMware vCenter Server in the VMWare environment. Management of Citrix XenServer does not need privacy CA since it supports direct anonymous attestation (DAA).

In the next section, we drill into the attestation server and understand the functions and the attestation process flows.

The Mt. Wilson Attestation Process

Figure 4-5 illustrates the attestation architecture in Mt. Wilson, with a drilldown of the attestation server component described in the previous session and depicted in Figure 4-4. The Mt. Wilson attestation process comprises three flows:

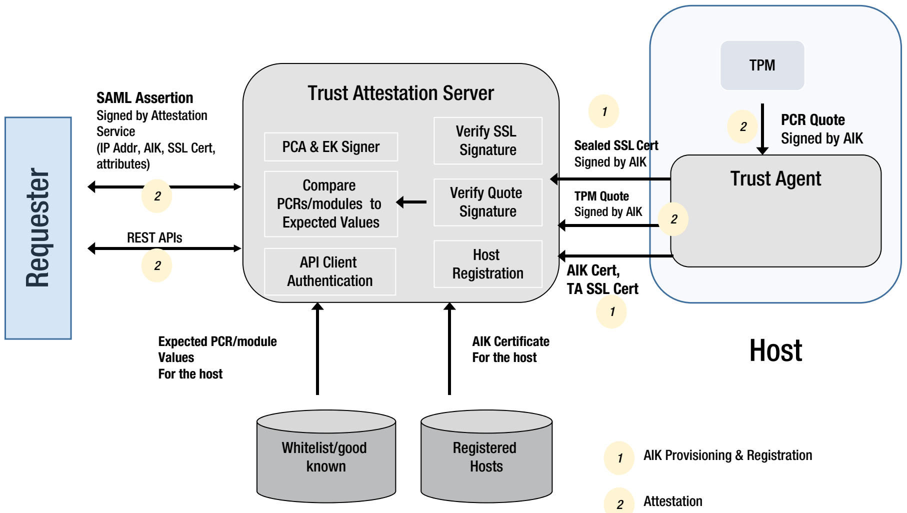


Figure 4-5. The Mt. Wilson attestation architecture

1. Provisioning the attestation identity keys (AIKs) and ensuring successful validation of the host
2. Registration of the host with Mt. Wilson
3. Actual attestation request and response

Attestation Identity Key Provisioning

The attestation identity key provisioning process is done in four steps.

1. *TPM on the host is validated.* According to the TCG specifications, compliant systems should contain an endorsement credential and a platform credential. These two credentials are installed by the OEM to certify that the TPM's endorsement key and the entire TCG subsystem are genuine. However, in practice these credentials are often missing. As a workaround, system administrators may inspect a system and generate equivalent credentials locally after being satisfied that the system is genuine. The trust agent software provides a password-protected mechanism in conjunction with the privacy CA service for the system administrator to easily generate and install the equivalent credentials. Additional credentials, known as the conformance credential and validation credential, are also possible but are seen even less in practice, and are not covered during the attestation identity key provisioning and host registration.
2. *The AIK is created by the platform and certified by the privacy CA.* This transforms the platform verification problem into an RSA encryption problem. It is critical for the system administrator to conduct an adequate inspection to ensure that the TPM is genuine and that Intel TXT is properly enabled on platforms that are missing the endorsement credential and the platform credential because, once the AIK is certified by the privacy CA, remote attestation services will trust TPM quotes signed with the corresponding AIK private key. The AIK certificate is imported into Mt. Wilson when the host is registered.
3. *An RSA key pair and transport layer security (TLS) certificate are generated.* These are for the trust agent to use for incoming attestation requests. Mt. Wilson provides a mechanism to import the trust agent TLS certificate on a per-host basis and verifies all attestation connections to that host using the same certificate.
4. *A second RSA key pair and TLS certificate are generated on the platform.* The private key bound to the TPM and the TLS certificate indicates the specifics of the TPM binding. This key pair facilitates applications of the trusted compute pool relying on attestation of the platform to authorize certain actions by providing a mechanism assure a third party that, when it connects to the attested platform, it is the same platform in the same trusted state as was attested. Mt. Wilson provides a mechanism to import the bound or sealed TLS certificate after a host is registered and to provide that certificate to its clients.

Host Registration and Attestation Identity Key Certificate Provisioning

Figure 4-6 depicts the sequence diagram showing the steps for host registration and the management of attestation identity key certificates. As mentioned earlier, these steps are applicable only for hosts running on Xen or KVM.

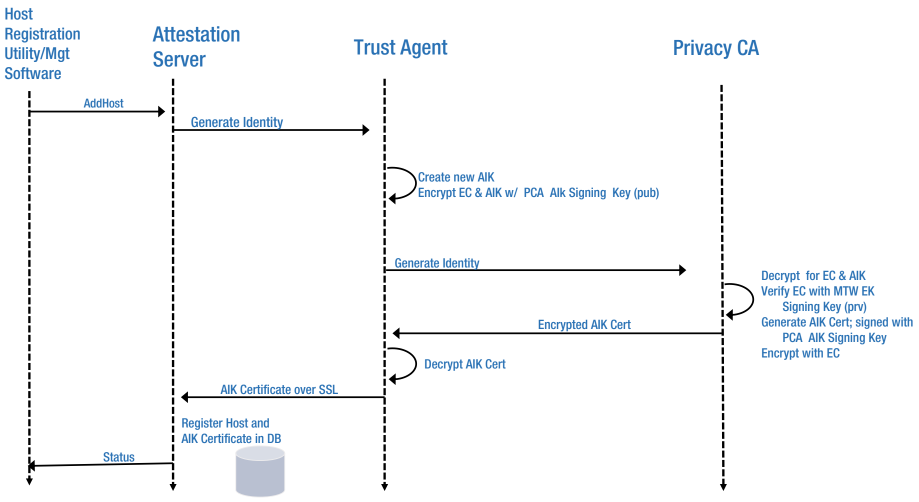

Host Registration Utility/Mgt Software
Attestation Server
Trust Agent
Privacy CA
AddHost
Generate Identity
Create new AIK
Encrypt EC & AIK w/ PCA Alk Signing Key (pub)
Generate Identity
Decrypt for EC & AIK
Verify EC with MTW EK Signing Key (prv)
Generate AIK Cert; signed with PCA AIK Signing Key
Encrypt with EC
Encrypted AIK Cert
Decrypt AIK Cert
AIK Certificate over SSL
Register Host and AIK Certificate in DB
Status

***Figure 4-6.** Flow of authority identity key certificate provisioning*

- The host registration process begins with an API request to the attestation server. This request may come from a system administrator using a management portal, or from an automated system in charge of managing hosts in the data center.
- The attestation server sends an attestation identity key provisioning request to the trust agent on the host using a TLS connection secured by the trust agent TLS certificate.
- The trust agent uses the TPM to create a new AIK private and public key pair. It sends the AIK public key and the endorsement credential to the privacy CA, encrypted using the privacy CA's public key to ensure privacy.
- The privacy CA decrypts the AIK public key and endorsement credential using its private key. It then generates a random challenge and encrypts it using the public key certified by the endorsement credential. It sends this challenge to the host.
- The host decrypts the challenge using the endorsement key, a private key corresponding to the endorsement credential. It re-encrypts the challenge using the privacy CA's public key for privacy and sends the re-encrypted challenge to the privacy CA.

- The privacy CA decrypts the challenge to verify it is correct, then certifies the host's AIK public key. The privacy CA sends the AIK certificate to the host, encrypted using the public key in the host's endorsement credential.
- The host decrypts the AIK certificate using its endorsement key.
- The host sends the AIK certificate to the attestation server over the trust agent TLS connection.
- The attestation server registers the host and stores the AIK certificate in the database.
- The attestation server responds to the system administrator or automated system, indicating the success or failure of the registration process.

Requesting Platform Trust

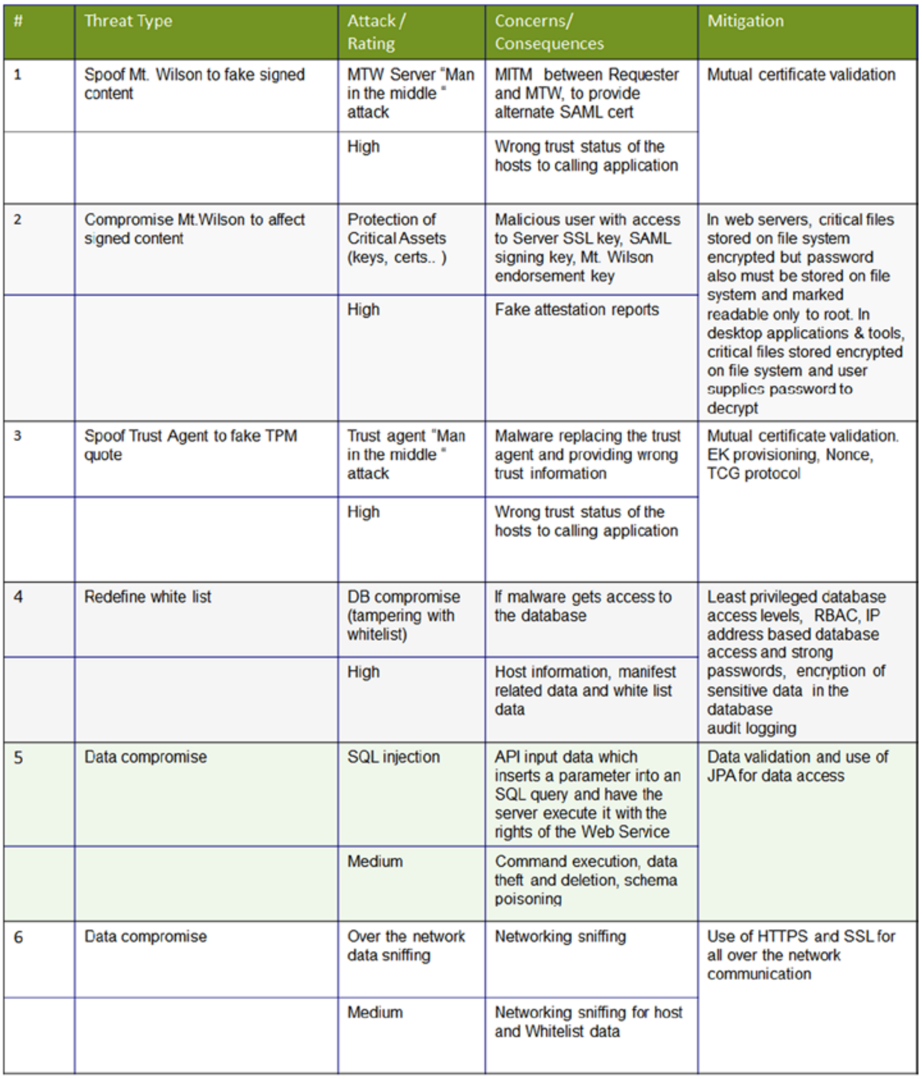
This is the invocation of the trust APIs by an entity requesting trust information. The API request is authenticated and the input parameters are validated and then handed to the appraiser component of the attestation server. The appraiser follows the remote attestation protocol to challenge the platform for the integrity measurements. Once the verification is done, Mt. Wilson summarizes all these steps by generating a SAML assertion of the platform compliance with its trust policy. Details of the SAML assertion and the security and integrity of the exchange are covered later in this chapter.

Security of Mt. Wilson

Security is integral to the Mt. Wilson platform. The ultimate objective of an adversary of Mt. Wilson would be to subvert and control the outcome of the attestation by:

- Spoofing the trust agent to attain a fake TPM quote
- Compromising the Mt. Wilson attestation server to subvert signed content
- Spoofing the Mt. Wilson attestation server to fake a signed content
- Hacking the whitelists
- Compromising the data on the network and repositories

Figure 4-7 shows the threat model considered during the design of Mt. Wilson, with articulation of the consequences when the adversary accomplishes the attack and possible mitigations implemented. We summarize the mitigating actions against the threats listed above.



#	Threat Type	Attack / Rating	Concerns/ Consequences	Mitigation
1	Spoof Mt. Wilson to fake signed content	MTW Server "Man in the middle " attack	MITM between Requester and MTW, to provide alternate SAML cert	Mutual certificate validation
		High	Wrong trust status of the hosts to calling application	
2	Compromise Mt.Wilson to affect signed content	Protection of Critical Assets (keys, certs..)	Malicious user with access to Server SSL key, SAML signing key, Mt. Wilson endorsement key	In web servers, critical files stored on file system encrypted but password also must be stored on file system and marked readable only to root. In desktop applications & tools, critical files stored encrypted on file system and user supplies password to decrypt
		High	Fake attestation reports	
3	Spoof Trust Agent to fake TPM quote	Trust agent "Man in the middle " attack	Malware replacing the trust agent and providing wrong trust information	Mutual certificate validation. EK provisioning, Nonce, TCG protocol
		High	Wrong trust status of the hosts to calling application	
4	Redefine white list	DB compromise (tampering with whitelist)	If malware gets access to the database	Least privileged database access levels, RBAC, IP address based database access and strong passwords, encryption of sensitive data in the database audit logging
		High	Host information, manifest related data and white list data	
5	Data compromise	SQL injection	API input data which inserts a parameter into an SQL query and have the server execute it with the rights of the Web Service	Data validation and use of JPA for data access
		Medium	Command execution, data theft and deletion, schema poisoning	
6	Data compromise	Over the network data sniffing	Networking sniffing	Use of HTTPS and SSL for all over the network communication
		Medium	Networking sniffing for host and Whitelist data	

Figure 4-7. *Mt. Wilson threat analysis*

- Registered API client calls (signed with their private key) can be verified by the Mt. Wilson attestation server using the corresponding public key. These keys get generated and stored by the API client during the registration process. Users are encouraged to secure their private keys using a password-based mechanism, at minimum. The Mt. Wilson Java API Client Library includes convenient functions for this purpose, using the Java Key store format to secure the private keys.

- The communication channels between the hosts and the users are encrypted using SSL. When a new user registers with Mt. Wilson, the Mt. Wilson SSL TLS certificate is verified and stored by the user to secure subsequent communication between the user and Mt. Wilson. The trust agent stores its SSL TLS certificate with Mt. Wilson upon registration of a new host to secure all future communication between Mt. Wilson and the trust agent.
- Trust agents store their TLS private keys in a password-protected Java Keystore file.
- Users are allowed to call into APIs based on their existing roles. Users request roles during registration with Mt. Wilson and these are approved by the Mt. Wilson administrator.
- The attestation status of the hosts is returned as signed SAML assertions that can be verified by the end consumer. The Mt. Wilson SAML certificate is stored by users when they register with Mt. Wilson in order to later verify SAML assertions.
- A public and private key pair is the preferred authentication mechanism for management of the whitelist and host trust policies.

Mt. Wilson Trust, Whitelisting, and Management APIs

Mt. Wilson provides a rich set of APIs for all interactions with it. In fact, the primary communication with the Mt. Wilson attestation authority is via authenticated APIs. There are five categories of APIs:

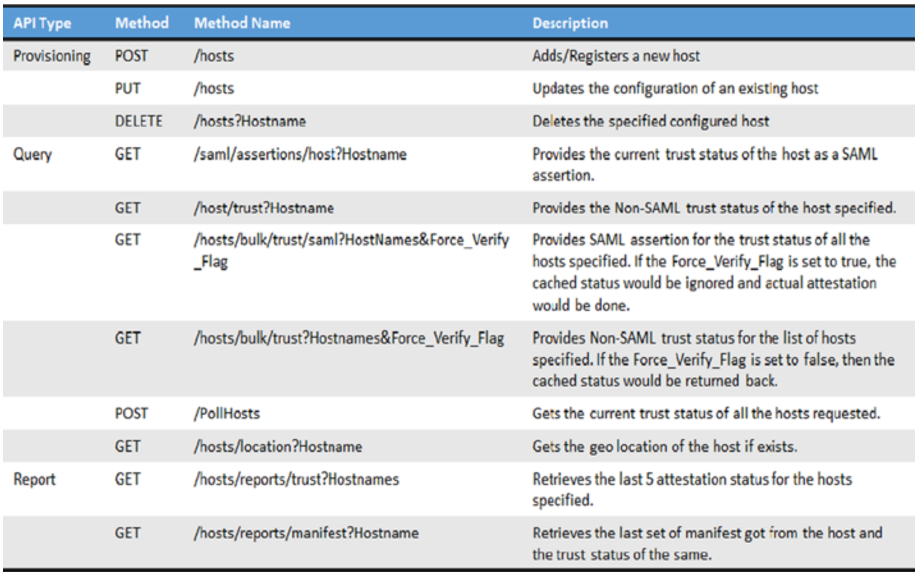
1. *Provisioning APIs,* for registering hosts and requesting AIKs.
2. *Query APIs,* the trust APIs that requesting entities (requesters/ API clients) invoke to get a trust assertion.
3. *Reporting APIs,* providing details about hosts registered with Mt. Wilson, including the current measurements and the whitelists.
4. *Automation APIs,* allowing an administrator to easily register all hosts within a VMware cluster or create an MLE using a known-good host in a trusted environment.
5. *Management APIs,* enabling registering users, managing their authorized roles, and downloading various certificates managed by Mt. Wilson.

Calls to the API must be sent over SSL TLS. All APIs are REST-based. Mt. Wilson APIs use a client-server model without third-party intervention to provide authentication. The authentication model is very similar to OAuth 1.0 and HTTP Digest, and it provides a stateless scheme for use with clusters and load balancers. However, it does not work with URL-rewriting proxies because the URL is covered by the client's signature. Every API client—that is, any entity invoking the APIs, such as portals, schedulers, other subsystems or policy engines—needs two RSA keys, as follows:

- *API signing key*. The public portion of the API signing key is stored in the Mt. Wilson keystore. The API client retains the private portion of this key in an encrypted and secure keystore
- *SAML assertion validation key*. This is the public portion of the Mt. Wilson SAML signing key and is stored with the API client
- An API client registers with Mt.Wilson via a credential management server to acquire the RSA keys. A Mt.Wilson instance can register a number of API clients.

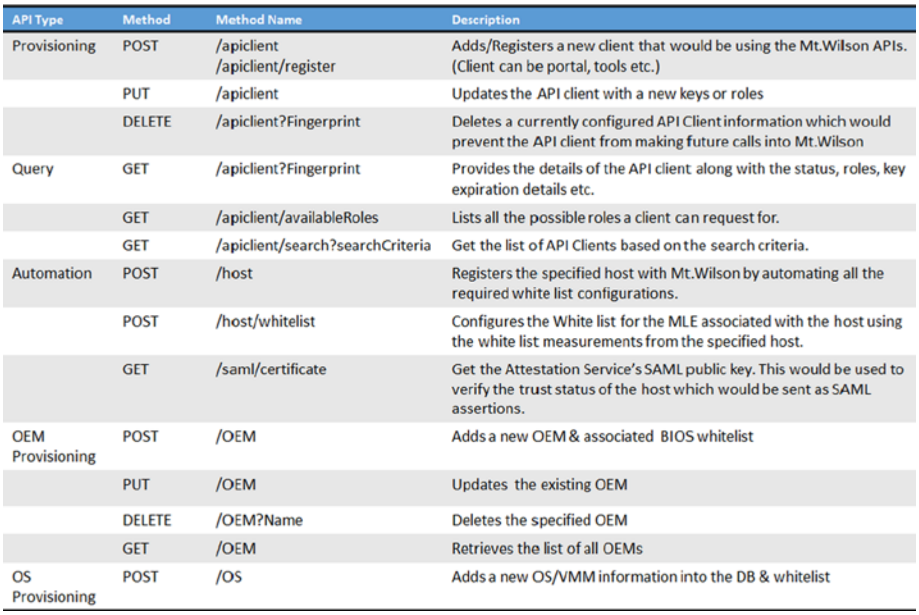
Mt. Wilson APIs

Figures 4-8 and 4-9 show the core APIs for the Mt. Wilson provisioning and trust query API and the management and whitelisting API.



API Type	Method	Method Name	Description
Provisioning	POST	/hosts	Adds/Registers a new host
	PUT	/hosts	Updates the configuration of an existing host
	DELETE	/hosts?Hostname	Deletes the specified configured host
Query	GET	/saml/assertions/host?Hostname	Provides the current trust status of the host as a SAML assertion.
	GET	/host/trust?Hostname	Provides the Non-SAML trust status of the host specified.
	GET	/hosts/bulk/trust/saml?HostNames&Force_Verify_Flag	Provides SAML assertion for the trust status of all the hosts specified. If the Force_Verify_Flag is set to true, the cached status would be ignored and actual attestation would be done.
	GET	/hosts/bulk/trust?Hostnames&Force_Verify_Flag	Provides Non-SAML trust status for the list of hosts specified. If the Force_Verify_Flag is set to false, then the cached status would be returned back.
	POST	/PollHosts	Gets the current trust status of all the hosts requested.
	GET	/hosts/location?Hostname	Gets the geo location of the host if exists.
Report	GET	/hosts/reports/trust?Hostnames	Retrieves the last 5 attestation status for the hosts specified.
	GET	/hosts/reports/manifest?Hostname	Retrieves the last set of manifest got from the host and the trust status of the same.

Figure 4-8. *Provisioning and trust query API*



API Type	Method	Method Name	Description
Provisioning	POST	/apiclient /apiclient/register	Adds/Registers a new client that would be using the Mt.Wilson APIs. (Client can be portal, tools etc.)
	PUT	/apiclient	Updates the API client with a new keys or roles
	DELETE	/apiclient?Fingerprint	Deletes a currently configured API Client information which would prevent the API client from making future calls into Mt.Wilson
Query	GET	/apiclient?Fingerprint	Provides the details of the API client along with the status, roles, key expiration details etc.
	GET	/apiclient/availableRoles	Lists all the possible roles a client can request for.
	GET	/apiclient/search?searchCriteria	Get the list of API Clients based on the search criteria.
Automation	POST	/host	Registers the specified host with Mt.Wilson by automating all the required white list configurations.
	POST	/host/whitelist	Configures the White list for the MLE associated with the host using the white list measurements from the specified host.
	GET	/saml/certificate	Get the Attestation Service's SAML public key. This would be used to verify the trust status of the host which would be sent as SAML assertions.
OEM Provisioning	POST	/OEM	Adds a new OEM & associated BIOS whitelist
	PUT	/OEM	Updates the existing OEM
	DELETE	/OEM?Name	Deletes the specified OEM
	GET	/OEM	Retrieves the list of all OEMs
OS Provisioning	POST	/OS	Adds a new OS/VMM information into the DB & whitelist

Figure 4-9. *Management and whitelisting API*

To facilitate interoperability, consistency, and seamless integration, we expect the industry to converge toward a standardized set of APIs related to attestation. We offer these as a starting point for the industry to help drive interoperability across different attestation solution implementations.

The API Request Specification

All API calls are http requests with one required header: "`Authorization: X509 <authentication-info>`". Any unauthorized request is challenged with a standard header: "`WWW-Authenticate: X509 <challenge-info>`".

Each API request includes the following parameters:

- Fingerprint (base64-encoded SHA-256 digest of the client API certificate)
- Signature method (RSA-SHA256)
- Time stamp from standard http Date header (RFC 822 date format)
- Client nonce (base64-encoded) in http X-Nonce header
- http request method
- Signature over the above and also:

    - Original request URL including query string
    - http message body (required, use empty string if not applicable)
    - Any other custom headers specified besides Date and X-Nonce in the "headers" field of the Authorization line, in the order specified
    - Signature created using client's RSA private key, and it is base64-encoded
- Strongest method is RSA-SHA256

Figure 4-10 is an example of a sample API request using authentication.

```
WWW-Authenticate: X509 realm="Attestation"
```

```
GET /reports/trust?hostname=example
Host: attestationservice.example.com
Authorization: X509
  realm="Attestation",
  algorithm="SHA256withRSA",
  fingerprint="0685bd9184jfhq2bafweK...",
  headers="Date,X-Nonce"
  signature="wOJIO9A2W5mFwDgiDvZbTSMK%2FPY%3D"
X-Nonce: 0123456789abcdef
Date: Sun, 06 Nov 1994 08:49:37 GMT
```

Figure 4-10. *API request including authentication*

API Response

Mt. Wilson asserts all API responses. Responses are signed SAML assertions. Assertions are signed with the Mt.Wilson RSA SAML signing key. There is one SAML signing key for each installation of Mt.Wilson. An API client validates the signature with the SAML public key and uses the trust information. Here is an example of an API invocation with a SAML assertion. This Java example uses the Apache HttpClient library to obtain the SAML assertion for "192.168.1.121" by sending a *GET* request to Mt. Wilson:

```
ApiClient api = KeystoreUtil.clientForUserInDirectory(directory, username,
password, server);
String samlForHost = api.getSamlForHost(new Hostname("192.168.1.121"));
```

Here's how to interpret the SAML response:

```
TrustAssertion trustAssertion = api.verifyTrustAssertion(samlForHost);
if( trustAssertion.isValid() )
        for(String attr : trustAssertion.getAttributeNames())
                System.out.println("Attr: "+attr+":"+trustAssertion.
getStringAttribute(attr));
```

Attributes for subject's trust status in the SAML response are:

- *Trusted*: True if both *Trusted_BIOS* and *Trusted_VMM* are true.
- *Trusted_BIOS:* True if the BIOS measurements on the subject match the whitelist (known-good values)
- *Trusted_VMM*: True if the VMM measurements on the subject match the whitelist (known-good values)

Attributes for subject's measured launch environment in the SAML response are:

```
BIOS_Name, BIOS_Version, BIOS_OEM, VMM_Name, VMM_Version, VMM_OSName, VMM_OSVersion
```

Mt. Wilson API Usage

There are two options for the requesters of attestation information to call into Mt. Wilson APIs. A direct invocation of the REST APIs is the most basic approach to use and integrate with Mt. Wilson. The user is required to implement the complete API request specifications. This would mean pre-processing the creation and handling of keys and authentication, and post-processing of information for a successful API invocation, and the correct processing of the responses. An API toolkit (called API Client Library) is available to simplify the invocation of the APIs, with bindings for different languages like Java, C#, and Python. This toolkit encapsulates multiple API calls, creation and handling of RSA keys and certificates, and authentication and processing of API responses (which are SAML signed assertions). Using this toolkit, the users can make Java (or C# or Python) function calls to communicate with the system. The sample code and examples that are used in this chapter use the Java binding of the API toolkit.

There are three different options for the *.jar* file:

1. Zip file containing the *api-client.jar* and related dependencies
2. Single *.jar* with dependencies
3. Single *.jar* with dependencies shaded to prevent conflicts with other libraries

Deploying Mt. Wilson

There are multiple models for deploying attestation components in a data center. Ideally, attestation is transparent to applications, carrying its function quietly in the background. In practice, it's far from that. How unobtrusive attestation technology is depends upon the deployment method. Some of the possible models include:

- Dedicated virtual appliances
- Dedicated physical appliances
- Integrated as a function in security application software

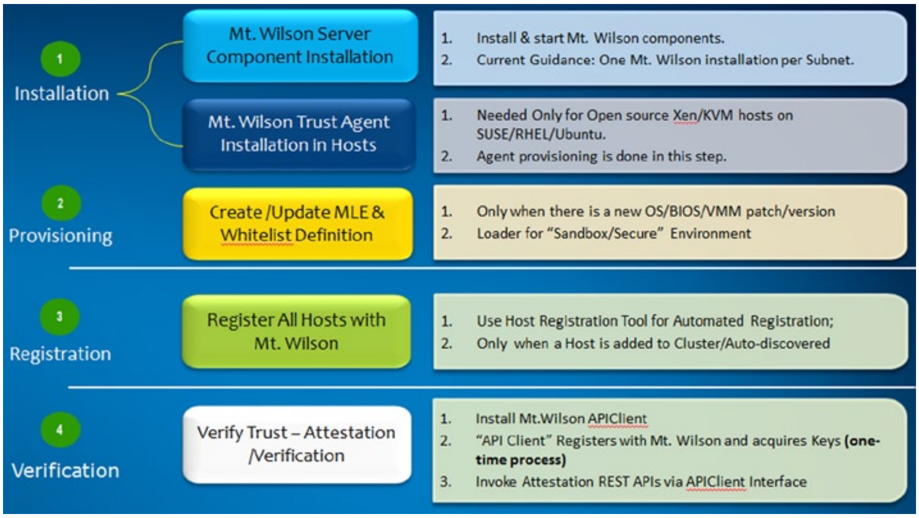
- Integrated in cloud and virtualization management software
- Offered as a component of a cloud service
- Integrated as a attestation of a service

Mt. Wilson is delivered today as a virtual appliance, and it is being integrated into security software applications such as HyTrust's Cloud Control, as well as cloud management software such as Virtustream's xStream. An initial approach for adoption is to package and deliver Mt. Wilson software as a separate appliance with cloud management and security management independent software vendor offerings. As the usage and experiences increase with increased design and development of attestation-based solutions, other models with tighter integration will become possible.

As attestation APIs become standardized and integral to the interactions and operations of a trusted cloud infrastructure, there is opportunity for providing value-added services on top of the core attestation APIs. This could lead security management and cloud service providers to offer attestation as a service, with granular control to the usage and evolution of the APIs.

Mt. Wilson Programming Examples

In this section, we look at how to invoke the attestation APIs to get trust information about a server in a data center. Figure 4-11 shows the high-level steps involved in setting up the system and configuring it for use.


1
Installation
Mt. Wilson Server Component Installation
Mt. Wilson Trust Agent Installation in Hosts
2
Provisioning
Create /Update MLE & Whitelist Definition
3
Registration
Register All Hosts with Mt. Wilson
4
Verification
Verify Trust – Attestation /Verification

1. Install & start Mt. Wilson components.
2. Current Guidance: One Mt. Wilson installation per Subnet.

1. Needed Only for Open source Xen/KVM hosts on SUSE/RHEL/Ubuntu.
2. Agent provisioning is done in this step.

1. Only when there is a new OS/BIOS/VMM patch/version
2. Loader for "Sandbox/Secure" Environment

1. Use Host Registration Tool for Automated Registration;
2. Only when a Host is added to Cluster/Auto-discovered

1. Install Mt.Wilson APIClient
2. "API Client" Registers with Mt. Wilson and acquires Keys **(one-time process)**
3. Invoke Attestation REST APIs via APIClient Interface

Figure 4-11. *Mt. Wilson high-level programming steps*

After the installation of the Mt.Wilson server and trust agent on the hosts, required only for Xen or KVM hosts, users need to include the *.jar* file provided as part of the API toolkit in their project and import the following packages:

```
import com.intel.mtwilson.*;
import com.intel.mtwilson.crypto.*;
import com.intel.mtwilson.datatypes.*;
import java.io.File;
import java.net.URL;
```

API Client Registration Process

Before the user can make any API calls into the system, the user has to register and the access has to be approved. Below are steps for how to register with Mt. Wilson and how to make API calls after the registration has been accepted. The following code creates a keystore "test1.jks" in the home directory. The keystore contains an RSA keypair that is used to authenticate the API calls to the system. The keystore would also contain the Mt. Wilson SSL certificate and SAML signing certificate, which are downloaded from the server.

```
File directory = new File(System.getProperty("user.home", "."));
String username = "test1"; // you choose a username
String password = "changeit"; // you choose a password
URL server = new URL("https://mtwilson.example.com:8181"); // attestation server
String[] roles = new String[] { "Attestation", "Whitelist" };
KeystoreUtil.createUserInDirectory(directory, username, password, server,
roles);
```

After the request is created, the user has to contact the system administrator to approve the access request (offline step). After the request is approved, based upon the roles the user has, appropriate APIs can be executed, such as maintaining a whitelist, adding hosts, and obtaining a trust assertion on one or more hosts.

To use the API, the user needs first to create an `ApiClient` object configured with the credentials and the attestation server. Notice that the variables directory, username, password, and servers are the same as what was used during registration.

```
File directory = new File(System.getProperty("user.home", "."));
String username = "test1"; // username created during registration
String password = "changeit"; // password created during registration
URL server = new URL("https://mtwilson.example.com:8181");
ApiClient apiClientObj = KeystoreUtil.clientForUserInDirectory(directory,
username, password, server);
```

Once an APIClient object is created, the user can use that to configure whitelists and also to register the hosts with Mt. Wilson so that they attest when challenged.

Whitelisting and Host Registration

Here's some sample code for how to create a whitelist and register the host with Mt. Wilson—for VMware ESXi hosts:

```
TxtHostRecord gkvHostObj = new TxtHostRecord();
gkvHostObj.HostName = "hostname-in-vcenter";
gkvHostObj.AddOn_Connection_String =
"vmware:https://vcenter.example.com:443/sdk;Username;Password";
boolean configureWhiteList = apiClientObj.configureWhiteList(gkvHostObj);

boolean registerHost = apiClientObj.registerHost(gkvHostObj);
```

Verify Trust: Trust Attestation

Once hosts are registered with Mt Wilson, it is now possible to request a trust assertion in SAML format using `getSamlForHost`. You can verify the signature on the assertion and get easy access to the details using `verifyTrustAssertion`.

■ **Note** If you are directly calling into the REST APIs, you have to implement the verification of the SAML assertion using the SAML certificate that needs to be downloaded explicity. The API toolkit downloads this certificate as part of the registration itself.

```
String samlForHost = apiClientObj.getSamlForHost(new Hostname("hostname-in-
vcenter"));
TrustAssertion trustAssertion = apiClientObj.verifyTrustAssertion(samlForHost);
if(trustAssertion.isValid()) {
for(String attr : trustAssertion.getAttributeNames())
        System.out.println("Attr:"+attr+":"+trustAssertion.
getStringAttribute(attr));
}
```

As shown in this above example, using the API Client Library is a very simple way of using the Mt. Wilson attestation mechanism. The Mt. Wilson software is being licensed by many ISV and CSPs to integrate trust into the software and service offerings. More and more organizations are moving to clouds, and they are asking for assurance of trust of the platform on which their workloads are running; they are also asking the CSPs to provide proof of a chain of trust. The attestation solution is fast becoming a critical security component in the security toolset. For developers favoring a DIY approach, the open-source OpenAttestation (OAT) is a good starting point for attestation.

■ **Note** OAT is the open-source version of Mt. Wilson code, and is provided and maintained by Intel Corporation. You can download the documentation, code, and installation/deployment scripts from the OAT website.

Summary

In this chapter we covered attestation as a foundational function of trusted computing environments that provides proof of trustability and auditability of trust for various computing devices. We covered the TCG remote attestation protocol, and we described the vision and architecture of Intel's Trust Attestation Platform, followed by a detailed look one of the first attestation solutions, called Mt. Wilson. The chapter reviewed the security architecture and the attestation APIs, and explained how requesters of trust and attestation information can invoke these APIs and process the assertions for decision making. There are many usages in data centers that would utilize the attestation information. As shown in the previous chapter, attestation is used in the creation of trusted compute pools and the attestation-based policy enforcement in these pools. Thus, attestation can be used to provide granular trust-based access control to consumer and BYOD devices, and the kind of services they can access within the cloud data centers. Attestation as a security management component will become an integral component of virtualization and cloud management, and it's becoming a critical requirement in cloud data centers to assert the integrity and compliance of platforms and systems. ISVs and security management vendors may also start offering it as a SaaS offering. We believe that, over time, value-added capabilities will emerge around the attestation function and will enable monetization possibilities.

Chapter 5 will introduce a new concept and control, called hardware-assisted *asset tag*, which can be used to provide isolation, segregation, placement, and migration control of workload execution in multi-tenant cloud environments. Additionally, as a specialization of asset tags, geolocation/geo-tagging can be enabled to definitively provide increased visibility to the physical geolocation of the server, which may enable many controls that require hardware-based roots of trust to assert the location of workloads and data. These attributes and the associated controls are dependent on the boot integrity assertion of the platform; hence, they become a great adjacency to trusted compute pools and boot integrity.