POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Attestazione remota di sistemi cloud

Relatori
prof. Antonio Lioy
dott. Marco De Benedictis

Candidato
Davide Renna

Anno accademico 2017-2018

A mio padre

† A mio nonno Pino

Sommario

Dati e codice all'interno di un ambiente Cloud costituiscono risorse molto importanti per gli utenti del Cloud stesso. Spesso all'interno di un'infrastruttura Cloud si deve far fronte ad un problema di gestione di dati e software dovuto alla presenza di un numero molto elevato di macchine. Un'azione accidentale o intenzionale che coinvolga in qualche modo software e/o dati potrebbe minare alla fruizione dei servizi al generico utente. Sarebbe molto importante garantire l'integrità di codice e dati, riuscendo quindi a comprendere la presenza di un'eventuale manipolazione. Il TC permette di definire delle architetture di Cloud robuste a questo tipo di minacce proponendo una soluzione al problema dell'Attestazione Remota. L'Attestazione Remota è un metodo che consente all'hardware e al software di un host di essere autenticato tramite un altro host remoto in modo da poter definire lo stato di integrità della piattaforma. Per trovare la soluzione di attestazione utile agli scopi del lavoro di tesi sono state analizzate varie soluzioni tra le quali: Excalibur, Cobweb, Intel Software Guard Extensions, Open Attestation e Open CIT. La scelta è ricaduta su OpenCIT un progetto open source realizzato da Intel. Tale progetto sfrutta le specifiche di attestazione del TC per recuperare delle informazioni chiave che consentano di definire lo stato di affidabilità della macchina. Il problema principale di questa tecnologia è che il processo di attestazione attualmente implementato si limita a verificare le misure dei componenti software a "load time", ossia in fase di avvio della macchina, ma non tiene conto di eventuali modifiche ai componenti software eseguiti a "run time", ossia in fase di esecuzione dei nodi Cloud. In questo lavoro di tesi ci si pone come obiettivo quello di estendere Open CIT integrando al suo interno un'architettura nota come IMA. Tale soluzione prevista per kernel Linux, permette di effettuare un controllo di integrità a runtime, consentendo di avere un flusso di informazioni costante che indichi il comportamento della macchina in esecuzione. Le informazioni raccolte dall'host prendono il nome di misure. Una misura è un digest calcolato su un componente della macchina (e.g. file, hardware, software). Le misure raccolte dall'host danno la possibilità di avere uno stato che rispecchi la situazione attuale della macchina. Il lavoro di tesi sviluppato propone due differenti scenari per realizzare la logica di verifica delle misure IMA raccolte dall'host. Il primo scenario adotta la logica di verifica proposta da Open CIT consentendo di verificare le misure attraverso la definizione di valori di Whitelist statici al momento della registrazione dell'host. Nel secondo scenario si assiste ad una soluzione più dinamica, dal momento che la verifica delle misure IMA si sposta al di fuori di Open CIT e non viene definito a priori alcun valore che venga utilizzato nelle operazioni di verifica. Questo grazie al fatto che la logica di verifica esterna si basa su un Verifier che prende le informazioni necessarie alle operazioni di verifica da un database costantemente aggiornato. In questo modo non si ha il bisogno di registrare nuovi valori di Whitelist per le misure IMA ogni qualvolta venga eseguito nuovo software. Inoltre sfruttando il Verifier esterno è possibile ricavare delle informazioni aggiuntive sui pacchetti di cui le misure IMA fanno parte. Queste informazioni includono il numero dei pacchetti che hanno bisogno di un aggiornamento di sicurezza e il numero di pacchetti che hanno bisogno di un aggiornamento dovuto alla presenza di bug.

Indice

Capitolo 1

Introduzione

Il NIST [1] definisce il *Cloud Computing* come un modello che permette un accesso di rete on-demand ad un insieme condiviso di risorse computazionali (e.g. reti, server, storage, applicazioni e servizi) che possono essere rapidamente forniti con un minimo sforzo di gestione e di interazioni tra i vari fornitori dei servizi. I vantaggi che il Cloud offre sono:

- **Risparmio sui costi**: il cloud computing elimina le spese di capitale associate all'acquisto di hardware e software e alla configurazione e alla gestione di data center locali, che richiedono rack di server, elettricità 24 ore su 24 per alimentazione e raffreddamento ed esperti IT per la gestione dell'infrastruttura;
- **Scalabilità globale:** significa avere la possibilità di poter, in modo flessibile e dinamico, fornire la giusta quantità di risorse IT, ad esempio una quantità maggiore o minore di potenza di calcolo, risorse di archiviazione e larghezza di banda, quando è necessario e dalla posizione geografica appropriata;
- **Prestazioni:** i servizi che il cloud offre vengono eseguiti su data center sicuri, aggiornati regolarmente all'ultima generazione di hardware, veloce ed efficiente. Inoltre grazie alla ridotta latenza di rete è possibile fornire i vari servizi più rapidamente;
- **Affidabilità:** il cloud computing aumenta la semplicità e riduce i costi di backup dei dati, ripristino di emergenza e continuità aziendale, grazie alla possibilità di eseguire il mirroring dei dati in più siti ridondanti nella rete del provider di servizi cloud;
- **Accessibilità del sistema:** se si possiede una connessione ad Internet è possibile accedere al sistema ovunque e inoltre non si hanno vincoli legati al tipo di dispositivo che si usa per poter accedere.

La crescente diffusione del cloud lo ha reso però un obiettivo sensibile per gli attaccanti. Infatti secondo i dati riportati dal Security Report di Microsoft [2] il numero di attacchi sul cloud è incrementato del 300% dal 2016 al 2017.

È a tal proposito che la *Cloud Security Alliance*, un'organizzazione no-profit che si pone come obiettivo quello di definire quali sono le migliore procedure per creare un ambiente cloud sicuro, ha stilato una classifica [3] indicando le dodici maggiori minacce che interessano un ambiente cloud. Da questa classifica si può evincere come cancellazione e modifiche dei dati, modifiche al software che garantisce il corretto funzionamento del cloud, versioni non aggiornate dello stesso, possano costituire dei seri problemi, minacciando l'affidabilità dell'ambiente stesso.

Normalmente un sistema Cloud ospita un numero abbastanza elevato di nodi, ovvero macchine server utilizzate per fornire vari servizi all'utilizzatore dello stesso. L'elevata quantità di macchine, unita all'altrettanto grande quantità di software che viene eseguito in esse, rendono difficile la gestione da parte dei fornitori dei servizi. Questo è sicuramente un punto a vantaggio di un

attaccante, che approfittando delle difficoltà nel rendere sicuro un sistema di tali proporzioni, può lanciare i suoi attacchi e mettere in crisi il sistema stesso così come gli utenti che ne usufruiscono.

Nel corso degli anni si sono fatte strada alcune soluzioni a questi problemi ed in particolare è emerso il concetto di *Remote Attestation*, ovvero un metodo che consente all'hardware e al software di un host di essere autenticato tramite un altro host remoto in modo da poter definire lo stato di *integrità* della piattaforma. L'integrità è una proprietà di sicurezza che consente di capire se è avvenuta o meno la modifica e/o la cancellazione di una qualche informazione del sistema. La tecnica di Remote Attestation però non ha ancora uno standard che definisca in che modo debba essere implementata e questo ha fatto sì che in letteratura fossero sviluppati approcci differenti (Shi *et al.* [4], Garfinkel *et al.* [5], Petroni *et al.* [6]). In questo lavoro di tesi invece si farà riferimento alla proposta avanzata dal *Trusted Computing Group* (TCG).

Il TCG è un'organizzazione non-profit nata con l'obiettivo di sviluppare, definire e promuovere standard aperti a supporto di una "root of trust" basata su un hardware noto come *Trusted Platform Module* (TPM).

Il TCG [7] basa il processo di attestazione sul concetto di *Integrity Measurement*, ovvero un digest definito come un "riassunto" a lunghezza fissa dell'elemento da proteggere, sia esso codice o sia esso dati, calcolato attraverso una funzione crittografica, nota come funzione di *hash*. Il TCG individua tre componenti fondamentali (figura 1.1) coinvolti nel processo di attestazione remota:

- il Requestor, ossia la macchina che deve essere attestata per comprenderne lo stato corrente;
- il Verifier, ossia la macchina che ha il compito di verificare che le misure provenienti dal Requestor siano valide;
- il Relying Party, ossia colui che si fida del risultato prodotto durante il processo di verifica e che ha la possibilità di utilizzarlo per i suoi scopi.

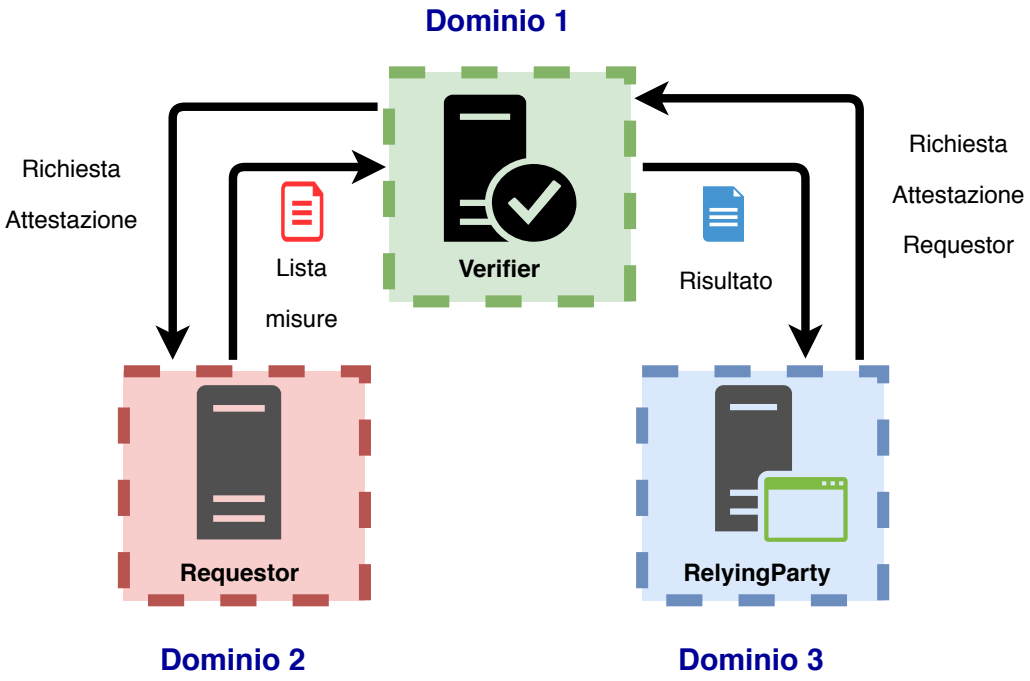

Figura 1.1. Modello base di attestazione remota secondo il TCG

È molto importare garantire che le misure che vengono inviate al Verifier non vengano alterate e possano così essere considerate affidabili. Per tale motivo tali misure, una volta calcolate, vengono memorizzate all'interno di un dispositivo hardware conosciuto come *Trusted Platform Module*

(TPM). Il TPM permette di "implementare un meccanismo fiduciario su generiche piattaforme di elaborazione" (ISO/IEC 11889 [8]) consentendo di garantire che una piattaforma implementi le tre proprietà che caratterizzano una "piattaforma fidata", ovvero:

- utilizzo di funzioni protette che siano in grado di accedere in modo sicuro alle locazioni protette;
- capacità di calcolare le misure di integrità che caratterizzano la piattaforma;
- reporting, ossia attestare la misure precedentemente calcolate.

Partendo dalle specifiche di attestazione definite dal TCG, sono nati alcuni progetti Open Source come *Open Attestation* (OAT)[9] e *Open Cloud Integrity Technology* (Open CIT)[10], il cui scopo è quello di attestare le macchine presenti all'interno di un'infrastruttura cloud, mettendo in atto il processo di misurazione definito dal TCG. Il problema principale di queste due tecnologie è che il processo di attestazione attualmente implementato si limita a verificare le misure dei componenti software a "load time", ossia in fase di avvio della macchina, ma non tiene conto di eventuali modifiche ai componenti software eseguiti a "run time", ossia in fase di esecuzione dei nodi cloud.

I dati e il codice all'interno dei nodi del Cloud, come si è visto, costituiscono delle risorse molto importanti per gli utenti che fanno uso dello Cloud stesso, quindi sarebbe importante mantenere e controllare costantemente l'integrità degli stessi con lo scopo di verificare che siano rimasti intatti. Più specificatamente per integrità dei dati si intende che i dati dovrebbero essere protetti da modifiche non autorizzate. Invece per integrità di computazione si intende che l'esecuzione del programma dovrebbe essere protetta da malware, insider o utenti malintenzionati che potrebbero cambiare l'esecuzione del programma e rendere un risultato incorretto.

L'obiettivo che questo lavoro di tesi si propone è quello di integrare nel sistema di attestazione di Open CIT una tecnologia di misurazione, nota come *Integrity Measurement Architecture*(IMA)[11], integrata nel kernel Linux e sviluppata da IBM. Tale soluzione permette di effettuare un controllo d'integrità a runtime per eventi eseguiti a livello di sistema operativo, intendendo per evento l'apertura di un file in lettura o l'esecuzione di un binario. In questo modo sarà possibile ottenere costantemente delle informazioni di integrità, per poter verificare che ogni macchina, presente nel sistema, sia in uno stato ritenuto affidabile.

Capitolo 2

Trusted Computing e Attestazione Remota

2.1 Obiettivi del Trusted Computing

Con il termine Trusted Computing si intende un insieme di idee e di proposte per un'architettura PC "locked-down", cioè un'architettura messa in sicurezza che sia capace di dare delle garanzie circa il software in esecuzione e che permetta alle applicazioni di comunicare in modo sicuro tra di loro e con i server. Il TCG promuove i seguenti principi:

i. **Sicurezza:** i componenti del sistema dovrebbero garantire accesso controllato ai dati considerati critici e dovrebbero misurare ed effettuare un reporting affidabile delle proprietà di sicurezza del sistema. Il processo di reporting dovrebbe essere completamente sotto il controllo del proprietario della macchina.

ii. **Privacy:** i componenti del sistema dovrebbero essere progettati ed implementati seguendo le specifiche di privacy suggerite dalla legge.

iii. **Interoperabilità:** l'implementazione e la messa in campo di una soluzione conforme alle specifiche del TCG dovrebbero facilitare l'interoperabilità e non introdurre ostacoli se non per scopi di sicurezza.

iv. **Portabilità dei dati:** la messa in campo della soluzione dovrebbe supportare i principi e le pratiche stabilite per la proprietà dei dati.

v. **Controllabilità:** ogni proprietario dovrebbe avere l'effettiva scelta e controllo su quali caratteristiche del TCG seguire. Di conseguenza anche l'utente dovrebbe avere la possibilità di scegliere a quali caratteristiche del TCG aderire non violando però la policy del proprietario.

vi. **Facilità d'uso:** gli utenti non tecnici dovrebbero trovare comprensibili ed usabili le funzionalità conformi al TCG.

Questi principi difficilmente possono essere garantiti in maniera isolata senza che vadano ad interferire tra loro, tuttavia nella maggior parte dei casi i principi che sembrano in conflitto in realtà si supportano e sono complementari tra loro.

2.2 Applicazioni del Trusted Computing

Ci sono molte possibili applicazioni per il Trusted Computing. In questa sezione ne verranno illustrate solo alcune.

Balacheff *et al.* [13] e Spalka, Cremers e Langweg [14] hanno proposto, in maniera indipendente, di utilizzare le funzionalità del trusted computing per migliorare il processo di firma digitale. Balacheff cerca di rendere più sicuro il processo di firma di un documento su una piattaforma utilizzando un'immagine che rappresenta il documento, un *trusted display controller* e una *smart card* posseduta dal firmatario che comunica con il TPM. Spalka, Cremers e Langweg invece cercano di proteggere le operazioni di firma dalla presenza di virus come *Trojan horses*, attraverso l'utilizzo del modello *Intelligent Adjunct* definito dal TC. Tale modello cerca di assicurarsi che, prima di effettuare qualunque operazione di firma, i componenti chiave della piattaforma siano integri.

Schechter *et al.* [15] propongono l'utilizzo del TC per aumentare la sicurezza nei nodi terminali delle reti *peer-to-peer* (P2P) per proteggere i contenuti multimediali e i lettori di contenuti multimediali da chi vuole estrarre o copiare questi contenuti. L'azione di proteggere tali contenuti può essere effettuata attraverso la cifratura degli stessi e la trasmissione delle chiave di decifratura solo a quelle piattaforme ritenute affidabili. In maniera indipendente anche Kinateder e Pearson [16] hanno proposto l'uso del Trusted Computing come meccanismo per consentire la realizzazione di un sistema di reputazione distribuita nell'ambito del P2P con lo scopo di trasmettere informazioni differenti in base alla piattaforma che le richiede.

Il Trusted Computing può essere usato anche per fornire supporto alle soluzioni di Single-Sign On (SSO) come specificato da Pashalidis [17]. In tal caso una piattaforma che aderisce ai principi del TC può comportarsi come un Authentication Service Provider (ASP) in quanto è il TPM che, essendo identificato univocamente tramite un certificato, può permettere di verificare l'identità dell'host che di conseguenza consentirà all'utente di accedere ai vari Service Provider (SP).

Chen *et al.* [18] discutono un approccio che cerca di utilizzare le funzionalità offerte dal Trusted Computing per migliorare la sicurezza e la privacy in un processo di autenticazione biometrica dell'utente. Questo processo coinvolge tre attori: una *Smart Card* (SC) che contiene le credenziali biometriche dell'utente, un *Biometric Reader* (BR) che cattura le credenziali immesse dall'utente e una *Computing Platform* (CP), ovvero la piattaforma alla quale l'utente vuole accedere e che effettuerà il confronto tra le credenziali contenute nella SC e quelle catturate dal BR. Attraverso l'utilizzo del TPM si possono memorizzare le informazioni di integrità della piattaforma e del BR con lo scopo di verificarle durante il processo di autenticazione prima di concedere l'accesso all'utente. Lo scopo di questa verifica è quello di evitare di utilizzare dei componenti non fidati nel processo che possano catturare le informazioni biometriche dell'utente.

L'utilizzo dei meccanismi di attestazione remota definiti dal TC possono essere utilizzati per tutelarsi contro il fenomeno del *phishing* [21], ovvero quel fenomeno che consente di ottenere informazioni sensibili dando l'illusione all'utente di trasmetterle ad entità fidate. La maggior parte dei siti web utilizza il protocollo *Transport Layer Security*(TLS) per permettere l'autenticazione del server e opzionalmente l'autenticazione del client. Grazie all'utilizzo del TPM è possibile generare e certificare delle chiavi, tramite un meccanismo di certificazione definito dal TC e noto come *Privacy CA*. In questo modo ogni client avrà sempre un proprio certificato alla quale vengono associati username e password usati per l'autenticazione sul server web, consentendo un'autenticazione a due fattori.

Infine il TPM e il meccanismo di attestazione remota possono essere utilizzati per assicurare l'integrità e la privacy delle piattaforme facenti parte dell'infrastruttura cloud, come verrà trattato in seguito.

2.3 Trusted Platform Module

Il *Trusted Platform Module* (TPM) nacque nel 2003 [22] su iniziativa del TCG. Questo modulo hardware simile ad un coprocessore crittografico, fu progettato per assicurare un buon livello di sicurezza alle macchine che ne facevano uso. La sua prima versione è la 1.1b che presenta delle incompatibilità a livello hardware, non essendo in grado di supportare le diverse interfacce e i diversi driver dei vari venditori. Questo ha portato al passaggio alla versione 1.2 sviluppata negli anni dal 2005 al 2009 attraverso varie release e che sfrutta l'algoritmo di digest SHA1[23]. Le successive pubblicazioni che rivelavano dei possibili attacchi all'algoritmo SHA1, fecero abbandonare la versione 1.2 per far posto alla versione più recente, ovvero alla 2.0.

I maggiori problemi che il *TPM 1.2* vuole risolvere sono i seguenti:

Identificazione dei dispositivi: il TPM infatti possiede una chiave che essendo certificata può univocamente identificare il dispositivo che lo ospita.

Generazione sicura di chiavi: il TPM possiede un generatore di numeri random che consente di generare delle chiavi sicure. In passato molte soluzioni di sicurezza venivano colpite a causa della non buona generazione di chiavi.

Memorizzazione sicura di chiavi: grazie ad una gerarchia di chiavi è possibile cifrare le chiavi che verranno utilizzate per le varie operazioni crittografiche, prima che vengano memorizzate al di fuori del TPM.

Cifratura di dati: è possibile usare il TPM per le operazioni di cifratura simmetrica e asimmetrica andando a cifrare file, cartelle ed anche l'intero disco.

Memoria NVRAM: l'utilizzo di una memoria non volatile permette di memorizzare vari oggetti, come i certificati.

Attestazione dello stato di un dispositivo: prima dell'utilizzo dei TPM, le organizzazioni erano solite stabilire lo stato di una macchina mediante software. I risultati che ne venivano fuori potevano indicare che la macchina era in uno stato sicuro, quando in realtà era stata compromessa. Con un meccanismo hardware come il TPM si ha invece la possibilità di memorizzare lo stato della macchina senza che nessuno possa modificarlo e tale risultato può essere utilizzato in un processo di attestazione.

Il *TPM 2.0* invece implementa ulteriori caratteristiche oltre a quelle del TPM 1.2:

Flessibilità nell'uso degli algoritmi: con il TPM 2.0 è possibile usare un vasto numero di algoritmi di cifratura. Questo significa che se anche qualche algoritmo dovesse essere considerato debole in futuro, non ci sarà bisogno di modificare le specifiche del TPM.

Meccanismi migliori di autorizzazione: rispetto al TPM 1.2 sono stati aggiunti nuovi meccanismi di *autorizzazione*. Per autorizzazione si intende una funzione che permette di controllore i privilegi di accesso di un utente alle risorse del sistema. Questi nuovi meccanismi di autorizzazione possono permettere la creazione di policy ottenute attraverso la combinazione logica dei differenti tipi di autorizzazione.

Caricamento rapido di chiavi: nella versione del TPM 1.2 veniva richiesto un tempo non trascurabile per il caricamento delle chiavi crittografiche. Con il TPM 2.0 le chiavi possono essere caricate più velocemente utilizzando cifratura simmetrica piuttosto che asimmetrica.

Gestione flessibile: si possono separare i differenti tipi di autorizzazione, permettendo una gestione più flessibile delle risorse del TPM.

Identificare le risorse in base al nome: vengono utilizzati dei nomi sicuri da assegnare alle risorse del TPM.

2.3.1 Integrity Measurement and Reporting

Il processo che porta alla definizione del "trust state" di un host, intendendo il livello di affidabilità garantito per lo specifico host, prende il nome di *Integrity Measurement*. Esso si basa sulla creazione di una "chain of trust", letteralmente una catena di fiducia, costruita mediante il calcolo di un digest per ogni determinato componente facente parte della piattaforma stessa. Il digest calcolato per un componente prende il nome di *misura* ed il risultato di tale processo sarà quindi un insieme di misure che definiscono lo stato della macchina. Ogni componente della catena misura il successivo mentre il primo componente che da origine alla catena non viene invece misurato da nessuno.

Alla base della catena di fiducia si trovano dei componenti che prendono il nome di *Root of Trust*. Queste sono i componenti di cui bisogna fidarsi a priori perché un loro malfunzionamento

non è rilevabile. Non è possibile determinare se una Root of Trust si stia comportando in maniera appropriata, di esse si può solo conoscere come sono state implementate. Questo grazie alla presenza dei certificati che possono assicurare che una Root of Trust è stata implementata in modo tale da renderla affidabile (ad esempio un certificato potrebbe identificare il costruttore e il livello stimato di garanzia di un TPM).

Esistono tre tipi di Root of Trust in una piattaforma fidata:

- **Root of Trust for Storage (RTS):** è un'entità di cui ci si fida per memorizzare le informazioni. Il TPM può agire come RTS in quanto al suo interno memorizza vari oggetti come la parte privata delle chiavi, difendendoli da accessi inappropriati. L memorizzazione sicura è resa possibile dalla presenza di una coppia di chiavi asimmetriche nota come *Storage Root Key* (SRK). Tale coppia di chiavi indica lo specifico proprietario del TPM, viene creata tramite un'operazione nota come *Take Ownership* e può essere cambiata tutte le volte che un nuovo utente diventa il proprietario del TPM. L'SRK risiede nella memoria non volatile del TPM e viene usata come punto di partenza per costruire una gerarchia di chiavi utili per la cifratura dei dati. Esistono due tipi di chiavi necessarie ad effettuare l'operazione di cifratura:

  Storage Key: chiave RSA utilizzata per cifrare altre chiavi;

  Binding Key: chiave RSA utilizzata per cifrare piccole quantità di dati e chiavi simmetriche, ma non chiavi RSA.

  Entrambe vengono cifrate secondo la gerarchia mostrata in figura 2.1 e memorizzate nel disco dell'host ospitante il TPM;

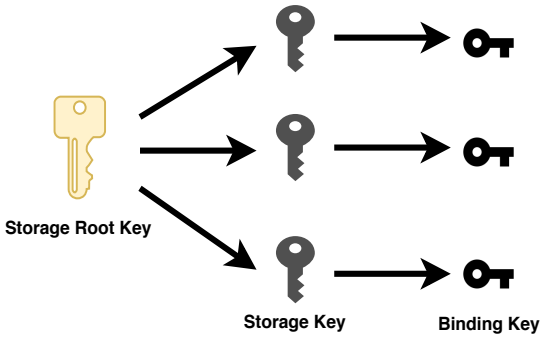

Figura 2.1. Gerarchia di cifratura delle chiavi

- **Root of Trust for Measurement (RTM):** è l'entità che si fa carico di registrare lo stato del sistema. Il RTM ha il compito di inviare le misure al RTS. Ha come obiettivo quello di misurare lo stato del sistema inserendolo in dei registri noti come *Platform Configuration Register* (PCR). La CPU è il componente che viene controllato dal *Core Root of Trust for Measurement* (CRTM). Il CRTM è il primo insieme di istruzioni che viene eseguito non appena viene stabilita una nuova catena di fiducia. Questa dovrebbe idealmente trovarsi all'interno del TPM, ma per ragioni legate all'architettura viene spesso collocata in altri dispositivi di difficile accesso per un attaccante remoto. Il CRTM ad esempio potrebbe trovarsi in un blocco di codice del BIOS accessibile in sola lettura. Il RTM una volta fatte le misurazioni le registra nel RTS;
- **Root of Trust for Reporting (RTR):** è l'entità responsabile di riportare all'esterno le informazioni riguardanti lo stato della macchina e coincide con il TPM stesso. In particolare il TPM riporta all'interno dei PCR i risultati che vengono fuori dal processo di misurazione. L'identità del RTR (quindi del TPM) è definita da una coppia di chiavi asimmetriche che prende il nome di *Endorsement Key* (EK). L'EK identifica univocamente la piattaforma e rimane attiva per tutta la vita del TPM, a differenza della SRK che cambia ogni qualvolta un nuovo utente entra in possesso del TPM.

L'obiettivo del processo di misurazione è quello di stabilire a "boot time", ovvero quando la macchina viene avviata, se quest'ultima è stata compromessa o meno, quindi definire, come è stato detto in precedenza, il suo trust state. Esistono due modi che consentono di effettuare il processo di misurazione:

Static Root of Trust of Measurement (SRTM): Prevede un processo di misurazione nel quale si parte dal CRTM, dal momento che è considerato un componente affidabile e immutabile. Successivamente vengono effettuati i seguenti passi (figura 2.2):

1. Il CRTM misura il successivo pezzo di codice che deve essere eseguito ovvero il BIOS e gli passa il controllo;
2. Il BIOS misura l'hardware e il loader del sistema operativo e cede il controllo a quest'ultimo;
3. Il loader del sistema operativo misura il kernel del sistema operativo e passa il controllo al sistema operativo stesso.

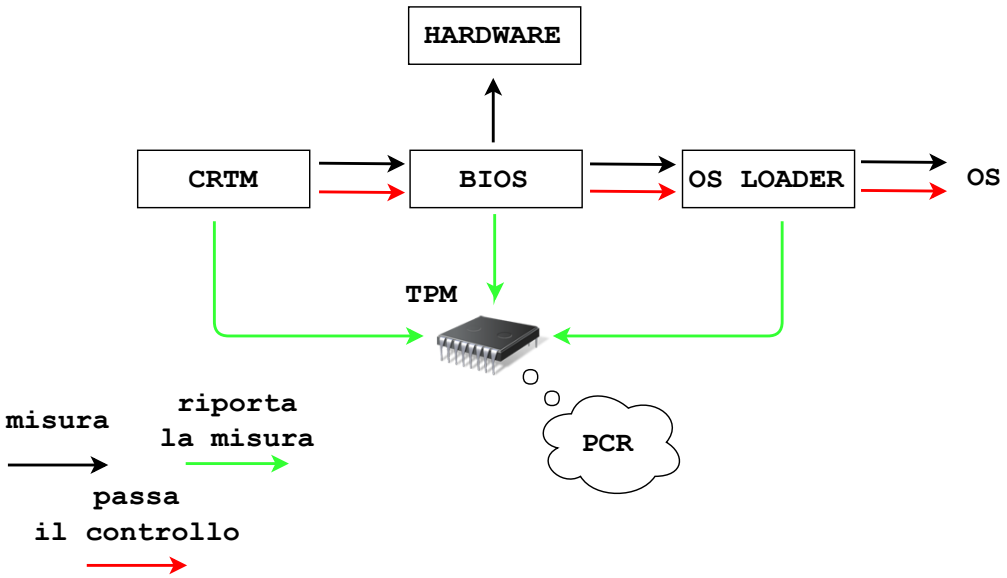

Figura 2.2. Processo di misurazione SRTM

Tutti i componenti che vengono misurati durante il SRTM fanno parte del *Trusted Computing Base* (TCB). Il TCB è un insieme di componenti hardware e software che provvedono alla sicurezza della piattaforma. Il SRTM richiede un riavvio della piattaforma per poter nuovamente essere eseguito. Al termine di ogni passo del SRTM viene effettuata l'operazione di *estensione* dei valori dei PCR e il valore finale di ognuno di questi indicherà il risultato finale del processo di misurazione della catena. Tali valori potranno essere utilizzati per essere confrontati con dei valori attesi per capire se la macchina viene eseguita o meno in un ambiente fidato, ma questo verrà discusso in seguito.

L'operazione di *Estensione* dei PCR consiste nell'applicare una funzione di hash tante volte quanti sono i componenti da misurare per quel determinato PCR. Si parte dal calcolo di un hash sul risultato ottenuto dalla concatenazione del valore iniziale del PCR con il digest del primo componente. Successivamente il valore del PCR dipenderà dal calcolo di un hash sul risultato della concatenazione del valore del PCR precedentemente ottenuto e il digest dell'attuale componente. In particolare

$$PCR_{\text{new}} := H_{\text{alg}}(PCR_{\text{old}}||D)$$

dove H indica l'algoritmo di hash utilizzato; || indicano che i valori vengono accodati; D rappresenta il digest calcolato sull'oggetto misurato. Quindi supponendo che:

- la piattaforma preveda di utilizzare il PCR0 per le misure del BIOS;
- il registro sia inizializzato con tutti zero;
- vengano misurate 10 componenti del BIOS;

si avrà che:

$$D_1 = H(Componente_1) \qquad PCR0_1 = H(0....0||D_1)$$
$$D_2 = H(Componente_2) \qquad PCR0_2 = H(PCR0_1||D_2)$$

fino ad avere il valore finale

$$D_{10} = H(Componente_{10}) \qquad PCR0_{10} = H(PCR0_9||D_{10})$$

Dynamic Root of Trust of Measurement (DRTM) [24] prevede che il processo di misurazione possa essere effettuato a "run time" senza quindi aver bisogno di riavviare la macchina, come nel caso del SRTM. Lo scopo del DRTM è quello di garantire che il TCB non venga compromesso. Quindi sarà necessario eseguire un ulteriore processo di misurazione che porti la piattaforma in uno stato noto come *Dynamically Launched Measured Environment* (DLME). Nel DLME si ha la certezza che il codice sia stato misurato e si possa verificare, tramite i valori riportati nei PCR, che il TCB sia stato stabilito correttamente. Per convenzione lo stato di hardware e software non definito della piattaforma prima che abbia inizio il processo di DRTM prende il nome di *Gap*. Le fasi in cui si articola il DRTM sono le seguenti (figura 2.3):

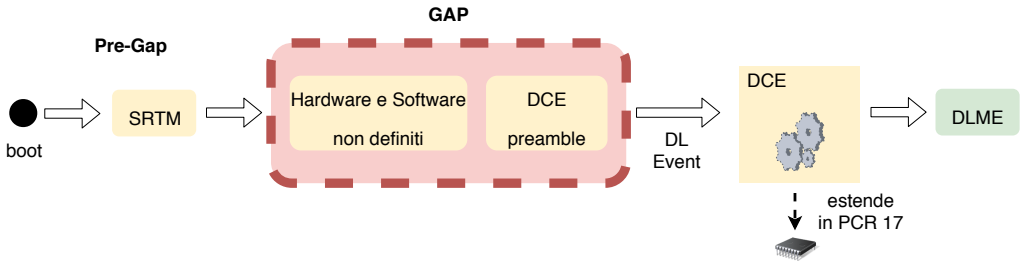

Figura 2.3. Processo di misurazione DRTM

1. La piattaforma viene accesa e inizia il processo di SRTM;
2. Una volta terminato, l'SRTM lancia il codice che porterà la piattaforma in uno stato di Gap;
3. Viene invocato il *DRTM Configuration Environment preamble* (DCE preamble), ovvero un codice fornito dal costruttore della piattaforma, il cui compito è quello di configurare l'ambiente prima di far partire il processo di DRTM;
4. Il DCE preamble esegue il *Dynamic Launch Event* (DL Event), ovvero il comando che porterà ad invocare un software noto come *DRTM Configuration Environment* (DCE) il cui compito è quello di effettuare la misurazione di hardware e software che si trovano in uno stato indefinito.
5. Il DCE resetta i PCR dal 17 al 22, attribuendoli un valore pari a tutti zero. Successivamente effettuerà il processo di misurazione di tutti gli elementi che possono compromettere il TCB della piattaforma. Tra questi:

   DMA remapping: le specifiche del DRTM richiedono che il sistema abbia un metodo per proteggere la memoria dagli accessi *Direct Memory Access* (DMA) come il *DMA remapping*. Questo implica che i periferici non possono in alcun modo compromettere il TCB. Tuttavia deve essere necessario misurare le unità di DMA remapping per capire se si stanno comportando correttamente.

System Management Mode (SMM): il SMM è un insieme di funzioni necessarie a garantire l'affidabilità e la sicurezza del sistema, come ad esempio proteggere il sistema da danni termici. Tali funzioni hanno molti privilegi di accesso, tra cui quello di essere in grado di poter accedere alla memoria e quindi compromettere il TCB.

Non-Host Platforms (NHP): sono un insieme di *controller* che eseguono operazioni critiche all'interno della piattaforma come la gestione dell'energia e la configurazione della piattaforma stessa. In particolare esistono dei controller noti come *updatable NHP* che essendo riprogrammabili possono nuocere al TCB.

Advanced Configuration and Power Interface tables (ACPI tables): per consentire al TCB di proteggere se stesso una volta terminato il processo DRTM, il TCB deve ottenere delle informazioni riguardanti l'hardware che potrebbe comprometterlo. Queste informazioni si trovano all'interno di tabelle note come ACPI tables che dovranno essere validate attraverso il processo del DRTM.

6. Una volta completato il processo di misurazione si raggiunge il DLME.

Le misure calcolate dal DCE vengono estese all'interno del PCR 17. Intel permette di realizzare il processo di DRTM attraverso un'architettura nota come *Intel Trusted Execution Technology* (Intel TXT).

2.3.2 Componenti del TPM

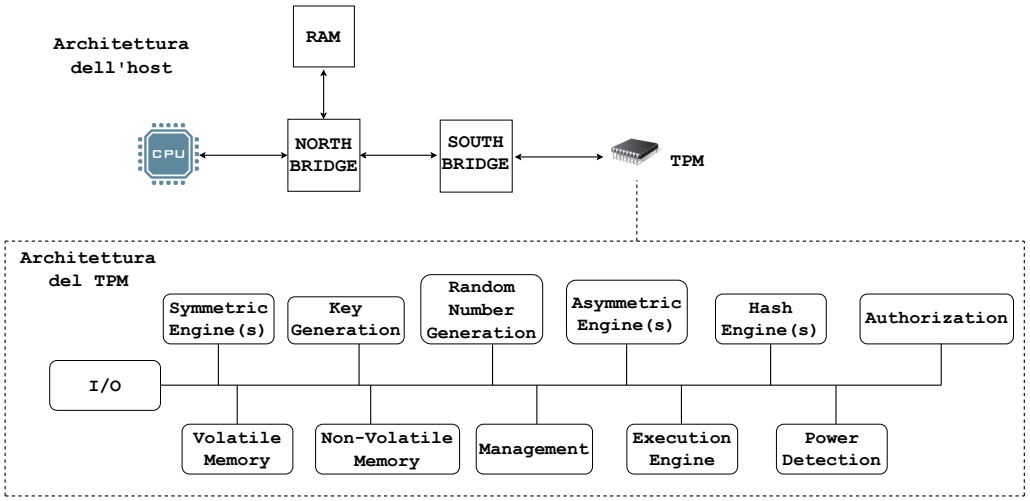

Figura 2.4. Architettura dell'host e architettura di un TPM

In figura 2.4 viene rappresentata l'architettura di un TPM con i suoi principali componenti. Il TPM [8] permette di mettere in comunicazione i propri componenti tramite l'*I/O buffer*. Le operazioni crittografiche effettuate dal TPM sono rese possibili dalla presenza di vari moduli come il *Symmetric Engine* per la cifratura simmetrica; il *Key Generation* per la generazione di chiavi che avviene sfruttando la creazione di valori casuali tramite il *Random Number Genration*; l'*Asymmetric Engine* per la cifratura asimmetrica caratterizzata da algoritmi come RSA [25] e ECC [26] (solo per la versione 2.0); l'*Hash Engine* usato per le operazioni che prevedono l'utilizzo di algoritmi di hash come SHA1[23] e SHA256 [27] (solo per la versione 2.0).

Il TPM presenta una propria memoria non volatile, in cui memorizza il proprio stato e le chiavi EK e SRK. Presenta inoltre una memoria volatile, caratterizzata dalla presenza di locazioni di memoria protette note come *Shielded Locations* (SL). I PCR sono contenuti nella memoria volatile del TPM all'interno di SL e sono utilizzati per la validazione delle misure. Essi vengono inizializzati a tutti zero o a tutti uno; dipende dalle specifiche del TPM della piattaforma. A seconda della versione del TPM i PCR possono avere una determinata dimensione: nella versione 1.2 la loro

dimensione è di 160 bit ognuno, mentre nella 2.0 è di 256 bit ognuno. In totale si hanno 24 registri e, a seconda delle specifiche della piattaforma, possono registrare misure differenti. Ad esempio, un PCR può essere usato per memorizzare le misure del BIOS, un altro per il loader del Sistema Operativo (SO) ed altri per il supporto alle applicazioni.

2.3.3 Le credenziali

Il TCG definisce il concetto di *credenziale* [28] come una prova astratta che deve essere creata prima che possa avvenire uno scambio tra due entità. Un certificato digitale può essere considerato una prova, dove per certificato digitale si intende una struttura dati che segue le specifiche ISO/IEC/ITU-T X.509 version 3 [29].

Il TCG definisce i seguenti tipi di credenziali [28] [30]:

Endorsement Key (EK) Credential. È un certificato X.509 v3 che contiene la chiave pubblica della coppia di chiavi EK di cui si era parlato in precedenza, memorizzato all'interno di una Shielded Location. Mentre la parte pubblica di una EK può essere letta dal TPM, quella privata non deve mai essere esposta. Tale coppia di chiavi viene generata dal costruttore prima che l'utente entri in possesso della piattaforma, oppure può essere generata dal TPM usando il comando TPM2_CreatePrimary. La coppia EK è costituita da chiavi RSA a 2048 bit nel caso della versione 1.2 del TPM, mentre per la versione 2.0 è costituita da chiavi RSA a 2048 bit o ECC NISTP a 256 bit.

Nel TPM 1.2 la EK poteva essere usata solo per decifrare e difficilmente per firmare. Con il TPM 2.0 invece si ha maggiore flessibilità, avendo completa libertà nella definizione degli attributi di una EK che potrà quindi essere usata sia per decifrare ma anche per firmare.

L'uso principale di un EK Credential è assistere le *Certificate Authority* (CA), ovvero le entità che emettono i certificati digitali, nel processo di attestazione per l'emissione delle *Attestation Key* (AK), chiavi utilizzate per le operazioni di firma. Inoltre un EK Credential può fornire una prova che le Attestation Key risiedono sullo stesso TPM. Inoltre un EK Credential asserisce che il possessore della chiave privata della coppia EK è un TPM conforme alle specifiche del TCG. Un EK Credential deve contenere la chiave pubblica della coppia di chiavi EK, la versione specifica del TPM, il costruttore del TPM e la versione del firmware del TPM. Queste informazioni sono utili per una CA per poter attestare se il TPM è conforme alla specifica versione del TPM di cui è richiesta l'attestazione. In base a questo la CA può decidere se il TPM dovrebbe o meno ricevere una Attestation Key Credential.

Platform Credential. Serve per attestare che una specifica piattaforma contiene un unico TPM e un *Trusted Building Block* (TBB). Un TBB è un componente o una collezione di componenti che vengono richiesti per creare una Root of Trust e che non si trovano all'interno di Shielded Location. Normalmente include solo la CRTM e le funzioni di inizializzazione del TPM. In generale la Platform Credential viene emessa dal costruttore della piattaforma.

Attestation Identity Key Credential. Una *Attestation Identity Key* (AIK) non è altro che una coppia di chiavi RSA a 2048 bit, generata dal TPM, che fornisce un alias per la EK con l'obiettivo di identificare il TPM ma allo stesso tempo non esporre l'EK stessa. Il TPM può generare un numero arbitrario di AIK che vengono usate in un protocollo Requestor-Verifier-Relying party e servono per attestare le informazioni originate dal TPM stesso. Infatti un AIK viene usata solo per firmare dati (e.g. i valori contenuti nei PCR) ed essere sicuri che quegli stessi dati provengano da quel TPM, grazie al fatto che tale chiave porta con se l'identità dello stesso. Nella gerarchia di chiavi di figura 2.1 un AIK si trova al livello delle storage key.

Un AIK Credential è un certificato X.509 v3 che contiene la chiave pubblica della coppia AIK ed originato attraverso un servizio noto come *Privacy Certification Authority (PCA)*. La prova fornita da un AIK Credential consiste nel dimostrare che quella AIK appartiene ad una EK Credential valida e ad una Platform Credential valida. In questo modo da una parte la PCA si assicura di interagire con un TPM valido, mentre dall'altra il TPM ha la possibilità di ottenere un certificato per ognuna delle AIK che ha creato.

Il protocollo utilizzato per ottenere tale credenziale è il seguente [31] (figura 2.5) :

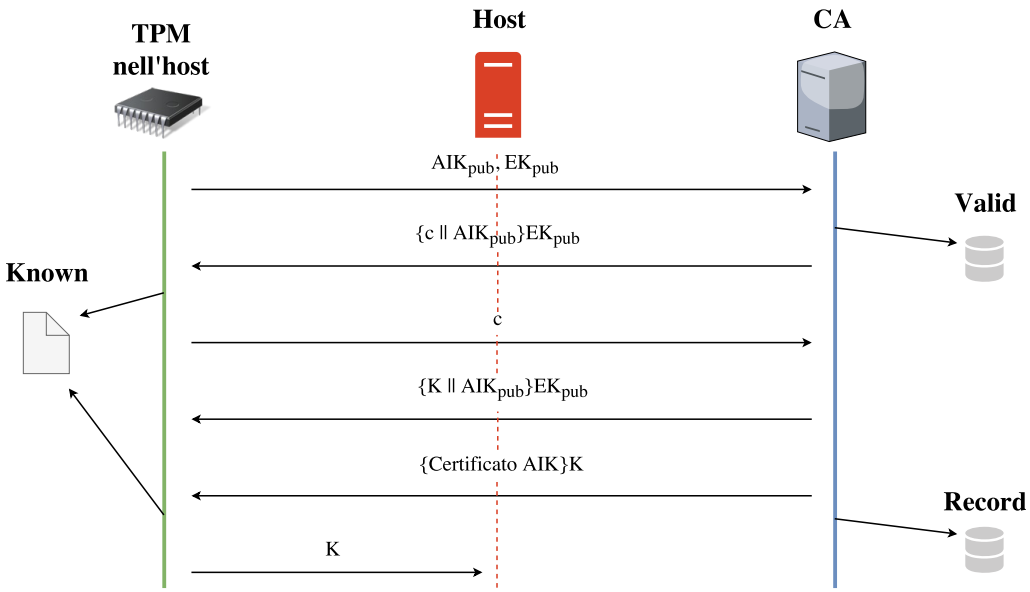

Figura 2.5. Servizio di PCA

1. L'host inizia il protocollo inviando al suo TPM un pacchetto dati in cui chiede di caricare la coppia di chiavi AIK (quindi chiave privata e chiave pubblica) che si vuole certificare. Nel frattempo l'host invia la parte pubblica dell'AIK e la parte pubblica dell'EK del TPM alla CA.
2. La CA verifica se nella lista *Valid*, ovvero una lista di coppie chiave pubblica della CA e chiave pubblica dell'EK del TPM, sia presente la chiave pubblica dell'EK che le è stata appena mandata. Se questo è vero, allora la CA sceglie un nonce c e calcola il ciphertext a:

$$a = \{c||AIK_{\text{pub}}\}EK_{\text{pub}}$$

dove

- {} indicano che il contenuto nelle parentesi è stato cifrato utilizzando una chiave. In tal caso la chiava utilizzata per cifrare è la chiave EK_{pub};
- || indica l'operazione di concatenazione;
- AIK_{pub} è la chiave pubblica della coppia AIK del TPM che è stata appena mandata alla CA.

Il ciphertext a, così ottenuto, sarà inviato all'host che lo inoltrerà al suo TPM.

3. Il TPM decifra il ciphertext a usando la chiave privata della coppia EK, preleva AIK_{pub} e controlla se tale chiave è presente nella lista *Known*, ovvero la lista delle coppie AIK create dal TPM. Inoltre controlla anche che la coppia AIK di cui fa parte la chiave pubblica appena prelevata dal ciphertext sia stata caricata. Se il controllo va a buon fine, il TPM rilascia c all'host che lo inoltra alla CA.
4. La CA verifica se il nonce c che ha ricevuto è quello che ha usato per la sfida. Se è così, allora la CA crea una chiave simmetrica K, un certificato su AIK_{pub}, *cer*, e calcola il ciphertext d:

$$d = \{K||AIK_{\text{pub}}\}EK_{\text{pub}}$$

Nel frattempo la CA calcola anche il ciphertext b:

$$b = \{cer\}K$$

Quindi invia i ciphertext d e b all'host. Infine la CA memorizza la tripla costituita da EK_{pub}, AIK_{pub}, *cer* in una lista chiamata *Record*.

5. Dopo aver ricevuto il ciphertext d dall'host, il TPM fa le stesse azioni del punto 3, quindi verifica la presenza di AIK_{pub} nella lista *Known* e che la coppia AIK sia stata caricata. Se il controllo va a buon fine, il TPM rilascia la chiave K all'host.
6. L'host decifra il ciphertext b usando la chiave K e ottiene il certificato *cer* dell'AIK_{pub}, ovvero l'AIK Credential, per la quale era stato richiesto inizialmente un nuovo certificato.

2.4 TCG Software Stack

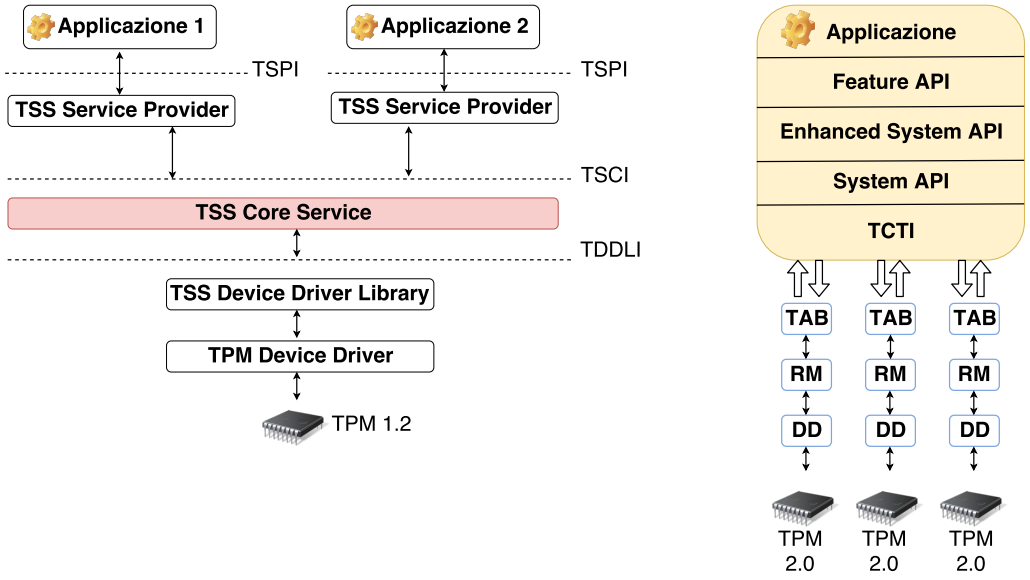

Figura 2.6. stack TSS a confronto

Il Trusted Computing Group Software Stack (TSS) è un insieme di componenti software (figura 2.6) che consente di creare un'astrazione di alto livello delle funzioni offerte dal TPM, permettendo un'interazione più semplice. In questo modo sarà possibile effettuare la comunicazione con il TPM eseguendo dei comandi che permettano alle applicazioni di ricavare informazioni utili sullo stato della macchina a seguito del processo di misurazione.

Nel mio lavoro di tesi l'interazione tra le applicazioni e il TPM ha permesso di:

- generare una coppia di chiavi AIK per effettuare la comunicazione con il Verifier;
- recuperare i valori contenuti nei PCR;
- calcolare una *quote*, ovvero un report firmato utilizzando la chiave AIK del TPM.

2.5 Il processo di attestazione remota

Il concetto di *Attestazione* è uno dei concetti fondamentali che, secondo il TCG [37], deve caratterizzare qualsiasi piattaforma si definisca *Trusted*, cioè una piattaforma che si trova in uno stato

affidabile. Infatti il processo di attestazione viene indicato [38] anche come un tassello fondamentale nel ciclo di vita di una piattaforma *Trusted*, grazie alla possibilità che offre di trasferire lo stato di integrità della piattaforma (il *Requestor*) verso un'altra macchina *il Verifier*. Nel caso in cui la macchina da attestare e quella che effettua l'attestazione richiedano l'utilizzo di un protocollo di rete per comunicare si parla di *Attestazione Remota*.

La figura 1.1 mostra come nel processo di attestazione qualsiasi transazione tra il Requestor e il Relying Party debba coinvolgere il Verifier. il Relying Party dovrà quindi fare affidamento sul risultato dell'attestazione che viene fornito dal Verifier. Tale risultato fornirà una valutazione sullo stato del Requestor, permettendo al Relying Party di prendere una decisione finale sull'azione da intraprendere a seguito del risultato stesso. Dal momento che le tre entità potrebbero trovarsi in tre *domini* differenti (e.g. domini amministrativi, domini giuridici, domini di sicurezza), è necessario che usino la stessa semantica e preferibilmente anche la stessa sintassi per comunicare tra loro, in modo che tutti siano in grado di interpretare allo stesso modo le informazioni e i risultati ricevuti.

2.5.1 Principi di attestazione remota

Gli autori dell'articolo "*Principles of remote attestation*" [39] individuano cinque principi fondamentali che dovrebbero essere garantiti nello sviluppo di un sistema di attestazione.

Fresh Information. Le asserzioni riguardanti la macchina da attestare dovrebbero riflettere lo stato del sistema durante la sua esecuzione e non solo in fase di *start-up*, cioè quando la macchina viene avviata.

In questo lavoro di tesi, come già accennato nell'introduzione, si ritiene molto importante la soddisfazione di tale principio poiché la "freschezza" delle informazioni provenienti dall'host sono un punto fondamentale per poter capire in maniera immediata ciò che sull'host sta accadendo e procedere con le eventuali contromisure.

Comprehensive Information. I meccanismi di attestazione dovrebbero essere capaci di fornire informazioni esaustive riguardanti la macchina da attestare, il cui stato interno dovrebbe essere accessibile agli strumenti locali di misurazione.

Constrained Disclosure. La macchina da attestare dovrebbe essere capace di applicare delle policy che indichino quali misure devono essere inviate ad ognuno dei Verifier con lo scopo di limitare la divulgazione delle informazioni riguardanti la macchina stessa. Pertanto un'architettura di attestazione deve permettere che il Verifier sia identificato dalla macchina da attestare, consentendo a quest'ultima di poter attestare lo stato di un Verifier prima che invii a questo il suo di stato.

Semantic Explicitness. Il contenuto semantico delle attestazioni dovrebbe essere espresso in maniera esplicita, permettendo al Verifier di trarre delle conclusioni in seguito alle attestazioni effettuate sulla macchina da attestare.

Trustworthy Mechanism. Lo stesso meccanismo di attestazione dovrebbe essere attendibile e l'architettura di attestazione dovrebbe essere identificata sia dalla macchina da attestare sia dal Verifier.

I principi di attestazione dovrebbero essere sempre soddisfatti, ma spesso i sistemi di attestazione non riescono a farlo ed introducono delle variazioni.

2.5.2 Il flusso di attestazione

Il flusso di attestazione prevede un insieme di passi che devono essere effettuati affinché il Verifier possa essere in grado di attestare lo stato del Requestor quando deve richiedere una risorsa o un servizio ad un Relying Party che vuole sapere se può o meno fidarsi di quel Requestor (figura 2.7).

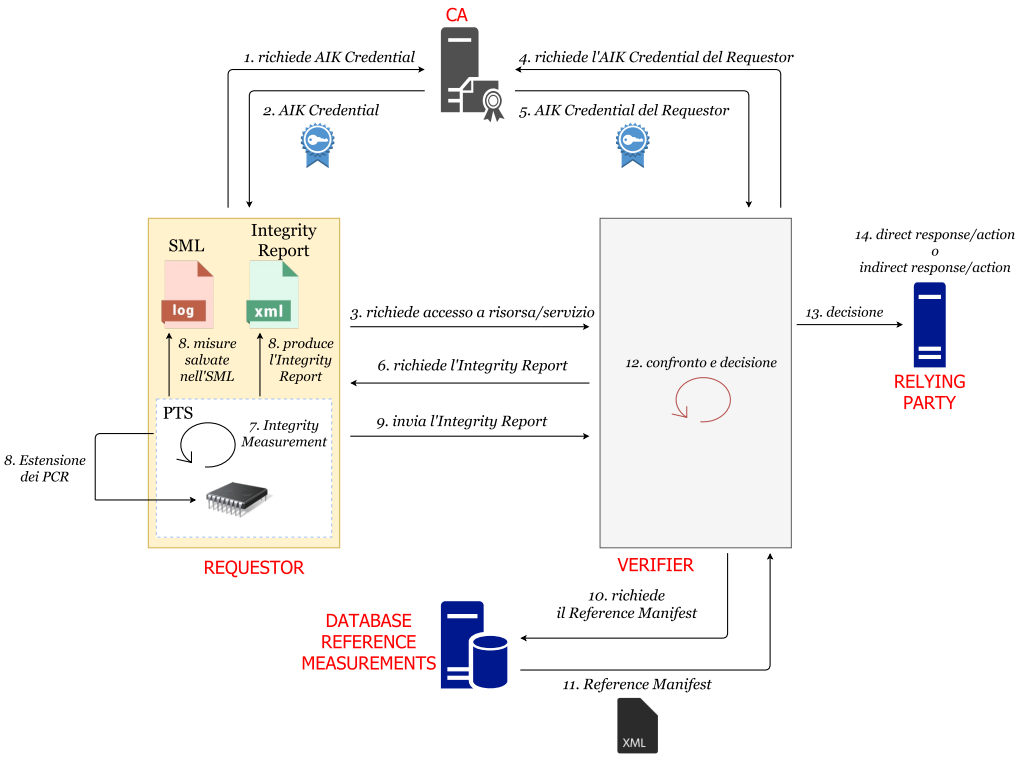

Figura 2.7. flusso di attestazione

Identity Credential Enrollement: è il primo passo fondamentale che serve a permettere ad un Requestor di comunicare con il mondo esterno e consiste nel richiedere ad una CA un AIK Credential, cioè un certificato per la chiave pubblica della coppia AIK. Questo avviene secondo il processo discusso precedentemente.

Identity Credential Publish: il Verifier prima di poter attestare il Requestor ha bisogno di conoscere la sua identità. Per tale motivo avrà bisogno di comunicare con la CA che ha emesso l'AIK Credential per quel Requestor, in modo che possa ottenere una copia della stessa da utilizzare per identificare quello specifico Requestor.

Platform Integrity Measurement: È il processo che permette di ottenere le misure di integrità della piattaforma che vengono memorizzate in log che prendono il nome di *Stored Measurement Logs* (SML) che si trovano al di fuori del TPM ma all'interno dello stack TSS. I digest di queste misure invece vengono inseriti all'interno dei PCR secondo l'operazione di estensione vista in precedenza. Tali misure sono molto importanti per il Verifier perché gli permettono di stabilire se il Requestor è in uno stato ritenuto affidabile o meno. Il Relying Party invece non ha bisogno di avere le misure in dettaglio, ma gli interessa solo ottenere un risultato complessivo delle stesse.

Platform Integrity Reporting: È la fase che permette di fare un report delle misure che sono state calcolate per poterle trasferire al Verifier. È importante tener conto di due aspetti quali: forma e struttura del log che contiene le misure e che viene trasmesso al Verifier, e protocollo di trasporto utilizzato per il trasferimento del log. La struttura che contiene l'insieme delle misure prodotte prende il nome di *Integrity Report* [40].

Un Integrity Report definito dal TCG è un file XML ed è costituito da un insieme di elementi che prendono il nome di *Snapshot*. Esiste uno Snapshot per ogni componente misurato, costituiti da:

- *integrity assertions* ovvero un insieme di affermazioni che permettono di identificare il componente e riflettere la sua qualità;

- *integrity values* ovvero un insieme di digest che costituiscono le misurazioni riguardanti quel componente.

Il processo che ha luogo all'interno del Requestor e che ha il compito di effettuare le misurazioni prende il nome di *Platform Trust Services* (PTS). Esso fornisce tutti i servizi per la misura dei componenti, per la creazione, la sincronizzazione degli Snapshot e la generazione dell'Integrity Report. Il PTS interagisce anche con il TPM quando disponibile sulla piattaforma.

Evaluation Reporting: il Verifier deve comunicare il risultato dell'attestazione al Relying Party. Tale risultato deve essere significativo; il Verifier e il Relying Party infatti devono aver stabilito a priori una metrica di valutazione che permetta al Verifier stesso di ottenere un risultato che si basi sulla metrica prestabilita. In tal modo il Relying Party sarà quindi in grado di interpretarla e di conseguenza prendere delle decisioni. Il tipo di metrica dipende dallo specifico caso d'uso.

Direct Response/Action: Consequenzialmente alla ricezione del risultato da parte del Verifier, il Relying Party può generare una risposta e/o eseguire un'azione che può interessare solo il dominio del Relying Party o quello del Requestor.

Indirect Response/Action: È una modalità differente dalla precedente ed è il caso in cui il Requestor ha richiesto un servizio al Relying Party e la comunicazione viene mediata dal Verifier. Quindi le risposte del Relying Party passano attraverso il Verifier che funge da *Policy Decision Point* (PDP).

2.5.3 La fase di verifica

La fase di verifica delle misure prodotte dal Requestor è importante poiché da essa dipende l'esito dell'attestazione.

I componenti che caratterizzano la piattaforma del Requestor possono non essere conosciuti dal Verifier, dal momento che non si può assumere che Verifier e Requestor siano nello stesso dominio. Pertanto il Verifier deve cercare delle *authoritative information* riguardanti ognuno dei componenti che caratterizzano la piattaforma. Queste authoritative information prendono il nome di *Reference Measurements* e vengono rese disponibili da un fornitore quale il costruttore dell'hardware o il produttore del software, oppure un fornitore fidato per entrambi. Le Reference Measurements sono contenute all'interno di una struttura dati definita dal TCG sotto forma di file XML che prende il nome di *Reference Manifest* [41]. Inoltre sono delle misure di riferimento statiche che definiscono per ogni componente misurato quale dovrà essere il valore della misurazione prodotto per quel particolare componente. Il Reference Manifest si trova nel *Reference Manifest Database* che può essere collocato all'interno del Verifier stesso oppure all'esterno, nello stesso dominio o in un dominio differente rispetto a quello del Verifier stesso.

Quando un Requestor deve essere valutato da un Verifier, deve fornirgli l'Integrity Report. Il Verifier una volta che è riuscito ad identificare ciascun componente della piattaforma del Requestor, potrà così procedere a comparare le misure riportate nell'Integrity Report con quelle presenti nel Reference Manifest. Finita questa operazione il Verifier potrà essere in grado di valutare il livello di attendibilità della piattaforma del Requestor e fornire quindi la decisione al Relying Party.

2.6 Criticità del Trusted Computing

Come si è potuto evincere, il Trusted Computing offre un'insieme di opportunità che consentono di garantire una sicurezza basata sull'hardware e quindi molto più forte degli approcci che si affidano al solo software. Tuttavia ci sono stati alcuni punti del TC che hanno generato scetticismo e controversie [42].

Il problema principale che si ha nel processo di attestazione definito dal TC è che coinvolge una terza parte. Quest'ultima può ottenere informazioni sensibili riguardanti il dispositivo dell'utente

e influenzarne la privacy collegando le richieste provenienti da uno stesso utente dal momento che esiste un'unica Endorsement Key per ognuno di essi.

Un altro problema sono i guasti dell'hardware a disposizione. Spesso tutti i dispositivi, compresi quelli forniti dal TC stesso, possono guastarsi, essere aggiornati o rimpiazzati. Questo può essere un problema per l'utente perché c'è il rischio che venga tagliato fuori dall'accesso alle proprie informazioni. Ad esempio il TPM ha un'unica chiave che lo identifica e tutte le chiavi più importanti vengono memorizzate e usate all'interno di esso. Quindi un suo malfunzionamento potrebbe provocare problemi all'utente che lo utilizza, che quindi non potrebbe essere in grado di provare la sua identità.

Il TCG non definisce delle procedure che permettano di effettuare dei test. Definisce solamente delle specifiche, senza fornire ai venditori un modo per testare se si è o meno conformi alle specifiche stesse. Quindi per un utente finale è difficile sapere se la sua piattaforma rispetta o meno le intere specifiche o un sottoinsieme di esse perché non c'è alcun prototipo che sia in grado di testare il TPM. Inoltre molto spesso accade che ci siano problemi di interoperabilità tra i vari software stack dei venditori e il trusted stack proposto dal TCG.

Il TC mette a rischio l'esistenza di applicazioni e sistemi operativi free perché vengono bloccati dal TPM. Infatti il TC richiederebbe che ogni programma sia autorizzato dallo sviluppatore del sistema operativo prima che possa essere installato. Inoltre il TC può portare ad un malfunzionamento di software o applicazioni. Questo perché l'idea sottolineata dal TC è che il computer includa un dispositivo di cifratura digitale e firma, e che le chiavi vengano mantenute segrete dai costruttori. Il TPM verrà usato da programmi proprietari per verificare quali programmi possono essere eseguiti, a quali documenti o dati si può accedere e a quali applicazioni questi possono essere affidati. Questi programmi continueranno a scaricare nuove regole di autorizzazione attraverso Internet che se bloccate dal TPM, potrebbero comportare dei malfunzionamenti alle applicazioni a cui erano destinate.

2.7 Integrity Measurement Architecture

IMA è un software integrato nel kernel Linux e sviluppato da IBM che, avvalendosi dell'uso del TPM, ha come obiettivo quello di verificare l'*integrità* di un sistema sulla quale sono eseguiti dei programmi. L'integrità di un programma [43] è una proprietà binaria che indica se il programma e/o il suo ambiente d'esecuzione hanno subito delle modifiche non autorizzate. Le informazioni di integrità del sistema si possono ricavare grazie alla raccolta di un insieme di digest, chiamati *misure*, che possono essere verificati confrontandoli rispetto ad un altro insieme di misure dette *known-good*, ovvero misure delle quali ci si fida perché calcolate in un ambiente fidato.

Il livello di verifica dell'integrità di una piattaforma è quello definito da Clark e Wilson [44] e si basa su un insieme di punti che devono essere raggiunti. A meno che il flusso di informazioni tra i vari processi non abbia vincoli particolari, è necessario misurare l'integrità di tutti i processi in esecuzione sulla macchina. Per ogni processo si deve essere in grado di ottenere le informazioni di integrità, indipendentemente dal fatto che questo sia caricato dal sistema operativo, da un loader dinamico come avviene per le librerie, o dall'applicazione stessa. I dati con una semantica di integrità simile a quella degli eseguibili devono essere trattati allo stesso modo degli eseguibili stessi. Quindi bisogna essere sicuri di essere in grado di catturare tali dati indipendentemente dal fatto che provengano dal sistema operativo, dai loader dinamici o dall'applicazione stessa. Il calcolo dell'integrità dei dati dinamici che non hanno una semantica d'integrità identificabile, dipende dall'integrità del processo che ha modificato quei dati e può essere verificata solo tramite policy di sicurezza o grazie alla storia stessa dei dati.

Vengono inoltre definite da IMA delle assunzioni molto importanti, senza le quali un attaccante potrebbe essere sempre in grado di ingannare un client remoto. Utilizzare servizi e protezioni dichiarate negli standard offerti dal TCG potrebbe essere utile per stabilire una sorta di fiducia nell'ambiente di misurazione e assicurarsi che le misure provenienti da tale ambiente non siano state modificate. Anche il TPM deve funzionare secondo le specifiche definite dallo standard in modo da essere sicuri di misurare correttamente il BIOS e le altre parti coinvolte nel processo

di misurazione. Un'assunzione importante è quella che il TPM non sia in grado di prevenire gli attacchi diretti all'hardware del sistema, in modo da escluderli nella trattazione del modello. Inoltre la misurazione del codice si può considerare sufficiente a determinare il comportamento dell'applicazione. È fondamentale che il *Requestor* possegga un certificato valido associato alla chiave pubblica AIK_{pub} del TPM e usato dal *Verifier* per validare il contenuto della risposta. Il *Verifier* può decidere come verificare le misure ricevute comparandole con una lista di valori fidati o affidandole ad una terza parte che deciderà in base a delle policy definite. Controllando l'accesso al servizio di attestazione si può considerare già soddisfatto il requisito di confidenzialità sui dati riguardanti la misurazione.

2.7.1 Il Modello

Il modello su cui si basa IMA è costituito da tre componenti principali:

Measurement Mechanism determina cosa deve essere misurato dell'ambiente di esecuzione, quando misurare e come conservare in modo sicuro le misure acquisite;

Integrity Challenge Mechanism permette ai vari *challenger*, ovvero coloro che desiderano ricevere le misure di un ambiente di esecuzione, di recuperare la lista di misure e verificare la loro completezza e freschezza, cioè capire se sono misure ottenute recentemente o meno;

Integrity Validation Mechanism verifica che la lista di misure sia completa, non sia stata manipolata e che ciascuna misura rispecchi il codice ritenuto affidabile.

Measurement Mechanism

L'idea alla base del meccanismo di misurazione è quella secondo cui il BIOS e il bootloader inizialmente misurano il codice del kernel, consentendo in seguito a quest'ultimo di misurare i propri cambiamenti ed ogni processo *user-level*. Questi a loro volta possono procedere alla misurazione di altri componenti. Ogni componente che da inizio ad una misurazione prende il nome di *Measurement Agent*.

Misurare significa calcolare un digest di tipo SHA1 sul file, in modo che i 160 bit risultanti possono identificare univocamente il contenuto del file stesso. I digest prodotti per ogni file verranno quindi raccolti all'interno di una *measurement list*, ovvero una lista che rappresenta lo stato di integrità del sistema che si vuole attestare.

È molto importante sottolineare che IMA non è in grado di prevenire la corruzione o la manipolazione del sistema o della lista di misure, tuttavia può essere in grado di rilevare qualunque comportamento che minaccia l'integrità del sistema, grazie all'utilizzo del TPM. Il TPM, infatti, utilizzando i suoi PCR fornisce una protezione sicura da modifiche. I PCR possono essere modificati solo da due funzioni: la prima che viene eseguita al riavvio del sistema e ha il compito di resettare tutti i PCR, assegnando ad essi un valore pari a tutti zero; la seconda invece è la funzione *TPM_execute* che prende un numero n da 160 bit (frutto della funzione SHA1 sul file misurato) e lo aggrega al contenuto dell'*i-esimo* PCR destinato a contenere il valore calcolato per quel file (esattamente come è stato descritto in precedenza nel processo di estensione). Se si assume che il TPM stia rispettando le specifiche del TCG e che non siano in corso degli attacchi fisici allora non ci sono altri modi per un sistema di cambiare i valori dei PCR. L'aggregato della lista di misure si può inserire all'interno di un PCR sia perché ogni misura viene aggregata nel PCR selezionato prima che il componente misurato possa influenzare e corrompere il sistema, sia perché il TPM costituisce un protezione sicura dagli attacchi. In questo modo un attaccante potrà sempre modificare la lista di misure ma non in modo tale che il nuovo aggregato vada a rispecchiare l'aggregato precedentemente salvato all'interno del PCR.

Integrity Challenge Mechanism

L'*Integrity Challenge Mechanism* si basa su un protocollo noto come *Integrity Challenge Protocol*. Questo protocollo descrive come un *Verifier* possa ottenere in modo sicuro la lista delle misure dal *Requestor*. Si assume che tale meccanismo venga attuato su una connessione sicura (usando ad esempio SSL) in modo da garantire autenticazione e confidenzialità.

Il protocollo è così descritto (figura 2.8):

1. Il Verifier crea un numero non predicibile e casuale di 160 bit, che prende il nome di *nonce*;
2. Il Verifier invia in un messaggio di richiesta *ChReq* sotto forma di sfida al Requestor;

3a. Il Requestor carica nel TPM la chiave *AIK*. Tale chiave viene cifrata dalla SRK, che come è stato detto in precedenza, è conosciuta solo dal TPM;

3b. Il Requestor richiede al TPM il calcolo di una *Quote*, ovvero un report firmato usando la chiave privata della coppia AIK e contenente i valori dei PCR e il *nonce* fornito dal Verifier, ovvero

$$Quote = \{PCR, nonce\}AIK_{\text{priv}}$$

In particolare la *Quote* contiene il valore dell'aggregato ricavato dalla lista di misure IMA e memorizzato all'interno di uno specifico PCR (di default il PCR 10);

3c. Il Requestor ricava la lista di misure dal kernel;

4. Il Requestor invia la risposta alla sfida *ChResp* contenete la *Quote* e la lista di misure;

5a. Il Verifier recupera il certificato associato alla chiave pubblica della coppia AIK;

5b. Il Verifier usa il certificato per validare la firma della risposta proveniente dal Requestor;

5c. Il Verifier valuta la freschezza della *Quote* grazie al *nonce* recuperato dalla riposta. Inoltre utilizza la lista di misure per ricalcolare l'aggregato e confrontarlo con il valore contenuto nel PCR di riferimento, cioè nel PCR nella quale il TPM aveva memorizzato il valore dell'aggregato.

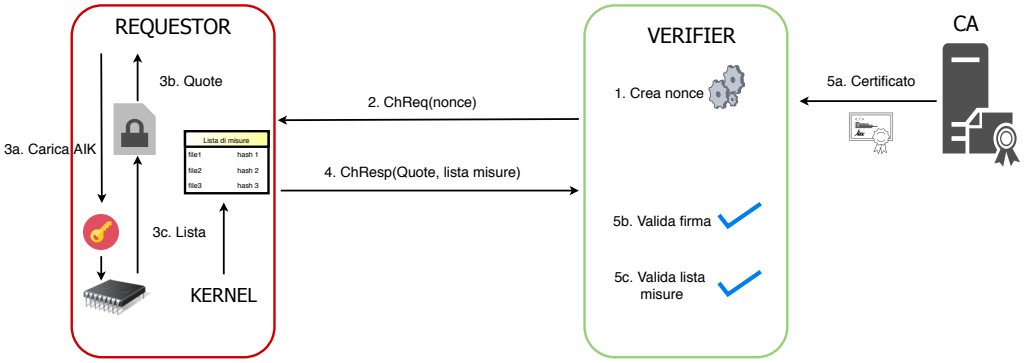

Figura 2.8. Integrity Challenge Protocol

Dal punto di vista della sicurezza questo protocollo dovrebbe essere in grado di proteggere le informazioni di attestazione dalle maggiori minacce come:

- **Replay Attacks:** un Requestor corrotto potrebbe replicare un informazione di attestazione (lista di misure e aggregato TPM) prima che il sistema venisse corrotto. Questo viene evitato grazie alla presenza del *nonce* che permette di distinguere le diverse richieste di attestazione (quindi le relative risposte). Inoltre il nonce riesce anche garantire la freschezza delle informazioni ricevute dal sistema da attestare;

- **Tampering:** un Requestor corrotto o un attaccante intermedio potrebbe manipolare la lista di misure e l'aggregato del TPM prima o mentre viene trasmesso al Verifier. Grazie alla validazione della firma fatta dal Verifier, ci si può accorgere se l'aggregato del TPM è stato manipolato perché la firma sarà invalida. Inoltre ricalcolando l'aggregato del TPM attraverso la lista di misure e comparandolo con l'aggregato contenuto nella Quote, è possibile comprendere se la lista è stata manipolata o meno;
- **Masquerading:** un Requestor corrotto o un attaccante intermedio potrebbe rimpiazzare la lista originale delle misure e l'aggregato del TPM con una lista di misure e un aggregato del TPM di un altro sistema non compromesso. Questo attacco può essere scoperto dal Verifier comparando l'identificativo univoco del Requestor con quello ricavato dal certificato della chiave pubblica AIK di quel Requestor. Questo presuppone chiaramente che quel certificato sia valido e che quindi non sia ad esempio presente in una qualche lista di revocazione (e.g. *Certificate Revocation List*, ovvero le liste che contengono i certificati digitali revocati dalle CA e che non sono più considerati fidati).

Integrity Validation Mechanism

È la fase in cui il Verifier, una volta ottenute le informazioni in seguito all'esecuzione del protocollo, può prendere delle decisioni circa le azioni da eseguire. Basandosi sull'utilizzo di policy, può quindi essere in grado di gestire i casi in cui si trovi di fronte rispettivamente alla mancanza, aggiunta e/o modifica della lista di misure.

Un esempio di come potrebbero essere usati questi meccanismi è l'*integrità di una transazione*. Per poter verificare l'affidabilità di una transazione tra un Requestor, quindi il sistema di cui non ci si fida, e il Verifier, si può osservare lo stato d'integrità del sistema prima e dopo la transazione, ovvero prima di inviare la richiesta Web e dopo aver ricevuto la risposta Web dal Requestor. Se in entrambi i casi il Requestor si trovava in uno stato ritenuto affidabile, dopo aver verificato la correttezza delle misure, allora anche la transazione potrà essere considerata tale. Questo è sempre vero tranne in un caso, ovvero quando il sistema viene riavviato. Infatti un attaccante potrebbe procedere a modificare il sistema dopo la prima sfida e prima della seconda, e successivamente effettuare un riavvio. Questo può essere un problema per quei sistemi che una volta riavviati ritornano in uno stato predefinito, poiché in tal caso non si è in grado di riconoscere eventuali modifiche dal momento che il sistema è ritornato nello stesso stato che aveva prima della prima sfida.

Tuttavia, anche in presenza di tali sistemi le possibilità di applicare delle modifiche tra le due sfide è remota perché richiederebbe che l'attaccante sia molto veloce a compromettere il sistema e che la macchina sia altrettanto veloce ad effettuare il riavvio. Queste due condizioni sono difficili da realizzare dal momento che solitamente tra un'attestazione e l'altra passa pochissimo tempo. Esiste però un modo per accorgersi se il sistema ha effettuato o meno un riavvio tramite dei contatori conosciuti come *TPM Counters*, in particolare vengono chiamati *Monotonic Counters* nella versione 1.2 del TPM e *NV counters* nella versione 2.0. Questi contatori vengono incrementati dal BIOS ogni qualvolta si verifica un riavvio della macchina, possono solo crescere e il raggiungimento del loro valore massimo fa si che il BIOS disabiliti il TPM.

Quindi includere nell'attestazione questi contatori potrebbe aiutare un Verifier a comprendere se il Requestor è stato o meno riavviato.

2.7.2 Implementazione di IMA

Un elemento che gioca un ruolo fondamentale nel processo di misurazione di un file è il framework LSM (*Linux Security Modules*) [45] che realizza un controllo accessi consentendo di caricare policy di sicurezza come moduli kernel. In particolare quando un processo utente effettua una *system call* viene attraversati il modulo che cerca e alloca le risorse, quello che esegue il controllo dell'errore e quello che realizza lo UNIX DAC *Discretionary Access Controls*. Prima che il kernel acceda ad un oggetto interno, un *LSM hook* effettua una chiamata ad un modulo per il controllo delle policy (figura 2.9).

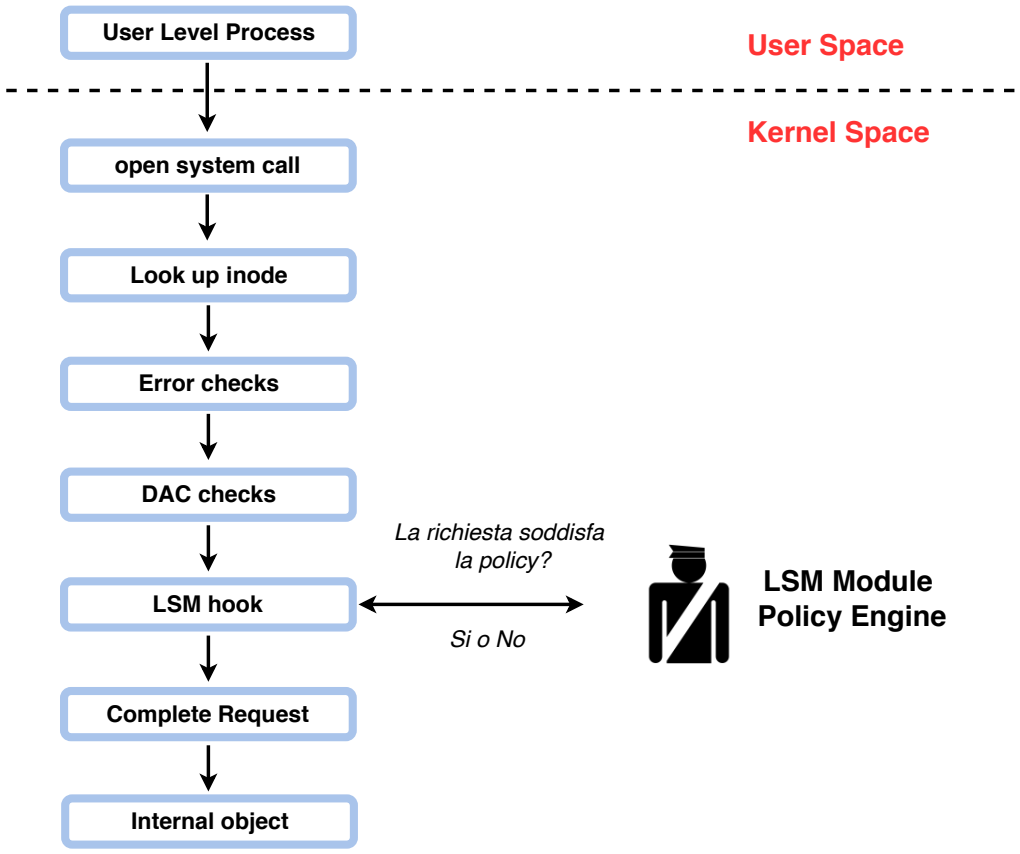

Figura 2.9. Funzionamento degli *LSM hook*

IMA sfrutta questo meccanismo per il processo di misurazione, usando, nella maggior parte dei casi, un LSM hook noto come *file_mmap*. Tale hook esegue una chiamata alla funzione *measure* ogni volta che un file o un'area di memoria devono essere misurati. La funzione *measure* riceve come argomento un puntatore ad una struttura che contiene il file che deve essere misurato. Da tale struttura può quindi prelevare l'*inode* del file, ovvero una riga di una tabella contenete le informazioni riguardanti un file, e i blocchi di dati sulla quale calcolare il digest SHA1.

La funzione measure, che come detto è responsabile di misurare i file, può essere invocata in tre modi differenti:

- dall'implementazione di una routine di scrittura/memorizzazione su uno pseudo file system /sys/security/measure usato dalle applicazioni del livello utente;
- dal LSM hook *file_mmap* nel caso di una misurazione di file che sono mappati in memoria come codice eseguibile;
- dalla routine *load_module* che misura i moduli kernel della memoria prima che la loro locazione venga cambiata.

Di seguito viene descritto il modo in cui si effettua la misurazione nei diversi casi.

Eseguibili del livello utente: tali eseguibili vengono invocati dalla system call *execve* e caricati dal loader del livello utente che eseguirà altre operazioni di caricamento e infine passerà il controllo alla funzione *main* dell'eseguibile. Nel caso di librerie statiche solamente il file binario viene caricato in memoria e verrà misurato grazie all'hook *file_mmap* che procederà a chiamare la funzione measure;

Librerie caricate in modo dinamico: il caricamento di una libreria dinamica è un processo effettuato dal loader del livello utente e completamente trasparente al kernel. Tuttavia, grazie al fatto che tali librerie vengono caricate nella memoria virtuale grazie alla system call *mmap*, che invoca l'hook *file_mmap*, è possibile applicare il processo di misurazione anche in questo caso;

Moduli del Kernel: i moduli kernel possono essere caricati dinamicamente in maniera esplicita, tramite *modprobe* o *insmod*, oppure implicita, tramite caricamento automatico, dopo che il sistema viene avviato. In entrambi i casi, a partire dalla versione 2.6 del kernel, i moduli vengono caricati in memoria e successivamente viene chiamata la funzione *sys_init_module* che informa il kernel della presenza di un nuovo modulo che verrà caricato nella memoria del kernel. Pertanto la misurazione dei moduli può essere eseguita da modprobe o inmod dopo che i moduli vengono caricati dal file system, oppure quando vengono trasferiti nella memoria del kernel, caso preferibile dal punto di vista della sicurezza perché previene da possibili exploit di vulnerabilità su modprobe e insmod. In ogni caso qui non è possibile applicare nessun LSM hook e quindi per chiamare la funzione measure bisogna utilizzare la funzione load_module chiamata a sua volta da sys_init_module;

Script: gli interpreti degli script vengono caricati e misurati come eseguibili comuni. Tuttavia gli interpreti caricano codice addizionale che determina il comportamento degli script stessi e sarebbe preferibile che gli interpreti fossero capaci di misurare gli input importanti dal punto di vista dell'integrità.

Ciò che garantisce che il file attualmente misurato sia esattamente quello che è presente in memoria, è l'accurata identificazione dell'inode del file e il rilevamento delle modifiche successive al file descritto da quell'inode. Questo è reso possibile dalla presenza di alcuni *lock* che proteggono il file da scritture finché questo è mappato in memoria come eseguibile. Nel caso dei moduli kernel che sono misurati quando sono già nella memoria del kernel queste inconsistenze non sono possibili. Per i file che invece vengono misurati nello spazio utente, si assume che le applicazioni mantengano aperto il descrittore del file finché il contenuto non viene letto o non viene richiesta una nuova misura, garantendo che il file letto sia lo stesso precedentemente misurato.

Un meccanismo molto importante dal punto di vista delle performance, che è stato introdotto dagli sviluppatori di IMA, è quello del *caching*. Questo meccanismo evita di eseguire nuovamente la misurazione se il file non è stato cambiato rispetto alla misurazione precedente, effettuando una nuova misurazione solamente se il file non è mai stato visto prima (*cache-miss*). Ogni misura viene inserita all'interno di una lista ordinata poiché conoscere l'ordine di misurazione è molto importante per poter riuscire a rilevare ogni modifica alla lista. Infatti se non fosse così non si riuscirebbe ad ottenere un aggregato che rispecchi il valore contenuto all'interno del PCR nel TPM.

Per ragioni di efficienza la lista è accompagnata da due tabelle di hash: una avente come chiavi il numero di inode e il numero di dispositivo del file misurato, l'altra avente come chiave il valore SHA1 del file misurato. In questo modo quando la funzione measure viene chiamata per effettuare la misurazione di un file potrà sfruttare queste tabelle e, usando l'inode corrispondente al descrittore del file, potrà controllare se il file è stato già misurato. Ogni riga contiene un bit che indica se il file è *CLEAN*, cioè non è stato modificato, o *DIRTY*, ovvero che possibilmente è stato modificato.

Di seguito viene evidenziato il comportamento adottato nelle differenti situazioni.

Misurare nuovi file: si cerca l'inode del file nella tabella avente come chiave l'inode e nel caso non venisse trovato si calcola il rispettivo valore SHA1. Il risultato così ottenuto lo si va a cercare all'interno della tabella avente come chiave il valore SHA1 del file. Se anche in questo caso non si ha alcun riscontro, si procede alla creazione di una nuova riga alla lista delle misure e si aggiornano le tabelle. In tal caso inoltre viene effettuata l'operazione di estensione del PCR. Se il valore SHA1 del file viene invece trovato nella seconda tabella, l'operazione di estensione non viene effettuata. Questo può succedere quando i file eseguibili sono delle copie e quindi hanno lo stesso valore SHA1, ma inode differente. In questo caso gli sviluppatori di IMA assumono che gli eseguibili sino equivalenti.

Rimisurare dei file: se il file viene trovato all'interno della tabella con chiave l'inode, significa che era già stato misurato in precedenza. Quindi si procede a controllare il bit di CLEAN/DIRTY. Se è CLEAN non viene eseguita alcuna operazione, altrimenti viene ricalcolato il valore SHA1 del file. Se il valore ottenuto è identico ad un valore nella lista di misure, lo stato del bit viene cambiato a CLEAN e il PCR non viene esteso poiché la misura è già conosciuta. Se invece il valore non viene trovato nella lista, si controlla se è presente nella seconda tabella, quella avente come chiave il valore SHA1. Se esiste, significa che lo stesso contenuto del file era già stato misurato (poiché ci sono delle copie del file corrente) e non viene seguita alcuna azione; altrimenti significa che il file corrente è stato cambiato e quindi occorre estendere il PCR e aggiornare lista e tabelle.

Dirty flagging: il bit di CLEAN/DIRTY viene impostato a DIRTY ogni qual volta:

- un file viene aperto con permessi di *write*, *create*, *truncate* o *append*;
- un file si trova su un file system del quale non si può controllare l'accesso (e.g. NFS);
- un file appartiene ad un file system che era non montato in precedenza.

Sembra abbastanza conservativo dire che un file è stato modificato solo perché viene aperto. Tuttavia la presenza della tabella che ha come chiave il valore SHA1 permette di ripulire il bit dallo stato di DIRTY se il file non è stato cambiato dopo che è stato aperto.

Misurare moduli kernel: vengono sempre misurati poiché non esiste alcuna informazione facilmente accessibile che permetta di definire un stato di DIRTY. Fortunatamente nella maggior parte dei casi sono pochi i moduli ad essere caricati. Per sapere se il modulo è stato già misurato, si effettua un controllo nella tabella avente come chiave il valore SHA1 e solo nel caso in cui non vi fosse alcun riscontro, si estende il valore del PCR all'interno del TPM.

Invalidare l'aggregato del TPM

Nel caso in cui l'architettura di misurazione progettata dagli sviluppatori di IMA non fosse in grado di fornire misurazioni corrette o venisse aggirata, si procede ad invalidare l'aggregato del TPM, estendendolo con un valore random e non registrato tale valore nella lista di misure. In questo modo il sistema che effettua la verifica sulla macchina da attestare, potrà facilmente accorgersi che qualcosa è andato storto e classificare la macchina come non affidabile.

Sebbene l'assunzione fatta in precedenza circa la possibilità che non ci possano essere attacchi hardware al TPM e il fatto che questo non possa essere manipolato in modo da mascherare gli attacchi software, minimizzano la possibilità che l'architettura possa essere aggirata e che l'aggregato sia consequenzialmente invalidato. Tuttavia qualsiasi utente che abbia accesso come *root* può aggirare i controlli eseguiti dagli hook e rompere la validità delle misure. Pertanto di seguito sono indicati dei casi in cui si riesce comunque a salvaguardare la sicurezza del sistema andando ad invalidare l'aggregato del TPM:

- **race conditions di tipo Time-of-measurement Time-of-use :** Per alcuni file come script o file di configurazione, il contenuto del file può essere cambiato tra quando il file viene misurato e quando viene realmente caricato. Per accorgersi di tale modifica, si utilizza un *measure count*, ovvero un contatore posto nell'inode di un file misurato. Tale contatore viene incrementato prima di chiamare la funzione measure e decrementato quando un file descriptor che era stato misurato viene chiuso. Quando arriva una richiesto di permesso write/append, si controlla se il *measure count* è maggiore di zero. Se si, allora l'aggregato nel TPM viene invalidato;
- **Aggirare il Dirty Flagging:** Ci sono alcuni processi eseguiti come root che potrebbero aggirare il controllo sul bit di DIRTY e riuscire quindi a cambiare il contenuto di un file. Fortunatamente questi tentativi vengono intercettati da IMA che procede di conseguenza ad invalidare il TPM;
- **Errori a run time tra funzioni di misurazione:** In caso di errori nella registrazione delle misure come ad esempio quelli che possono essere causati dall'insufficienza di memoria nel creare una nuova struttura di misurazione, portano all'invalidazione dell'aggregato del TPM.

Capitolo 3

Architettura utilizzata: OpenCIT

Open Cloud Integrity Technology (Open CIT) [10] è un Software Development Kit (SDK) utilizzato per l'attestazione di un sistema Cloud, che permette, grazie ad un'architettura nota come *Intel Trusted Execution Technology* (Intel TXT) [46], di misurare le componenti del sistema da attestare verificando che non siano state manipolate. Intel TXT è il punto di partenza di una *Chain Of Trust* e costituisce quindi il *Root of Trust* della catena, che, durante la fase di boot, misurerà dei software che misureranno altri software.

Open CIT si serve di un *Attestation Service* remoto per verificare che i componenti della macchina attestata non siano stati modificati rispetto ad un insieme di misure *known-good* precedentemente memorizzate.

Attualmente Open CIT è alla versione 3.2.1 e rispetto alle versioni precedenti, dove potevano essere misurati solamente il BIOS, il kernel del sistema operativo e i componenti dell'hypervisor, ora si possono definire delle policy, note come *Trust Policy*, che permettono di misurare file e cartelle appartenenti sia a macchine virtuali che a macchine fisiche.

Grazie al processo di misurazione iniziato con Intel TXT ed esteso con il processo di attestazione, i fruitori del Cloud potranno essere sicuri che i loro servizi vengano eseguiti su hardware e software che soddisfano dei criteri di sicurezza ben definiti.

3.1 Intel TXT

OpenCIT per poter avviare il processo di misurazione sul server da attestare, sfrutta un'architettura nota come Intel Trusted Execution Technology (Intel TXT) che si pone come una *Root of Trust* nella *Chain of Trust* che si viene a creare durante la misurazione.

Intel TXT [46] è un'architettura altamente scalabile che fornisce tecnologie di sicurezza basate sull'hardware con lo scopo di fornire delle protezioni contro le crescenti minacce verso le infrastrutture fisiche e virtuali. In modo particolare, Intel TXT è in grado di proteggere la piattaforma da minacce come: attacchi all'hypervisor, al BIOS, installazioni di *root kit*, attacchi al firmware e in generale a tutti gli altri attacchi basati sul software che minacciano le proprietà di integrità, confidenzialità, affidabilità e disponibilità del sistema.

Intel TXT permette di controllare il processo di boot della macchina mediante un *Measured Launch Environment* (MLE) che consente di confrontare tutti gli elementi critici dell'ambiente di lancio con una sorgente Known-Good. Intel TXT crea un identificatore crittografico univoco per ogni componente approvato coinvolto nel processo di lancio, fornendo meccanismi di esecuzione basati sull'hardware che permettono di bloccare il lancio di codice che non è conforme al codice approvato. I valori Known-Good vengono forniti attraverso il TPM, memorizzati all'interno della memoria non volatile dello stesso. In questo modo all'accensione della macchina avviene la misurazione del BIOS, il cui risultato verrà confrontato con il valore Known-Good nel TPM. Se i valori non coincidono, allora questo indicherà che lo stato della piattaforma è Untrusted, altrimenti se

i valori coincidono, vorrà dire che non c'è stata alcuna manipolazione e lo stato del sistema sarà Trusted. Si procederà quindi con il componente successivo e anche in questo caso verrà effettuato lo stesso controllo. Intel TXT fornisce:

Verified Launch: ovvero una catena di fiducia basata sull'hardware che permette il lancio di un MLE in uno stato Known-Good;

Launch Control Policy: un meccanismo che consente la creazione e l'implementazione di liste di codice approvato o Known-Good;

Secret Protection: meccanismo hardware che consente di rimuovere i dati in caso di un errato spegnimento di un MLE e di proteggere i dati dagli attacchi di reset e di *memory-snooping*;

Attestation: meccanismo che consente di fornire delle credenziali ad utenti o sistemi locali e/o remoti col fine di completare il processo di verifica e supportare attività di audit.

3.1.1 I componenti della piattaforma

I componenti [48] sul quale si basa Intel TXT sono i seguenti:

- **CPU and chipset hardware:** Il chipset contiene registri speciali utilizzati da Intel TXT, molti dei quali leggibili e/o scrivibili solamente dagli *Authenticated Code Modules* e dal *CPU microcode*;
- **CPU microcode:** è un firmware inserito direttamente all'interno del microprocessore per eseguire gruppi di micro-operazioni che vengono combinate con lo scopo di eseguire istruzioni in linguaggio assembler e altre funzioni interne alla CPU;
- **Intel Authenticated Code Modules (ACMs):** ACMs sta ad indicare un insieme di moduli costituiti da codice digitalmente firmato da Intel TXT ed invocati da una speciale funzione nota come *GETSEC*. Questi moduli entrano in gioco nelle SRTM e DTRM, indicate in precedenza. La scelta del modulo ACM che verrà invocato dipende da un registro che viene configurato in seguito all'esecuzione dell'istruzione GETSEC. Il modulo ACM, prima di essere eseguito, viene autenticato utilizzando la firma che è contenuta nell'*header*. In particolare tale modulo viene firmato con una chiave privata conosciuta solo da Intel, mentre la chiave pubblica viene incapsulata all'interno dei registri hardware del chipset e quindi solamente quei moduli firmati con la corrispondente chiave privata potranno essere eseguiti. Per i server Intel TXT esistono solo due moduli ACM:

  BIOS ACM: contiene molte funzioni tra le quali *Startup ACM* chiamata dal CPU microcode e che ha come compito quello di avviare la SRTM e misurare il BIOS, e la funzione *Lock Config* chiamata dal BIOS che ha il compito di proteggere alcuni registri da software o firmware malevolo;

  SINIT ACM: viene chiamato dal sistema operativo o dalle applicazioni sotto di esso con lo scopo di eseguire un *Measured Launch*(ML), ovvero del processo che porta ad avere un MLE.

  Entrambi i moduli ACM vengono eseguiti all'interno di una memoria speciale della CPU che è protetta da accessi *DMA* e da qualunque *snooping* di codice e dati dei moduli;
- **GETSEC:** è una funzione speciale scritta in linguaggio assembler il cui scopo è quello di eseguire i moduli ACM, uscire da essi e dal ML;
- **BIOS enabling for Intel TXT:** c'è una tabella all'interno del BIOS nota come *Firmware Interface Table* (FIT) che ha il compito di far sapere ai moduli ACM e al microcode se Intel TXT è abilitato, dove è posizionato il BIOS ACM e quali parti del BIOS devono essere misurate;
- **TPM:** i PCR all'interno del TPM vengono utilizzati per memorizzare le misure dei componenti coinvolti nel processo di boot. Alcuni PCR vengono estesi dal microcode e altri dai moduli ACM. La memoria NV all'interno del TPM è invece utilizzata per conservare i valori Known-Good utilizzati nel MLE.

3.1.2 Boot Sequence

In figura 3.1 viene illustrato il processo di ML nel tempo e come i vari componenti interagiscono con il TPM. Durante il processo che avviene a boot time e che porta alla misurazione dei vari componenti, in ognuno dei passi che verranno di seguito illustrati, è sempre possibile che ci siano degli errori che faranno si che in un registro del chipset venga inserito un valore del tipo *TXT.ERRORCODE*. Il processo è così articolato:

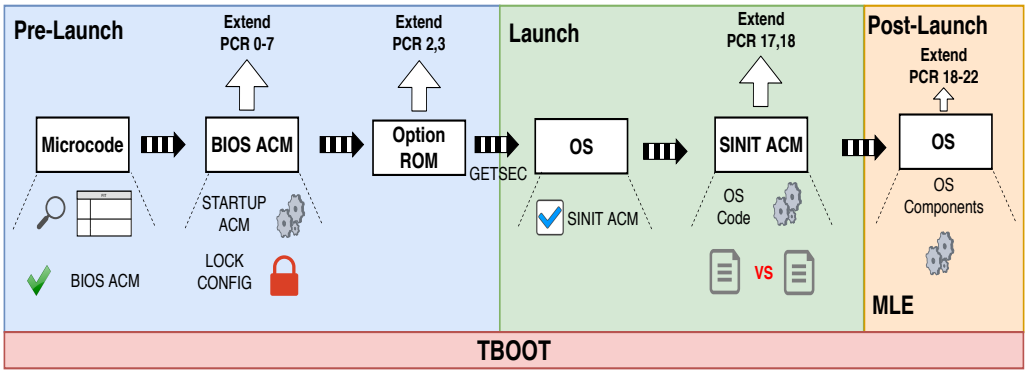

Figura 3.1. Intel TXT boot

1. Il microcode utilizza la tabella BIOS FIT per determinare dove si trova il modulo BIOS ACM nell'immagine del BIOS. Verifica quindi la firma del BIOS ACM e dopo altri controlli di sicurezza se tutto va bene e l'ACM risulta non corrotto, allora quest'ultimo verrà eseguito all'interno di una memoria protetta nella CPU;

2. Il microcode invoca la funzione *Startup ACM* misurerà quelle porzioni del BIOS che sono state specificate nella tabella FIT, configurate dall'OEM del BIOS. Non è necessario misurare tutta l'immagine del BIOS e qualunque regione della memoria flash poiché ci sono alcune regioni che anche in presenza di un normale boot, possono cambiare e quindi se misurate causerebbero sempre la presenza di uno stato Untrusted. Le porzioni del BIOS che verranno misurate, daranno origine a delle misure che saranno estese nel PCR0. Se la funzione *Stratup ACM* termina senza alcun errore, il BIOS viene eseguito;

3. Il BIOS misurerà altro codice del BIOS stesso estendendo dal PCR0 al PCR7, e crea un log di qualsiasi cosa venga misurata. Prima che venga eseguito qualsiasi codice al di fuori dei confini dei confini del BIOS, il BIOS ACM chiama la funzione *Lock Config* in modo da proteggere le configurazioni della piattaforma da codice non fidato;

4. Vengono quindi misurate le *Option ROM* che a meno che non vengano fornite dall'OEM, si trovano al di fuori dei confini di fiducia stabiliti dal BIOS e il loro codice verrà misurato ed esteso nel PCR2, mentre la loro configurazione nel PCR3;

5. Al termine dell'esecuzione del BIOS, questo avvia il sistema operativo che dovrà anche esso essere verificato;

6. Affinché il sistema operativo venga avviato in una modalità fidata, viene eseguita la funzione *GETSEC* che permette al microcode di verificare il SINIT ACM in modo simile a quanto avviene nei punti 1 e 2, e successivamente lo esegue;

7. Il SINIT ACM verifica che durante l'esecuzione dei passi precedenti non vi sia stato alcun errore analizzando il registro del chipset che dovrebbe contenere il valore *TXT.ERRORCODE*. Successivamente misura il codice del sistema operativo e attiva il meccanismo di LCP con lo scopo di confrontare le misure ottenute in seguito alla misurazione del sistema operativo e i valori dei PCR, estesi nei punti precedenti, con le liste di valori Known-Good. Se nel corso di questa operazioni si verificassero delle incongruenze di valori, il SINIT ACM procederà ad

un reset della piattaforma. Se invece tutto va bene, allora il SINIT ACM eseguirà il ML che porterà il sistema operativo in una modalità sicura, ovvero in un MLE. Le misure prodotte dal SINIT ACM vengono estese nei PCR17 e PCR18;

8. Una volta che il sistema operativo viene considerato fidato, andrà a misurare altri componenti del sistema operativo stesso e i risultati prodotti verranno estesi nei PCR dal 18 al 22;

9. Questo processo può essere sfruttato dalle applicazioni locali in ambito di una *Local Attestation*. Infatti queste potrebbero utilizzare i valori dei PCR per proteggere delle informazioni segrete che vengono prelevate solo alla fine del processo di misurazione, quando quindi la piattaforma sarà in un ambiente fidato. Ad esempio il sistema operativo potrebbe memorizzare delle chiavi di cifratura che utilizza per cifrare informazioni private e privilegiate. Solamente nel caso in cui il processo di ML ha avuto un esito positivo, il sistema operativo potrà recuperare le chiavi e decifrare i dati. Un altro ambito in cui tale processo può essere sfruttato è sicuramente la *Remote Attestation*, dove un agente esterno utilizza i valori dei PCR per prendere una decisione circa lo stato di fiducia della macchina.

3.1.3 Trusted Boot

Trusted Boot (TBOOT) [49] è un modulo pre-kernel/virtual machine manager (VMM), open source che usa Intel TXT per un ML del kernel/VMM di un sistema operativo. TBOOT è un eseguibile che comprende se la macchina supporta o meno Intel TXT e in questo caso chiamare la funzione GETSEC. Se TBOOT determina che il sistema non supporta Intel TXT o non è configurato correttamente, a causa ad esempio di un modulo SINIT ACM non corretto, allora lancerà direttamente il kernel senza apportare alcuna modifica.

OpenCIT prevede l'utilizzo di TBOOT per calcolare tutte le misure che, estese nei vari PCR, daranno origine all'aggregato che dovrà essere verificato nei confronti della WhiteList. In particolare ogni volta che viene effettuato il boot della macchina viene lanciato uno script che esegue il comando *txt-stat* che ha il compito di mostrare in output le misure calcolate da TBOOT. Lo script catturerà l'output del comando e creerà un file XML all'interno della quale inserirà i risultati raccolti, secondo uno specifico formato. Questi dati verranno utilizzati, come si vedrà, in fase di attestazione.

3.2 Soluzioni esistenti di verifica dell'integrità in ambienti Cloud

Nel corso degli ultimi anni si sono fatte strada diverse soluzioni per risolvere il problema dell'attestazione remota in ambiente cloud:

Excalibur: [50] è un sistema che offre una nuova astrazione del TC conosciuta come *policy-sealed data*, basata su uno schema crittografico noto come *Ciphertext Policy Attribute-Based- Encryption*(CPABE). In questo sistema un nodo è in grado di accedere ai dati del cloud solo se la sua configurazione soddisfa una determinata policy, ovvero una collezione di attributi a cui sono associati dei valori ben definiti. Excalibur si pone come obiettivo quello di garantire l'integrità e la confidenzialità dei dati nei nodi facenti parte del Cloud proponendo un meccanismo di attestazione remota basato su certificati. Un certificato è una collezione di coppie chiave-valore, ad esempio i valori dei PCR che un nodo deve avere e le chiavi AIK relative al nodo stesso. Il processo di attestazione viene effettuato da un'entità nota come *monitor*, il quale è in grado di attestare ogni nodo accendendo ad un database di certificati definiti per gli stessi e controllando che il nodo soddisfi i valori definiti all'interno di essi.

Cobweb: [51] è un framework di attestazione di ambienti cloud che vede il problema di attestazione come un *contextual graph problem*, ovvero un problema in cui un messaggio di attestazione viene visto come un grafo. Tale grafo deve fornire delle informazioni di contesto che evidenzino le relazioni esistesti tra i vari componenti software dei nodi del cloud, come una relazione di *fork()* tra processo padre e processo figlio o una relazione IPC tra processo *sender* e processo *receiver*.

Il grafo avrà come vertici i componenti software caratterizzati da un o più *label*, all'interno delle quali si inserisce il valore di hash del componente. Gli archi del grafo invece indicano le relazioni esistenti tra i componenti software. Il processo di attestazione prevede, come detto, l'invio di un grafo che verrà confrontato con delle policy che non sono altro che un insieme di *template graph* che vengono comparati con il grafo ricevuto durante l'attestazione. Questi grafi di valori attesi vengono prodotti in condizioni affidabili da un insieme di VM caratterizzate da software sempre aggiornato, in modo che il grafo prodotto rispecchi sempre le versioni più recenti dei vari componenti software.

Intel Software Guard Extensions (Intel SGX): [52] è un progetto disponibile per un'architettura Intel, in particolare a partire dalla sesta generazione di processori. Questo progetto si basa sul concetto di *enclave*, un container sicuro all'interno della quale un'applicazione può inserire i dati e codice che vuole proteggere da possibile software malizioso all'interno della macchina. Un'enclave è una zona di memoria che viene isolata dall'ambiente circostante e viene attestata attraverso un processo di attestazione che si verifica ogni volta che l'applicazione associata all'enclave richiede un servizio o delle informazioni ad un *service provider*, che, prima di fornire la sua risposta vuol capire se può fidarsi o meno di quell'enclave. Il processo di attestazione vede coinvolta un ulteriore entità nota come *quoting enclave* che ha il compito di firmare il valore di hash riguardante il contenuto dell'enclave e produrre una *quote*. Un server conosciuto come *Intel Attestation Server* (IAS) avrà il compito di ricevere la quote e verificarla tramite una *trusted configuration*, ovvero una configurazione attesa per quell'enclave e creata dall'autore dell'enclave.

Open Attestation (OAT): [9] è un SDK sviluppato da Intel, il cui obiettivo è verificare l'integrità dei nodi all'interno di un ambiente cloud. L'host periodicamente interroga l'*Appraiser*, ovvero il componente che ha il compito di effettuare l'attestazione e successivamente crea una *quote* che comprende l'insieme dei valori dei PCR sull'host. L'Appraiser procederà a verificare tali valori rispetto a delle *Whitelist* definite a priori in cui sono stati dichiarati dei valori ritenuti affidabili per i PCR in questione.

Tutte le tecnologie finora descritte riescono a verificare l'integrità dei nodi all'interno di un'infrastruttura cloud. Tuttavia in questo lavoro di tesi la scelta del framework di attestazione da utilizzare è ricaduta su Open CIT. Open CIT è il successore di OAT ed infatti lo sta sostituendo, tanto è che OAT non riceve più alcun aggiornamento. Come tutte le tecnologie di cui si è discusso, Open CIT realizza l'attestazione remota delle macchine all'interno del cloud, ma presenta alcune importanti differenze. Innanzitutto, ciò che è più rilevante è la presenza di Intel TXT che, come si è visto, è un meccanismo hardware che consente di valutare l'affidabilità di una piattaforma nelle sue fasi di avvio e di lancio, coinvolgendo nel processo il BIOS e l'hypervisor. Grazie ad Intel TXT viene introdotta, quindi la possibilità di poter definire un ambiente sicuro mantenendo le proprietà di confidenzialità, integrità e affidabilità, prima che qualsiasi applicazione possa essere eseguita. In questo modo è possibile quindi definire lo stato della macchina prima di effettuare qualunque altra operazione. Inoltre Open CIT permette di effettuare l'attestazione sia di macchine con TPM 1.2 che con TPM 2.0, invece OAT attesta solo macchine con TPM 1.2.

Open CIT prevede un meccanismo di validazione basato sulla configurazione di Whitelist statiche che non hanno bisogno di essere ricavate da un processo molto oneroso come mettere su un insieme di VM, come accade in Cobweb, o predisporre un insieme di certificati come in Excalibur, ma possono essere registrate insieme all'host al momento della registrazione quando l'host può essere considerato ancora affidabile.

Open CIT, a differenza delle altre soluzioni citate, permette anche di verificare i file e le cartelle che fungono da configurazione per l'ambiente di esecuzione, sia sui server fisici ma anche sulle VM, negando per queste ultime la loro esecuzione se non viene riscontrata una configurazione affidabile. Soluzioni come Cobweb e Intel SGX permettono tuttavia di ottenere informazioni di integrità circa il software che risulta essere in esecuzione sulla macchina, a differenza di Open CIT che invece non è in grado di fornire questo controllo di integrità e quindi non riesce a fornire informazioni che costantemente mostrino se quanto viene eseguito rispecchi o meno uno stato affidabile del sistema. Questo è un problema, in quanto possibili modifiche al software o l'esecuzione di software non previsto, possono compromettere gravemente il funzionamento dello stesso sistema che non sarà quindi in grado di garantire agli utenti che ne fanno uso un servizio affidabile. Di qui la scelta di introdurre in Open CIT un meccanismo come IMA che per registrare le misure a "runtime", in modo da ottenere costantemente delle informazioni di integrità relative al software in esecuzione sulla

macchina. L'introduzione di IMA in Open CIT consente di ottenere un altro vantaggio rispetto ad Intel SGX. Intel SGX permette di verificare l'integrità un'applicazione alla volta e solamente quando è l'applicazione a richiedere ad un Service Provider un servizio o delle informazioni. Al contrario Open CIT con l'integrazione di IMA permette di ottenere informazioni di integrità di tutte le applicazioni in esecuzione contemporaneamente.

3.3 I componenti di Open CIT

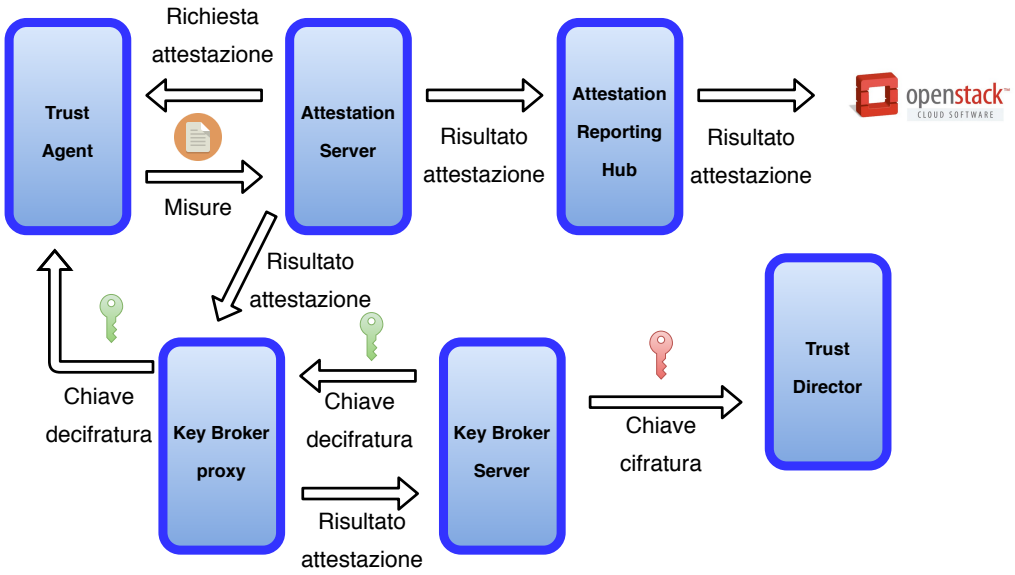

Figura 3.2. Componenti Open CIT e loro interazione

Open CIT è costituito da un insieme di componenti (figura 3.2) che permettono la gestione di un ambiente Cloud. Il componente utilizzato per le operazioni di verifica è l'*Attestation Server* (AS). L'AS esegue l'attestazione remota dei server facenti parte dell'infrastruttura Cloud, confrontando le misure provenienti da essi con un database contenente i valori delle stesse, misurati in ambiente fidato e che prendono il nome di misure *Known-Good*. Ogni server facente parte del Cloud deve essere registrato presso l'AS. Durante la fase di registrazione vengono stabilite le misure Known-Good per la determinata macchina che saranno utilizzate per le attestazioni future. Il risultato prodotto a seguito del processo di attestazione viene riportato tramite l'*Attestation Reporting Hub* (ARH) ad un servizio di schedulazione noto come *Open Stack* [47].

Ogni server dell'infrastruttura deve essere caratterizzato da un *Trust Agent* ovvero un componente che ha il compito di comunicare con tutti gli altri componenti di Open CIT per fornire le misure ad un AS e/o richiedere di lanciare una *Virtual Machine* (VM).

La generazione delle immagini delle VM, immagini Docker, Trust Policy e la cifratura delle immagini delle VM avviene attraverso l'utilizzo di un componente noto come *Trust Director* (TD). Per cifrare le immagini delle VM il TD collabora con il *Key Broker Server*, noto anche come *Key Management Server* (KMS). Il KMS gestisce le chiavi di cifratura/decifratura fornendo ad un *compute node*, ovvero un server del Cloud, che richiede il lancio di una VM, le chiavi per la decifratura dell'immagine. La chiave di decifratura viene fornita al compute node solamente se l'attestazione effettuata dall'AS soddisfa i requisiti di policy prestabilite.

Al fine di migliorare le prestazioni nella comunicazione tra il KMS e gli altri componenti dell'infrastruttura Cloud, viene utilizzato un componente noto come *Key Broker Proxy* (KMS proxy).

3.4 Whitelist

Il processo di attestazione basa la sua fase di verifica sui valori Known-Good. Questi sono importanti poiché possiono essere confrontati con i valori provenienti dalla macchina da attestare, per comprendere lo stato di affidabilità della macchina stessa. Lo stato di una macchina da attestare può transitare solamente tra tre possibili valori:

Trusted: quando tutte le misurazioni soddisfano i valori Known-Good;

Untrusted: quando esiste almeno una misurazione che non soddisfa i valori Known-Good;

Unknown: quando non si riesce a contattare la macchina da attestare.

L'insieme dei valori Known-Good viene inglobato all'interno di una struttura dati nota come *Whitelist* o *Measured Launch Environment* (MLE), insieme ad altre informazioni riguardanti l'host da attestare. È possibile distinguere due differenti tipi di MLE:

- **BIOS MLE:** riguarda le misure specifiche della piattaforma (e.g. il BIOS);
- **VMM MLE:** riguarda le misure del kernel del sistema operativo ed altri componenti dello stesso, quali ad esempio l'hypervisor e i file definiti nella Trust Policy inerente all'host. Infine comprende anche le misure di *tboot* ovvero un modulo molto importante che consente di attuare il processo di misurazione utilizzando Intel TXT e che verrà trattato in seguito.

I valori di Whitelist sono specifici di una precisa versione e configurazione dei file dell'host da attestare. Questo significa che ad esempio la versione 1.1 del BIOS avrà un insieme di misure differente rispetto alla versione 1.2; così come la versione 3.13.0-24-generic del kernel avrà misure differenti rispetto alla versione 3.13.0-30-generic. Se chiaramente uno di questi file venisse aggiornato da una versione all'altra, risulterebbe che il suo digest non sarebbe più uguale al valore registrato nella Whitelist.

È inoltre possibile applicare una stessa MLE a più di un server. Ad esempio se un data center usa solo un tipo di server, con la stessa versione del BIOS, stesso sistema operativo e stessi altri componenti misurati, allora si potranno utilizzare una BIOS MLE e una VMM MLE identiche per tutti gli host. Appare chiaro che se all'interno del data center dovessero invece essere presenti host che differiscono tra loro nelle versioni e nei differenti componenti da misurare, occorrerà definire delle BIOS MLE e VMM MLE specifiche per ognuno di essi.

Molto spesso le Whitelist, BIOS e VMM, vengono create al momento della registrazione dell'host usando i valori provenienti dall'host stesso, dal momento che in fase di registrazione l'host può essere considerato affidabile. In particolare durante la creazione delle Whitelist vengono definite le seguenti informazioni.

Host Type. Il tipo di host che si vuole attestare che può appartenere solo ai seguenti tipi: Linux, Linux KVM, Citrix XenServer, VMware ESXi, Windows or Xen.

Tipo di MLE e server su cui applicarla. Si indica se si vuole creare una BIOS MLE, una VMM MLE o entrambe. Inoltre si dichiara su quali server sarà applicata la Whitelist scegliendo tra:

- *OEM* (Original Equipment Manufacturer), indica che la Whitelist che si sta creando sarà applicabile solo per lo specifico OEM, ovvero che tutti i server appartenenti a quell'OEM potranno essere attestati con questa stessa WhiteList;
- *Global*, indica che la Whitelist è applicabile a qualsiasi tipo di OEM e quindi sarà possibile attestare allo stesso modo due server appartenenti a OEM differenti;
- *Host*, indica che la Whitelist è applicabile solo per quel host specifico e quindi anche se l'host appartiene allo stesso OEM di un altro host, non possono essere attestati utilizzando la stessa WhiteList definita per quel OEM.

PCR. Vengono qui indicati i PCR il cui valore sarà registrato all'interno delle WhiteList, con lo scopo di verificare durante l'attestazione che tale valore non sia cambiato. Inoltre un PCR può riguardare uno o l'atro tipo di Whitelist a seconda di quello che è misurato ed è esteso all'interno del PCR stesso. Ad esempio il PCR 0 misura la BIOS ROM per cui il valore che verrà fuori in seguito alla misurazione verrà esteso nel PCR0 e memorizzato nella Whitelist di tipo BIOS.

WhiteList host. Nome o indirizzo IP dell'host che si vuole attestare.

Port. Indica la porta che l'AS utilizzerà per comunicare con l'host.

TLS policy. L'AS valida l'autenticità delle connessioni attraverso l'uso di differenti policy TLS scelte tra tipi predefiniti.

Le WhiteList così create vengono memorizzate all'interno di uno specifico database sull'AS. Quando viene importata una nuova Whitelist ed una stessa versione esiste già nel database, il comportamento dell'AS è quello di controllare se sono cambiati i valori dei PCR o di qualche componente misurato. Se questo è vero, il comportamento di default dell'AS è quello di aggiungere una nuova Whitelist senza sovrascrivere l'originale, ma aggiungendo alla fine del suo nome un tag numerico (001, 002, ecc.). Sempre in fase di registrazione dell'host, è però possibile indicare se la Whitelist che si sta creando dovrà sovrascrivere la Whitelist esistente ed in questo caso l'AS, indipendentemente dal fatto che i valori dei PCR o dei componenti siano cambiati o meno, forzerà la sovrascrittura della Whitelist esistente.

3.5 L'utilizzo delle Trust Policy

Una Trust Policy è una lista di file e cartelle che vengono misurati e verificati attraverso l'utilizzo delle Whitelist. Le Trust Policy possono essere definite sia per i server fisici (in tal caso prendono il nome di *non-virtualized server Trust Policy*) che per le macchine virtuali.

Una Trust Policy è quindi un insieme di valori Known-Good usati per verificare le misure di file e cartelle, calcolate in fase di boot della macchina. Qualsiasi deviazione come aggiunta, cancellazione, rinomina, modifica di un file misurato in una directory o di un file specificatamente misurato rispetto alle misure attese porta a considerare la macchina in uno stato ritenuto non affidabile.

Dal momento che il processo di misurazione dei file e delle cartelle selezionate nella Trust Policy viene effettuato a boot time, è importante che facciano parte di una Trust Policy solamente quei file e cartelle considerati *statici*, ovvero che non cambiano tra un riavvio e l'altro della macchina.

La creazione delle Trust Policy viene affidata al Trust Director, il quale dà la possibilità di definire delle regole di inclusione/esclusione, basate su espressioni regolari, applicate alle cartelle.

L'applicazione di un filtro di esclusione comporterà l'esclusione, quindi la non misurazione, di tutti quei file che soddisfano l'espressione di esclusione. I filtri di esclusione vengono sempre applicati dopo i filtri di inclusione. Questo significa che una cartella con filtro di inclusione del tipo "*" e filtro di esclusione del tipo "*.txt", consentirà la misurazione di tutti i fie nella cartella, esclusi quei file il cui nome contenga la stringa ".txt". Le regole di inclusione/esclusione possono essere applicate ad una cartella in modo individuale o in modo ricorsivo a tutte le sotto cartelle.

Le Trust Policy per i server fisici sono rappresentate da un file XML che, al momento della creazione della policy, viene memorizzato sulla macchina da attestare sotto la cartella */boot/trust* con il nome "manifest.xml".

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Manifest xmlns="mtwilson:trustdirector:manifest:1.2" DigestAlg="sha256">
       <Dir Include=".*" Exclude="" FilterType="regex" Path="/opt/mtwilson"/>
```

```
<Dir Include=".*" Exclude="" FilterType="regex"
    Path="/opt/mtwilson/share"/>
<Dir Include=".*" Exclude="" FilterType="regex"
    Path="/opt/mtwilson/share/openssl"/>
<Symlink Path="/opt/mtwilson/share/openssl/openssl.cnf"/>
<Dir Include=".*" Exclude="" FilterType="regex" Path="/opt/tbootxm"/>
<File Path="/opt/tbootxm/bin/configure_host.sh"/>
<File Path="/opt/tbootxm/bin/functions.sh"/>
<File Path="/opt/tbootxm/bin/generate_initrd.sh"/>
<File Path="/opt/tbootxm/bin/measure_host"/>
```

Tale file viene utilizzato dal Trust Agent per capire quali sono i file e le cartelle che devono essere misurati al boot della macchina. Quindi quando la macchina viene riavviata il TA creerà un file XML noto come "measurement.xml" sotto il percorso */var/log/trustagent* o sotto il percorso */opt/trustagent/logs* contenente le misure calcolate.

```
<?xml version="1.0"?>
<Measurements xmlns="mtwilson:trustdirector:measurements:1.2"
    DigestAlg="sha256">
        <Dir Path="/opt/mtwilson">e3b0c44...852b855</Dir>
        <Dir Path="/opt/mtwilson/share">e3b0c44...852b855</Dir>
        <Dir Path="/opt/mtwilson/share/openssl">e018660...f407e1d</Dir>
        <Symlink
            Path="/opt/mtwilson/share/openssl/openssl.cnf">96bd00b...4d25f54
        </Symlink>
        <Dir Path="/opt/tbootxm">e3b0c44...52b855</Dir>
        <File
            Path="/opt/tbootxm/bin/configure_host.sh">e08353a...aada5b2</File>
        <File Path="/opt/tbootxm/bin/functions.sh">d1d9853...cbf4655</File>
        <File
            Path="/opt/tbootxm/bin/generate_initrd.sh">6b5adb9...2b2d540</File>
        <File Path="/opt/tbootxm/bin/measure_host">67e274d...6a57d52</File>
```

3.5.1 Trust Policy e VM

Il processo che porta alla creazione di una Trust Policy per le macchine virtuali è molto simile a quello visto per i server fisici, ma viene introdotto un nuovo parametro che prende il nome di *Workload Integrity*. Questo parametro consente di selezionare la politica che verrà adottata quando sarà lanciata una nuova istanza della macchina virtuale e può assumere solamente due valori:

- **Hash only:** permette di attestare l'immagine della macchine virtuale, verificando che soddisfi quanto definito all'interno della Trust Policy, non compiendo ulteriori azioni;
- **Hash and Enforce:** anche in questo caso si verifica che l'immagine della VM soddisfi la Trust Policy, tuttavia non permette di "accendere" la VM se la Trust Policy non viene soddisfatta.

Se lo stato della macchina coinvolta nel processo di attestazione assume il valore Untrusted in seguito ad un reboot, anche l'istanza della macchina virtuale assumerà questo stato perché ovviamente l'istanza non potrà mai essere Trusted se l'host che la ospita non lo è.

Il processo che permette di lanciare una nuova istanza di una VM su un host è articolato nei seguenti passi (figura 3.3):

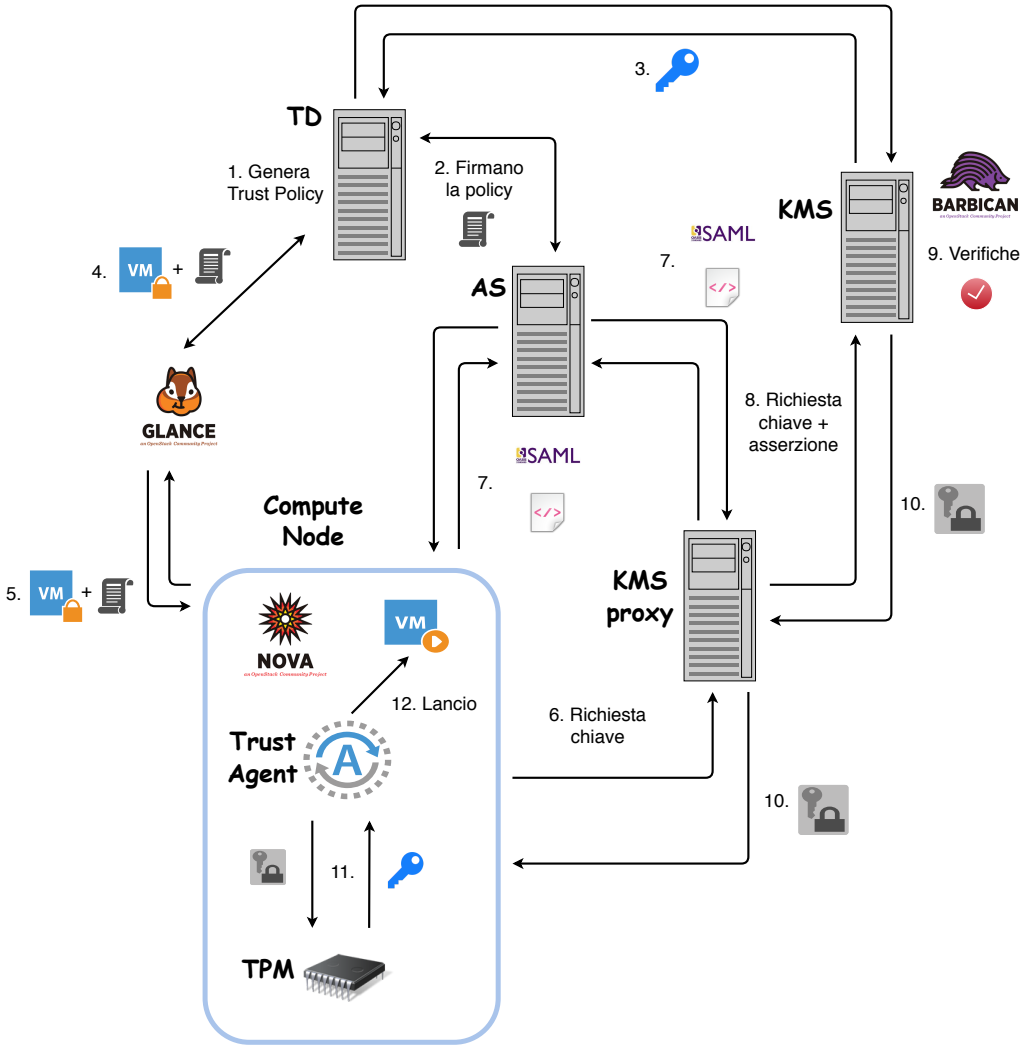

Figura 3.3. Esecuzione di una VM

1. Il Trust Director genera una Trust Policy;

2. La Trust Policy viene firmata dal Trust Director e dall'Attestation Server;

3. Il TD preleva dal KMS una chiave di cifratura e con questa cifra l'immagine. Il KMS per realizzare questa operazione comunica con OpenStack Barbican, un servizio che memorizza in modo sicuro, gestisce e fornisce dati come password, chiavi di cifratura e certificati X.509;

4. Il TD carica l'immagine cifrata e la Trust Policy associata su OpenStack Glance, ovvero su un servizio di memorizzazione delle immagini delle VM;

5. Quando attraverso OpenStack Horizon, un servizio per la visualizzazione di una dashboard, si vuole lanciare un'istanza di quella VM su un Compute Node tramite Nova, ovvero un servizio di OpenStack che permette di creare istanze di VM, quest'ultimo richiederà a Glance l'immagine della VM e la Trust Policy ad essa collegata;

6. Nova dovendo decifrare l'immagine appena ricevuta, contatta il Trust Agent risiedente sull'host che farà richiesta della chiave di decifratura al KMS proxy;

7. Il KMS proxy richiederà all'AS di effettuare una nuova attestazione per l'host che vuole lanciare la VM, ottenendo così l'asserzione SAML che certifica il risultato dell'attestazione;

8. Il KMS proxy prende la richiesta della chiave di decifratura e l'asserzione SAML e le inoltra al KMS;
9. Il KMS analizza l'asserzione ricevuta, si assicura che sia propriamente firmata dall'AS conosciuto e verifica che il risultato dell'attestazione dichiari che lo stato dell'host è Trusted, perché altrimenti la chiave di decifratura non verrà fornita;
10. Se tutti i requisiti vengono soddisfatti, il KMS recupera la chiave di decifratura da Barbican, la associa alla chiave pubblica della coppia AIK dell'host, la cifra usando quest'ultima e la invia all'host tramite il KMS proxy;
11. Il Trust Agent chiederà al TPM di recuperare la chiave di decifratura usando la chiave privata della coppia AIK. La chiave di decifratura non è mai memorizza sull'host, pertanto ogni qualvolta l'host effettuerà il reboot, dovrà richiederla di nuovo;
12. Infine l'immagine della VM viene decifrata e viene montato un disco virtuale cifrato dove viene collocata l'immagine. Inoltre prima di eseguire la VM, il Trust Agent verifica che la Trust Policy sia soddisfatta e a seconda del parametro Workload Integrity si deciderà se lanciare o meno la VM.

La verifica di una Trust Policy per una VM, oltre che al lancio dell'immagine, avviene quando l'istanza viene sospesa/ripresa, al reboot dell'istanza oppure allo spegnimento/accensione della stessa.

3.6 I processi di registrazione e di attestazione di un host

Per poter eseguire il processo di attestazione su una macchina è necessario che questa sia registrata presso l'Attestation Server. La registrazione include la configurazione di dettagli come il tipo di host (Linux, Windows ecc.), i parametri di connessione e come visto in precedenza la configurazione delle whitelist (BIOS MLE e VMM MLE).

Il processo di registrazione prevede i seguenti passi (figura 3.4):

1. L'AS chiede all'host di inviargli le sue informazioni attraverso un API REST;
2. L'AS genera un *nonce* (20 byte casuali) che salverà in un file che verrà utilizzato per le operazioni di verifica;
3. L'AS chiede al Trust Agent di inviargli il certificato (in formato PEM o DER) relativo alla chiave pubblica della coppia AIK;
4. Il TA genera il certificato per la sua chiave pubblica AIK se non lo ha ancora generato e lo invia all'AS;
5. L'AS memorizza il certificato ricevuto all'interno di un file e chiede all'host la quote inviandogli una richiesta in formato XML contente il nonce precedentemente generato, i PCR dei quali è richiesto il calcolo della quote e i banchi di PCR che devono essere selezionati. Di default OpenCIT richiede sempre che nella quote vengano inseriti tutti i 24 PCR sia per il banco SHA1 che per il banco SHA256;
6. L'AS una volta ricevuta la risposta, estrae la quote e la salva in un file. Quindi la verifica attraverso un file eseguibile che prende in input: il file contente il certificato della chiave pubblica AIK relativa all'host, il file con il nonce inviato al client e il file contenente la quote. L'eseguibile estrae dalla quote i valori dei PCR;
7. L'AS estrae il contenuto della risposta e salva i valori dei PCR, le misure TXT e le misure della Trust Policy in un database *PostgreSQL*. In particolare i valori della risposta che vengono presi in considerazione sono solamente quelli relativi ai PCR selezionati in fase di registrazione dall'utente. Se l'utente non ha selezionato alcun valore, di default vengono considerati i valori

Figura 3.4. Comunicazione TA-AS durante la registrazione

riguardanti il banco SHA-256 e i PCR 0,17,18 e 19 contenuti in tale banco, altrimenti vengono presi anche i valori degli altri PCR selezionati insieme alle misure utilizzate per l'operazione di estensione di quel PCR se presenti.

8. L'AS effettua la richiesta di una nuova quote all'host;
9. Il TA invia la quote all'AS;
10. L'AS effettua le verifiche del punto 5 e utilizza le nuove misure presenti nella risposta per confrontarle con i valori registrati in precedenza nel database;
11. Solamente in caso in cui non ci sia differenza tra i valori della prima quote, salvati nel DB, e i valori della seconda, l'host verrà registrato correttamente e il processo di registrazione avrà fine.

Il processo di attestazione è caratterizzato da un sottoinsieme del numero dei passi previsti per il processo di registrazione. Infatti i primi passi sono identici: recupero informazioni dall'host, creazione nonce, recupero certificato dell'host, ricezione e verifica della quote. Successivamente i valori della quote non vengono più registrati nel DB, come avveniva nella registrazione, ma vengono presi e confrontati con i valori di whitelist che riguardano quell'host precedentemente registrato. Verrà rilevata la presenza di qualsiasi valore che non coincide con il corrispondente valore di whitelist e questo naturalmente porterà a considera l'host in uno stato *Untrusted*.

L'AS ha un processo automatico in background che regolarmente controlla tutte quelle attestazioni che sono vicine alla scadenza fissata di 90 minuti. Quando un'attestazione si trova all'interno di un intervallo di 5 minuti dalla sua scadenza, l'AS inizia un nuovo processo di attestazione. L'attestazione può anche essere richiesta manualmente attraverso l'utilizzo dell'interfaccia grafica.

Capitolo 4

Progettazione della soluzione

La soluzione progettata si propone di utilizzare l'SDK Open CIT per effettuare l'attestazione remota delle macchine server caratterizzanti l'infrastruttura cloud. La scelta è stata condotta basandosi sui vantaggi che esso offre rispetto alle altre soluzioni analizzate in precedenza, come:

- la presenza di Intel TXT, utile ad attuare un processo di misurazione che coinvolga i componenti hardware e software a bassa livello come il BIOS, in modo da poter comprendere lo stato della macchina prima ancora che qualsiasi operazione di alto livello venga eseguita;
- la possibilità di poter attestare macchine che dispongono dell'ultima versione del TPM ovvero la 2.0;
- la possibilità di definire delle *Trust Policy* per la misurazione di file statici di configurazione della macchina sia per l'host fisico che per le VM che questo esegue.

Tuttavia Open CIT non offre la possibilità di effettuare un controllo di integrità a runtime, non permettendo quindi di monitorare l'ambiente di esecuzione della macchina per comprendere la presenza o meno di una compromissione in uno o più eseguibili. Questa possibilità è invece offerta da IMA che però da solo non è in grado di riportare lo stato delle misure raccolte ad un Verifier che possa effettuare una verifica. Pertanto si è deciso di integrare IMA nel progetto Open CIT in modo da sfruttare i vantaggi offerti da entrambe le tecnologie e garantire la definizione di uno stato di affidabilità dell'host che non si basi esclusivamente su informazioni raccolte a *boot time*.

Le modifiche effettuate nell'ambito del lavoro di tesi hanno coinvolto il Trust Agent e l'Attestation Server (figura 4.1). Sono stati analizzati e realizzati due differenti *scenari* per consentire la verifica delle misure IMA. Di seguito verranno analizzati gli scenari realizzati

4.1 Scenario: Whitelist statiche

Il primo scenario si pone come obiettivo quello di adottare la logica di verifica definita da Open CIT, ovvero quella che prevede la creazione di Whitelist statiche durante la fase di registrazione definendo dei valori utilizzati come riferimento per le successive attestazioni dell'host. Quindi in tal caso le misure IMA raccolte dall'host vengono confrontate con dei valori ottenuti in ambiente fidato con lo scopo di comprendere se le misure siano state manipolate o meno. Il problema principale della definizione di Whitelist statiche è dato dal fatto che ogni qualvolta si vuole eseguire nuovo software all'interno dell'host, dovrà essere definita una nuova Whitelist. Infatti se non si definisse una nuova Whitelist la presenza del nuovo software porterebbe a considerare l'host in uno stato *untrusted*, dal momento che quel software non era previsto nella Whitelist definita in precedenza. Quindi questo approccio è da utilizzare solamente in quegli ambienti in cui non si prevede un installazione frequente di nuovo software, perché altrimenti in un ambiente in continua evoluzione dal punto di vista software si dovrebbe sempre creare una nuova Whitelist e di conseguenza registrare nuovamente l'host.

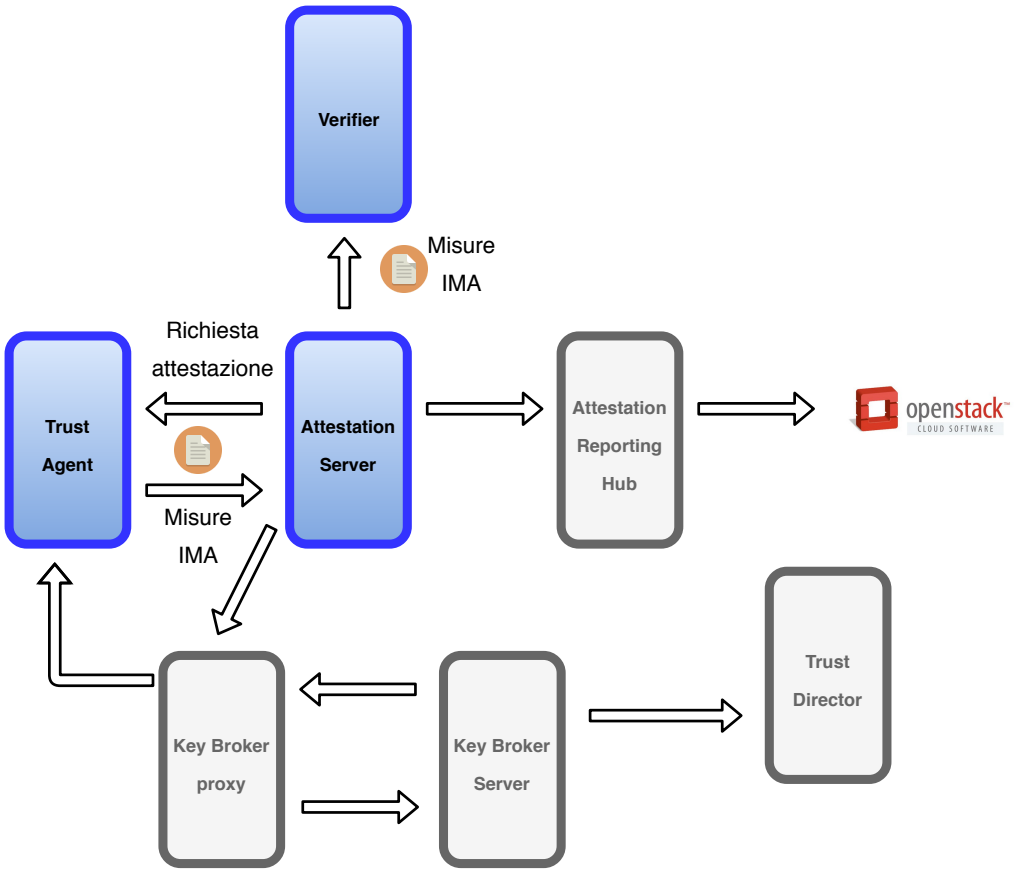

Figura 4.1. Componenti di Open CIT modificati

4.2 Scenario: uso del Verifier esterno

Il secondo approccio è invece più dinamico in quanto non vengono definite a priori delle Whitelist statiche ma le misure IMA vengono verificate tramite un Verifier esterno. Il Verifier è un'applicazione Web basata sul progetto *SECURED* [53] che consente di integrare più framework di attestazione grazie all'utilizzo di funzionalità note come *driver*. Ogni driver interagisce con il proprio framework di attestazione consentendo di recuperare da esso le informazioni di integrità utili per la fase di verifica (figura 4.2). Un driver viene progettato per consentire l'utilizzo di tre principali funzioni che sono esposte sotto forma di API REST:

funzione di stato: è la funzione che deve essere chiamata prima di qualunque altra funzione del driver, per saper se il framework di attestazione è attivo ed è pronto a ricevere richieste da parte del Verifier;

funzione di registrazione: il suo compito è quello di registrare l'host sia nel framework di attestazione invocando le funzioni necessarie a registrare l'host in quello specifico framework, sia all'interno del DB del Verifier. Ogni host che viene registrato sul Verifier è caratterizzato da tutte le informazioni necessarie per eseguire l'attestazione dell'host all'interno dello specifico framework a cui quell'host fa riferimento;

funzione di attestazione: è invocata ogni qualvolta si vuole effettuare l'attestazione di un host. Tale funzione si basa sull'utilizzo delle informazioni raccolte in fase di registrazione per poter consentire di attestare correttamente l'host. Il suo scopo è quello di utilizzare il framework di attestazione per recuperare le misure IMA dall'host e trasferirle alla logica di verifica del Verifier.

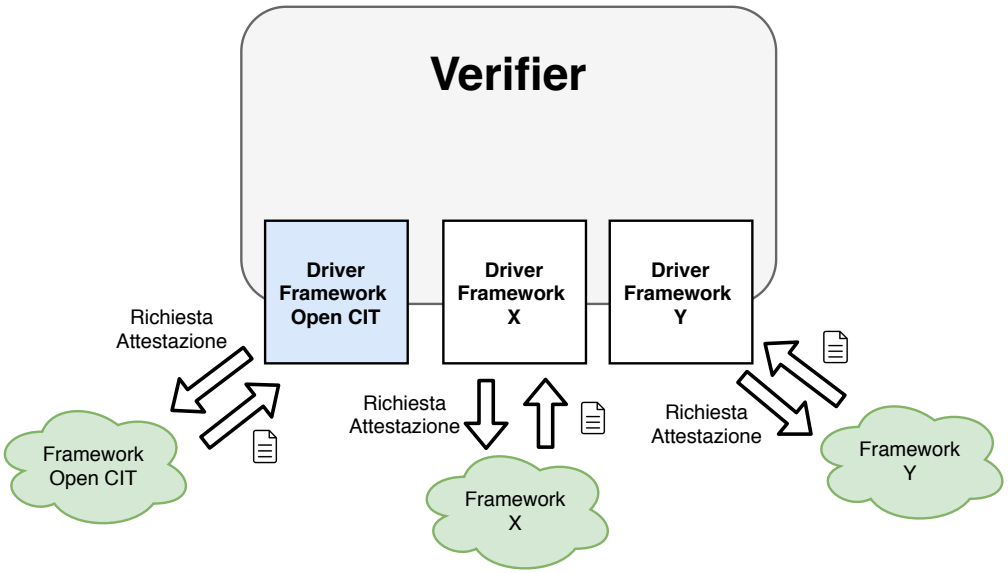

Figura 4.2. Comunicazione tra framework e Verifier

La logica di verifica del Verifier prevede l'utilizzo di un DB costantemente aggiornato con le informazioni di integrità riguardanti tutti i possibili pacchetti software previsti per il sistema operativo dell'host da attestare e scaricati dai vari *repository* riguardanti il sistema operativo stesso. In questo modo non c'è bisogno di definire una nuova Whitelist ogni qualvolta si aggiunge nuovo software e inoltre tale software potrà comunque essere verificato. Il limite di questo secondo approccio è che attualmente il Verifier permette di attestare solamente misure provenienti da un host *CentOS*.

4.3 Trust Agent e IMA

Come è stato detto in precedenza, Open CIT non prevede l'inserimento di alcuna informazione che permetta di comprendere lo stato della macchina a *run time*. Il componente di Open CIT che si occupa di raccogliere le informazioni dall'host ed inviarle all'Attestation Server è il Trust Agent. Questo componente è stato modificato per consentire di valutare l'integrità del software in esecuzione sulla macchina, integrando le misure IMA nella risposta inviata all'AS.

Il primo passo è stato quello di definire una policy per IMA. IMA offre la possibilità di utilizzare delle policy predefinite o delle policy *custom* per fornire delle indicazioni circa quello che deve essere misurato a *run time* sull'host. La policy che è stata scelta per la soluzione sviluppata è la *Execution Policy*. Questo perché la policy di default che viene applicata è la *Standard Policy* che permette la misurazione di un numero di elementi molto elevato dal momento che include anche tutti i file di configurazione letti durante il bootstrap e quelli letti da ogni servizio finché questi operano come root. La *Execution Policy*(figura 4.3) invece misura soltanto gli eseguibili, ovvero i binari eseguiti tramite *execve()* e le librerie condivise ad essi correlate, caricate tramite *mmap()*, consentendo di limitare il numero totale di eventi misurati.

Il TCG propone una soluzione al problema dell'attestazione remota definendo delle specifiche in cui si prevede di inviare al *Verifier* le informazioni di integrità raccolte sul *Requestor* tramite l' *Integrity Report* [40]. Tuttavia gli sviluppatori di Open CIT nell'inviare la *Quote* contenente i valori dei PCR e le varie misure, non seguono le indicazioni fornite dal TCG e non creano alcun *Integrity Report* ma si affidano ad un file con un formato differente. Questo file contiene tutti i valori dei PCR e le misure a loro correlate comprendendo anche quelle misure legate alla Trust Policy. Con lo scopo di essere conformi alle specifiche definite dal TCG si è cercato di modificare il Trust Agent per inserire l'Integrity Report, tuttavia questo non è stato possibile. Il TCG infatti

```
# Measure all files mapped in memory as executable
measure func=FILE_MMAP mask=MAY_EXEC
# Measure all files executed by the execve() syscall
measure func=BPRM_CHECK mask=MAY_EXEC
```

Figura 4.3. Execution Policy

```
<?xml version="1.0"?>
<Measurements xmlns="mtwilson:trustdirector:measurements:1.2"
    DigestAlg="sha256">
<Dir Path="/opt/mtwilson">e3b0c44298fc1c149afbf4c8996fb92427a
e41e4649b934ca495991b7852b855</Dir>
<File Path="/opt/tbootxm/bin/configure_host.sh">e08353a519caa66d6f2bd7e54
4ea24e5992bf90fb912907d1f2099e21aada5b2</File>
<File Path="/opt/tbootxm/bin/functions.sh">d1d98531fd130e616e38219473e
d0d08d17502cdffaa955ab9186859acbf4655</File>
.....................................
</Measurements>
```

Figura 4.4. File XML delle misure della Trust Policy

non ha ancora definito delle specifiche che consentissero la creazione di un report per supportare le informazioni riguardanti la versione 2.0 del TPM. Attualmente è possibile definire solamente un *Integrity Report* per la versione 1.2.

Alla luce di queste considerazioni si è deciso di adottare per le misure IMA la stessa logica che viene seguita per le misure della Trust Policy, ovvero quella di inserire le misure sotto forma di un file XML contenuto nella risposta (figura 4.4). Inoltre per consentire una maggiore flessibilità sul modo in cui si vuole attestare l'host è stata modificata la richiesta della quote da parte dell'AS per informare il Trust Agent se dovrà o meno includere le misure IMA nella risposta. In questo modo sarà possibile definire per quali host devono essere verificate le misure IMA e per quali invece deve essere seguito l'originale processo di attestazione previsto da Open CIT. Quindi il processo che ha luogo sul Trust Agent una volta che l'Attestation Server invia la richiesta di attestazione al TA è il seguente (figura 4.5):

1. Il TA effettua il parsing della richiesta prelevando informazioni come: PCR per cui è richiesta la verifica di integrità, i relativi banchi di PCR da selezionare ed un valore che consente di capire se è stata o meno richiesta un'attestazione con IMA;

2. Vengono prelevate da un file XML memorizzato sulla macchina le misure relative ai file definiti per la Trust Policy e inserite nella risposta;

3. Se è stata richiesta attestazione con IMA, vengono prelevate le misure IMA e inserite all'interno della risposta;

4. Si contatta il TPM per richiedere i valori dei PCR e il calcolo della Quote;

5. Infine viene inviata la risposta all'AS.

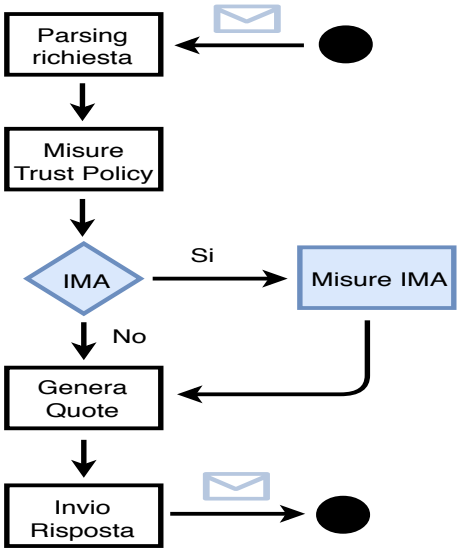

Figura 4.5. Processo sul Trust Agent

4.4 Attestation Server e IMA

Il componente utilizzato da Open CIT per effettuare l'operazione di attestazione dei nodi all'interno dell'infrastruttura cloud è l'Attestation Server. Poiché esso è il responsabile di effettuare l'analisi e la verifica delle informazioni di integrità provenienti dagli host, è stato modificato per integrare la logica di verifica delle misure IMA, seguendo due differenti scenari.

4.4.1 Scenario: Whitelist statiche

Si è potuto osservare come la logica di verifica definita da Open CIT si basi sulla creazione di Whitelist statiche al momento della registrazione dell'host. Utilizzando la prima *Quote* che viene richiesta all'host al momento della registrazione, vengono prelevati tutti i valori riguardanti i PCR e le misure utilizzate per ottenere i vari aggregati, comprese quelle misure che fanno parte della Trust Policy. L'Attestation Server estrae questi valori, li memorizza in un Database (DB) e li utilizza per le future attestazioni dello specifico host. L'obiettivo è stato quello di replicare questo comportamento anche per le misure IMA che, attraverso le modifiche previste dalla soluzione proposta, vengono incluse nella risposta inviata dal Trust Agent. Il processo di verifica che caratterizza le misure IMA nel primo scenario interessa due operazioni:

Calcolo aggregato: le misure IMA vengono estese nel TPM del Trust Agent all'interno del PCR 10. Quindi è importante verificare che l'insieme delle misure ricevute dal Trust Agent producano un aggregato che coincida con il valore del PCR 10 contenuto nella quote. Infatti è possibile che le misure vengano manipolate sia mentre si trovano ancora sul Trust Agent sia durante la trasmissione. Non è possibile invece manomettere il valore del PCR 10 contenuto all'interno della quote. Pertanto questo valore può essere utilizzato per effettuare un confronto con l'aggregato che l'AS calcola utilizzando la lista di misure IMA. In questo modo se la lista è stata manipolata l'aggregato non coinciderà con il valore contenuto nella quote e si potrà dedurre che l'host si trova in uno stato non affidabile. C'è inoltre una differenza rispetto alle verifiche condotte dall'AS sugli altri PCR. L'AS così come previsto in Open CIT registra nel DB tutti i valori dei PCR contenuti nella prima quote durante la fase di registrazione. Nel caso del PCR 10 questo comportamento deve essere evitato. Infatti anche se i componenti misurati da IMA sono sempre i medesimi e non vengono modificati, ogni qualvolta viene fatto il boot della macchina potrebbe cambiare l'ordine con il quale tali componenti vengono eseguiti e di conseguenza cambia anche l'aggregato che viene calcolato.

Questo farà si che il valore ottenuto dal calcolo dell'aggregato non coinciderà con il valore che verrebbe registrato nel DB anche se non è stata modificata alcuna misura. Si può quindi concludere che il valore del PCR 10 non viene memorizzato nel DB.

Confronto con il DB: l'altra verifica che viene effettuata dall'AS riguarda le singole misure IMA. Ogni misura viene memorizzata all'interno del DB e il suo valore utilizzato per le successive verifiche. Questo inoltre permette anche di verificare che sull'host non siano in esecuzione altri componenti non previsti oppure che qualcuno dei componenti che dovevano essere eseguiti in realtà non venga eseguito e/o misurato. In questo modo è possibile evidenziare quale componente misurato da IMA ha reso lo stato del sistema non affidabile.

Il processo che ha luogo sull'AS in fase di registrazione dell'host in entrambi gli scenari è il seguente (figura 4.6):

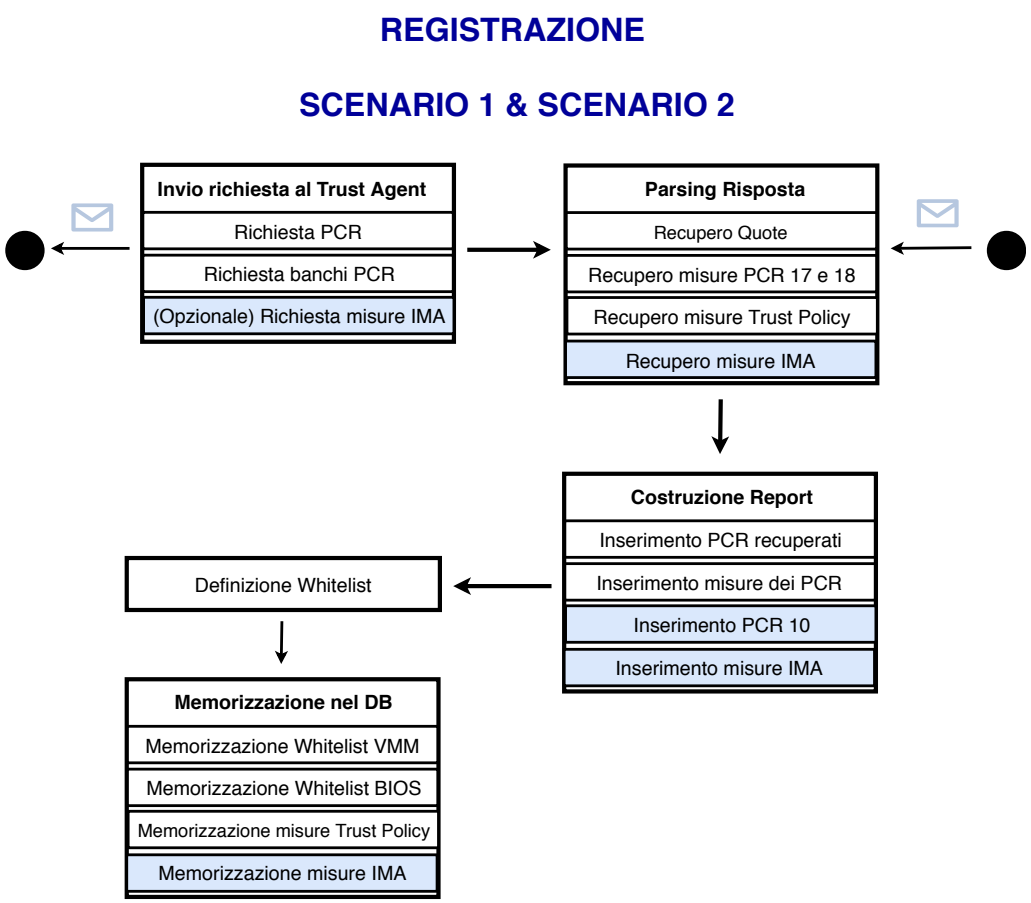

Figura 4.6. Processo di registrazione sull'Attestation Server

1. L'AS richiede al TA l'invio della *quote*, richiedendo se necessario l'inserimento delle misure IMA;
2. L'AS riceve la risposta dal TA ed effettua il parsing;
3. L'AS preleva i valori dei PCR e le misure relative ad ogni PCR comprese le misure della Trust Policy e le misure IMA se presenti e crea un report in formato XML, immettendo in esso solo quei PCR dei quali è stata richiesta la verifica;
4. Il report così creato viene utilizzato per la creazione delle Whitelist VMM e BIOS;
5. Le Whitelist vengono memorizzate all'interno del DB insieme alle misure riguardanti i PCR comprese le misure della Trust Policy e le misure IMA se presenti;

La fase di attestazione è differente a seconda dello scenario in cui ci si trova. Nel primo scenario infatti viene effettuata una fase di verifica delle misure IMA che realizza un confronto tra i valori appena ricevuti dal Trust Agent e i valori precedentemente memorizzati in fase di registrazione. Per un host per cui è prevista attestazione con IMA, il processo di attestazione svolto nel primo scenario è il seguente (4.7):

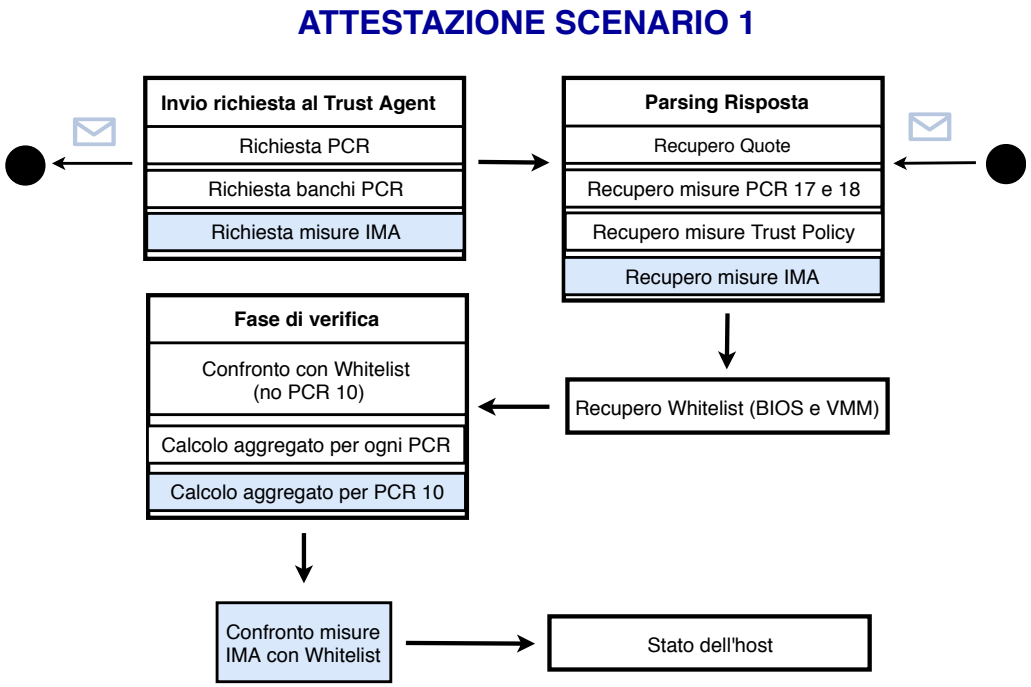

Figura 4.7. Processo di attestazione scenario 1

1. L'AS richiede al TA l'invio della quote comprendente le misure IMA;
2. L'AS riceve la risposta dal TA ed esegue il parsing;
3. Utilizzando le informazioni dell'host da attestare vengono recuperate dal DB le Whitelist VMM e BIOS registrate in precedenza;
4. Inizia la fase di verifica: per tutti i PCR, compreso il PCR 10, viene effettuata la fase di verifica prevista da Open CIT ovvero quella basata sulle Whitelist. La verifica riguardante il calcolo dell'aggregato viene effettuata anche per il PCR 10 dal momento che si vuole verificare che la lista delle misure non sia stata manipolata;
5. L'AS ottiene il risultato della verifica e definisce lo stato dell'host.

4.4.2 Scenario: uso del Verifier esterno

Il secondo scenario previsto per l'AS modifica la logica di verifica delle misure IMA, spostando parte di essa al di fuori dell'AS ed affidandola ad un Verifier esterno. In particolare la verifica dell'aggregato viene sempre effettuata dall'AS che avrà così modo di comunicare al Verifier esterno se la lista di misure è stata modificata durante la trasmissione, mentre non viene realizzato più alcun confronto con il DB. La registrazione delle misure IMA all'interno del DB può anche essere evitata dal momento che l'AS in questo scenario non le verifica, tuttavia è stata compiuta una scelta legata al fatto che l'Attestation Server è un'applicazione web che permette ad un utente di interagire con un'interfaccia grafica. Tale interfaccia permette all'utente di visualizzare i risultati dei processi di registrazione e attestazione, così come anche i valori dei PCR e le misure raccolte dall'host attestato. Per mantenere la compatibilità con Open CIT anche in questo scenario e non

alterare la sua logica di verifica e visualizzazione dei risultati consentendo ad un utente che fa uso dell'interfaccia grafica di avere la possibilità di visualizzare le misure ricavate dall'host attestato, si è deciso di memorizzare le misure IMA anche in fase di attestazione. Questo salvataggio è reso necessario dal fatto che la logica di Open CIT permette di visualizzare graficamente solamente ciò che viene salvato nel DB. Pertanto consentendo di aggiornare il DB ad ogni attestazione dell'host, memorizzando le nuove misure IMA ricevute, l'utente avrà comunque l'opportunità di visualizzare delle informazioni che rispecchiano l'ambiente di esecuzione attuale dell'host. In figura 4.8 viene mostrato il processo che viene attuato nella verifica delle misure IMA nel secondo scenario:

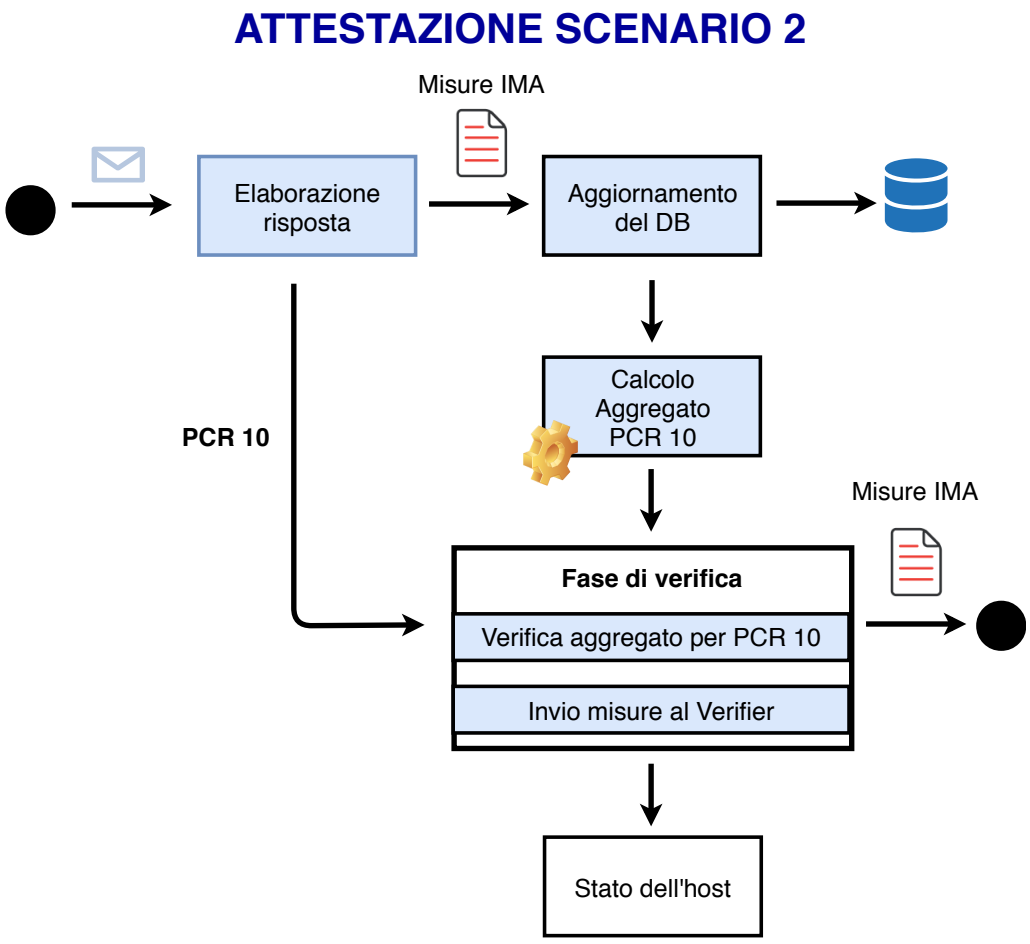

Figura 4.8. Processo attestazione scenario 2

1. l'AS riceve la risposta dal TA che viene elaborata eseguendo il parsing per poter recuperare le informazioni di integrità che devono essere verificate;

2. le misure IMA recuperate dalla risposta vengono utilizzate per aggiornare il DB;

3. viene effettuato il calcolo dell'aggregato sfruttando le misure IMA presenti nella risposta;

4. inizia la fase di verifica che prevede il calcolo dell'aggregato sfruttando il valore del PCR 10 contenuto nella quote e inviando le misure IMA la Verifier esterno;

5. Il risultato prodotto in seguito alla verifica dell'aggregato viene utilizzato per definire lo stato dell'host.

4.5 Il driver di attestazione per Open CIT

Il driver è una funzionalità del Verifier esterno prevista solamente per il secondo scenario ed ha come scopo quello di gestire la comunicazione con la logica di verifica delle misure IMA. Il Verifier esterno utilizza un DB costantemente aggiornato che contiene tutte le informazioni relative ai pacchetti scaricati dai repository di una distribuzione Linux (ad esempio *CentOS*) come i digest degli eseguibili e librerie contenute nei pacchetti. Il Verifier esterno riceve la lista di misure IMA ed effettua un controllo su di esse con lo scopo di definire un *livello di trust*. Esistono quattro differenti livelli di trust:

L1: è il caso in cui esiste almeno una misura IMA non conosciuta dal Verifier;

L2: tutte le misure sono conosciute dal Verifier, ma è presente almeno una misura per cui esiste un aggiornamento di sicurezza;

L3: tutte le misure sono conosciute dal Verifier e non hanno aggiornamenti di sicurezza. Tuttavia esiste almeno una misura che ha un *functional bug*;

L4: tutte le misure sono conosciute dal Verifier e non hanno alcuna vulnerabilità di sicurezza e *functional bug*.

Quindi il Verifier per ogni host di cui attesta le misure IMA è in grado di definire se l'host rispetta o meno un determinato *livello di trust*. In particolare il processo messo in atto dal Driver è il seguente (figura 4.9):

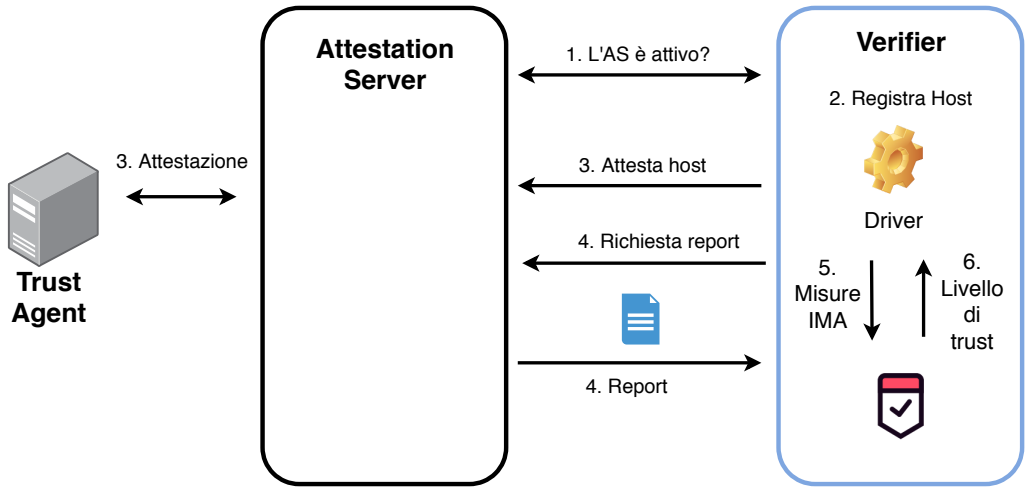

Figura 4.9. Processo del Driver

1. Il driver controlla se l'AS è attivo e pronto a ricevere richieste;
2. Ricevuta la risposta dall'AS, il driver registra l'host da attestare presso il Verifier esterno, se non è già stato registrato in precedenza;
3. Il driver invia una richiesta all'AS per chiedere di attestare l'host registrato in precedenza;
4. Il driver verifica che l'attestazione sia andata a buon fine e chiede all'AS di inviargli il report contente tutte le informazioni riguardanti l'host, tra le quali le misure IMA;
5. Il driver effettua il parsing della risposta ottenuta, preleva le misure IMA e le trasferisce alla logica di verifica;
6. Il driver ottiene il risultato dal processo di verifica indicante il livello di trust dell'host.

Capitolo 5

Implementazione

L'implementazione delle modifiche effettuate al codice di Open CIT ha coinvolto il Trust Agent per consentirgli di prelevare ed inviare le misure IMA, e l'Attestation Server per realizzare la logica di verifica nel caso del primo scenario. Successivamente per permettere l'implementazione di un secondo scenario sono state apportate delle modifiche al Trust Agent e all'Attestation Server, ed è stato creato un Driver per consentire la comunicazione con il Verifier esterno per la verifica delle misure IMA.

5.1 Modifiche al Trust Agent

L'obiettivo previsto per le modifiche al Trust Agent è quello di permettere la cattura e l'invio delle misure IMA. Le misure IMA vengono registrate all'interno di un file chiamato `ascii_runtime_measurements` sotto la directory `/sys/kernel/security/ima/`. La struttura del file è quella indicata di seguito.

```
PCR          template-hash              filedata-hash                   filename-hint
10 91f34b5c671d73504b274a919661cf80dab1e127 ima-ng sha1:1801e1be3e65ef1eaa5c16617bec8f1274eaf6b3 boot_aggregate
10 8b1683287f61f96e5448f40bdef6df32be86486a ima-ng sha256:efdd249edec97caf9328a4a01baa99b7d660d1afc2e118b69137081c9b689954 /init
10 ed893b1a0bc54ea5cd57014ca0a0f087ce71e4af ima-ng sha256:1fd312aa6e6417a4d8dcdb2693693c81892b3db1a6a449dec8e64e4736a6a524 /usr/lib64/ld-2.16.so
10 9051e8eb6a07a2b10298f4dc2342671854ca432b ima-ng sha256:3d3553312ab91bb95ae7a1620fedcc69793296bdae4e987abc5f8b121efd84b8 /etc/ld.so.cache
```

Tale file indica le seguenti informazioni:

- **PCR:** indica il PCR sulla quale sono state estese le misure;
- **filedata-hash:** contiene una stringa indicante l'algoritmo di hash utilizzato seguito dal digest del file;

  filename-hint: è il nome del file;
- **template-hash:** è un digest calcolato in un modo che dipende dal tipo di *template* che viene adottato. Esistono tre principali template: `ima`, `ima-ng` e `ima-sig`, ma è possibile anche crearne dei nuovi. In questo lavoro di tesi si è scelto di utilizzare il template `ima-ng`. Tale scelta implica che il template-hash venga calcolato attraverso la seguente formula

  $$template{-}hash := H(lunghezza\_totale||nome\_algoritmo||filedata{-}hash||lunghezza\_nome\_file||nome\_file)$$

  dove

  - `lunghezza_totale` è data dalla somma tra la lunghezza del digest del file e la lunghezza della stringa che costituisce il nome dell'algoritmo;
  - `filedata-hash` costituisce il filedata-hash presente nel file ma senza la stringa contenente il nome dell'algoritmo di hash.

```
POST https://compute1:1443/v2/tpm/quote

Content-Type: application/xml

<tpm_quote_request>
        <nonce>tHgfRQED1+pYgEZpq3dZC9ONmBCZKdx10LErTZs1k/k=</nonce>
        <pcrs>0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23</pcrs>
        <pcr_banks>SHA1 SHA256</pcr_banks>
        <ima>true</ima>
</tpm_quote_request>

Risposta:
HTTP/1.1 200 OK
Content-Type: application/xml

<tpm_quote_response>
        <timestamp>1491041233964</timestamp>
        <client_ip>192.168.1.105</client_ip>
        <error_code> 0</erro_code>
        <error_message>OK</error_message>
        <aik>MIIC....U0qt</aik>
        <quote>AAMB.....riJR</quote>
        <event_log>PG1l....Cg==</event_log>
        <tcb_measurement>.........</tcb_measurement>
        <ima_measurement>.........</ima_measurement>
        <selected_pcr_banks>SHA1 SHA256</selected_pcr_banks>
        <is_tag_provisioned>false</is_tag_provisioned>
        <asset_tag></asset_tag>
</tpm_quote_response>
```

Figura 5.1. Richiesta POST alla risorsa `tpm/quote`

Il Trust Agent viene modificato per poter prelevare da questo file le informazioni necessarie affinché l'Attestation Server possa svolgere le sue operazioni di verifica. In particolare da tale file si preleva il `template-hash` e il `filename-hint` di ogni misura. Non viene considerato il `filedata-hash` in quanto la ricostruzione dell'aggregato che viene confrontato con il valore del PCR 10 avviene fornendo in input il template-hash all'operazione di estensione. Nel secondo scenario invece il Trust Agent viene modificato per prelevare il `filedata-hash` e non il template-hash. Questo perché le misure IMA di cui ha bisogno il Verifier esterno per effettuare le sue operazioni di verifica coinvolgono solamente il `filedata-hash`.

Come detto in precedenza, per poter essere conformi al modello di attestazione previsto da Open CIT e quindi permette un'attestazione che non coinvolga le misure IMA, è necessario comunicare al Trust Agent se deve o meno includere le misure IMA nella risposta inviata all'AS. A tal scopo è stata modificata la classe `TpmQuoteRequest` rappresentante la richiesta, in modo da avere un valore *booleano* che indichi se è stata richiesta o meno un attestazione con IMA. Questo valore è configurabile attraverso un opportuno metodo di *set* e restituito da un metodo di *get*. Anche la risposta che il Trust Agent invia all'AS deve essere modificata per permettere l'invio delle misure IMA. A tal scopo nella classe `TpmQuoteResponse` che rappresenta la risposta inviata all'AS è stata aggiunta una stringa contenente un file XML che ingloba tutte le misure IMA raccolte. La richiesta proveniente dall'AS viene inviata attraverso una POST alla risorsa `/tpm/quote`(figura 5.1).

La risposta, il cui contenuto dei campi `aik`, `quote`, `event_log`, `tcb_measurement` e `ima_measurement` è stato volutamente compresso per motivi pratici, include i seguenti campi:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<IMA_Measurements DigestAlg="sha1" xmlns="mtwilson:ima:measurements:1.0">
    <File Path="boot_aggregate">4c4472872f65a6b86aeb9d0325cb135d460802e4</File>
    <File Path="/init">a68711aa498cf73bf3ed7b12e4e1f38c6f9b39f2</File>
    <File Path="/bin/sh">790ff4fe72889b071a0f7585112710be6d0084fe</File>
    ........................
</IMA_Measurements>
```

Figura 5.2. Struttura del file XML contente le misure IMA

timestamp: indica l'istante di tempo in cui è stata creata la quote;

client_ip: indica l'indirizzo ip dell'host dal quale proviene la quote;

error_code: indica il codice dell'errore nel caso in cui si verifichi un errore. Se non è presente alcun errore, il codice che verrà mostrato sarà pari al valore "0";

error_message: contiene il valore "OK" nel caso in cui non si sia verificato nessun errore, mentre in caso contrario mostrerà il relativo messaggio;

aik: è il certificato della chiave pubblica della coppia AIK che l'host invia in ogni sua risposta;

quote: contiene il risultato dell'esecuzione del comando `tpm2_quote` (una delle funzioni fornita dal package `tpm2-tools`), codificato in *Base64*. In particolare la quote contiene i dati firmati, quindi la lista di tutti i PCR e il nonce inviato dall'AS, e la firma;

event_log: costituisce l'insieme delle misure calcolate da TXT. Ovvero sono le misure presenti nel file *measureLog.xml* di cui si è detto in precedenza, codificate in *Base64*;

tcb_measurement: sono le misure riguardanti la Trust Policy che vengono inviate sotto forma di un file XML;

ima_measurement: sono le misure IMA inglobate in un file XML;

is_tag_provisioned: indica se la risposta è provvista o meno di *Asset Tag*. Gli Asset Tag sono delle coppie chiave-valore che contengono delle informazioni riguardanti il carico di lavoro sul server e che possono essere utili all'amministratore di sistema;

asset_tag: costituiscono gli Asset Tag richiesti per quell'host.

Per poter recuperare le misure IMA, innanzitutto all'interno della classe `Tpm` che espone la risorsa REST, viene inserito un controllo per capire se è stata richiesta o meno attestazione con IMA. Se così fosse viene utilizzata la classe `RetrieveImaMeasurement` il cui compito è quello di creare il file XML contenente le misure IMA e fornirlo alla risposta. Per realizzare questo obiettivo la classe `RetrieveImaMeasurement` chiama la classe `IMAReader` che preleva le misure dal file *ascii_runtime_measurements* e costruisce l'oggetto `IMAMeasurements`. La classe `IMAMeasurements` viene creata utilizzando il framework *Java Architecture for XML Binding* (JAXB) a partire da un file di tipo *XSD* appositamente creato. Il file XML contenente le misure IMA ha la struttura seguente (figura 5.2):

5.2 Modifiche all'Attestation Server

5.2.1 Problema sul PCR 17

Nel realizzare le modifiche all'AS si è dovuto innanzitutto risolvere un problema che rendeva l'host *untrusted* perfino durante la fase di registrazione e che riguardava il PCR 17. Questo accadeva per

```
TpmQuoteResponse tpmQuoteResponse = client.getTpmQuote(nonce, new int[]{0, 1,
    2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
    22, 23}, host.PcrBanks, isIMA());
```

Figura 5.3. Richiesta della quote al Trust Agent

un'errata costruzione del file *measureLog.xml* sotto la directory */opt/trustagent/var/*. Questo è un file contenente le misure relative al PCR 17 che vengono utilizzate per calcolare l'aggregato che sarà confrontato con il valore contenuto nella quote. Questo file viene inviato all'AS e viene popolato dallo script *module_analysis_da.sh* sotto la directory */opt/trustagent/bin* con le misure raccolte dall'esecuzione del comando `txt-stat`. Quello che si poteva notare è che durante la costruzione del file veniva omessa l'informazione riguardante il tipo di banco (SHA1 o SHA256) alla quale la misura apparteneva oppure poteva accadere che la misura era di tipo SHA1 ma veniva indicato un banco SHA256 e viceversa. Questo creava dei problemi al momento della verifica sull'AS che non era in grado di effettuare correttamente il parsing di tale file. Per risolvere tale problema si è quindi dovuto agire sul file *module_analysis_da.sh* affinché generasse correttamente il file *measureLog.xml*.

Nonostante il problema riguardante la creazione del file fosse stato risolto l'host continuava ad essere *untrusted* a causa del PCR 17. Questo accadeva perché, come detto in precedenza, il file `measureLog.xml` viene riempito con le misure riguardanti il PCR 17 a seguito dell'esecuzione del comando `txt-stat`, ma tale comando non era in grado di catturare tutte le misure necessarie a ricalcolare l'aggregato. Si è quindi provato ad installare il software di Open CIT su una macchina differente notando che non si verificava alcun problema di questo tipo e che quindi l'host in fase di registrazione risultava correttamente in uno stato *trusted*. Quindi si è potuto concludere che il TPM di cui è dotata la macchina di test abbia qualche problema nella raccolta delle misure riguardanti il PCR 17. Pertanto si è deciso di modificare il codice dell'AS in modo da eludere il controllo effettuato sul PCR 17 e far si che l'host in fase di registrazione risultasse *trusted*.

5.2.2 Interazione con il Trust Agent

Per interagire con il TA viene utilizzata la classe `TAHelper` la quale per poter inviare la richiesta della quote al TA utilizza il metodo `getQuoteInformationForHost`. Questa classe è stata modificata per consentire di includere un valore *booleano*, indicante un'attestazione con IMA, nella richiesta al TA.

Come si può vedere dalla figura 5.3, viene invocato il metodo `getTpmQuote` dell'oggetto client appartenente alla classe `TrustAgentClient`. Tale metodo è responsabile di eseguire una richiesta POST alla risorsa *quote* e restituisce un oggetto di tipo `TpmQuoteResponse`. La risposta così ricevuta viene elaborata per costruire un oggetto di tipo `PcrManifest`. Tale oggetto consente di memorizzare le informazioni ricevute dal TA, ovvero i valori dei PCR contenuti nella quote e le misure riguardanti ciascun PCR. Pertanto tale classe è stata modificata per poter accogliere al suo interno anche le misure IMA.

Il processo descritto sinora che porta ad interagire con il TA e recuperare la quote avviene attraverso la creazione di un *thread* tramite la classe privata `HostAttReport`. Lo scopo di questa classe è quello di costruire un report in formato XML che viene utilizzato per memorizzare solamente quei valori che dovranno essere verificati. Ad esempio se l'utente tramite l'interfaccia grafica ha selezionato solamente un insieme di PCR tale report XML dovrà contenere esclusivamente i valori riguardanti quei PCR selezionati e le misure che li caratterizzano. Quando l'AS richiede al TA di inviare la quote, viene indicato di inserire al suo interno tutti i PCR. L'utente tramite l'interfaccia grafica potrebbe però aver selezionato solamente un gruppo di PCR e non vorrebbe verificare e analizzare anche i PCR che non gli interessano. Quindi la costruzione del report XML è utile per agire come un filtro, facendo passare solamente quei valori dei PCR che sono stati selezionati dall'utente con le relative misure. Tuttavia ci si è resi conto che la costruzione del report si basa

su un'altro criterio di selezione dei PCR. Questo criterio consente di inserire nel report solamente quei PCR che sono stati calcolati utilizzando l'algoritmo di hash più robusto supportato dall'host da attestare. Ad esempio la macchina di test utilizzata è in grado di supportare solamente banchi SHA 1 e SHA 256. Quindi secondo il criterio di selezione definito, all'interno del report XML vengono inseriti solamente i PCR selezionati dall'utente e che sono stati calcolati attraverso l'algoritmo di hash SHA 256, tutti i PCR di tipo SHA 1 vengono esclusi. Dal momento che il PCR 10 che è quello adoperato da IMA appartiene al banco SHA 1, secondo quanto detto, viene escluso e non inserito nel report. Quindi per permettere l'inserimento del valore SHA 1 del PCR 10 è stato necessario apportare una modifica nel metodo `getHostAttestationReport` della classe `TAHelper` per poter eludere il controllo che non considerava i PCR appartenenti al banco SHA 1.

5.2.3 Memorizzare le misure IMA nel DB

I valori provenienti dal report XML di cui si è parlato in precedenza, vengono utilizzati per la definizione delle Whitelist (BIOS e VMM) in fase di registrazione dell'host. La fase di verifica definita da Open CIT prevede infatti che i valori provenienti dall'host vengano confrontanti con dei valori ricavati in ambiente fidato. Per rendere necessaria quest'operazione di verifica bisogna utilizzare un DB per le memorizzazione di tali dati. Open CIT prevede l'utilizzo di un DB relazionale come *PostgreSQL*. All'interno di tale DB, modificando uno script eseguito in fase di installazione dell'AS, è stato fatto in modo che venisse generata una tabella chiamata `mw_ima_measurement_xml` 5.1 con lo scopo di contenere le misure IMA recuperate in fase di registrazione ed utilizzate successivamente per la fase di attestazione.

id	*mleId*	*content*
cc29544c-133a-442b-9a3c-ef41dd2c1696	150	<?xml version=”1.0”....

Tabella 5.1. Tabella delle misure IMA.

Le colonne all'interno della tabella indicano rispettivamente l'id utilizzato per identificare univocamente il file XML contenente le misure; l'id della Whitelist a cui quelle misure fanno riferimento; il contenuto del file XML contenente le misure IMA.

Per interfacciarsi con il DB e permettere la memorizzazione delle misure IMA si è fatto uso dell'interfaccia *Java Persistence API* (JPA). Quest'interfaccia premette allo sviluppatore di memorizzare, aggiornare, modificare, prelevare dati da un DB gestendoli come se fossero degli oggetti Java. In questo modo viene definita la classe `MwImaMeasurementXml` con lo scopo di consentire la memorizzazione nel DB delle misure IMA. Questa classe infatti fornisce una rappresentazione in Java di un oggetto così come deve essere memorizzato nella tabella `mw_ima_measurement_xml`. Come si può vedere dalla figura 5.4, un oggetto della classe `MwImaMeasurementXml` avrà degli attributi che corrispondono alle colonne della tabella rappresentata in 5.1. Inoltre è stata anche creata una classe `MwImaMeasurementXmlJpaController` la cui funzione è quella di gestire oggetti di tipo `MwImaMeasurementXml` e consentirne quindi la corretta cancellazione, creazione e aggiornamento all'interno del DB.

La creazione delle Whitelist avviene nel metodo `configureWhiteListFromCustomData` all'interno della classe `HostBO`. Qui vengono memorizzati nel DB tutti quei valori che dovranno essere utilizzati per la fase di verifica. Per la memorizzazione delle misure IMA è stato creato il metodo privato `configureImaMeasurementXmlLog`. Tale metodo permette la creazione di un oggetto `MwImaMeasurementXmlJpaController` associato al file XML delle misure IMA ed è chiamato dal metodo `configureWhiteListFromCustomData` solamente nel caso in cui nel report XML siano presenti le misure IMA.

Nel secondo scenario dove non viene prevista alcuna fase di verifica delle misure IMA che coinvolga l'AS, è necessario memorizzare nuovamente le misure IMA ogni qualvolta viene effettuata l'attestazione dell'host. Il metodo `getTrustReportForHost` della classe `HostTrustBO` è il metodo responsabile di chiamare i metodi per l'interazione con il TA e la verifica delle Whitelist in fase di attestazione. All'interno di questo metodo, prima che venga effettuata la fase di verifica, è stata inserita la chiamata ad un nuovo metodo privato chiamato `configureImaMeasurementXmlLog`.

```
@Id
@Basic(optional = false)
@Column(name = "id")
private String id;
@JoinColumn(name = "mleId", referencedColumnName = "id")
@ManyToOne(optional = false)
private TblMle mleId;
@Column(name = "content")
private String content;
```

Figura 5.4. Attributi della classe `MwImaMeasurementXml`

Questo metodo consente di utilizzare le misure IMA arrivate con la nuova quote per aggiornare quelle presenti nel DB. In questo modo l'AS non effettuerà alcuna fase di verifica, ma semplicemente mostrerà le misure IMA sull'interfaccia grafica.

5.2.4 Verifica delle misure IMA

La fase di verifica delle misure IMA segue due differenti scenari. Nel primo scenario è l'AS a verificare direttamente le misure IMA basandosi sulla Whitelist statica costruita in fase di registrazione dell'host. Per effettuare le verifiche Open CIT si basa su un meccanismo di creazione di oggetti che implementano l'interfaccia `Rule`. A seconda del tipo di verifica che si vuole effettuare esiste una differente implementazione di tale interfaccia:

PcrMatchesConstant: utilizzata per confrontare il valore del PCR con il valore dello stesso PCR definito dalla Whitelist;

PcrEventLogIntegrity: utilizzata per calcolare l'aggregato dei PCR 17 e 18 che sono coinvolti nell'architettura Intel TXT. In tal caso ogni misura facente parte del PCR 17 e 18 viene considerata come un *EventLog*;

PcrEventLogEquals: utilizzata per verificare che ciascun *EventLog* coincida con il rispettivo valore di whitelist memorizzato all'interno del DB;

PcrEventLogIncludes: utilizzata per controllare che nessun *EventLog* venga a mancare perché non incluso tra le misure che vengono inviate dal TA;

XmlMeasurementLogEquals: è responsabile di confrontare ciascuna misura riguardante la Trust Policy con il corrispettivo valore memorizzato nel DB come valore di whitelist;

XmlMeasurementLogIntegrity: utilizza le misure riguardanti la Trust Policy per calcolare l'aggregato e confrontarlo con il valore del PCR 19 contenuto nella quote inviata dal TA.

Per poter effettuare la verifica delle misure IMA sono state create due classi che come le classi precedenti estendono l'interfaccia `Rule`:

XmlImaMeasurementLogEquals: è la classe responsabile di confrontare le misure IMA arrivate in fase di attestazione con i valori di Whitelist presenti all'interno del DB. Il suo scopo è quello di comprendere se le misure IMA appena ricevute sono state compromesse rispetto a quelle memorizzate nel DB, oppure se qualche misura attesa non è presente o se sono presenti delle misure non attese. Ogni misura che crea dei problemi viene utilizzata per creare un oggetto che appartiene ad una classe che estende la classe `Fault`. Le classi che estendono la classe `Fault` si differenziano a seconda del tipo di "errore" riscontrato nella fase di verifica: cancellazione di una misura attesa, misura non prevista e misura non corrispondente al valore memorizzato in fase di registrazione;

```
char c = '\0';

byte[] fhashhex = HexUtil.toByteArray(m.getValue().toString());


byte[] fname=(m.getLabel()+c).getBytes();

byte[] algname=("sha1:"+c).getBytes();

int tot_len = fhashhex.length + algname.length;

byte[] bytes1 = ByteBuffer.allocate(4).order(ByteOrder.LITTLE_ENDIAN)
                              .putInt(tot_len).array();
byte[] bytes2 = ByteBuffer.allocate(4).order(ByteOrder.LITTLE_ENDIAN)
                              .putInt(fname.length).array();
try{
    MessageDigest hash = MessageDigest.getInstance("SHA-1");
    hash.update(bytes1);
    hash.update(algname);
    hash.update(fhashhex);
    hash.update(bytes2);
    hash.update(fname);
    return hash.digest();
}catch(NoSuchAlgorithmException e){

    log.debug("NoSuchAlgorithmException: algorithm not found");

}
return null;
```

Figura 5.5. Calcolo del `template-hash` a partire dal `filedata-hash`

XmlImaMeasurementLogIntegrity: utilizzata per calcolare l'aggregato date le misure IMA contenute nella risposta proveniente dal TA. L'aggregato calcolato viene confrontato con il valore del PCR 10 contenuto nella quote. Anche in questo caso qualora i valori confrontati non coincidono, viene rilevata la presenza di un errore utilizzando delle classi che estendono la classe `Fault`. Come è stato discusso in precedenza nella sezione riguardante il TA, i digest che vengono prelevati dal file *ascii_runtime_measurements* sono diversi. Nel primo scenario si preleva il template-hash mentre nel secondo il filedata-hash. Dal momento che l'aggregato del PCR 10 viene calcolato utilizzando il template-hash, nel secondo caso è stato necessario aggiungere alla classe `XmlImaMeasurementLogIntegrity` un metodo privato (figura 5.5) che consentisse il calcolo del template-hash di ogni misura a partire dal filedata-hash.

L'utilizzo di questo nuovo metodo è reso necessario dal fatto che anche nel secondo scenario si vuole garantire che la lista di misure IMA che viene inviata all'AS non sia stata modificata in alcun modo. Il calcolo dell'aggregato e il successivo confronto con il valore del PCR 10 contenuto nella quote permettono all'AS di capire se sia o meno avvenuta qualche modifica alla lista. Inoltre in questo modo si riesce a mantenere la compatibilità con la logica di verifica di Open CIT, in quanto l'AS potrà comunque riuscire a produrre un risultato che indichi lo stato di affidabilità dell'host che potrà essere comunicato al Verifier esterno.

5.2.5 Inclusione delle misure IMA nel Report grafico

Come discusso in precedenza, per mantenere la compatibilità con Open CIT e poter attestare un host senza richiedere l'invio delle misure IMA, è stato aggiunto un campo nell'interfaccia grafica che permetta di effettuare opzionalmente l'attestazione di un host con IMA (figura 5.6)

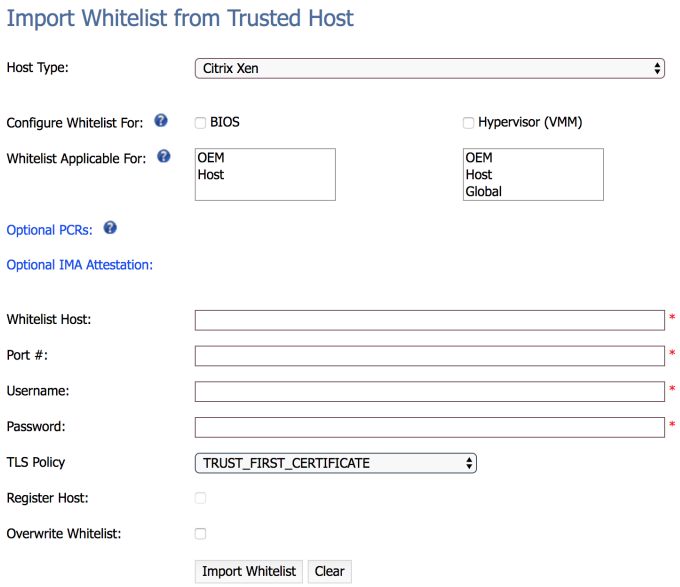

Figura 5.6. Interfaccia di registrazione di un host

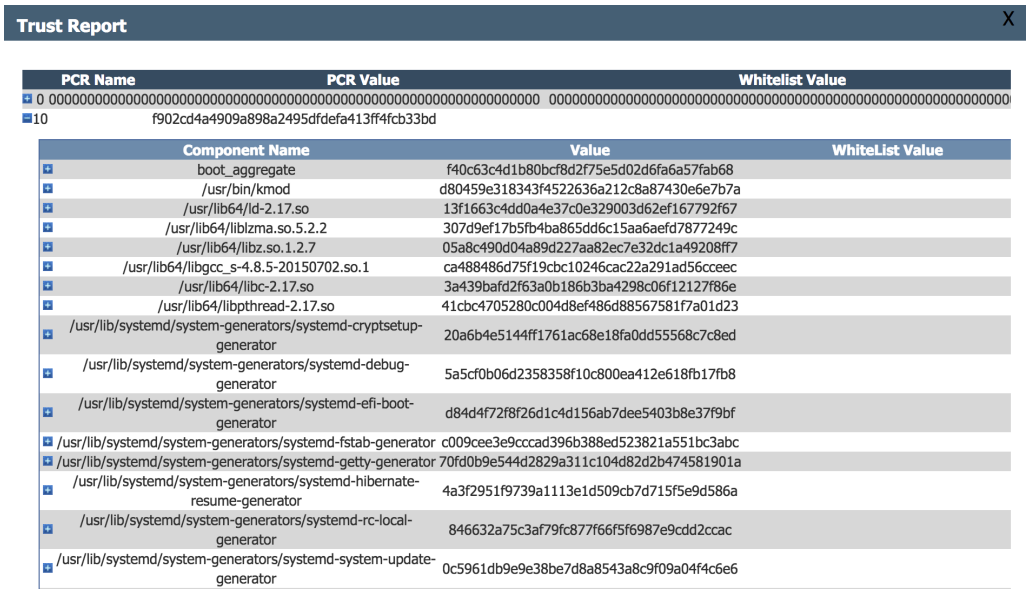

Figura 5.7. Report grafico nel secondo scenario

Per poter inserire tale campo è stato modificato il file `WhiteListConfig.jsp` aggiungendo un `div` con una funzione richiamata quando tale `div` viene selezionato dall'utente. In questo modo è

possibile raccogliere l'informazione indicante che l'utente ha scelto per lo specifico host di effettuare un attestazione con IMA. Questa informazione viene trasferita alla logica dell'AS riguardante l'invio della richiesta al TA tramite la funzione `fnUploadWhiteListConfigurationData` nel file `WhiteListConfig.js`, opportunamente modificata per consentire di specificare la presenza o meno di una richiesta con attestazione IMA.

La costruzione del report grafico che viene mostrato all'utente passa attraverso il metodo `logPcrTrustStatus` della classe `HostTrustBO` dove vengono creati degli oggetti di tipo `TblTaLog`. Un oggetto `TblTaLog` rappresenta un PCR all'interno del DB, ma non è il PCR registrato come valore di Whitelist (costituito dall'oggetto `TblPcrManifest`), bensì il valore attuale del PCR, ovvero quello appena ricevuto in fase di attestazione. Ognuno di questi oggetti è caratterizzato da zero o più oggetti di tipo `TblModuleManifestLog`. Un oggetto `TblModuleManifestLog` rappresenta gli oggetti di tipo `Fault` all'interno del DB e rileva la presenza di inconsistenze tra i valori di Whitelist e i valori ricevuti dal TA durante la fase di attestazione. Pertanto il metodo `logPcrTrustStatus` è stato modificato per permettere la creazione di oggetti `TblModuleManifestLog` nel caso in cui ci fossero dei problemi nella verifica delle misure IMA e creare un oggetto `TblTaLog` anche per il PCR 10. Infatti nella fase di creazione degli oggetti `TblTaLog` è presente un controllo che porta all'esclusione di qualsiasi PCR che non appartenga al banco ritenuto più robusto per l'host (ovvero il banco SHA 256 per la macchina di test).

Gli oggetti di tipo `TblTaLog` e `TblModuleManifestLog` vengono utilizzati all'interno del metodo `addManifestLogs` della classe `ReportsBO`. Per ogni PCR vengono creati degli oggetti di tipo `ModuleLogReport` che costituiscono ognuno la singola misura, caratterizzata da valore attuale e valore di whitelist, e riguardante uno specifico PCR. Il metodo `addManifestLogs` è stato modificato per permettere la creazione di oggetti `ModuleLogReport` per ciascuna misura IMA riguardante il PCR 10. Un oggetto `ModuleLogReport` , sfruttando gli oggetti `TblModuleManifestLog`, permette anche di visualizzare le misure non previste e le misure che non soddisfano il loro valore di Whitelist. La creazione degli oggetti `ModuleLogReport` che non riguardano misure affette da un qualche tipo di errore, quindi misure ritenute *trusted*, avviene sfruttando i valori di Whitelist delle misure stesse memorizzati nel DB in fase di registrazione. Quindi la logica seguita da Open CIT nel caso di misure *trusted* è quella di non utilizzare i valori delle misure ricevuti durante la fase di attestazione ma quelli precedentemente registrati come valori di whitelist dal momento che se sono *trusted* sicuramente i propri valori coincidono. Quindi anche nel caso delle misure IMA il metodo `addManifestLogs` è stato modificato per creare oggetti `ModuleLogReport` partendo dall'oggetto `MwImaMeasurementXml` memorizzato in fase di registrazione dell'host.

Nel secondo scenario dal momento che la fase di verifica delle misure IMA viene effettuata dal Verifier esterno il metodo `addManifestLogs` è stato modificato per far si che alla creazione degli oggetti `ModuleLogReport` per le misure IMA, non venisse configurato alcun valore di whitelist. In questo modo nel report grafico le misure IMA appariranno solamente con il loro valore attuale e non avranno alcun valore di whitelist (figura 5.7).

5.3 Creazione del Driver

Il Driver è una classe realizzata nel linguaggio di programmazione *Python* il cui compito è quello di interagire con la logica di verifica del Verifier esterno al fine di definire un livello di trust delle misure IMA. Il Driver permette di registrare l'host da attestare all'interno del Verifier utilizzando un identificativo univoco `host_uuid` e il suo `hostname`. Inoltre espone la funzione `getStatus` il cui compito è quello di inviare una richiesta alla url sulla quale è in ascolto l'AS per vedere se è attivo o meno. La funzione più importante offerta dal Driver è la funzione `pollHost` la quale permette di effettuare l'attestazione delle misure IMA provenienti dall'host. Tale funzione per realizzare il suo obiettivo sfrutta le API REST definite da Open CIT. Innanzitutto viene effettuata una richiesta POST all'Attestation Server passando un oggetto JSON contenente l'identificativo univoco associato all'host che si vuole attestare. Tale identificativo viene creato dall'AS in seguito alla fase di registrazione dell'host e può essere recuperato dal DB dell'AS. La richiesta POST permette all'AS di effettuare una nuova attestazione per l'host il cui identificativo è specificato nel corpo della richiesta. Il risultato prodotto in seguito all'esecuzione di questa richiesta è la

creazione di un'asserzione SAML, memorizzata nel DB dell'AS e contente l'informazione circa lo stato di affidabilità dell'host attestato, oltre che alla data ed ora in cui l'host è stato attestato.

```
POST https://indirizzo_ip_AS:8443/mtwilson/v2/host-attestations

Content-Type: application/json

{"host_uuid":"c455c03f-c578-4098-bee0-04864ccf62d0"}

Risposta:
HTTP/1.1 200 OK
Content-Type: application/saml

<?xml version="1.0" encoding="UTF-8"?>
<saml2:Assertion xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
    ID="HostTrustAssertion" IssueInstant="2018-05-14T08:17:46.691Z"
    Version="2.0">
        ....................................
               <saml2:Attribute Name="Trusted">
                      <saml2:AttributeValue
                          xmlns:xs="http://www.w3.org/2001/XMLSchema"
                              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                  xsi:type="xs:anyType">true</saml2:AttributeValue>
                      </saml2:Attribute
        ....................................
</saml2:Assertion>
```

Una volta che l'host è stato attestato viene effettuata una richiesta GET verso l'AS con lo scopo di prelevare il report contenente tutte le informazioni circa lo stato dell'host e le misure che lo caratterizzano. La richiesta prende come parametro il nome dell'host del quale si vuole che l'AS restituisca il report.

```
GET
    https://indirizzo_ip_AS:8443/mtwilson/v2/host-attestations?nameEqualTo=hostname

Risposta:
HTTP/1.1 200 OK
Content-Type: application/xml

<host_attestation_collection>
       <host_attestations>
              <host_attestation>
                     <id>1e647b2c-365b-4f84-835d-51fa8dcc323e</id>
                     <hostUuid>c455c03f-c578-4098-bee0-04864ccf62d0<hostUuid>
                     <hostName>compute2</hostName>
                     ....................................
                     <imaMeasurementXml>.............</imaMeasurementXml>
                     ....................................
                     <saml2:Assertion .......... </saml2:Assertion>
              </host_attestation>
       </host_attestations>
</host_attestation_collection>
```

Come si può osservare il report contiene le misure IMA che sono state prelevate dall'host in fase di attestazione, così come l'asserzione SAML creata precedentemente durante l'esecuzione della POST. Una volta ricevuto il report se ne effettua il *parsing* con lo scopo di prelevare informazioni come: lo stato dell'host secondo l'AS, l'istante di creazione dell'asserzione SAML e il file XML contenente le misure IMA. Le prime due informazioni vengono utilizzate per la costruzione di un oggetto JSON che il Verifier restituirà dopo aver effettuato la verifica. Il file XML riguardante le misure IMA viene utilizzato per la costruzione di tanti oggetti `IMARecord` quante sono le misure. Gli oggetti `IMARecord` vengono utilizzati dal Verifier per le operazioni di verifica delle misure. Alla fine del processo di verifica viene costruito un oggetto JSON che definisce lo stato dell'host, l'eventuale presenza di misure che non sono state trovate nel DB del Verifier e il numero di pacchetti ai quali le misure appartengono, che hanno bisogno di un aggiornamento di sicurezza o di *bug fixing*.

Capitolo 6

Risultati

I risultati raccolti hanno permesso di evidenziare come la soluzione proposta che consente l'integrazione di IMA in Open CIT e la conseguente verifica delle misure IMA, sia efficiente dal punto di vista prestazionale. Infatti riesce a raggiungere il suo obiettivo senza portare ad eccessivi ritardi nel processo di attestazione dell'host all'interno dell'infrastruttura cloud. Di seguito vengono analizzati i risultati ottenuti in entrambi gli scenari.

6.1 Scenario: Whitelist statiche

In questo scenario le misure IMA vengono verificate attraverso la logica di verifica prevista da Open CIT consistente nella generazione di valori di Whitelist durante la fase di registrazione dell'host. Per permettere la realizzazione di questo scenario sono stati modificati i componenti Trust Agent e Attestation Server facenti parte del framework Open CIT. La raccolta dei risultati è stata effettuata su questi componenti con lo scopo di analizzare quanto la presenza delle nuove funzionalità vada ad inficiare le prestazioni complessive del sistema. Alla luce di queste considerazioni si è scelto di analizzare le seguenti tempistiche sul Trust Agent:

Recupero misure TCB: è il tempo necessario affinché il Trust Agent possa recuperare dal file `measurement.xml` le misure riguardanti la Trust Policy.

Recupero misure IMA: è il tempo necessario affinché il Trust Agent possa recuperare le misure IMA dal file `ascii_ runtime_measurements`.

Costruzione Quote: è il tempo necessario affinché il Trust Agent collabori con il TPM per calcolare la Quote contenente i valori dei PCR.

Costruzione risposta: è il tempo necessario per inserire nella risposta tutte le informazioni richieste dall'Attestation Server (e.g. Quote, misure Trust Policy, misure IMA).

Le tempistiche che invece sono state prese in considerazione sull'Attestation Server sono:

Informazioni host: è il tempo necessario al recupero delle informazioni riguardati l'host da attestare dal DB (e.g. hostname, indirizzo IP, identificativo univoco, banchi PCR supportati);

Ricezione ed elaborazione risposta: è la somma del tempo impiegato affinché l'Attestation Server riceva le informazioni di integrità dal Trust Agent e il tempo necessario all'Attestation Server per effettuare il parsing della risposta e ricavare le informazioni sullo stato dell'host. Bisogna inoltre tener presente che il tempo di ricezione della risposta è la somma del tempo necessario al Trust Agent per effettuare il processo che porta alla costruzione della risposta e il tempo di trasmissione;

Recupero BIOS e VMM MLE: è il tempo impiegato dall'Attestation Server per recuperare dal DB le Whitelist registrate per lo specifico host da attestare;

Verifica delle misure: è il tempo necessario affinché l'Attestation Server verifichi i valori dei PCR ricevuti e le loro misure utilizzando i valori di Whitelist precedentemente registrati.

Innanzitutto osservando le tabelle in figura 6.1 si può notare come la soluzione progettata per il primo scenario non infici eccessivamente sui tempi impiegati da Open CIT per attestare l'host. Si può notare la notevole differenza tra il tempo di recupero delle misure TCB e il tempo di recupero delle misure IMA. Questo è una conseguenza del fatto che le misure TCB sono sempre disponibili immediatamente ad ogni attestazione e si trovano già in formato XML all'interno del file `measurement.xml` che viene costruito ogni qualvolta si effettua il boot della macchina e quindi non cambia finché non viene riavviata la macchina a meno di modifiche. Per le misure IMA invece il file in formato XML non è disponibile immediatamente, ma è necessario prima catturare le misure dal file `ascii_ runtime_measurements` e successivamente costruire il file XML delle misure da inviare all'Attestation Server. Le tempistiche meno interessanti sono quella riguardante la costruzione della quote e quella riguardante la costruzione della risposta. Per la prima si può notare che sia nel caso di attestazione con IMA sia in caso di attestazione senza IMA non ci sia alcuna differenza. Questo è naturale dal momento che, come descritto nei capitoli precedenti, la Quote si basa solamente sui valori dei PCR e non riguarda la presenza o meno delle misure IMA. Anche il tempo di costruzione della risposta non varia sebbene ci sia un nuovo elemento da inserire nella risposta, i.e. le misure IMA. Questo è dovuto al fatto che la risposta che dovrà essere restituita all'Attestation Server viene costruita a partire da un oggetto Java costruito passo dopo passo quando si recuperano le informazioni utili. Quindi l'azione che viene compiuta è solamente una costruzione di un "oggetto" XML che contenga le informazioni prelevate dall'oggetto Java di cui detto. Come si può osservare dal tempo totale, le prestazioni sul Trust Agent anche introducendo le misure IMA rimangono quasi invariate.

	Tempo (ms) - IMA NO	Tempo (ms) - IMA SI
Recupero misure TCB	1.3	1.6
Recupero misure IMA	/	22.6
Costruzione Quote	2057.8	2057.3
Costruzione risposta	3.2	3.3
Totale	**2062.3**	**2084.8**

	Tempo (ms) - IMA NO	Tempo (ms) - IMA SI
Informazioni host	14.8	17
Ricezione ed elaborazione risposta	2979.1	3015.8
recupero BIOS e VMM MLE	125.3	292.8
Verifica delle misure	278.1	755.5
Totale	**3397.3**	**4081.1**

Figura 6.1. Confronto Open CIT con o senza IMA

Per quanto riguarda le tempistiche inerenti all'Attestation Server, si può notare che l'inserimento delle misure IMA ha comunque portato ad un peggioramento di circa un secondo (20%). Le maggiori differenze si notano nei tempi di recupero delle Whitelist e verifica delle misure. Questo perché le Whitelist nel caso di attestazione con IMA sono caratterizzate dalla presenza delle misure IMA che comunque a seconda del tipo di policy IMA applicata possono anche essere in numero abbastanza elevato. Di conseguenza più misure sono presenti più aumenta il tempo necessario ad

effettuare la verifica. Il tempo di ricezione ed elaborazione della risposta varia di poco in quanto comunque bisogna tenere conto che nella risposta che arriva dal Trust Agent è presente un nuovo elemento, i.e. le misure IMA. Il tempo necessario alla raccolta delle informazioni dell'host è ininfluente. Complessivamente un peggioramento del 20% nelle prestazioni da parte dell'Attestation Server è comunque un peggioramento ragionevole dal momento che come è stato spiegato le misure IMA possono essere tante e naturalmente necessitano di tempi di verifica più lunghi.

6.1.1 Analisi al crescere del numero delle misure

Un'altra analisi che è stata effettuata ha permesso di ottenere dei risultati nel caso in cui aumenta il numero delle misure IMA ritenute *untrusted*. Questo è il *worst case* dal momento che un aumento delle misure considerate *trusted*, come dimostrato da prove sperimentali, non porta ad un considerevole aumento delle tempistiche. Infatti per come è costruita la logica di verifica di Open CIT, essa è più lenta nelle verifiche quando sono presenti delle misure non previste. I casi che sono stati analizzati prevedono di calcolare le tempistiche quando si hanno rispettivamente 20, 1000 e 2000 misure *untrusted*. Le tabelle 6.1 e 6.2 mostrano i tempi complessi medi necessari rispettivamente al Trust Agent e all'Attestation Server per compiere le operazioni descritte dettagliatamente in precedenza. Come si può osservare entrambe le tabelle mostrato che non c'è un significativo peggioramento delle prestazioni come si potrebbe pensare.

Numero misure IMA Untrusted	*Tempo (ms)*
20	2086.1
1000	2099.4
2000	2110.3

Tabella 6.1. Confronto dei tempi sul Trust Agent con IMA

Numero misure IMA Untrusted	*Tempo (ms)*
20	4080.5
1000	4343.3
2000	4643.2

Tabella 6.2. Confronto dei tempi sull'Attestation Server con IMA

Più dettagliatamente i grafici della figura 6.2 permettono di evidenziare l'andamento al crescere del numero delle misure per i tempi che più incidono sul tempo totale, ovvero il tempo di recupero delle misure IMA sul Trust Agent e il tempo di verifica sull'Attestation Server. Il tempo necessario al recupero delle misure IMA risulta essere crescente con il numero di misure dal momento che cresce la dimensione del file `ascii_runtime_measurements` e cresce il tempo necessario alla costruzione del file XML che contiene le misure. Anche il tempo di verifica ha un andamento crescente con il numero delle misure, tuttavia in entrambi i casi è evidente come il peggioramento non sia tale da causare una decadenza delle prestazioni nel procedimento di attestazione dell'host.

Un'analisi rilevante può essere effettuata alla luce dei risultati mostrati nella figura 6.3. Questa figura infatti mostra il tempo di attestazione nei tre casi in esame in relazione al tempo totale di risposta in seguito ad un'attestazione effettuata dall'utente attraverso l'utilizzo dell'interfaccia grafica o attraverso l'utilizzo dell'API REST di attestazione dell'host. I risultati mostrano dei tempi di risposta davvero elevati quando aumenta il numero di misure IMA ritenute untrusted. Quindi una volta che l'attestazione è stata completata c'è una parte del sistema che porta ad avere dei tempi di risposta molto elevati. La causa di questi tempi non è una progettazione non buona della soluzione sviluppata, bensì essa risiede nella logica di creazione delle informazioni che saranno utili all'interfaccia grafica, se utilizzata. Infatti prima di individuare la causa di tale problema sono state effettuate delle prove di attestazione senza IMA, ma con un incremento via via sempre maggiore delle misure untrusted della Trust Policy. Eseguendo tali test si è notato che i tempi di risposta sono molto vicini a quelli che si hanno con le misure IMA nel caso in cui

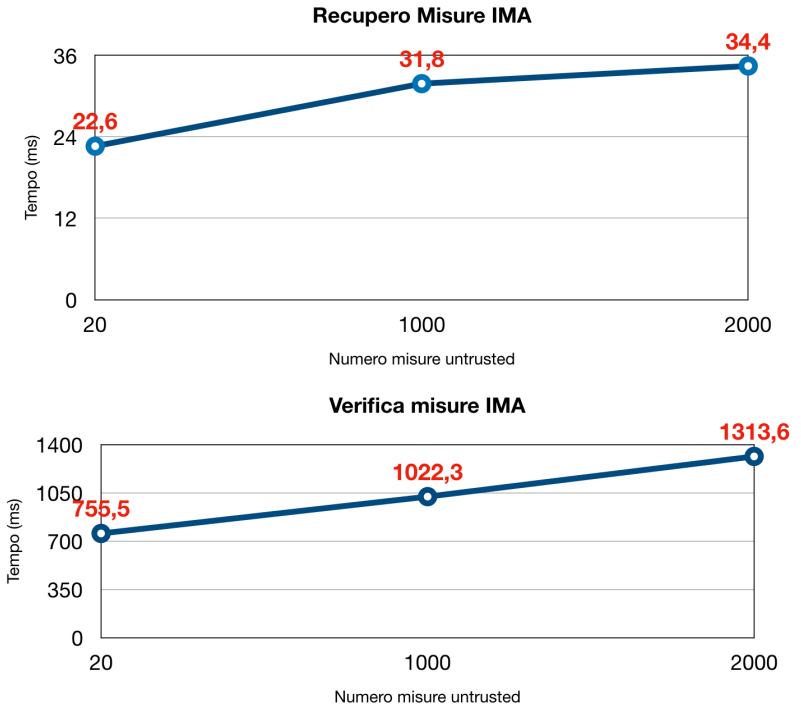

Figura 6.2. Andamento dei tempi di recupero e verifica delle misure all'aumentare del numero di misure untrusted

sia presente un numero elevato di misure untrusted. Quindi la causa di questo comportamento non risiede nell'introduzione delle misure IMA e delle varie logiche di verifica previste per esse. Il problema è nel fatto che dopo aver terminato la fase di verifica, Open CIT prevede la creazione di un report caratterizzato da un numero abbastanza elevato di oggetti da inserire all'interno del DB. Questo report è quello che viene utilizzato per mostrare all'utente tutti i PCR e le misure che caratterizzano l'host attestato. La sua costruzione avviene attraverso un numero molto elevato di istruzioni condizionali e cicli annidati che analizzano tutti gli oggetti Java creati a seguito della presenza di misure untrusted.

Numero IMA Misure Untrusted	Tempo attestazione	Tempo totale (risposta e creazione report)
20	4080.5 ms	4.82 sec
1000	4343.3 ms	33.89 sec
2000	4643.2 ms	98 sec

Figura 6.3. Tempi di attestazione in relazione ai tempi di risposta all'aumentare del numero di misure untrusted

6.2 Scenario: utilizzo del Verifier esterno

Il secondo scenario consente di effettuare la verifica delle misure IMA attraverso l'utilizzo di un Verifier esterno. Quest'ultimo utilizza il framework di attestazione per prelevare le misure dall'host e le trasferisce alla sua logica di verifica la quale indica il livello di trust per l'host in esame. Le tempistiche che sono state prese in considerazione sono le seguenti:

Tempo attestazione: è il tempo impiegato dal framework Open CIT, quindi dall'AS, per effettuare l'attestazione dell'host;

Tempo recupero e parsing report: è il tempo dato dalla somma del tempo impiegato dal Verifier per richiedere all'AS il report riguardante tutti valori dei PCR e le relative misure appartenenti all'host precedentemente attestato, e il tempo impiegato per fare il parsing del report e recuperare le misure IMA trasferendole alla logica di verifica;

Tempo verifica: è il tempo impiegato dal Verifier per effettuare le operazioni di verifica e restituire i risultati.

Numero misure IMA Untrusted	*Tempo (sec)*
20	4.95
1000	11.77
2000	20.37

Tabella 6.3. Confronto dei tempi sul Verifier

Così come nel primo, anche nel secondo scenario si è voluto analizzare il `worst case`, considerando i casi in cui il numero di misure ritenute untrusted cresce. In particolare i test sono stato eseguiti quando le misure untrusted sono rispettivamente 20, 1000 e 2000, su macchine CentOS, dal momento che il secondo scenario è disponibile solo per macchine con tale sistema operativo. Come si può notare dalla tabella 6.3 il tempo impiegato dal Verifier per verificare le misure IMA cresce all'aumentare delle misure untrusted. Come si può evincere dal grafico in figura 6.4 gran parte del tempo totale viene utilizzato per le operazioni di verifica. Dal grafico si può notare la differenza tra i due scenari dove il processo di verifica nel primo scenario ha dei tempi migliori rispetto al secondo scenario. Ma si deve tener conto che, mentre nello primo scenario si devono effettuare dei controlli con un DB di misure IMA molto piccolo (all'incirca un migliaio di misure), nel secondo scenario il DB ha dimensioni notevoli e quindi la ricerca nel DB può portare ad avere dei tempi abbastanza lunghi. Come si può notare dai grafici 6.5 e 6.6 i tempi di attestazione e di recupero del report crescono all'aumentare delle misure untrusted ma contribuiscono di meno al tempo totale rispetto al tempo di verifica. Inoltre si può notare come i tempi di attestazione nei casi in esame siano inferiori nel secondo scenario rispetto al primo. Questo è spiegabile con il fatto che nei tempi di attestazione del primo scenario è già inclusa la parte di verifica, mentre nel secondo scenario la parte di verifica, escludendo il controllo sull'aggregato del PCR 10, viene lasciata al Verifier esterno.

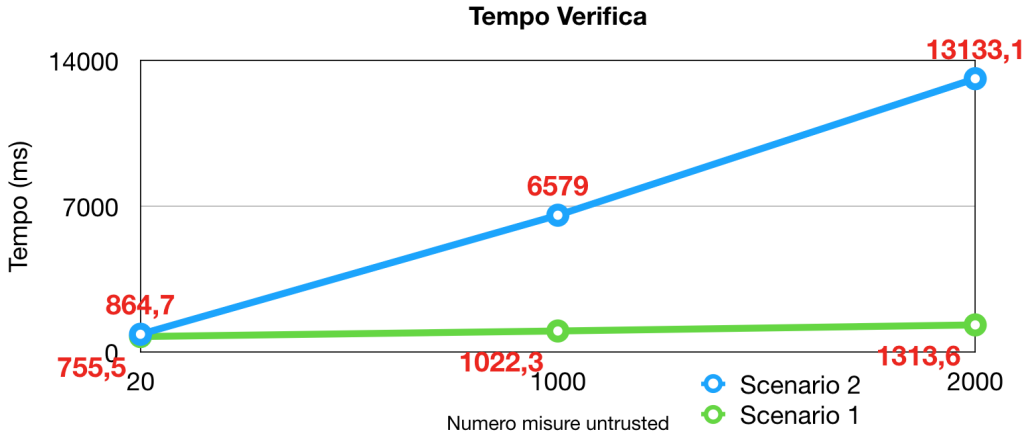

Figura 6.4. Differenze tra i due scenari nei tempi di verifica

Per quanto riguarda l'aumento dei tempi di recupero e di parsing della risposta, è naturale che con l'aumentare del numero delle misure cresca la dimensione del report e di conseguenza il tempo necessario ad effettuare il parsing dello stesso.

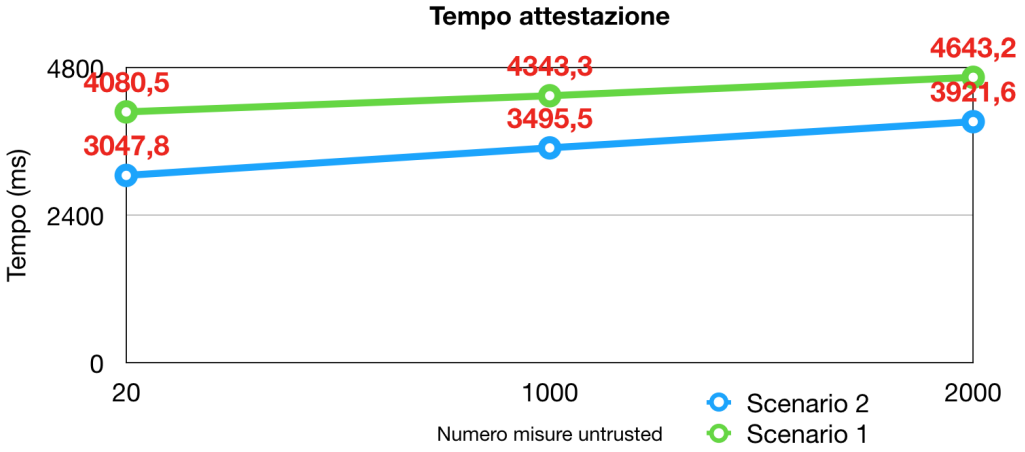

Figura 6.5. Differenze tra i due scenari nei tempi di attestazione

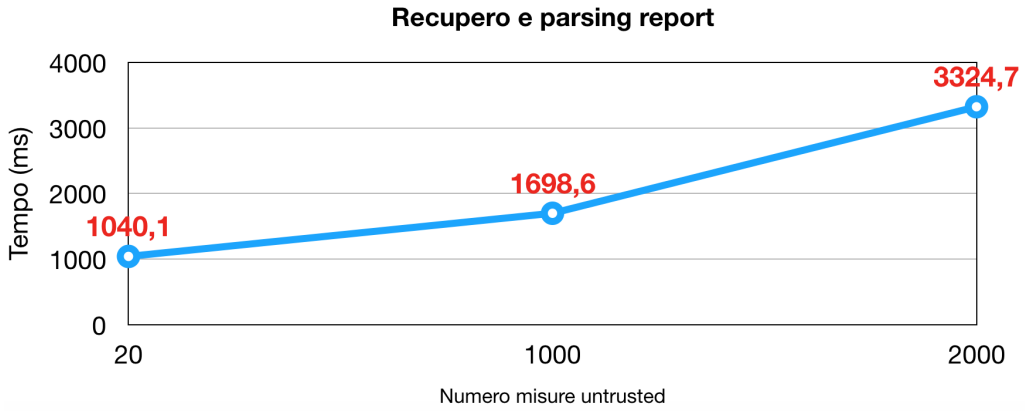

Figura 6.6. Grafico sui tempi di recupero e parsing del report

Per concludere nella tabella 6.7 si vogliono sottolineare le differenze dei tempi totali impiegati dall'uno e dall'altro scenario per svolgere le operazioni. Si può notare come i tempi necessari nel secondo scenario siano molto inferiori rispetto al primo, ma come si è detto in precedenza, il problema di tempi così lunghi nel primo scenario è legato alla creazione e memorizzazione del report grafico. Migliorando la logica di costruzione del report, il primo scenario avrebbe dei tempi migliori rispetto al secondo, tuttavia continuerebbe ad esserci il limite dell'utilizzo delle Whitelist statiche, non adattabili ad ambienti in continua evoluzione dal punto di vista software.

Numero IMA Misure Untrusted	Tempo Totale (Scenario 1)	Tempo totale (Scenario 2)
20	4.82 sec	4.95 sec
1000	33.89 sec	11.77 sec
2000	98 sec	20.37 sec

Figura 6.7. Differenze tra i due scenari

Capitolo 7

Conclusioni

Questo lavoro di tesi si è posto come obiettivo quello di consentire il miglioramento delle funzioni di sicurezza all'interno di un'infrastruttura cloud grazie all'utilizzo di una tecnica nota come *attestazione remota* che consente di definire lo stato di integrità di ogni piattaforma facente parte del cloud stesso. Per realizzare l'attestazione remota all'interno del sistema sono state analizzate varie soluzioni decidendo di utilizzare un progetto esistente sviluppato da Intel noto come Open CIT. La scelta è stata compiuta sulla base dei vantaggi che esso offre rispetto agli altri framework di attestazione, ovvero quelli di usufruire dell'architettura Intel TXT, di attestare macchine dotate di TPM 2.0 e di garantire l'integrità di file e cartelle attraverso l'utilizzo delle Trust Policy. Sono stati individuati i limiti dell'SDK, rappresentati dall'incapacità di fornire delle informazioni di integrità che rispecchino lo stato attuale di esecuzione dell'host, dal momento che Open CIT è in grado solamente di fornire informazioni di integrità definite a boot time e non a run time. Il superamento di questi limiti è stato ottenuto tramite l'integrazione dell'architettura IMA grazie alle possibilità che esso offre di rilevare eventuali manipolazioni che coinvolgono l'esecuzione di un binario.

Le modifiche effettuate ad Open CIT hanno previsto l'implementazione di due differenti scenari che, sebbene propongano delle logiche di verifica differenti, rispettano comunque la progettazione originale proponendo delle modifiche che non vanno ad intaccare in alcun modo la logica di funzionamento dell'SDK. Infatti sia il primo che il secondo scenario consentono di raccogliere le misure IMA rispettando la logica di raccolta delle informazioni di integrità proposta da Open CIT che consiste nell'inserire le misure relative ad un PCR all'interno di un file XML. Inoltre nel primo scenario la logica di verifica delle misure IMA è esattamente la stessa seguita da Open CIT per verificare tutte le misure riguardanti uno specifico PCR. Nel secondo scenario, invece, sebbene parte della logica di verifica venga affidata ad un Verifier esterno, si lascia comunque ad Open CIT la possibilità di definire lo stato di affidabilità dell'host attraverso una verifica dell'aggregato delle misure IMA e la loro visualizzazione all'interno dell'interfaccia grafica proposta all'utente.

Un aspetto molto importante scaturito dall'analisi delle prestazioni della soluzione proposta, ha evidenziato come l'SDK abbia dei tempi di risposta estremamente lunghi nel caso in cui siano presenti delle misure non previste nella definizione delle Whitelist dovuti alla creazione del report grafico. Tale problema non si verifica invece nel secondo scenario dove, come dimostrano i risultati ottenuti, la verifica delle misure è abbastanza veloce, ma può essere utilizzato solamente per l'attestazione di host CentOS. Un altro aspetto che non dipende in sé per sé da Open CIT è legato al fatto che per inviare le informazioni il Trust Agent non utilizzi un Integrity Report che segua le specifiche del TCG, ma questo come detto dipende dal fatto che non sono ancora presenti delle specifiche che consentano la sua definizione.

Una possibile evoluzione di questo lavoro di tesi consisterebbe nel consentire l'analisi dell'ambiente di esecuzione di VM e/o container Docker[54]. Open CIT al momento è in grado solo di verificare che non sia stata cambiata la configurazione di una VM e/o di un container Docker durante la sua fase di boot, attraverso l'utilizzo delle Trust Policy. Sarebbe importante raccogliere delle informazioni a runtime sullo stato delle VM ospitate su un host e/o dei container Docker

poiché la maggior parte dei servizi che il cloud offre vengono resi disponibili da questi, e quindi garantire l'integrità del software in esecuzione su di essi permetterebbe a chi gestisce l'infrastruttura di conoscere lo stato di esecuzione di ogni VM e/o container docker. Ovviamente rilevare lo stato della macchina host sulla quale si trovano le VM e i container è ancora più importante, poiché se è lo stesso host a non essere sicuro anche le sue VM e i suoi container non lo saranno. Quindi questa tesi può costituire un buon punto di partenza nella definizione futura di un lavoro che permetta di estendere lo stato di affidabilità dell'host in modo che coinvolga anche lo stato delle VM e/o dei container docker in esecuzione su di esso.

Appendice A

Manuale utente

La versione di Open CIT che si intende installare in questo manuale è la versione `3.2.1` che si appoggia su un'installazione standard di OpenStack `Mitaka` (https://docs.openstack.org/mitaka/install-guide-ubuntu/). L'ambiente di test prevede l'utilizzo delle seguenti macchine:

Trust Agent: Installato su una macchina NUC Intel NUC5i5MYBE con processore Intel Core i5-5300U da 2.30 GHz, architettura x_86 e memoria RAM da 16 GB;

Attestation Server: Installato su un Acer Espire E5-575G con processore Intel Core i7-6500U da 2.50 GHz, architettura x_86 e memoria RAM da 12 GB;

OpenStack controller: Installato su una macchina virtual con sistema operativo Ubuntu Server 16.04 e memoria RAM da 4 GB.

In questa sezione vengono descritti i passi necessari per poter ricreare l'ambiente di test. Sono presenti due differenti scenari:

- nel primo viene descritto il procedimento da seguire nel caso in cui si voglia installare il Trust Agent su un host con Sistema Operativo (SO) Ubuntu 16.04 server. In tal caso le misure IMA vengono verificate dall'Attestation Server;
- nel secondo si descrive il procedimento da seguire nel caso si voglia installare il Trust Agent su un host con SO CentOS 7. In questo caso le misure IMA vengono verificate da un Verifier esterno.

Le differenze nel processo di installazione dei due differenti scenari verranno indicate dove necessario. In entrambi gli scenari valgono i seguenti prerequisiti. La macchina dove si deve installare l'Attestation Server deve avere almeno i seguenti requisiti:

Sistema Operativo supportato: Ubuntu 14.04;

RAM: 4 GB raccomandati, 2 GB minimo;

Spazio sul disco: 2 GB per installare i servizi dell'Attestation Server, ma potrebbe essere necessario ulteriore spazio se venisse installato il DB sullo stesso disco;

La macchina dove si deve installare il Trust Director deve avere almeno i seguenti prerequisiti:

Sistema Operativo supportato: Ubuntu 14.04 Server;

RAM: 2 GB minimo;

Spazio sul disco: 1 GB

Open CIT prevede che ogni macchina deve essere dotata di accesso SSH come utente root. Il build di ogni progetto deve essere effettuato sulla macchina dove viene installato il Trust Agent che è chiamata *compute1* ed è equipaggiata con Intel TXT e il TPM 2.0, affinché sia possibile effettuare il build del progetto `opencit-tboot-xm`.

A.1 Preparare l'ambiente di build

Innanzitutto bisogna installare i pacchetti richiesti tramite il seguente comando eseguito in una shell:

```
$ apt-get install -y ssh ant makeself git make gcc g++ openssl libssl-dev
    unzip nsis zip
```

Per host CentOS eseguire i comandi:

```
$ yum update

$ yum install -y openssh ant git make gcc gcc-c++ openssl openssl-devel unzip
    zip patch wget trousers-devel

$ wget http://mirror.centos.org/centos/7/extras/x86_64/Packages
              /epel-release-7-9.noarch.rpm

$ rpm -ivh epel-release-7-9.noarch.rpm

$ yum install -y makeself nsis
```

Successivamente bisogna definire le seguenti variabili d'ambiente in un file come `.bashrc`:

```
JAVA_HOME=/usr/lib/jvm/jdk1.8.0_131
export JAVA_HOME PATH=$PATH:$JAVA_HOME
export MAVEN_HOME=/usr/local/apache-maven-3.3.1
export M2=$MAVEN_HOME/bin
export MAVEN_OPTS="-Xmx1024m -XX:MaxPermSize=1024m" PATH=$PATH:$M2
```

Per installare `Java` versione `1.8.0.0_131` devono essere eseguiti i seguenti comandi:

```
$ curl -b oraclelicense=accept-securebackup-cookie -L -s -o
              jdk-8u131-linux-x64.tar.gz
              http://download.oracle.com/otn-pub/java/jdk/
              8u131-b11/d54c1d3a095b4ff2b6607d096fa80163/
              jdk-8u131-linux-x64.tar.gz

$ gzip -dc jdk-8u131-linux-x64.tar.gz | tar xf -

$ mv jdk1.8.0_131 /usr/lib/jvm

$ cd /etc/alternatives/

$ ln -s /usr/lib/jvm/jdk1.8.0_131/bin/java
```

```
$ ln -s /usr/lib/jvm/jdk1.8.0_131/bin/javac

$ cd /usr/bin/

$ ln -s /etc/alternatives/java

$ ln -s /etc/alternatives/javac
```

Per installare `Maven` versione `3.3.1` eseguire i seguenti comandi:

```
$ wget http://archive.apache.org/dist/maven/maven-3/3.3.1
              /binaries/apache-maven-3.3.1-bin.zip

$ unzip apache-maven-3.3.1-bin.zip

$ mv apache-maven-3.3.1 /usr/local
```

Scaricare il codice relativo ad Open CIT eseguendo i seguenti comandi come utente `root`:

```
$ cd /root

$ git clone https://github.com/opencit/opencit-external-artifacts

$ git clone https://github.com/opencit/opencit-contrib

$ git clone https://github.com/opencit/opencit-util

$ git clone https://github.com/opencit/opencit-privacyca

$ git clone https://github.com/opencit/opencit-tboot-xm

$ git clone https://github.com/dav10re/opencit-trustagent.git

$ git clone https://github.com/dav10re/opencit.git

$ git clone https://github.com/opencit/opencit-vrtm

$ git clone https://github.com/opencit/opencit-policyagent

$ git clone https://github.com/opencit/opencit-docker-proxy

$ git clone https://github.com/opencit/opencit-kms

$ git clone https://github.com/opencit/opencit-director

$ git clone https://github.com/opencit/opencit-openstack-extensions

$ git clone https://github.com/opencit/opencit-openstack-controller-extensions
```

```
$ git clone https://github.com/opencit/opencit-attestation-cache-hub

$ git clone https://github.com/opencit/opencit-quickstart

$ git clone https://github.com/opencit/opencit-tpm-tools-windows
```

A.2 Build dei progetti di Open CIT

Il primo progetto di cui deve essere fatto il build è il progetto `Open CIT External Artifacts`. I comandi da eseguire a tal scopo sono i seguenti:

```
$ cd /root/opencit-external-artifacts

$ git checkout v1.0+cit-3.2.1

$ wget --no-check-certificate https://mmonit.com/monit/dist/monit-5.5.tar.gz

$ mv monit-5.5.tar.gz monit/monit-5.5-linux-src.tgz

$ wget --no-check-certificate https://archive.apache.org/dist/tomcat
             /tomcat-7/v7.0.34/bin/apache-tomcat-7.0.34.tar.gz

$ mv apache-tomcat-7.0.34.tar.gz apache-tomcat/apache-tomcat-7.0.34.tar.gz

$ wget --no-check-certificate https://sourceforge.net/projects/vijava
             /files/vijava/VI%20Java%20API%205.5%20Beta/vijava55b20130927.zip

$ unzip vijava55b20130927.zip

$ mv vijava55b20130927.jar vijava/vijava-5.5.jar

$ wget --no-check-certificate
    https://sourceforge.net/projects/kmip4j/files/KMIP4J-V1.0/kmip4j-bin-1.0.zip

$ unzip kmip4j-bin-1.0.zip

$ mv jar/kmip4j.jar kmip4j/kmip4j-1.0.jar

$ wget --no-check-certificate
    https://sourceforge.net/projects/ext2fsd/files/Ext2fsd/0.62/Ext2Fsd-0.62.exe

$ mv Ext2Fsd-0.62.exe ext2fsd/Ext2Fsd-0.62.exe

$ wget --no-check-certificate https://sourceforge.net/projects/trousers
             /files/tpm-tools/1.3.8/tpm-tools-1.3.8.tar.gz

$ mv tpm-tools-1.3.8.tar.gz tpm-tools/tpm-tools-1.3.8.tar.gz

$ ant
```

Quindi procedere con il build di tutti gli altri progetti di Open CIT eseguendo i seguenti comandi:

```
$ cd /root/opencit-contrib

$ git checkout v1.0+cit-3.2.1

$ ant

$ cd /root/opencit-util

$ git checkout v1.0+cit-3.2.1

$ ant

$ cd /root/opencit-privacyca

$ git checkout v3.2.1

$ ant

$ cd /root/opencit-tboot-xm

$ git checkout v3.2.1

$ ant
```

Nel caso di host CentOS il build del progetto `opencit-trustagent` fallisce a causa di un errore dovuto alla mancanza di una dipendenza. Per risolvere tale problema eseguire i seguenti comandi:

```
$ cd
    /root/.m2/repository/com/intel/mtwilson/tbootxm/packages/tbootxm/3.2.1-SNAPSHOT/

$ mv tbootxm-3.2.1-SNAPSHOT-centos.bin tbootxm-3.2.1-SNAPSHOT-rhel.bin
```

Nel caso l'host sia Ubuntu o CentOS è disponibile l'utilizzo del primo scenario e quindi si possono eseguire i seguenti comandi

```
$ cd /root/opencit-trustagent

$ git checkout --track origin/ima_attestation

$ ant

$ cd /root/opencit

$ git checkout --track origin/ima_attestation

$ ant
```

Nel caso di host CentOS è disponibile il secondo scenario descritto in precedenza quindi per il build del progetto `opencit-trustagent` eseguire i seguenti comandi:

```
$ cd /root/opencit-trustagent

$ git checkout --track origin/centos_ima_att

$ ant

$ cd /root/opencit

$ git checkout --track origin/centos_ima_att

$ ant
```

Successivamente effettuare il build degli altri progetti attraverso i seguenti comandi:

```
$ cd /root/opencit-vrtm

$ git checkout v3.2.1

$ ant

$ cd /root/opencit-policyagent

$ git checkout v3.2.1

$ ant

$ cd /root/opencit-docker-proxy

$ git checkout v3.2.1

$ ant

$ cd /root/opencit-kms

$ git checkout v3.2.1

$ ant

$ cd /root/opencit-director

$ git checkout v3.2.1

$ ant
```

Solamente per host CentOS il build del progetto `opencit-openstack-extensions` fallisce a cause di un errore dovuto alla mancanza di una dipendenza. Per risolvere questo problema eseguire i seguenti comandi:

```
$cd /root/.m2/repository/com/intel/mtwilson/vrtm/packages/vrtm/3.2.1-SNAPSHOT

$mv vrtm-3.2.1-SNAPSHOT-centos.bin vrtm-3.2.1-SNAPSHOT-rhel.bin
```

Procedere quindi con l'esecuzione dei seguenti comandi:

```
$ cd /root/opencit-openstack-extensions

$ git checkout v3.2.1

$ ant

$ cd /root/opencit-openstack-controller-extensions

$ git checkout v3.2.1

$ ant

$ cd /root/opencit-attestation-cache-hub

$ git checkout v3.2.1

$ ant

$ cd /root/opencit-quickstart

$ git checkout v3.2.1

$ ant
```

A.3 Installazione di Open CIT

Ci sono due possibilità. È possibile installare l'Attestation Server, il Trust Director e il Trust Agent attraverso il *Deployment Wizard* oppure installare solo il Trust Agent se l'Attestation Server e il Trust Director sono già stati installati. Nel primo caso bisogna copiare nella cartella `/root/` gli eseguibili sotto la cartella `opencti-quickstart/packages/cit-quickstart-linux/target/` ed eseguire i seguenti comandi:

```
$ cd /root/

$ ./cit-quickstart-linux-3.2.1-SNAPSHOT.bin

$ ./cit-quickstart-linux-3.2.1-SNAPSHOT-integrity.bin

$ ./cit-quickstart-linux-3.2.1-SNAPSHOT-confidentiality.bin
```

Nel caso in cui venga mostrato il messaggio d'errore `Error while executing config: Specified protection parameters do not match encrypted data header` è necessario installare JCE attraverso i seguenti comandi

```
$ add-apt-repository ppa:webupd8team/java

$ apt update

$ apt install oracle-java8-unlimited-jce-policy
```

Si può accedere all'installer attraverso la url `http://indirizzo_ip_trust_agent`. Da qui è necessario effettuare le seguenti selezioni

1. **Environment:** Private Network (Enterprise or Datacenter)
2. **Features:** Container Integrity (Docker) e Integrations with OpenStack Nova & Horizon, Glance
3. **Layout:** One package per remote host, OpenStack Glance installed separately
4. **Settings:** Inserire la url `http://controller`, dove controller è l'hostname del controller OpenStack, come hostname di base per Glance e KeyStone. Inserire `glance` come username e la sua password, e `service` come tenant.
5. **Credentials:** Inserire gli indirizzi IP o gli hostname per ciascun componente (e.g. Attestation Server, Trust Director, Attestation Reporting Hub e OpenStack Extensions). Per ciascuno di essi inserire la password per accesso SSH come root e spuntare la checkbox `Accept this SSH key fingerprint`. Inoltre bisogna impostare `compute1` come hostname per il Trust Agent e la sua password di root.

Nel caso in cui si voglia installare esclusivamente il Trust Agent e gli altri componenti siano già stati installati è necessario copiare sotto la directory `/root/` l'eseguibile di installazione che si trova sotto la directory `/root/opencit-trustagent/packages/trustagent-linux/target/`. Nel caso di host CentOS il nome dell'eseguibile da copiare è `trustagent-linux-3.2.1-SNAPSHOT-rhel.bin`. Inoltre bisogna creare sempre sotto la directory `/root/` il file `trustagent.env`. Il contenuto che deve avere tale file è il seguente:

```
MTWILSON_API_URL=https://ip_attestation_server:8443/mtwilson/v2
MTWILSON_API_USERNAME=<PrivacyCA username, vedere file mtwilson.env sull'AS>
MTWILSON_API_PASSWORD=<PrivacyCA password, vedere file mtwilson.env sull'AS>
MTWILSON_TLS_CERT_SHA256=<SHA 256 certificato AS>
REGISTER_TPM_PASSWORD=y
MANIFEST_PATH="/boot/trust/manifest.xml"
GRUB_FILE=<percorso_file_grub.cfg>
TRUSTAGENT_LOGIN_REGISTER=true
```

Il digest SHA 256 del certificato dell'AS lo si può ricavare dall'esecuzione del comando `mtwilson tomcat-status`.

Per l'installazione del Trust Agent è necessario eseguire queste operazioni sia nel caso di installazione con *Deployment Wizard* che in caso di installazione del solo Trust Agent.

A.4 TPM ownership

È necessario effettuare l'operazione di `clear ownership` sul TPM. Di seguito sono riportati i passi da seguire per realizzare questa operazione su una macchina NUC, ma tali passi possono essere applicati in modo simile ad altre macchine.

1. Accedere al BIOS della macchina e diasbilitare Intel TXT;
2. Entrare in configurazione `Maintainence Mode` attraverso lo spostamento di un jumper di colore giallo;
3. Premere `4` per effettuare l'operazione di `clear ownership` del TPM;
4. Spegnere la macchina e riportare il jumper alla posizione di partenza;
5. Riavviare la macchina.

A.5 Installazione del modulo SINIT ACM

È necessario scaricare il modulo SINIT ACM per la propria macchina a seconda del tipo di processore. Per farlo bisogna trovare il *codename* del processore. Lo si può trovare eseguendo il comando `lscpu`. Questo comando fornisce il modello del processore che può essere utilizzato per cercare sul sito di Intel il *codename*. Successivamente bisogna scaricare il modulo SINIT ACM dalla url http://software.intel.com/en-us/articles/intel-trusted-execution-technology/, scegliendo il file da scaricare a seconda del proprio *codename*. Per la macchina NUC utilizzata come test il *codename* è *Broadwell* e il relativo modulo SINIT ACM da scaricare è 5th_gen_i5_i7_SINIT_79.zip. Una volta estratto il contenuto dal file zip, copiare il file binario sotto la directory `/boot/` del Trust Agent. Nel caso del NUC il nome del binario è `5th_gen_i5_i7_SINIT_79.BIN`.

A.6 Installazione dei package tss2 e tpm2-tools

Dalla directory opencit-trustagent/features/mtwilson-trustagent-tpm2-packages/src/resources/ copiare le directory `tss2` e `tpm2-tools` nella directory `/root/` ed eseguire gli script `setup.sh` contenuti nelle directory. Infine verificare che il servizio `tcsd2` è in esecuzione eseguendo il comando `systemctl status tcsd2`. Se non dovesse essere eseguito lanciare manualmente il binario `/usr/local/sbin/resourcemgr` e controllare gli errori.

A.7 Installazione del Trust Agent

Nel caso di host CentOS prima di installare il Trust Agent controllare che sia presente il file `grub.cfg` sotto la directory `/boot/grub2/` altrimenti è necessario generarlo tramite il comando:

```
$ grub2-mkconfig -o /boot/grub2/grub.cfg
```

Mentre per host UEFI controllare la presenza del file `grub.cfg` sotto la directory /boot/efi/EFI/centos/ altrimenti generarlo è necessario generarlo tramite il comando:

```
$ grub2-mkconfig -o /boot/efi/EFI/centos/grub.cfg
```

Inoltre sempre nel caso di host CentOS può esserci un errore dovuto alla non presenza dei comandi `multiboot2` e `module2`. Per risolvere il problema controllare se sotto la directory /usr/-lib/grub/x86_64-efi sono presenti i file `multiboot2.mod` e `relocator.mod`. Se sono presenti allora eseguire i seguenti comandi:

```
$ mkdir /boot/efi/EFI/centos/x86_64-efi

$ cp /usr/lib/grub/x86_64-efi/multiboot2.mod
    /boot/efi/EFI/centos/x86_64-efi/multiboot2.mod

$ cp /usr/lib/grub/x86_64-efi/relocator.mod
    /boot/efi/EFI/centos/x86_64-efi/relocator.mod
```

Se invece non sono presenti cercare il package che contiene `multiboot2.mod` ed installarlo. Per fare questo eseguire i seguenti comandi:

```
$ yum install yum-utils

$ repoquery --whatprovides '*multiboot2.mod'

$ wget url_pacchetto

$ rpm -ivh pacchetto.rpm
```

Il comando `repoquery` ritorna il nome del pacchetto che contiene multiboot2.mod, mentre il comando `rpm` installa il pacchetto.

Nel caso in cui sia stato eseguito il *Deployment Wizard*, sotto la directory /root/ dovrebbero essere presenti i file `cit3-openstack-trusted-node-ubuntu.bin` e `mtwilson-openstack.env`. Invece nel caso in cui si voglia installare solo il Trust Agent devono essere presenti sotto la directory /root/ l'eseguibile d'installazione e il file `trustagent.env`. In ogni caso eseguire gli eseguibili indicati. La prima esecuzione viene bloccata poiché richiede di riavviare la macchina in modalità *tboot*. Infatti riavviando la macchina dovrebbe comparire una entry nel menu di grub riguardante tboot. Quindi avviare il sistema in modalità tboot e rieseguire l'eseguibile precedente. Una volta terminata l'installazione è necessario copiare lo script `module_analysis_da.sh` sotto la directory `/opt/trustagent/bin/`. Questo risolve il problema nella creazione del file `measureLog.xml`. Riavviare nuovamente la macchina e selezionare la entry "TCB-Protection Ubuntu GNU/Linux".

A.8 Disinstallare Open CIT

Sull'Attestation Server eseguire i seguenti comandi:

```
$ mtwilson erase-data

$ mtwilson erase-users --all
```

```
$ mtwilson uninstall

$ director uninstall

$ attestation-hub uninstall

$ apt purge postgresql*
```

Sul Trust Agent (Ubuntu 16.04) eseguire i seguenti comandi:

```
$ rm /root/cit3-openstack-trusted-node-ubuntu.bin

$ tagent uninstall

$ apt purge tboot
```

Sul Trust Agent (CentOS 7) è necessario innanzitutto modificare lo script `tboot-xm-uninstall.sh` sotto la directory /opt/tbootxm/bin/ aggiungendo subito dopo la riga 66 le seguenti istruzioni:

```
elif [ $os_version == "centos" ] && [ -n "which grub2-install
    2>/dev/null" ] ; then
               grub2-install --version | grep " 2."
               GRUB_VERSION=2
               return
```

Inoltre bisogna modificare la riga 121 dello stesso script nel seguente modo:

```
if [ $os_version == "fedora" ] || [ $os_version == "rhel" ] || [ $os_version
    == "centos" ]; then
```

Quindi eseguire i seguenti comandi:

```
$ tagent uninstall

$ yum remove tboot

$ grub2-mkconfig -o /boot/efi/EFI/centos/grub.cfg  (solamente con macchina
    UEFI)

$ grub2-mkconfig -o /boot/grub2/grub.cfg    (solamente per macchina non UEFI)
```

A.9 Scenario d'uso

In questa sezione si descrivono i passi da seguire per avviare il processo di registrazione dell'host con la conseguente verifica delle misure. A partire dalla versione `3.2.1` di Open CIT per poter registrare un host è necessario innanzitutto creare una Trust Policy. Questa può essere creata solamente dal Trust Director dove per accedere è necessario creare un utente attraverso il seguente comando:

```
$ director password <username> <password> --permissions *:*
```

Una volta che l'utente è stato creato si può creare la Trust Policy seguendo i seguenti passi:

1. accedere alla URL del Trust Director "https://ip_trust_director:19444/" ed inserire le credenziali dell'utente creato;
2. selezionare "Trust Policy", quindi selezionare "Non-Virtualized Server" e premere "Add New Policy";
3. fornire un nome per la nuova Trust Policy così come i dettagli SSH (utente root e relativa password);
4. selezionare cartelle e file da includere nella Trust Policy.

Una volta creata la Trust Policy è possibile registrare l'host attraverso i seguenti passi:

1. accedere alla URL dell'Attestation Server "https://attestation_server_ip:8443/mtwilson-portal", effettuare il login e selezionare "Whitelist", quindi selezionare "Import from Trusted Host";
2. selezionare il tipo di host (Linux, Linux KVM, Citrix XenServer, VMware ESXi, Windows or Xen);
3. selezionare le MLE che si vogliono creare (BIOS, VMM o entrambe);
4. selezionare "IMA attestation";
5. inserire i dettagli di connessione all'host da attestare, ovvero nome dell'host e porta 1443;
6. lasciare come policy TLS quella di default;
7. selezionare il riquadro di registrazione dell'host e di sovrascrittura della Whitelist;
8. infine premere il pulsante "Import Whitelist".

Andando alla pagina della dashboard sarà possibile visualizzare l'host registrato e attraverso il report grafico la presenza delle misure IMA.

Appendice B

Manuale del programmatore

In questa sezione vengono illustrate le classi Java che sono state modificate per la realizzazione della soluzione proposta, suddividendo l'appendice in due sezioni rappresentati gli scenari realizzati.

B.1 Scenario: Whitelist statiche

È lo scenario in cui la verifica delle misure IMA avviene utilizzando la logica di verifica prevista da Open CIT, ovvero il confronto tra i valori provenienti dall'host ed i valori registrati nel DB come valori di Whitelist.

B.1.1 Classi del Trust Agent

Le classi coinvolte nelle modifiche del Trust Agent sono le seguenti:

TpmQuoteRequest: è la classe utilizzata per fare il parsing XML della richiesta, caratterizzata dai vari metodi di `get` e `set`. È stata modifica aggiungendo un valore booleano che permette di capire se è stata o meno richiesta attestazione con IMA. Tale classe si trova nel progetto `mtwilson-trustagent-model` di `opencit-trustagent`.

Tpm: è la classe che espone la risorsa REST `/tpm`. In particolare la richiesta della quota arriva al metodo `tpmQuote` che è il responsabile di effettuare tutto il processo di creazione della risposta come il calcolo della quote e la raccolta delle misure della Trust Policy. Tale metodo è stato modificato per creare un oggetto di tipo `RetrieveImaMeasurement` per recuperare le misure IMA. Tale classe si trova nel progetto `mtwilson-trustagent-ws-v2` di `opencit-trustagent`.

TrustAgentClient: è la classe responsabile di effettuare la richiesta POST alla risorsa `/tpm/quote`. Questa classe è stata modifica per inserire un valore booleano che consenta al TA di comprendere se nella risposta dovrà o meno includere le misure IMA. È possibile trovare questa classe nel progetto `mtwilson-trustagent-client-jaxrs2` di `opencit-trustagent`.

IMAReader: è una classe creata per recuperare le misure IMA dal file ascii_runtime_measurements. Ogni riga di questo file viene utilizzata per creare oggetti di tipo `FileMeasurementType` che vengono utilizzati per popolare l'oggetto di tipo `List` esposto dalla classe `IMAMeasurements`. Gli oggetti di tipo `FileMeasurementType` sono caratterizzati da un attributo path nella quale viene inserito il nome del file misurato e un attributo value nella quale viene inserito il `template-hash`. Le classi `FileMeasurementType` e `IMAMeasurements` sono generate automaticamente in fase di build a partire dallo schema XSD `IMA_Schema.xsd`. La classe `IMAReader` si trova all'interno del progetto `mtwilson-trustagent-ws-v2` di `opencit-trustagent`.

RetrieveImaMeasurement: è una classe creata appositamente per costruire l'oggetto di tipo `String` che conterrà il file XML delle misure IMA. Utilizza il metodo `execute` per creare il file XML a partire dall'oggetto di tipo `IMAMeasurements` costruito dalla classe `IMAReader`. Tale classe si trova nel progetto `mtwilson-trustagent-ws-v2` di `opencit-trustagent`.

TADataContext: è la classe che viene utilizzata per fornire alla classe `BuildQuoteXMLCmd` le informazioni utili per costruire la risposta che il TA deve inviare all'AS. Questa classe è stata modifica per recuperare le misure IMA a partire dalla classe `RetrieveImaMeasurement` ed è possibile trovarla nel progetto `mtwilson-trustagent-ws-v2` di `opencit-trustagent`.

TpmQuoteResponse: è la classe utilizzata per fare il parsing XML della riposta che il TA deve fornire all'AS. È stata modificata per consentire l'inserimento delle misure IMA ed è possibile trovarla nel progetto `mtwilson-trustagent-model` di `opencit-trustagent`.

B.1.2 Classi dell'Attestation Server

Le classi coinvolte nelle modifiche all'Attestation Server sono le seguenti:

PcrEventLogIntegrity: è la classe che viene utilizzata per effettuare il controllo sugli aggregati del PCR 17 e 18, ed è stata modificata per aggirare il problema legato al PCR 17, eludendo i controlli sull'aggregato dello stesso. Tale classe è presente nel progetto `mtwilson-trust-policy` di `opencit`.

TAHelper: è la classe che è caratterizzata dai metodi che consentono l'interazione con il TA. Tra questi c'è il metodo `getInformationForHost` che consente all'AS di richiedere la Quote al TA e che è stato modificato per inviare al TA un valore booleano che permetta di includere o meno le misure IMA nella risposta del TA. Inoltre è stato modificato il metodo `getHostAttestationReport` per consentire l'inserimento del PCR 10 all'interno del report XML. Tale classe è presente nel progetto `mtwilson-intel-hostagent` di `opencit`.

IntelHostAgent2: è la classe necessaria a creare un oggetto di tipo `PcrManifest` contente tutte le informazioni presenti nella risposta del TA, attraverso il metodo `getPcrManifest`. Tale classe contiene il metodo `getHostAttestationReport` modificato per verificare se per l'host da attestare è stata specificata o meno un'attestazione con IMA in modo da trasferire questa informazione all'oggetto di tipo `TAHelper` che provvederà a richiedere la quote all'host. Questa classe si trova nel progetto `mtwilson-intel-hostagent` di `opencit`.

PcrManifest: è la classe necessaria a contenere tutte le informazioni presenti nella riposta proveniente dal TA. È stata modificata per consentire l'inserimento nei propri oggetti delle misure IMA contenute nella risposta. Tale classe si trova nel progetto `mtwilson-api` di `opencit`.

MwImaMeasurementXml: è la classe utilizzata per rappresentare l'insieme delle misure IMA all'interno del DB. Infatti è caratterizzata da un oggetto di tipo `String` all'interno del qual viene inserito il file XML contenente le misure IMA. Tale classe è costruita secondo le specifiche JPA e si trova all'interno del progetto `mtwilson-attestation-jpa` di `opencit`.

MwImaMeasurementXmlJpaController: è la classe che funge da controller JPA per gestire oggetti di tipo `MwImaMeasurementXml` e consentirne la memorizzazione, cancellazione e aggiornamento nel DB. Tale classe si trova all'interno del progetto `mtwilson-attestation-jpa` di `opencit`.

MyJpa: è la classe utilizzata per permettere la creazione delle classi controller che verranno utilizzate per l'interazione con il DB. Questa classe è stata modificata per consentire la creazione di oggetti `MwImaMeasurementXmlJpaController` e si trova nel progetto `mtwilson-my` di `opencit`.

HostBO: è la classe che è caratterizzata dai metodi che interagiscono con gli oggetti utili alla creazione delle Whitelist in fase di registrazione. In particolare per la creazione delle Whitelist viene utilizzato il metodo `configureWhiteListFromCustomData`. Tale metodo è stato

modificato per invocare il metodo privato `configureImaMeasurementXmlLog` il cui compito è quello di consentire la memorizzazione delle misure IMA nel DB. Questa classe può essere trovata nel progetto `mtwilson-management` di `opencit`.

XmlImaMeasurementLogIntegrity: è una classe che implementa l'interfaccia `Rule` ed ha il compito di verificare l'aggregato delle misure IMA durante la fase di attestazione confrontandolo con il valore del PCR 10 presente all'interno della Quote. Tale classe si trova nel progetto `mtwilson-trust-policy` di `opencit`.

XmlImaMeasurementLogEquals: è una classe che implementa l'interfaccia `Rule` e che viene utilizzata per effettuare il confronto di ciascuna misura IMA con il rispettivo valore di Whitelist memorizzato all'interno del DB. Questa classe si trova nel progetto `mtwilson-trust-policy` di `opencit`.

IntelTpmDaHostTrustPolicyFactory: è la classe che fornisce i metodi per la creazione di oggetti le cui classi implementano l'interfaccia `Rule`. In particolare sono presenti i metodi `loadTrustRulesForBios` e `loadComparisonRulesForVmm` con quest'ultimo che è stato modificato per consentire la creazione di oggetti `XmlImaMeasurementLogIntegrity` e `XmlImaMeasurementLogEquals` se per lo specifico host si effettua attestazione con IMA. Tale classe è presente nel progetto `mtwilson-attestation` di `opencit`.

JpaPolicyReader: è una classe i cui metodi vengono utilizzati per la creazione di oggetti le cui classi implementano l'interfaccia `Rule`. In particolare per la creazione di oggetti di tipo `XmlImaMeasurementLogIntegrity` e `XmlImaMeasurementLogEquals` viene utilizzato il metodo `loadXmlImaMeasurementLogRuleForVmm`. La classe `JpaPolicyReader` si trova nel progetto `mtwilson-attestation` di `opencit`.

XmlImaMeasurementLog: è la classe che viene utilizzata per fare il parsing del file XML contenente le misure IMA salvato nel DB, con lo scopo di creare tanti oggetti di tipo `Measurement` quante sono le misure IMA contenute nel file XML. Tale classe si trova nel progetto `mtwilson-api` di `opencit`.

MleBO: è la classe che offre i metodi per la gestione delle Whitelist, compresa la memorizzazione, la cancellazione e l'aggiornamento del DB. Tale classe è stata modificata per consentire di cancellare le misure IMA dal DB quando viene cancellata la Whitelist. Inoltre è contenuta all'interno del package `mtwilson-whitelist` di `opencit`.

ReportsBO: è la classe che fornisce i metodi per la creazione del report grafico. In particolare è stato modificato il metodo `addManifestLogs` per consentire l'inserimento nel report delle misure IMA. Inoltre è stato modificato il metodo `getPcrModuleManifest` per consentire l'inserimento del PCR 10 nel report grafico. Tale classe appartiene al progetto `mtwilson-attestation` di `opencit`.

HostTrustBO: è la classe responsabile di effettuare l'attestazione dell'host, generare gli oggetti che la classe `ReportsBO` userà per la creazione del report, e la creazione dell'asserzione SAML in seguito al processo di attestazione. Il metodo `logPcrTrustStatus` è quello responsabile di creare gli oggetti che verranno utilizzati dalla classe `ReportsBO` ed è stato modificato per consentire la creazione di un oggetto `TblTaLog` per il PCR 10. Gli oggetti `TblTaLog` rappresentano un PCR, ma non quello il cui valore è stato memorizzato come valore di Whitelist, ma il PCR ricevuto nella quote in fase di attestazione. La classe `HostTrustBO` si trova all'interno del progetto `mtwilson-attestation` di `opencit`.

Tra gli altri file modificati si ha:

WhiteListConfiguration.jsp: è il file caratterizzato dal codice HTML necessario a costruire l'interfaccia grafica mostrata durante la registrazione dell'host. Tale file è stato modificato per aggiungere una checkbox che consentisse di selezionare per lo specifico host un'attestazione con IMA. Tale file si trova nel progetto `mtwilson-portal` di `opencit`.

WhiteListConfig.js: è un file Javascript che espone tutte quelle funzioni che vengono utilizzate per interagire con l'interfaccia grafica mostrata durante la registrazione dell'host. Tale file è stato modificato per aggiungere nella funzione `fnGetWhiteListConfigData` un insieme di istruzioni per consentire di comprendere se l'utente ha o meno richiesto attestazione con IMA. Tale file si trova nel progetto `mtwilson-portal` di `opencit`.

B.2 Scenario: uso del Verifier esterno

Il secondo scenario è quello in cui l'AS verifica solo che l'aggregato delle misure IMA corrisponda al valore del PCR 10 contenuto nella quote, mentre la verifica di ciascuna misura viene effettuata utilizzando un Verifier esterno.

Le modifiche riguardanti il Trust Agent hanno coinvolto solamente la classe `IMAReader` che viene utilizzata per recuperare le misure IMA dal file ascii_runtime_measurements. Ogni riga di questo file viene utilizzata per creare oggetti di tipo `FileMeasurementType` che vengono utilizzati per popolare l'oggetto di tipo `List` esposto dalla classe `IMAMeasurements`. Gli oggetti di tipo `FileMeasurementType` sono caratterizzati da un attributo path nella quale viene inserito il nome del file misurato e un attributo value nella quale viene inserito il `filedata-hash`. Le classi `FileMeasurementType` e `IMAMeasurements` sono generate automaticamente in fase di build a partire dallo schema XSD `IMA_Schema.xsd`. La classe `IMAReader` si trova all'interno del progetto `mtwilson-trustagent-ws-v2` di `opencit-trustagent`.

Le modifiche riguardanti l'Attestation Server hanno riguardato le seguenti classi:

XmlImaMeasurementLogIntegrity: è una classe che implementa l'interfaccia `Rule` ed ha il compito di verificare l'aggregato delle misure IMA durante la fase di attestazione confrontandolo con il valore del PCR 10 presente all'interno della Quote. Per il secondo scenario è stato aggiunto un metodo privato che consentisse il calcolo del `template-hash` a partire dal `filedata-hash`, operazione necessaria poiché il calcolo dell'aggregato si basa sul `template-hash`. Tale classe si trova nel progetto `mtwilson-trust-policy` di `opencit`.

ReportsBO: è la classe che fornisce i metodi per la creazione del report grafico. In particolare è stato modificato il metodo `addManifestLogs` per consentire l'inserimento nel report delle misure IMA senza l'inserimento dei valori di Whitelist associati poiché non previsti dallo scenario. ale classe appartiene al progetto `mtwilson-attestation` di `opencit`.

HostTrustBO: è la classe responsabile di effettuare l'attestazione dell'host, generare gli oggetti che la classe `ReportsBO` userà per la creazione del report, e la creazione dell'asserzione SAML in seguito al processo di attestazione. Il metodo `getTrustReportForHost` è uno dei metodi principali coinvolti nel processo di attestazione in quanto effettua le operazioni necessarie a richiedere la quote e a verificare le misure ricavate dall'host. Tale metodo è stato modificato per consentire l'aggiornamento delle misure IMA presenti nel DB. La classe `HostTrustBO` si trova all'interno del progetto `mtwilson-attestation` di `opencit`.

Infine per consentire la verifica delle misure IMA per mezzo del Verifier esterno, è stata creata una classe `Python` chiamata `DriverCit` che è caratterizzata da tre metodi:

registerNode: è il metodo responsabile di effettuare la registrazione dell'host.

pollHost: è il metodo responsabile di effettuare l'attestazione dell'host, recuperare le misure IMA e trasferirle alla logica di verifica del Verifier.

getStatus: è il metodo che consente di capire se l'Attestation Server è attivo ed è pronto a ricevere richieste.

Bibliografia

[1] Mell, P. and T. Grance, "The NIST definition of cloud computing", 2011, Computer Security Division, Information Technology Laboratory, National Institute of Standards and Technology Gaithersburg

[2] Microsoft, "Microsoft Security Intelligence Report. January through March, 2017", August 2017, http://download.microsoft.com/download/F/C/4/FC41DE26-E641-4A20-AE5B-E38A28368433/Security_Intelligence_Report_Volume_22.pdf

[3] Cloud Security Alliance, "The Treacherous 12. Cloud Computing Top Threats in 2018", February 2017, https://downloads.cloudsecurityalliance.org/assets/research/top-threats/treacherous-12-top-threats.pdf

[4] E.Shi, A.Perrig, L.V.Doorn, "BIND: A Fine-Grained Attestation Service for Secure Distributed Systems", 2005, IEEE Symposium on Security and Privacy, Washington (DC, USA), May 8-11, 2005, pp. 154-168, DOI 10.1109/SP.2005.4

[5] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, D. Boneh, "Terra: a virtual machine-based platform for trusted computing", SIGOPS Operating System Review, Vol. 37, No. 5, October 2003, pp. 193-206, DOI 10.1145/1165389.945464

[6] N. L. Petroni,Jr., T. Fraser, J. Molina, W. A. Arbaugh, "Copilot - a coprocessor-based kernel runtime integrity monitor", 13th conference on USENIX Security Symposium, San Diego (CA, USA), August 9-13, 2004, pp. 179-194

[7] TCG Infrastructure Working Group, "Reference Architecture for Interoperability (Part I), specification version 1.0", June 2005, https://trustedcomputinggroup.org/wp-content/uploads/IWG_Architecture_v1_0_r1.pdf

[8] Trusted Computing Group, "ISO/IEC 11889-1:2015 Information technology - Trusted Platform Module - Part 1: Overview", ISO/IEC 11889-1, August 2015

[9] The Open Attestation project, https://01.org/blogs/2014/openattestation-oat-project

[10] Open Cloud Integrity Technology (Open CIT), https://01.org/opencit

[11] Integrity Measurement Architecture, https://sourceforge.net/p/linux-ima/wiki/Home/

[12] Trusted Computing Group, "Design, Implementation, and Usage Principles Version 2.0", December 2005, https://www.trustedcomputinggroup.org/wp-content/uploads/Best_Practices_Principles_Document_V2_0.pdf

[13] B.Balacheff, L.Chen, D.Plaquin, G.Proudler "A trusted process to digitally sign a document", Proceedings of the 2001 workshop on New security paradigms, Cloudcroft, New Mexico, September 10-13, 2001, pp. 79-86, DOI 10.1145/508171.508184

[14] A.Spalka, A.B.Cremers, H.Langweg "Protecting the Creation of Digital Signatures with Trusted Computing Platform Technology Against Attacks by Trojan Horse Programs", Trusted Information. SEC 2001. IFIP International Federation for Information Processing, Boston, MA, 2001, DOI 10.1007/0-306-46998-7_28

[15] S.E.Schechter, R.A.Greenstadt, M.D.Smith, "Trusted Computing, Peer-to-Peer Distribution, and The Economics of Pirated Entertainment", Economics of Information Security. Advances in Information Security, Boston, MA, 2004, DOI 10.1007/1-4020-8090-5_5

[16] M.Kinateder, S.Pearson, "A Privacy-Enhanced Peer-to-Peer Reputation System", E-Commerce and Web Technologies. EC-Web 2003. Lecture Notes in Computer Science, 2003, DOI 10.1007/978-3-540-45229-4_21

[17] A.Pashalidis, C.J.Mitchell, "Single Sign-On Using Trusted Platforms", Information Security. ISC 2003. Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2003, DOI 10.1007/10958513_5

[18] L.Chen, S.Pearson, A.Vamvakas, "On enhancing biometric authentication with data protection", Proceedings of the Fourth International Conference on Knowledge-Based Intelligent Engineering System and Allied Technologies, Los Alamitos, CA, 2000, pp. 249-252, DOI 10.1109/KES.2000.885804

[19] M.C.Mont, S.Pearson, P.Bramhall, "Towards accountable management of identity and privacy: sticky policies and enforceable tracing services", 14th International Workshop on Database and Expert Systems Applications, Prague, Czech Republic, 2003, pp. 377-382, DOI 10.1109/DEXA.2003.1232051

[20] J.S.Erickson, "Fair use, DRM, and trusted computing", Communications of the ACM - Digital rights management, Volume 46 Issue 4, April 2003, Pages 34-39 DOI 10.1145/641205.641228

[21] A.Alsaid, C.J.Mitchell, "Preventing phishing attacks using trusted computing technology", Proceedings of the 6th International Network Conference (INC 06), 2006, pp. 221-228

[22] W.Arthur, D.Challener, K.Goldman, "History of the TPM" nel libro "A Practical Guide to TPM 2.0" a cura di Apress, Berkeley, CA, 2015, pp. 1-4, DOI 10.1007/978-1-4302-6584-9_22

[23] D.Eastlake, P.Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC-3174, DOI 10.17487/RFC3174

[24] Trusted Computing Group, "TCG D-RTM Architecture", June 2013, https://trustedcomputinggroup.org/wp-content/uploads/TCG_D-RTM_Architecture_v1-0_Published_06172013.pdf

[25] R.Rivest, A.Shamir, L.Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", Communications of the ACM, pp. 120-126, February 1978

[26] S.Blake-Wilson, N.Bolyard, V.Gupta, C.Hawk, B.Moeller "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC-4492, DOI 10.17487/RFC4492

[27] D.Eastlake, T.Hansen, US Secure Hash Algorithms (SHA and HMAC-SHA), RFC-4634, DOI 10.17487/RFC4634

[28] Trusted Computing Group, "TCG EK Credential Profile For TPM Family 2.0; Level 0", November 2014, https://trustedcomputinggroup.org/wp-content/uploads/Credential_Profile_EK_V2.0_R14_published.pdf

[29] D.Cooper, S.Santesson, S.Farrell, S.Boeyen, R.Housley, W.Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC-5280, DOI 10.17487/RFC5280

[30] Trusted Computing Group, "TCG Credential Profiles For TPM Family 1.2 Level 2", July 2003, https://trustedcomputinggroup.org/wp-content/uploads/Credential_Profiles_V1.2_Level2_Revision8.pdf

[31] L.Chen, B.Warinschi, "Security of the TCG Privacy-CA Solution", IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, Hong Kong, China, 11-13 Dec. 2010, DOI 10.1109/EUC.2010.98

[32] Trusted Computing Group, "TCG Software Stack (TSS) Specification Version 1.2, Level 1, Errata A", March 7 2007, https://trustedcomputinggroup.org/wp-content/uploads/TSS_1_2_Errata_A-final.pdf

[33] Trusted Computing Group, "TCG Software Stack Feature API", November 7 2014, https://trustedcomputinggroup.org/wp-content/uploads/TSS-Feature-API-version-.12_Review.pdf

[34] Trusted Computing Group, "TCG TSS 2.0 Enhanced System API (ESAPI) Specification", August 24 2017, https://trustedcomputinggroup.org/wp-content/uploads/TSS_TSS-2.0-Enhanced-System-API_V0.9_R03_Public-Review-1.pdf

[35] Trusted Computing Group, "TSS System Level API and TPM Command Transmission Interface Specification", January 26 2015, https://trustedcomputinggroup.org/wp-content/uploads/TSS-system-API-01.pdf

[36] Trusted Computing Group, "TSS TAB and Resource Manager Specification", February 3 2015, https://trustedcomputinggroup.org/wp-content/uploads/TSS-TAB-and-Resource-Manager-00-91-PublicReview.pdf

[37] Trusted Computing Group, "TCG Infrastructure Working Group Architecture Part II - Integrity Management", November 17 2006, https://trustedcomputinggroup.org/wp-content/uploads/IWG_ArchitecturePartII_v1.0.pdf

[38] Trusted Computing Group, "TCG Infrastructure Working Group Reference Architecture for Interoperability (Part I)", June 16 2005, https://trustedcomputinggroup.org/wp-content/uploads/IWG_Architecture_v1_0_r1.pdf

[39] G.Coker, J.Guttman, P.Loscocco et al., "Principles of remote attestation", International Journal of Information Security, Volume 10, Issue 2, pp. 63-81, 2011, DOI 10.1007/s10207-011-0124-7

[40] Trusted Computing Group, "TCG Infrastructure Working Group Integrity Report Schema", August 24 2011, https://trustedcomputinggroup.org/wp-content/uploads/IWG_Integrity_Report_Schema_v2.0.r5.pdf

[41] Trusted Computing Group, "TCG Infrastructure Working Group Reference Manifest (RM) Schema Specification", August 24 2011, https://trustedcomputinggroup.org/wp-content/uploads/Reference_Manifest_Schema_Specification_v2.0.r5.pdf

[42] N.F.B.Awang, "Trusted Computing - Opportunities & Risk", Proc. 5th International Conference on Collaborative Computing: Networking Application and Worksharing, Washington, DC, USA, May 11-14 Nov. 2009, pp. 1-5, DOI 10.4108/ICST.COLLABORATECOM2009.8402

[43] R.Sailer, X.Zhang, T.Jaeger, L.van Doorn, "Design and implementation of a TCG-based integrity measurement architecture", 13th conference on USENIX Security Symposium - Volume 13, San Diego (CA, USA), August 9-13, 2004, pp. 223-238

[44] D.D.Clark, D.R.Wilson, "A Comparison of Commercial and Military Computer Security Policies", 1987 IEEE Symposium on Security and Privacy, Oakland (CA, USA), April 27-29, 1987, pp. 184-194, DOI 10.1109/SP.1987.10001

[45] C.Wright, C.Cowan, S.Smalley, J.Morris, G.Kroah-Hartman, "Linux Security Modules: General Security Support for the Linux Kernel", 11th USENIX Security Symposium, San Francisco (CA, USA), August 5-9, 2002, pp. 17-31

[46] Intel, "Intel Trusted Execution Technology", https://www.intel.com/content/www/us/en/architecture-and-technology/trusted-execution-technology/trusted-execution-technology-security-paper.html

[47] OpenStack project, https://www.openstack.org

[48] W.Arthur, D.Challener, K.Goldman, "Platform Security Technologies That Use TPM 2.0" nel libro "A Practical Guide to TPM 2.0", Apress, 2015, pp. 331-348, DOI 10.1007/978-1-4302-6584-9_22

[49] Trusted Boot project, https://sourceforge.net/projects/tboot/

[50] N.Santos, R.Rodrigues, K.P. Gummadi, S.Saroiu, "Excalibur: Building Trustworthy Cloud Services" Technical Report MPI-SWS-2011-004, MPI-SWS, 2011

[51] F.Wang, Y.Joung, J.Mickens, "Cobweb: Practical Remote Attestation Using Contextual Graphs", 2017

[52] Intel, "Intel Software Guard Extensions project", https://01.org/intel-softwareguard-extensions

[53] European Commission, "The SECURED project (SECURity at the network EDge)", https://www.secured-fp7.eu

[54] Docker, https://www.docker.com