UNIVERSITÀ DI PISA

FACOLTÀ DI SCIENZE MATEMATICHE FISICHE E NATURALI

CORSO DI LAUREA SPECIALISTICA IN TECNOLOGIE INFORMATICHE

ATTESTAZIONE REMOTA DELL'INTEGRITÀ SEMANTICA DI UN SISTEMA

Tesi di Laurea Specialistica

Diego Cilea

Relatore: Prof. Fabrizio Baiardi

Relatore: Ing. Luca Guidi

Controrelatore: Prof. Maurizio Bonuccelli

ANNO ACCADEMICO 2007-2008

UNIVERSITÀ DI PISA

FACOLTÀ DI SCIENZE MATEMATICHE FISICHE E NATURALI

CORSO DI LAUREA SPECIALISTICA IN TECNOLOGIE INFORMATICHE

ATTESTAZIONE REMOTA DELL'INTEGRITÀ SEMANTICA DI UN SISTEMA

Tesi di

Diego Cilea

Relatore:

Prof. Fabrizio Baiardi . . . . . .

Relatore:

Ing. Luca Guidi . . . . . .

Controrelatore:

Prof. Maurizio Bonuccelli . . . . . .

Candidato:

Diego Cilea . . . . . .

UNIVERSITÀ DI PISA

13 FEBBRAIO 2009

Sommario

Negli ultimi anni si è avuta una significativa evoluzione nel caso dei sistemi di controllo industriali. Infatti, si è modificata la situazione in cui il controllo di risorse e dispositivi avveniva manualmente con personale in loco. L'innovazione tecnologica permessa dall'informatica ha permesso di controllare tali risorse da remoto, con un effettivo risparmio di gestione ed una capacità elevata di raccolta di informazioni in tempo reale sullo stato dei sistemi da monitorare. Un esempio di tali sistemi di controllo sono i sistemi SCADA, particolarmente diffusi in ambiti quali la distribuzione di beni di primaria importanza come energia elettrica e forniture idriche. Per la loro importanza tali infrastrutture sono dette *Infrastrutture Critiche*. La veloce diffusione di sistemi di controllo ed il loro repentino inserimento nel contesto di gestione delle risorse "critiche", ha però comportato una serie di nuovi rischi. Un esempio di tali rischi è l'accesso remoto illegale di utenti a risorse critiche presenti ad esempio in una Intranet aziendale.
Le politiche di protezione basate sull'autenticazione e controllo degli accessi applicate di norma a protezione di queste stesse reti non possono essere adeguate se consideriamo che una percentuale elevata di attacchi è generata da sistemi "conosciuti" cioè correttamente autenticati che, nella maggioranza dei casi ed all'insaputa del possessore, eseguono operazioni che portano alla modifica od all'ottenimento di informazioni riservate.
Il framework descritto in questa tesi considera un caso, molto frequente in ambito SCADA, di servizio per l'accesso remoto a risorse e cerca di garantire la massima sicurezza nella sua implementazione. Esso, infatti, permette ad un host remoto fornitore di servizi di verificare l'integrità di un client,implementato mediante virtualizzazione, prima e durante la connessione ad una rete od a risorse. L'approccio adottato integra tecniche di introspezione e l'implementazione di un modello di trust tra i componenti per poter definire strumenti che effettuano misure affidabili sull'host monitorato.
La tesi dimostra come la gestione degli accessi remoti ad un sistema SCADA richieda l'attestazione remota dell'integrità di client remoti e propone un modello teorico di trust, un insieme di architetture e un protocollo di attestazione remota basato su tecnologie di virtualizzazione. Viene inoltre descritta una prima realizzazione prototipale del protocollo e dell'architettura di cui si discutono le prestazioni.

Indice

Elenco delle figure

Elenco delle tabelle

Introduzione

Il lavoro descritto in questa tesi è centrato sull'analisi del problema dell'integrità di un sistema nel contesto della sicurezza di infrastrutture critiche. In questo contesto, non sono rare le occasioni di attacco da parte di utenti o software ostili e, la particolare caratteristica di criticità di tali infrastrutture richiede la garanzia di un adeguato livello di sicurezza che non può utilizzare solo tecniche standard. Partendo da questa analisi, si è definito un framework che permette di assicurare l'affidabilità di un host che vuole eseguire operazioni o accedere ad una o più risorse protette. Le linee principali che hanno guidato il lavoro di tesi sono state il concetto di *trust* e di *misura* d'integrità di un sistema, la *virtualizzazione* e l'*attestazione remota* delle proprietà di un sistema. Questi concetti hanno costituito le basi per la progettazione e realizzazione di *Virtual machine Integrity Measurement System*(VIMS), una versione prototipale di un insieme di architetture che verificano l'integrità semantica di un sistema mediante strumenti di misurazione quali introspezione di macchine virtuali e componenti di trusted computing. L'architettura utilizza anche un protocollo di attestazione remota dell'integrità. Nel seguito descriviamo brevemente questi concetti ed il lavoro svolto per la tesi.

Trusted Computing

Negli ultimi anni è emerso nella comunità scientifica un framework per la misura dell'integrità promosso da un consorzio di industrie, il Trusted Computing Group(TCG), che definisce le specifiche per componenti hardware e software che devono garantire un determinato livello di affidabilità [56]. La misura dei componenti critici definita da questi standard avviene mediante il calcolo di un hash crittografico di tipo SHA-1 dei componenti prima che essi possano essere caricati ed eseguiti sul sistema. Il fondamento del sistema di verifiche ideato dal TCG è il cosiddetto Trusted Platform Module(TPM), un dispositivo hardware che possiede capacità crittografiche e una piccola memoria dove custodisce in sicurezza i valori delle misure effettuate

sul sistema.
Questa architettura, integrata con una libreria software che gestisce le misure, fornisce un buon modello per la determinazione dell'integrità del sistema e del software eseguito durante la sua inizializzazione. Inoltre, il TPM sta diventando nel tempo uno standard, infatti, è sempre più presente su personal computer di ultima generazione.
Tuttavia, questo framework soffre di alcuni problemi. Il principale è quello dell'integrità del software a tempo di esecuzione, non permettendo di rilevare attacchi successivi al caricamento del software. Se l'ambiente d'esecuzione del sistema può essere attaccato mediante le sue interfacce hardware e software, il problema non può essere risolto solamente mediante l'analisi e la misura dell'integrità dei file eseguibili, ma richiede periodiche misure d'integrità dei processi eseguiti e presenti in memoria [49]. Questa proprietà pone nuove problemi perchè, prima di tutto, alcuni dati dei processi eseguiti sul sistema variano consistentemente durante tutto l'arco di esecuzione e questo implica hash diversi in misure effettuate in tempi diversi. Infine, alcuni cambiamenti nei dati a tempo di esecuzione e l'integrità della rappresentazione non può essere garantita solo mediante hash.

TPM e vTPM

I sistemi forniti del chip TPM [19] [41] lo utilizzano come *root-of-trust* verificatrice dei componenti caricati nel sistema durante il processo di start-up dell'ambiente software. Esso può assicurare che un sistema ed il software da esso eseguito su di esso siano in uno stato corretto e non compromesso. Inoltre, per poter implementare questo meccanismo, gestisce delle coppie di chiavi asimmetriche e dei registri di memoria volatile che memorizzano informazioni sulla affidabilità del sistema.
Per poter interagire con i livelli software, il TPM definisce delle API che permettono di richiedere operazioni da esso implementate. Un esempio di queste operazioni eseguite sui registri del TPM sono la *extend* e la *quote*. La prima ha un valore `V` come input e calcola l'hash di tipo SHA-1 del contenuto corrente del registro esteso con `V`, memorizzando il risultato nel medesimo registro. La seconda, invece, genera un messaggio cifrato dalla chiave privata del TPM con il contenuto dei registri.
Le funzionalità del TPM possono essere estese in un ambiente virtualizzato, dove più macchine virtuali interagiscono condividendo lo stesso componente hardware. Se presente il TPM, una generica macchina virtuale può accedere al proprio TPM virtuale mediante il vTPM [21] [34] che permette al VMM di emulare le funzionalità del TPM, esportando una interfaccia condivisa verso tutte le macchine virtuali.

Attestazione remota

L'attestazione remota [11] [19] è un metodo mediante il quale due componenti, quello che misura l'integrità di un sistema e quello che autentica tale misura, interagiscono con un partner remoto per comunicare lo stato del sistema che misurano. L'obiettivo dell'attestazione è di fornire, in remoto, la prova che un sistema ha una certa configurazione e che tale dato possa essere assunto affidabile. In questo modo si può evitare che informazioni riservate siano ricevute da un sistema non sicuro od infetto, o che il comportamento di tale sistema non sia conforme a quanto atteso. Per poter implementare una attestazione remota è necessario stabilire l'affidabilità delle misure. A questo scopo, il concetto di *root of trust* introdotto dal TPM può fornire l' affidabilità richiesta da una corretta implementazione [26]. L'attestazione remota, in generale, viene implementata mediante crittografia a chiave pubblica, in modo tale che le informazioni possano essere lette solo dai componenti dell'architettura di attestazione.

Virtualizzazione

La virtualizzazione [68] è, ad oggi, un concetto molto vasto e denota, in generale, una versione virtuale, cioè astratta, di una determinata risorsa fisica. Le tecnologie di virtualizzazione hanno come componente portante un livello software che permette di gestire, in modo uniforme, un insieme di risorse differenti.
Nel caso in cui le risorse da astrarre siano componenti hardware, le tecnologie di virtualizzazione vengono realizzate a livello del sistema operativo tramite l'introduzione del Virtual Machine Monitor (VMM). Esempi molto noti sono VMWare [12], QEmu [8] e Xen [13] [32]. Il VMM è un livello software che si frappone tra gli ambienti di esecuzione virtuali, detti Virtual Machine, VM, ed i componenti hardware, creando un livello ulteriore di interfaccia. Esso astrae le risorse fisiche permettendo ad ogni sistema virtuale di poterne usufruire con gli stessi privilegi degli altri presenti sullo stesso host fisico. Questa tecnologia, diffusasi negli ultimi anni, offre numerosi benefici, tra cui:

- Consolidamento, cioè l'impiego di un numero minore di macchine fisiche;
- Migrazione, cioè il bilanciamento del carico tra server in modo rapido e sicuro;
- Isolamento e protezione, cioè la garanzia che i sistemi eseguiti sulla stessa macchina fisica non interferiscano tra loro.

Introspezione delle macchine virtuali

L'introspezione delle macchine virtuali [37] è una tecnica di verifica *trasparente* dello stato di un sistema in esecuzione in un ambiente virtualizzato.
Lo stato del sistema è infatti incapsulato in una VM e ciò permette a questa tecnica di poterlo analizzare tale stato in dettaglio. L'analisi dello stato avviene mediante accesso diretto alle locazioni di memoria assegnate alla VM od a registri della CPU *virtuale* assegnata al sistema.
Lo strumento che permette di effettuare tali verifiche è il Virtual Machine Monitor, il componente che gestisce l'ambiente di esecuzione su cui sono eseguite le macchine virtuali. Mediante delle opportune interfacce, il VMM permette di esaminare lo stato della macchina virtuale che si vuole monitorare da un punto di vista "esterno" e da un livello più basso.
In precedenza, l'introspezione poteva essere implementata solo mediante componenti hardware, il vantaggio che introduce la virtualizzazione è quello di una implementazione via software, rendendo il processo di verifica più flessibile e potente.

Vims framework

Virtual machine Integrity Measurement System (VIMS) è una architettura che implementa un approccio di misura dell'integrità e che può essere adottato per proteggere le reti mediante il controllo degli estremi delle connessioni. VIMS valuta l'integrità degli host remoti mediante l'applicazione di una o più politiche di sicurezza prima che la piattaforma remota possa connettersi ad una rete esistente e possa richiedere determinati servizi. Più in dettaglio, VIMS può garantire l'integrità di un sistema client che tenta di connettersi ad una rete. La nozione di integrità adottata include non solo la corretta configurazione del sistema e del software che esso esegue, ma anche che il client non esegua software ostile, come malware o virus, che possa modificare a tempo di esecuzione il comportamento del sistema o di singoli processi. A questo scopo, VIMS sfrutta la tecnologia di virtualizzazione per permettere al client di eseguire due macchine virtuali gestite da un VMM, la *Client-VM* e la *Assurance-VM*. La *Client-VM* esegue un sistema general purpose, utilizzabile ed estendibile, che possiede, tra i suoi processi, un software client VPN o un browser. La *Assurance-VM*, una macchina virtuale privilegiata e nascosta all'utente, che applica una serie di controlli di sicurezza sulla memoria della Client-VM mediante introspezione. Questi controlli misurano, per conto di un server *Remote-S*, l'integrità del software eseguito dalla Client-VM. La Assurance-VM può applicare questi controlli di consistenza periodicamente o su richiesta esplicita di Remote-S e la loro complessità è funzione del livello di affidabilità che il Remote-S richiede. Appena la Assurance-VM rileva un comportamento anomalo, o un processo ostile, ad esempio un rootkit nella memoria della Client-VM,

essa contatta il Remote-S mediante un protocollo di attestazione sicuro. A questo punto, il Remote-S applicherà la sua politica sulla connessione con Client-VM.
VIMS aggiunge inoltre una significativa innovazione mediante un modello formale di trust tra i componenti dell'architettura descritta, che ha portato ad estendere la *trusted computing base* del framework realizzato fino ai livelli inferiori dell'ambiente di esecuzione[71]. Essa, infatti, basa la sua affidabilità sull'implementazione e l'utilizzo di una *root of trust* hardware.

Struttura della tesi

Nei prossimi capitoli verranno descritti approfonditamente i concetti fin qui introdotti. La struttura della tesi è organizzata nella maniera seguente:

Capitolo 1 - Sicurezza nelle infrastrutture critiche Viene data una panoramica del contesto di gestione di sistemi SCADA e delle sue problematiche di sicurezza.

Capitolo 2 - Integrità ed affidabilità di un sistema Questo capitolo presenta una dettagliata analisi di come è affrontato il problema dell'integrità di un sistema nell'ambito di ricerca della sicurezza informatica.

Capitolo 3 - Xen ed il controllo d'integrità delle Macchine Virtuali Capitolo introduttivo della tecnologia di virtualizzazione e dell'hypervisor Xen, che ne descrive gli aspetti essenziali dell'architettura per poi descrivere diverse tecniche di verifica dell'integrità delle macchine virtuali.

Capitolo 4 - Architettura di VIMS Questo capitolo esamina l'architettura del framework VIMS. Inoltre, viene descritto un primo prototipo di VIMS realizzato in Java che utilizza Xen come tecnologia di virtualizzazione, due diverse librerie di introspezione e una libreria di accesso ad un TPM.

Capitolo 5 - Test e conclusioni Questo capitolo presenta i test eseguiti per verificare l'efficacia e le performance del prototipo di VIMS. Presenta inoltre le prime conclusioni del lavoro svolto e propone degli spunti per future estensioni.

Capitolo 1

La sicurezza nelle infrastrutture critiche

Indice

Il tema della security in ambiente IT è ormai da diversi anni un tema all'attenzione dei responsabili delle infrastrutture informatiche. Solo negli ultimi tempi, con il proliferare delle tecnologie IT anche in ambienti che utilizzano sistemi di controllo di tipo SCADA(Supervisory Control and Data Acquisition) finora considerati al sicuro da incidenti come intrusioni informatiche e perdita di dati, si è iniziato a guardare con molto interesse alla tematica della sicurezza. Da sempre, infatti, i sistemi SCADA sono stati pensati per essere molto semplici da utilizzare, ma anche affidabili, robusti ed efficienti, e spesso, non essendo stati progettati per interagire e comunicare con altri sistemi(anche remoti), risultano deboli dal punto di vista della sicurezza informatica. Dopo i fatti dell'11 Settembre 2001, principalmente negli Stati Uniti ed in Europa,

l'affacciarsi della minaccia terroristica ha evidenziato la necessità di esaminare criticamente questo ambito per la messa in sicurezza delle infrastrutture critiche, pubbliche e private. Nel seguito verranno descritti i più diffusi sistemi per reti SCADA per gli impianti di produzione, le strategie di implementazione e le relative vulnerabilità, analizzeremo infine le linee guida per poterle eliminare ed i problemi ancora aperti.

1.1 Reti e sistemi di controllo degli impianti

In uno stabilimento, impianti e macchinari sono gestiti da sistemi di automazione e controllo che sempre più spesso vengono collegati in rete per le interazioni necessarie ai processi controllati. Queste reti sono a loro volta connesse alla rete amministrativa aziendale, principalmente per la raccolta dati utilizzati sia per la gestione a scopo commerciale. In questa sezione cercheremo di introdurre in breve i principali dispositivi che appartengono alle reti degli impianti e le comuni applicazioni di questi sistemi.

Programmable Logic Controller(PLC). I PLC sono controllori basati su microprocessore ed eseguono, in genere, task ripetitivi. Sono dotati di schede di I/O a cui possono essere collegati sensori ed attuatori installati sull'impianto o sul macchinario associato al PLC. Gli I/O possono essere remoti, e spesso utilizzano Ethernet, e gli stessi PLC possono comunicare tra di loro mediante lo stesso protocollo. I diversi produttori di PLC, fino a pochi anni fa [75] utilizzavano solo protocolli proprietari, la cui implementazione rimaneva "oscura" agli utilizzatori. Da qualche tempo però è emersa, grazie all'intervento di enti statunitensi come il NERC ed il NIST, la necessità di una standardizzazione di tali protocolli che ha portato i produttori ad utilizzare pochi protocolli ormai diventati standard di mercato o di fatto, alcuni esempi sono ModBUS [6], OPC [7] e DNP3 [4].
La molteplicità dei protocolli utilizzati in impianti non ancora aggiornati, spesso nasconde ancora molte delle vulnerabilità più diffuse nelle reti di controllo.

Human Machine Interface(HMI) e Supervisory Control and Data Acquisition(SCADA). Sono le postazioni utilizzate dagli operatori per la verifica dello stato dell'impianto mediante i dati inviati dai PLC che forniscono una interfaccia grafica di raccolta dei dati provenienti dai sensori dell'impianto ed espongono interfacce per l'azionamento degli attuatori.
Gli HMI sono i cosiddetti "pannelli operatore", dotati di schermo e basati su microprocessore, fino a poco tempo fa con sistemi operativi proprietari, ma ultimamente sviluppati con versioni embedded di Windows o Linux che supportano applicativi di mercato adatti alla presentazione

dei dati all'operatore. Sono collegati ai PLC mediante interfacce ethernet o seriali ed utilizzano i medesimi protocolli. SCADA e' l'acronimo di una classe di pacchetti software applicativi che consentono di comporre un'interfaccia intelligente per la gestione di uno o più PLC mediante un PC. Utilizzano sistemi operativi di tipo Windows ed comunicano mediante il protocollo OPC(basato su DCOM), standard sponsorizzato da Microsoft e da alcuni produttori. OPC viene utilizzato principalmente per rendere più semplice la configurazione e la gestione dei sistemi utilizzati.

Distributed Control System(DCS). Sono sistemi di controllo integrati e distribuiti formati da controllori e Pc a cui lavorano gli operatori, ideati per impianti dove la criticità di determinate operazioni impone vincoli elevati di sicurezza in termini di disponibilità ed affidabilità dei componenti e dell'applicazione. Di solito i DCS sono impiegati nel controllo della produzione di energia elettrica, nei processi delle raffinerie di petrolio e delle piattaforme petrolifere, negli impianti chimici, ove pericolosità e criticità del processo di produzione impone questa scelta. Date le particolari operazioni ed i vincoli imposti su di esse, sia i controllori che le reti su cui si scambiano i messaggi sono in genere reti ridondanti e protette.

Sistemi SCADA(Sistemi di telecontrollo) Anche se si utilizza lo stesso acronimo dei precedenti,i sistemi per il telecontrollo sono pensati per la gestione di dispositivi distribuiti sul territorio. Sistemi di telecontrollo sono utilizzati anche per la gestione di reti elettriche, gas, autostrade, ferrovie: ovunque sia necessario monitorare da remoto lo stato di impianti e gestire il loro funzionamento. I dispositivi utilizzati possono essere sia PLC che RTU(Remote Terminal Unit), apparati appositamente pensati per la gestione locale di un limitato numero di dispositivi I/O con funzionalità di trasmissione e ricezione da e verso una sala di controllo di comandi e dati sullo stato di funzionamento. Nei centri di controllo ci sono reti di computer con applicativi atti alla gestione degli RTU. Spesso la complessità di tali applicativi risiede principalmente nell'integrazione delle diverse tecnologie utilizzate dai vari trasmettitori, che sono state installate in diversi periodi ma che in qualche modo devono convivere in uno stesso sistema interconnesso.

1.2 Struttura di una rete Scada

Il layout di una rete SCADA rappresenta il primo mezzo con cui una organizzazione può difendere i propri impianti ed i propri dati di produzione. Come suggerito nelle *best practices* pubblicate dagli enti statunitensi NIST [47] e NISCC [24] il il principio base che ispira il progetto della rete dell'impianto è l'isolamento dei sistemi di controllo(Process Control Network - PCN) in modo tale da evitare connessioni non filtrate verso Internet e la rete aziendale(Enterprise

Network - EN).
Una comune topologia di rete SCADA è quella indicata in figura 1.1.

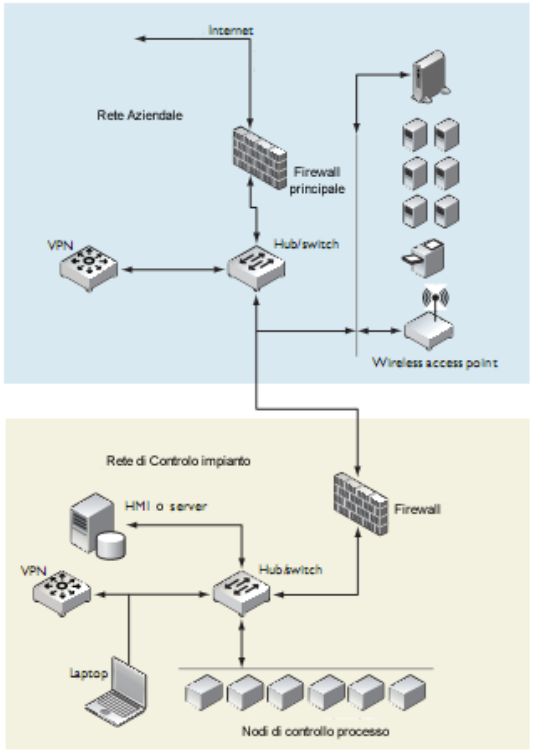


Figura 1.1: Un'esempio di rete informatica per impianti industriali.

La realizzazione della rete seguendo queste linee guida fa uso massiccio di comuni apparati di rete quali firewall, IDS, routers, gateway vpn, aprendo e standardizzando di fatto le reti di controllo degli impianti all'integrazione con infrastrutture IT. Ciò offre sicuramente numerosi vantaggi ma anche svantaggi e minacce, ereditate dal mondo IT: intrusioni remote, attacchi DoS, worms e virus. Purtroppo, nel caso delle infrastrutture critiche, tali minacce risultano essere molto più rilevanti e dannose che in un normale sistema IT, infatti, nel caso di attacchi ai sistemi, non si ha solo un danno economico ma possono sussistere rischi anche per persone od addirittura nazioni intere. Tralasciando le problematiche di sicurezza ereditate dall'ambiente IT, che tratteremo nel paragrafo 3, in primo luogo analizzeremo in dettaglio i dispositivi connessi in rete cercando di individuare i vantaggi ed i svantaggi della loro adozione, dal punto di vista sia della semplificazione della gestione del flusso di informazioni sia nelle prestazioni generali.

1.2.1 Firewall

Nella implementazione della rete dell'impianto il firewall può essere d'aiuto a ridurre un determinato insieme di rischi dovuti ad accessi non autorizzati alla rete di controllo. Come indicato in [59] alcuni obiettivi di una strategia per ridurre tali rischi sono:

- Eliminare connessioni dirette dalla EN e da internet alla rete di controllo PCN ed ai sistemi SCADA;
- Limitare l'accesso alla rete PCN da utenti connessi alla EN;
- Abilitare, dietro richiesta di credenziali, utenti della rete EN ad accedere a dati/report dell'impianto e della produzione;
- Stabilire connessioni sicure verso la rete PCN;
- Definire regole certe per tutto il traffico permesso sulla rete di controllo PCN;
- Monitorare il traffico per impedire accessi non autorizzati e traffico non voluto sulla rete PCN;
- Stabilire connessioni sicure per la gestione degli stessi apparati di rete;

Come ampiamente descritto in [24] è imperativo l'utilizzo di uno o più firewalls tra le varie sezioni della rete SCADA per garantire la separazione tra le varie sezioni(EN e PCN) e i punti di ingresso/uscita del traffico da e verso la rete esterna.

Firewall e protocolli industriali Molti dei più diffusi protocolli utilizzati nel mondo industriale, come ad esempio OPC, DCOM, MODBUS/Tcp, DNP3 e molti dei protocolli proprietari utilizzati dai software SCADA, presentano notevoli rischi per la i sistemi e per le reti. Questi protocolli sono spesso sconosciuti a dispositivi nell'ambito IT, e quindi anche ai firewalls. Scrivere regole che permettano di trasferire dati via OPC e DCOM attraversando un firewall può essere un'impresa ardua ed errori che lasciassero aperte e non presidiate porte di comunicazione non previste porterebbero in ogni caso ad ulteriori vulnerabilità nei sistemi.

Il problema principale da affrontare è che, mentre per le applicazioni IT tradizionali i firewall sono utilizzati per il controllo dei protocolli standard come IP,TCP/IP etc., il loro utilizzo in ambiente SCADA è da valutare con attenzione, in quanto non sono spesso in grado di gestire i protocolli proprietari per queste tipologie di rete. I firewall e gli apparati di monitoraggio del traffico, inoltre, spesso inducono latenze, che per applicazioni real-time possono non essere adeguate. In modo particolare possono nascere problemi per applicazioni critiche che richiedono garanzie per caratteristiche sui tempi di risposta,ad esempio nel trattamento di un allarme.

1.2.2 Rete DMZ.

Una DMZ(DeMilitarized Zone) è un segmento isolato della rete LAN raggiungibile sia dalle reti interne che dalle reti esterne e che permette, però, connessioni solo verso l'esterno: gli host attestati sulla DMZ non possono connettersi alla rete aziendale interna.
Tale configurazione viene normalmente utilizzata per permettere ai server posizionati sulla DMZ di fornire servizi verso l'esterno senza compromettere la sicurezza dei sistemi interni nel caso una di tali macchine sia sottoposta ad attacco informatico. In una rete di tipo SCADA, l'utilizzo di questi reti intermedie, separati dalla rete aziendale (EN) mediante firewalls e routers aggiuntivi, è divenuto un metodo abbastanza comune per dividere la parte "business" della rete dalla parte controllo e SCADA ed è altamente consigliato [24].

1.2.3 Intrusion Detection.

I firewall ed altri semplici apparati di controllo della rete non hanno "intelligenza" adeguata per osservare, rilevare ed identificare particolari signature o pattern di attacchi alla rete od al sistema che possono essere presenti nel traffico che monitorano o nei file di log che collezionano. Questa mancanza spiega come device tipo gli Intrusion Detection System(IDS)[31] o Intrusion Prevention System(IPS) possano effettivamente concorrere nel mantenere una più granulare ed adeguata sicurezza nella rete.
Un sistema di rilevamento delle intrusioni (IDS: Intrusion Detection System) è un sistema automatico che ha il compito di rilevare tentativi di intrusione e attacchi ad un sistema informatico. Un sistema per la prevenzione delle intrusioni (IPS: Intrusion Prevention System,) è invece un sistema che, oltre ad eseguire gli stessi compiti di un IDS, cerca di bloccare i tentativi di intrusione prima che questi compromettano il sistema. Il suo principale obiettivo è quello di rilevare e segnalare usi non autorizzati, utilizzi impropri e abusi di un sistema informatico sia da parte di utenti autorizzati che da intrusi [47]. Esso deve essere in grado di rilevare le intrusioni in tempo reale e in maniera accurata: in particolare deve minimizzare il numero di falsi positivi [17] ,quando viene erroneamente segnalato un attacco, e di falsi negativi, quando invece un attacco non viene rilevato.

Obiettivi degli IDS Gli IDS possono essere configurati anche per rilevare comportamenti che violano le politiche di sicurezza: in questo senso, gli IDS agiscono in maniera simile ai firewall e riconoscono il traffico che non rispetta certe regole. Inoltre, gli IDS possono essere utilizzati per:

- duplicare i controlli effettuati dal firewall. Ad esempio, l'IDS può ripetere gli stessi tipi di controlli del firewall per individuare del traffico sulla rete che il firewall avrebbe dovuto

bloccare, ma che è comunque riuscito ad entrare nella rete interna, ad esempio grazie ad un errore di configurazione del firewall;

- fornire funzioni di logging per documentare gli attacchi e i tentativi di intrusioni effettuati sulla rete;
- segnalare gli amministratori della rete, tramite un alert (allarme), ogni volta che si verificano eventi meritevoli di attenzione.

Gli IPS possono inoltre:

- bloccare un attacco. Azioni da eseguire per questo scopo sono: terminare una connessione di rete, bloccare il traffico destinato/inviato a/da un dato IP e/o porta;
- modificare le politiche di sicurezza quando viene rilevata una minaccia per la rete. Possono, ad esempio, aggiungere una regola per il firewall per bloccare traffico ritenuto sospetto;
- modificare il contenuto del pacchetto di rete. Ad esempio, possono normalizzare il traffico di rete, per cui il payload del pacchetto viene incapsulato in un nuovo header per cercare di prevenire attacchi basati su header malformati, o rimuovere allegati infetti, come virus allegati a e-mail.

Un buon dislocamento degli IDS sulla rete SCADA con un adeguato livello di analisi effettuato in ogni momento può contribuire sensibilmente ad identificare tentativi di intrusione o compromissione dei dati tra quelli che sono comunemente riscontrabili su una rete TCP/IP.

1.2.4 Application Security

Oltre all'hardening della rete, rendere sicuri gli accessi del personale al sistema SCADA fornisce un ulteriore livello di difesa. L'autenticazione e l'autorizzazione sono dei meccanismi che, implementati in un singolo punto di controllo e di accesso, abilitano il sistema all'identificazione del personale autorizzato ad accedere al sistema SCADA. Il sistema di autenticazione e di autorizzazione, una volta accordato l'accesso, dovrà monitorare(*audit*) e controllare con la più fine granularità le operazioni effettuate sui componenti del sistema di controllo. Non può invece possedere backdoors che possano permettere di saltare passi della procedura di controllo delle credenziali richieste. Inoltre, è necessario [43] che i dati sensibili quali password, profili utente, ruoli e politiche di accesso siano conservati su storage sicuro e con una politica di accesso ad-hoc.

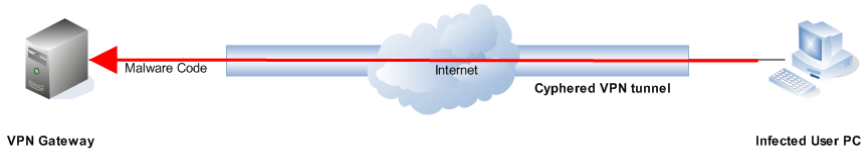


Figura 1.2: Il rischio delle infezioni malware.

1.2.5 Connessioni esterne.

La rete SCADA "ideale" dovrebbe essere un sistema "chiuso", controllato solo dal personale addetto dell'impianto in cui è utilizzato, accessibile solo dall'interno della rete aziendale. In realtà, spesso sono necessari accessi dall'esterno, da parte di utenti temporanei, più o meno fidati: manutentori, personale esterno dei fornitori, personale dell'azienda stessa che da altre sedi devono poter accedere ai dati locali all'impianto via rete. Questi tipi di collegamento portano con sè diverse minacce alla rete SCADA e sono il vero e proprio elemento critico per la sicurezza. Una rete esterna è una qualsiasi rete che non è parte del sistema SCADA. Alcuni esempi possono essere interfacce verso la rete aziendale, connessioni verso servizi di terze parti,o ad altri sistemi Scada. Spesso tali connessioni sono utilizzate per il controllo remoto, o per effettuare valutazioni economiche per facilitare le contrattazioni sul mercato dell'energia dalla parte amministrativa dell'azienda. Anche queste tipologie di connessione sono considerate "esterne".

Le interfacce verso sistemi esterni possono assumere che la rete esterna sia affidabile(*trusted*), e ciò implica che la sicurezza dell'infrastruttura SCADA possa dipendere da più di una organizzazione. Se l'interfaccia verso l'esterno è una possibile fonte di minacce, è importante che i link verso queste reti terze siano innanzitutto protetti utilizzando algoritmi di crittografia sicuri. Ci sono due modi per implementare una connessione sicura con l'obiettivo di trasferire dati dalla rete SCADA a reti esterne. Il primo metodo, più semplice e "spartano", è attraverso l'utilizzo di storage condiviso (CD, DVD, floppy disk). Il secondo metodo, invece, presuppone una connessione di rete sicura tra rete SCADA e l'organizzazione esterna. Questa connessione deve far sì che le richieste di servizio di clients remoti autenticati possano essere servite senza consentire l'accesso diretto alle risorse della rete di controllo PCN. Come descritto in precedenza, è molto utilizzata la tecnica delle reti DMZ per delimitare una "zona intermedia" dove è possibile accedere ai dati ma non alle risorse ed all'impianto.
Tuttavia, in molti casi non basta poter accedere a reports ed a dati elaborati dall'impianto. In questo caso, il livello di rischio nell'accesso a particolari apparati e dispositivi di rete è molto più elevato, in quanto non è detto che risiedano in zone ben delimitate della rete.

La prima necessità a cui la rete SCADA deve rispondere è fornire ad un link sicuro, e questo

normalmente viene garantito utilizzando i tunnel VPN per comunicare verso la rete d'impianto. Con le Virtual Private Network, i dati acceduti dal personale abilitato sono protetti dal tunnel cifrato stabilito dai due endpoint attraverso la rete "insicura". I device VPN, legati ai gateway ed ai firewall situati nei punti di accesso esterno alla rete SCADA, garantiscono la connessione solo agli utenti ed ai device abilitati e conosciuti escludendo determinati tipi di connessione rischiosi e poco affidabili come Telnet ed SNMP.

1.2.6 Architettura di rete: Casi d'uso e configurazioni

In [75] [24] vengono descritte diverse topologie di rete realizzate mediante diverse combinazioni degli strumenti descritti nelle sezioni precedenti.
Tali topologie, a seconda dell'utilizzo della rete SCADA effettuato, sono consigliate per prevenire e ridurre eventuali rischi.
Qui di seguito ne vengono illustrate alcune tra le più importanti e meno banali. Per ognuna di esse verrà descritta la configurazione ed eventuali problemi rispetto a tutti i vincoli che una rete SCADA deve soddisfare.

1.2.6.1 Configurazione 1: Firewall a due porte tra rete di controllo PCN e rete aziendale EN

L'introduzione di un Firewall tra la rete aziendale e la rete di controllo, definisce una prima topologia di rete separata che contribuisce sensibilmente al miglioramento della security rispetto alla completa e libera interconnessione tra le due reti. Molti Firewall presenti sul mercato possiedono funzionalità di "stateful inspection" [78] per i pacchetti TCP, in aggiunta a possibili NIDS e/o proxy delegati al controllo di ulteriori protocolli e pattern di possibili attacchi. Una configurazione del firewall di tipo "aggressivo" che limiti la maggior parte dei protocolli applicativi non necessari come FTP, SMTP,riduce sensibilmente la probabilità che un attacco abbia successo dall'esterno della rete PCN. Come descritto in [43], [24], [75] questa è una delle soluzioni più adottate dalle aziende che utilizzano sistemi di controllo Scada e PCN.

Problematiche e possibili implementazioni. Il primo problema da considerare è la scelta della parte della rete in cui collocare i server con i dati di produzione, consultabili sia da personale dell'impianto che da operatori esterni. Se essi sono piazzati sulla rete aziendale, di devono definire delle regole sul firewall che permettono la comunicazione tra tali server ed i sistemi della rete di controllo da cui acquisire i dati. Un pacchetto proveniente da un host "ostile" o configurato non correttamente sulla rete aziendale, che in apparenza potrebbe sembrare proveniente dal server dati e server web, potrebbe infatti essere inoltrato direttamente ad un

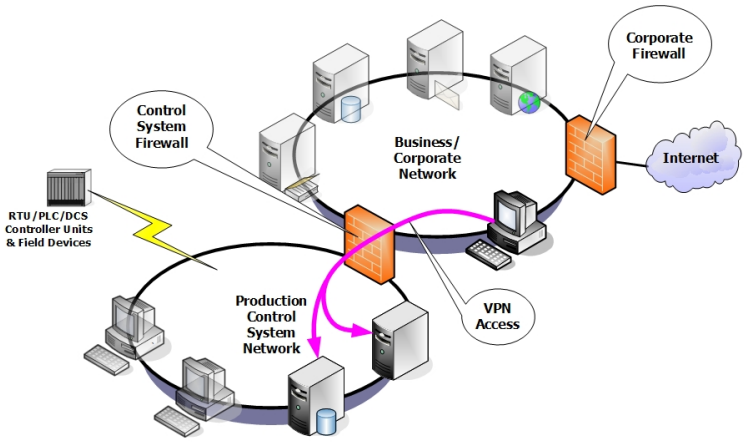


Figura 1.3: Esempio di rete di controllo con firewall.

singolo HMI o PLC sulla rete di controllo PCN. Se invece tali server risiedono sulla rete di controllo PCN, i firewall saranno configurati in maniera tale da permettere l'accesso solo agli utenti abilitati all'utilizzo di tali dati. Di solito, le comunicazioni effettuate da e verso tali server sono messaggi HTTP o richieste SQL ed un accesso ostile al livello di codice applicativo può compromettere sia i server dati che i server web. In cascata anche gli altri nodi sulla rete di controllo PCN diventerebbero vulnerabili ad una propagazione di codice virale con conseguente esposizione ad attacchi.

1.2.6.2 Configurazione 2:Firewall con DMZ tra la rete di controllo e la rete aziendale

Un notevole miglioramento avviene con l'utilizzo di firewall che possano gestire una o più DMZ tra la rete di controllo e la rete aziendale. Nella DMZ mettiamo i componenti "critici", come ad esempio i server con i dati della produzione, i web-server, i database, i punti di accesso per la rete esterna etc. L'utilizzo di firewall permette di segmentare e "segregare" le DMZ creando zone intermedie di rete che, nell'ambiente SCADA sono dette PIN(Process Information Network) [16]. I firewall utilizzati per creare la DMZ possiedono tre o più porte, ad una delle porte è collegata la rete aziendale(EN), ad un'altra la rete di controllo SCADA/PCN, mentre alle rimenenti si collegano la DMZ(o più DMZ) contenente i dispositivi "non sicuri". La figura 1.4 ne illustra un esempio.

Ponendo i dispositivi accessibili dalla rete aziendale in una DMZ, non è più richiesto permettere comunicazioni tra la EN e la rete di controllo PCN, ogni rete termina nella DMZ. Tutto ciò è valido se e solo se nella configurazione del firewall vengono ben specificate le tipologie di traffico e i corrispettivi path di destinazione.

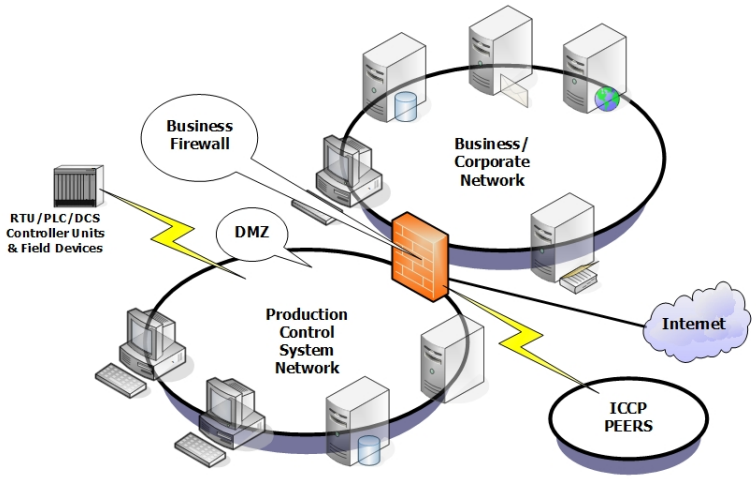


Figura 1.4: Un'esempio di rete SCADA con DMZ.

Problematiche e possibili implementazioni. Il rischio primario di questo tipo di architettura è che, se un computer nella DMZ viene compromesso, esso possa essere utilizzato per lanciare attacchi verso la PCN attraverso traffico "applicativo", di norma permesso tra la DMZ e la rete di controllo PCN. Questo rischio può essere molto ridotto se la robustezza dei server nella DMZ viene aumentata mediante periodiche verifiche delle funzionalità ed applicazione delle patch [16] per ridurre le vulnerabilità e se le regole del Firewall permettono che le comunicazioni tra la zona PCN e la DMZ possa essere iniziata solo dai dispositivi presenti nella rete di controllo PCN.
E' chiaro che l'utilizzo delle sole patch non risolva tutti i problemi.
Infatti, alcune delle riserve espresse su diversi security reports per SCADA [24] rispetto a questa soluzione riguardano principalmente la complessità di configurazione. Alcuni esperti infatti pensano che il firewall, avendo più porte, contribuisca ad aumentare la complessità della gestione degli utenti registrati nella ACL(*Access Control List*), aumentando quindi anche la possibilità di errori di configurazione e di gestione. In generale, questa configurazione è comunque quella maggiormente adottata e largamente suggerita [15].

1.3 Sicurezza delle reti di controllo e nelle reti Scada

Il governo degli Stati Uniti, in un report del ministero dell'energia del 2002 [59] individua 5 tendenze che hanno determinato la consapevolezza dei rischi legati alle reti SCADA:

1. L'adozione di tecnologie standard con vulnerabilità conosciute;

2. Connessione dei sistemi di controllo ad altre reti;

3. Connessioni remote non sicure;

4. Informazioni tecniche diffuse e disponibili circa i sistemi di controllo;

5. Impedimenti nell'utilizzo di politiche e tecnologie di sicurezza;

Queste tendenze hanno portato le reti SCADA da reti proprietarie e chiuse nell'*arena* dell'information technology, con tutti i suoi costi, benefici di performance, e sfide nel campo della security.

La maggior parte dei sistemi SCADA, sistemi di automazione, sistemi e reti di controllo sono quindi sempre più basati su tecnologie e prodotti tipici degli ambienti informatici, come ad esempio Ethernet, TCP/IP, sistemi operativi general purpose come Windows e Linux, ed utilizzati spesso in ambiti come la comunicazione di rete. Tali strumenti hanno portato sicuramente un beneficio tanto che queste infrastrutture si sono trovate inserite in un contesto eterogeneo basato sull'interoperabilità e la comunicazione, ma dall'altro lato hanno introdotto nuove vulnerabilità in un ambito dove il fondamento della protezione del sistema si è spesso basato sull'ambiente proprietario ed in genere circoscritto ad un utilizzo locale dell'impianto in cui è dislocato.
Purtroppo non ci sono ancora dei metodi di protezione di cui sia stata certificata l'efficacia da parte di gestori e produttori dei sistemi di controllo, ma esistono dei criteri approvati da enti americani [47] e [27] ed adottati anche in Europa, che forniscono delle linee guida per una adeguata implementazione delle policy di sicurezza su una rete di tipo SCADA.

1.3.1 Security IT e Security Scada

Studi approfonditi sull'applicabilità delle tecniche di security IT ai sistemi di controllo SCADA hanno dimostrato come metodologie normalmente utilizzate in un contesto puramente IT non possano essere applicate direttamente al contesto industriale così come riportato in [43].

In particolare, prendendo spunto dallo standard ISO2700 [22], ed il [59], le priorità negli obiettivi dell'IT Security e i corrispettivi nell'ambiente industriale sono differenti.
Per l'IT Security i capisaldi sono la garanzia di

- Riservatezza
- Integrità
- Disponibilità

dell'informazione.

Nei sistemi di controllo, invece, disponibilità ed integrità hanno maggior priorità rispetta alla riservatezza dei dati. A partire da questo dato, alcune delle priorità dei sistemi di controllo, diverse da quelle dell'ambito IT, dei sistemi di controllo sono:

- I rischi;
- I vincoli da soddisfare per la disponibilità dei servizi;
- I tempi di risposta richiesti per le operazioni;
- Tempi di risposta alle emergenze;
- Risorse Hw/Sw e loro manutenzione;
- Integrità di dati ed informazioni;

A partire da queste priorità verranno quantificate e valutate le vulnerabilità già presenti su un sistema SCADA per poi applicare le adeguate contromisure in termini di infrastruttura di rete e di security.

1.3.2 Minacce, vulnerabilità e contromisure nelle reti di automazione

Prima di analizzare ed implementare una rete SCADA interconnessa con altri servizi e/o reti, è utile individuare le vulnerabilità comunemente riscontrabili in una rete di controllo di processo di un impianto di produzione per poi individuare le corrette contromisure, sia in termini di apparati da utilizzare sia in termini di politiche da applicare.
Le minacce verso i sistemi di controllo SCADA possono provenire da molte fonti, quali gruppi terroristici, governi nemici, spionaggio industriale, personale licenziato od anche errori di sistema ed umani, disastri naturali. Secondo uno studio fatto dal Govern Accountability Office nel 2005 [53] le potenziali minacce si suddividono nelle seguenti percentuali:

Potenziale Minaccia	**Percentuale**
Hackers	22%
Personale Scontento	17%
Aziende o Governi Concorrenti	13%
Virus/Malware automatici	38%
Gruppi terroristici	10%

Tabella 1.1: Suddivisione percentuale delle minacce

Ogni vulnerabilità rilevata su un sistema è correlata ad una o più minacce, per cui lo studio della security di un sistema ha come passi: 1) individuare tutte le possibili minacce, 2) definizione delle vulnerabilità esistenti o possibili, 3)stesura di un piano di contromisure.

In [16] si identificano una serie di vulnerabilità identificate con l'acronimo "*core vulnerabilities*", suddivise in tre macro-categorie:

1. delle politiche e delle procedure di sicurezza;
2. delle piattaforme software utilizzate;
3. di rete.

In generale, l'analisi deve individuare tutte le vulnerabilità del sistema in esame, ma anche effettuare un ordinamento in base all'impatto potenziale che il verificarsi di un certo evento può determinare sulla gestione dell'impianto, in particolare su quei sistemi e quelle informazioni necessarie al management dell'azienda per perseguire i propri interessi economici, preservare le proprie attività, rispettare le proprie responsabilità legali e mantenere la propria operatività.

E' necessario quindi un metodo per valutare ed ordinare i rischi di possibili vulnerabilità in specifici impianti ed attrezzature.
Il rischio indotto da una data vulnerabilità è influenzato da un insieme di fattori:

- Condizioni della rete e dei computer connessi;
- Contromisure già presenti;
- Difficoltà tecniche di effettuare un attacco;
- Probabilità di individuazione dell'attacco;
- Conseguenze dell'incidente;
- Costo dell'incidente;

Nelle seguenti sezioni verranno illustrate in breve alcune possibili minacce e specifiche contromisure adottate su reti e sistemi SCADA per renderle conformi alle specifiche di sicurezza necessarie.

1.3.2.1 Vulnerabilità delle politiche e delle procedure di sicurezza

Alcune vulnerabilità sono introdotte sui sistemi di controllo a causa della mancanza di documentazione sulla sicurezza dell'infrastruttura.

Talvolta la vulnerabilità è dovuta a documentazione obsoleta o inappropriata.Inoltre, il problema può coinvolgere le politiche e le procedure che sono il caposaldo di ogni programma di sicurezza. La politica di sicurezza dell'azienda, infatti, può ridurre sensibilmente le vulnerabilità attraverso, ad esempio, l'indicazione di vincoli di utilizzo di determinate risorse.
Nella tabella 1.2 sono indicate alcune vulnerabilità riguardanti questa area.

Vulnerabilità	**Descrizione**
Politiche inadeguate per la rete di controllo	Sono introdotte vulnerabilità dovute all'inadeguatezza od alla mancanza di politiche specifiche per la sicurezza dei sistemi di controllo
Mancanza di formazione e consapevolezza sulla sicurezza dei sistemi industriali	Una formazione adeguata sui problemi e sulle procedure di sicurezza permettono l'aggiornamento delle conoscenze dello staff sulle procedure e *best practices* per il mantenimento di elevati standard di sicurezza.
Inadeguatezza dell'architettura di sicurezza	E' necessario che la scelta degli operatori che realizzano la rete industriale abbiano le conoscenze adeguate sugli standard di sicurezza da ottenere e sulle problematiche specifiche di questi sistemi.
Mancanza di documentazione sugli standard e sulle politiche utilizzate	La documentazione e i device di sicurezza uitilizzati devono essere mantenuti aggiornati e le politiche di sicurezza conosciute da tutto il personale.
Mancanza di periodici controlli di sicurezza su device e reti di controllo	Audit incentrati sulla sicurezza delle reti industriali richiesti a società terze specializzate garantiscono l'affidabilità dell'implementazione della rete e un monitoraggio certificato. Queste procedure aiutano ad individuare eventuali falle dovute a nuove vulnerabilità o a fattori prima non considerati.

Tabella 1.2: Vulnerabilità delle politiche

1.3.2.2 Vulnerabilità delle piattaforme software.

La configurazione errata, falle nel software o cattiva manutenzione delle piattaforme hardware e software, come i sistemi operativi e le applicazioni su di essi installate, provocano la presenza di vulnerabilità. Queste vulnerabilità possono essere attenuate attraverso diversi controlli e verifiche: patches per i sistemi operativi e le applicazioni, controllo dell'accesso ai

computer, utilizzo di software di sicurezza, antivirus.
Nella tabella 1.3 sono indicate alcune delle vulnerabilità più critiche.

Vulnerabilità	**Descrizione**
Patch dei sistemi rilasciate non immediatamente	Il tempo che intercorre tra il rilevamento di un bug nel software ed il rilascio della patch, spesso espone i sistemi ad una finestra di vulnerabilità non adeguata alla criticità dei sistemi.
Mantenimento dei sistemi operativi e del software legacy	La mancanza di patch ed aggiornamenti del software legacy ha risvolti catastrofici se vengono rilevate nuove vulnerabilità.
Uso di configurazioni di default	La cattiva gestione del software parte dall'inappropriatezza della configurazione. In particolare fonte di vulnerabilità è l'utilizzo delle configurazioni di default inserite nei device, lasciando al caso la possibilità di presenza di servizi non necessari e non controllati.
Backup e salvataggio delle configurazioni critiche	In presenza di un piano di disaster recovery è necessario effettuare backup delle configurazioni per un possibile rispristino.
Mancanza di procedure di protezione dei dati	I dati sensibili devono essere protetti da furto o manomissione, per questo sono richieste politiche di protezione adeguate.
Mancanza di politiche di gestione dei dati di accesso	La politica di gestione dei device, del software e dell'accesso ai dati deve garantire la sicurezza mediante una oculata definizione di procedure a protezione dei dati d'accesso.

Tabella 1.3: Vulnerabilità delle piattaforme

1.3.2.3 Vulnerabilità di rete.

Infine, è da considerarsi l'aspetto dell'interconnessione della rete di controllo dell'impianto a reti di diverso tipo, anche non SCADA. Questo tipo di interconnessione comporta, come descritto nelle sezioni precedenti, diversi problemi da affrontare legati sia ai diritti di accesso ai dati, sia ai meccanismi di controllo di tali accessi. E' possibile attenuare queste vulnerabilità mediante diversi controlli di sicurezza derivanti dal progetto, dalla configurazione ed il design dell'infrastruttura di rete, dalla tipologia di apparati di rete utilizzati ed anche dalle politiche stabilite ed applicate per l'accesso alle risorse.
Possono esistere quindi diverse tipologie di vulnerabilità dovute alla rete:

- Configurazioni di rete errate;
- Hardware di rete con configurazioni non conformi alla sicurezza necessaria;
- Layout di rete non sicuro;
- Monitoraggio e logging non adeguato o modificabile;
- Configurazione delle comunicazioni non sicura.

1.3.3 Fattori di rischio

Diversi sono i fattori che contribuiscono all'esposizione a minacce ed all'incremento del rischio nei sistemi di controllo, e che nelle seguenti sezioni saranno discussi in dettaglio:

- Adozione di protocolli e tecnologie con vulnerabilità note;
- Connettività dei sistemi di controllo verso altre reti;
- Connessioni insicure e/o dannose;
- Disponibilità di informazioni pubbliche sui sistemi di controllo;

1.3.3.1 Adozione di protocolli e tecnologie con vulnerabilità

Negli ultimi anni, i produttori di sistemi di controllo hanno iniziato a rilasciare le specifiche dei propri algoritmi e protocolli per permettere di integrare queste tecnologie con apparati e sistemi diversi e per ottenere compatibilità tra i vari strumenti che interoperano sulle reti SCADA. Inoltre, per ridurre i costi ed aumentare le performance generali delle loro reti, le organizzazioni passano, seppur lentamente, da sistemi proprietari a sistemi e tecnologie ormai standard nel mondo IT come Windows o Linux, favorite anche dalla diffusione capillare di protocolli come OPC, che permettono la comunicazione e l'interoperabilità di infrastrutture diverse nel contesto SCADA/Reti di controllo. Il passaggio a questi standard di fatto del mondo IT se da un lato favorisce un ritorno economico e tecnico, dall'altro incrementa la probabilità di attacco su tali sistemi dovuta in particolare, alla grande facilità di realizzazione e di utilizzo di tool che riescono a sfruttare al meglio le vulnerabilità ereditate.

1.3.3.2 Connettività verso sistemi diversi

I sistemi SCADA e le reti aziendali sono spesso reti interconnesse. La richiesta di accessi remoti verso la rete di produzione ha spinto molte organizzazioni ad utilizzare connessioni

remote per monitorare e controllare i sistemi SCADA da punti *esterni* la rete di controllo. Molte organizzazioni, inoltre, prevedono delle connessioni tra rete di controllo e rete dell'azienda per permettere ai managers di accedere in tempo reale ai dati di produzione ed ai dati *critici* come lo stato operativo dei sistemi. Nelle implementazioni più recenti questo può essere fatto attraverso applicazioni software ad hoc od attraverso servers gateway OPC. Tuttavia, in precedenza questo era effettuato attraverso i normali protocolli di rete e applicazioni standard basate su TCP/IP, FTP, XML.
Software standard e protocolli, essendo utilizzati senza una completa conoscenza dei corrispondenti rischi di sicurezza.
Inoltre, è da sottolineare come le reti delle società siano anch'esse connesse ad altre reti dei propri partners strategici oltre che ad Internet. Queste reti fanno largo uso di WAN ed Internet per la trasmissione di dati da e verso i device/sensori remoti dislocati su aree geografiche anche molto distanti. C'è da notare come l'integrazione delle reti dei sistemi di controllo con la rete pubblica e quella aziendale comporta l'aumento di probabilità di un attacco che sfrutti le vulnerabilità degli stessi sistemi.
A meno che non siano utilizzati appropriati controlli di sicurezza, queste vulnerabilità possono esporre anche tutti i livelli della rete di controllo agli attacchi derivanti dalla complessità del sistema, da worms, malware ed attacchi esterni.

1.3.3.3 Connessioni non protette

Molti produttori di sistemi SCADA realizzano i propri sistemi con la possibilità di instaurare connessioni remote per facilitare la manutenzione da parte del personale tecnico.
Questi accessi remoti, in molti sistemi di controllo, forniscono un accesso privilegiato al sistema agli utenti autenticati con valide credenziali.
I punti di accesso, spesso implementati con modem dial-up, permettono attacchi mediante tools progettati per ottenere il controllo di un sistema tramite questi tipi di connessione.
Inoltre, problema abbastanza diffuso anche nel mondo IT, le password predefinite fornite dal produttore degli apparati sono spesso le medesime per tutti, ed ancor più spesso una volta inserite in rete le password permangono identiche. Questo tipo di disattenzione rende naturalmente molto vulnerabile il sistema.
L'utilizzo di connessioni wireless negli impianti è in costante crescita ma, anche in questo caso, non è spesso valutata l'effettiva sicurezza necessaria all'impianto prima di adottare questo tipo di collegamento.
In [16] vengono elencati diversi incidenti legati all'utilizzo di connessioni wi-fi che hanno portato a numerosi incidenti.

Il primo inconveniente è il mezzo di trasporto dell'informazione, cioè l'etere. Quindi, la prima contromisura in questi casi è la cifratura del segnale. La seconda considerazione è la criticità dei dati che vengono trasportati attraverso connessione wi-fi. Infatti, è poco complesso effettuare attacchi di tipo *denial of service* ad una rete wireless.
Un'altro problema riguarda l'interconnessione delle reti aziendale (EN) e quella di controllo degli impianti(PCN).
Molte delle interconnessioni tra la rete aziendale e la rete di controllo degli impianti richiedono l'integrazione di sistemi con standard di comunicazione differenti. Il risultato è una infrastruttura ideata ed ingegnerizzata per muovere dati con successo tra le due entità.
Spesso, però, la complessità dell'integrazione di sistemi molto eterogenei, può comportare la mancata verifica di tutti i fattori di rischio da eliminare per garantire i livelli di sicurezza necessari alla rete. Come indicato in [75], questo comportamento nella gestione delle politiche di accesso EN-PCN fa sì che il controllo degli accessi tra le due reti sia minimale, esponendolo a minaccia sia interna che esterna.
Direttamente conseguenti a questo, come già descritto in precedenza, sono le possibili minacce derivanti anche dall'utilizzo di protocolli convenzionalmente insicuri come UDP e TCP/IP per il trasporto di dati tra le due reti, che espongono il sistema a minacce dal livello applicativo.

1.3.3.4 Accesso pubblico alle informazioni dei sistemi SCADA

Informazioni pubbliche sul progetto degli apparati, manutenzione, interconnessioni, e tipi di comunicazione sono disponibili su Internet per creare competizione tra i produttori. Gli stessi produttori, inoltre, forniscono toolkits per permettere agli sviluppatori di contribuire alla creazione di software di supporto per le reti e per gli standard del contesto SCADA. Informazioni e risorse sono quindi disponibili a potenziali avversari o intrusi di ogni tipo. Con le informazioni disponibili è molto semplice e possibile per chi ha conoscenze approfondite di un sistema di controllo ottenere accessi privilegiati all'infrastruttura SCADA mediante attacchi conosciuti di vario genere e con diversi impatti.

1.3.4 Statistiche e principali cause degli incidenti in reti SCADA

Una statistica accurata degli incidenti occorsi nei sistemi SCADA è difficile da determinare. Tuttavia molti studi effettuati negli ultimi 10 anni dimostrano come la crescita degli incidenti nei sistemi IT è direttamente proporzionale alla crescita degli incidenti nelle reti/sistemi di controllo industriale.
Nel 2001 è stato creato presso il British Columbia Institute of Technology, il ISID(Industrial Security Incident Database)[25] un sistema di tracciamento degli incidenti occorsi presso sistemi

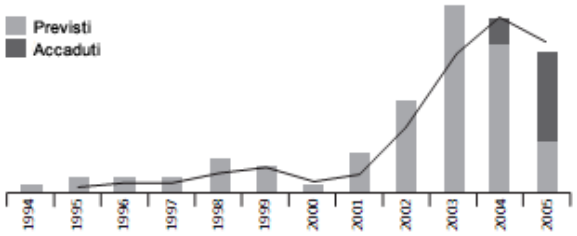


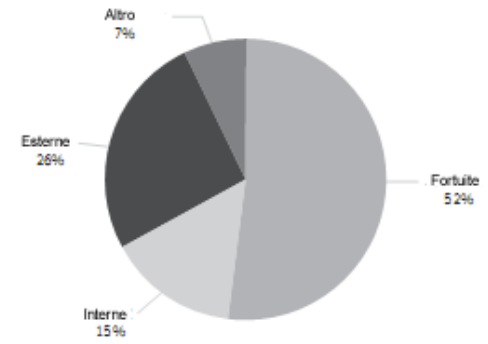
Figura 1.5: Trends incidenti informatici su sistemi SCADA.

di controllo industriali. I dati sono ottenuti da notizie pubbliche di incidenti o da sottomissioni private effettuate da personale di organizzazioni colpite da attacco.

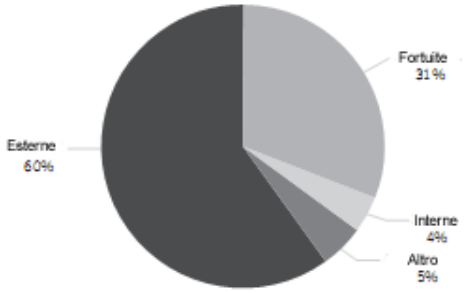
Fino a Gennaio 2007[25] gli incidenti riportati sul database ISID sono stati circa 150, a fronte di un numero di incidenti conosciuti, nei soli Stati Uniti, pari a 500-600[60]. Ciò sta a significare chiaramente come il numero di incidenti, sia per le scarse conoscenze del personale, sia per la poca sensibilità al problema, possa essere di un numero circa triplo rispetto a quanto riportato nell'iniziativa della BCIT.
La figura 1.5 illustra i trend degli incidenti dal 1986 al 2006, con un marcato picco di incidenti a partire dal 2001.La complessità dei moderni sistemi di controllo ed automazione industriale espongono, come già illustrato, diverse vulnerabilità sfruttabili come vettore d'attacco. Gli attacchi provengono da diverse fonti come la rete aziendale, Internet, Virtual Private Networks(VPN), reti wireless e modem dial-up.
Alla fine del 2001, il profilo tipico degli incidenti accaduti fino ad allora era come mostrato in figura 1.6(a). Dopo pochi anni il profilo è radicalmente cambiato come si può notare dalla figura 1.6(b).

I principali fattori dietro questo cambiamento sono stati chiaramente lo sviluppo e la diffusione di virus e worms automatici che, come noto, non necessitano di attivazione da parte dell'utente, e della massiccia adozione del protocollo ethernet per le reti di controllo.
Il database ISID, inoltre, ha tracciato la natura dei cyber-attacchi, mostrando come il malware primeggiasse su tutte le altre cause come si vede nel grafico di figura 1.7.
Almeno i $2/3$ degli incidenti è causato da worms e virus, mentre un numero considerevole è causato da azioni di semplice auditing dei sistemi con conseguenti alti costi di gestione e ripristino.

Dopo gli attacchi terroristici dell'$11/09$, anche la comunità hacker ha trovato nei sistemi SCADA un nuovo ambito in cui sperimentare le proprie abilità. E' ormai comune per le conferenze Black-Hat presentare interventi su vulnerabilità e test effettuate su sistemi SCADA



(a)



(b)

Figura 1.6: Classi di incidenti fino al 2001 (a); Classi di incidenti dal 2001 al 2006 (b)

di pubblici impianti non sicuri [51].
La figura 1.8 mostra, con riferimento alle statistiche sugli attacchi presenti in ISID, una serie di vie di accesso “critiche” in una normale rete SCADA.

1.3.5 Defence-in-depth

L’analisi dei dati presente su ISID mostra come le diverse organizzazioni operanti con device SCADA hanno diverse buone ragioni per occuparsi di cyber-security.
Gli incidenti dovuti a virus e worms compongono una significativa parte del numero totale degli incidenti che hanno coinvolto i sistemi di controllo.
L’alta frequenza di incidenti causati da malware indica che le strategie per aumentare la sicurezza in molti sistemi di automazione e controllo industriale sono insufficienti. Per esempio, un firewall perimetrale che protegge la rete aziendale offre poca protezione contro il malware eventualmente

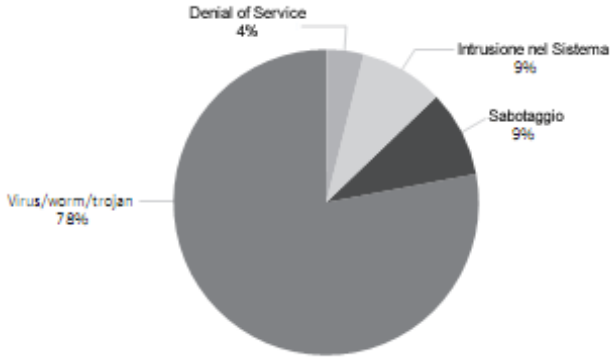


Figura 1.7: Incidenza delle minacce degli incidenti informatici su sistemi SCADA.

Causa Incidente	Anno	Costo Stimato
Melissa	1999	$0.3 Milioni
I Love You	2000	$8.0 Milioni
Code Red	2001	$2.6 Milioni
Slammer	2003	$1.0 Milioni

Tabella 1.4: Costo dei danni causati dai worms/trojans più famosi

presente sui laptop del personale connesso internamente alla rete di controllo PCN.
La security in SCADA, a partire da questi dati può essere in un primo step migliorata in due aree distinte:

- *a)* Il grande numero di incidenti che hanno coinvolto vettori di attacco ben conosciuti ed individuabili indica che molti dei problemi legati alla sicurezza vanno affrontati attraverso migliori policies, abitudini, e corsi di formazione del personale piuttosto che esclusivamente attraverso soluzioni tecnologiche.
  Ad esempio un elevato numero di incidenti dovuto al virus ***Slammer*** è stato sottomesso su ISID molti mesi dopo il rilascio delle patches. In questi casi le carenze sono proprio nella scarsa consapevolezza e conoscenza delle nozioni basilari di sicurezza del personale che gestisce gli impianti. Il fattore umano è ancora una volta la parte critica dell'equazione che porta al successo od al fallimento di un programma di sicurezza.

- *b)* L'esistenza di numerose *"vie secondarie"* all'interno del sistema SCADA/PCN fa riflettere sulla necessità oggettiva di una strategia di difesa in profondità(***defence-in-depth***), che coinvolga tutti i livelli del sistema da controllare. Questa strategia include l'hardening degli apparati di accesso ed endpoint, l'installazione e al corretta configurazione di firewall ed IDS per garantire diversi livelli di sicurezza

Metodi moderni di sicurezza indicano in maniera vincolante che garantire una sicurezza effettiva su un sistema richiede una strategia di difesa in profondità, dove i sistemi possano essere protetti da diversi **livelli di sicurezza**. Questo presuppone, ad esempio, che le diverse aree del sistema da difendere siano chiaramente individuate e separate e trattate in maniera differente rispetto al loro utilizzo. Infatti, il livello di sicurezza adeguato per la rete aziendale non è lo stesso necessario per la rete di controllo. Ad esempio, il firewall della rete EN deve, tipicamente, permettere agli utenti di poter navigare su internet usando protocolli come `HTTP`, la rete PCN invece richiede policies di sicurezza che lo vietino.
Un altro esempio sono le varie workstation presenti sulla rete, in questo caso è consigliato l'estensione dei livelli di sicurezza mediante utilizzo di personal firewall ed antivirus aggiornati come indicato in [24].
Una buona e basilare strategia di sicurezza del sistema deve offrire, quindi, livelli diversi di protezione a seconda dell'area e dell'hardware presente. Ad esempio, si può inizialmente prevedere un firewall dedicato al sistema di controllo dell'impianto per poi progressivamente prevedere politiche specifiche a seconda delle minacce e possibili vulnerabilità dei diversi device. Questo livello di sicurezza è, secondo, la base da cui partire. Infatti, anche utilizzando

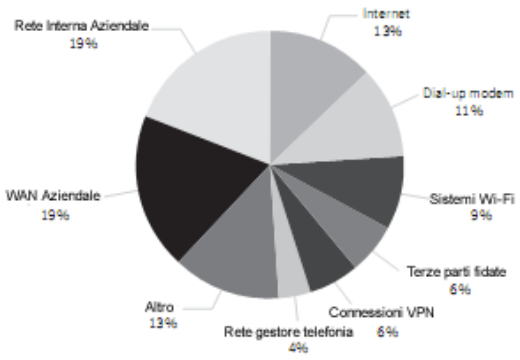


Figura 1.8: Analisi vie di accesso critiche su sistemi SCADA.

le ultime tecnologie nel campo della sicurezza e con un adeguato sistema di auditing di rete, le vulnerabilità in un sistema informatico sono comunque,un problema all'ordine del giorno.
Sia le policies di sicurezza, sia l'accesso ai dati sulla rete possono introdurre una serie di vulnerabilità conosciute e non conosciute che è difficile arginare solo con delle *best practice* [16].
Come ben noto nel contesto SCADA, "un sistema sicuro è un sistema non connesso alla rete". Infatti, 0-days exploit, virus e worms in ambiente IT si diffondono molto facilmente sfruttando bugs di applicazioni e protocolli, per cui per molti versi la sicurezza di un sistema connesso

comprende il problema di conoscere l'integrità dei partner remoti, ed in particolare per un ambiente come quello delle infrastrutture critiche, è necessaria l'assicurazione che questi siano affidabili durante tutto l'arco della connessione.

Capitolo 2

Integrità ed affidabilità di un sistema

Indice

Il concetto di integrità è un aspetto fondamentale della sicurezza informatica che indica: *la condizione in cui un componente è integro, cioè intatto*. Nel campo della sicurezza, il problema dell'integrità riguarda la modifica dei dati non autorizzata, in altre parole, la manomissione delle informazioni(*tampering*). In molti considerano che il problema dell'integrità possa essere risolto definitivamente con l'utilizzo della crittografia a chiave pubblica, funzioni hash, e firma

digitale, che forniscono una modalità robusta per determinare l'integrità dei dati. Questa visione proviene anche dalla constatazione che la ricerca rispetto ad altri campi della sicurezza, ad esempio la privacy, non è molto sviluppata. Tuttavia, la determinazione della genuinità di un sistema è solo una parte del problema generale d'integrità, che riguarda anche il mantenimento e l'attestazione della stessa ad un sistema esterno.
Nella letteratura sono presenti diversi lavori sull'integrità di un sistema e dei dati su di esso presenti. Anche se non è presente una definizione formale, il significato della parola *tampering* è "*l'atto di modifica non autorizzata dei dati*", e risulta evidente che l'idea di *tamper-detection* sia sinonimo di integrità. Il problema della individuazione delle modifiche è quindi strettamente correlato alla verifica dell'integrità sui dati.

Questo capitolo discute il problema della determinazione della genuinità in un sistema informatico. In particolare le sezioni saranno focalizzate sui metodi per la rilevazione di manomissioni e corruzione di dati.
La prima sezione introdurrà i fattori riguardanti la sicurezza dell'informazione, a cui è strettamente correlato il concetto di integrità.
La seconda sezione analizzerà il problema dell'integrità così come studiato sulla letteratura, partendo dai vari modelli esistenti fino alla definizione di sistema affidabile (*trusted*).L'ultima sezione è infine dedicata al *Trusted Computing*, un campo che si è sviluppato con forza negli ultimi anni e che segna un ulteriore sviluppo nel contesto generale della determinazione dell'integrità di un sistema.

Per poter discutere ed approfondire il concetto di integrità è necessario, quindi, partire da alcuni aspetti e definizioni utilizzate dalla teoria della sicurezza informatica, dal quale poi si potrà evincere più facilmente l'origine del problema e delle soluzioni che saranno trattate in seguito.

2.1 La sicurezza delle informazioni

La sicurezza informatica si basa su tre proprietà fondamentali[22]:

- Confidenzialità;
- Integrità;
- Disponibilità.

L'interpretazione e le relazioni tra questi tre aspetti variano a seconda del contesto informatico considerato e che comprende la tipologia di risorsa su cui applicare la sicurezza, i requisiti dell'organizzazione, leggi, decisioni manageriali, scelte e tipologie di personale.
Ad esempio, è ragionevole assumere che in un ambiente militare si ponga più enfasi sulla *confidenzialità*, in ambiente finanziario sull'*integrità*, in ambiente medico o industriale sulla *disponibilità ed integrità*.

La gestione della sicurezza è quindi il processo che cerca di garantire un determinato ***livello*** di sicurezza dell'informazione e dei servizi informatici espresso in termini di confidenzialità, integrità e disponibilità.

2.1.1 Confidenzialità

La confidenzialità è la garanzia che le informazioni possano essere lette solo da chi ne ha diritto. L'importanza di tale vincolo è cresciuta da quando i sistemi informatici sono usati in ambiti sensibili come quelli governativi, industriali o militari. L'accesso alle informazioni, quindi, è divenuto un problema di primaria importanza tanto che l'utilizzo di politiche di controllo per l'accesso ai dati sensibili sono ormai abbastanza comuni in molte organizzazioni.
I meccanismi di controllo degli accessi contribuiscono alla confidenzialità ed un ulteriore meccanismo utilizzato è quello della crittografia.
La confidenzialità riguarda anche l'esistenza dei dati, che tante volte è più decisiva del dato in sè. Ad esempio, sapere il numero esatto di quanti cittadini non hanno fiducia in un politico è meno significativo che sapere dell'esistenza di questo sondaggio. I meccanismi di controllo degli accessi, spesso, riguardano quindi anche l' esistenza del dato e non solo la protezione del suo contenuto.
Il mascheramento delle risorse è un'altro importante aspetto della confidenzialità. Un'organizzazione, ad esempio, potrebbe non fornire informazioni sulle apparecchiature che utilizza per dare determinati servizi.
Tutti i meccanismi che richiedono la confidenzialità necessitano di procedure di supporto fornite dal sistema dove saranno implementati perciò l'affidabilità di questi meccanismi è alla base della confidenzialità offerta dal sistema. In un computer general purpose, si deve quindi assumere, che tali proprietà possano essere fornite dal kernel del sistema, mentre i dati saranno trattati al livello dei processi.

Minacce alla confidenzialità dei dati L'accesso e la visibilità dei dati da parte di processi od utilizzatori non autorizzati è la sorgente principale di minacce alla confidenzialità.

Principalmente, la violazione può sfruttare:

- un sistema di **autenticazione debole**. Questo punto è correlato alla password personale che deve essere il primo dato a dover essere protetto;
- l'**ascolto ed intercettazione del traffico di rete**. Difficile ovviare a questo inconveniente, ma la soluzione sta nel rendere illeggibili i dati che transitano in rete tramite la crittografia che permetta **canali criptati**;
- la **cattiva progettazione o implementazione dei controlli di accesso ai dati** in ambito applicativo o banche dati. Da qui l'importanza che gli stessi siano progettati da specialisti;
- **virus, spyware, malware:** forme di virus che consentono di carpire informazioni all'insaputa dell'utilizzatore.

2.1.2 Integrità

l'integrità valuta, come detto in precedenza, l'integrità di dati o risorse, ed è normalmente associata alla prevenzione di cambiamenti impropri o non autorizzati.

L'integrità include:

- **Integrità dei programmi**;
- **Integrità dei dati**;
- **Integrità dell'origine dei dati**(detta anche *autenticazione*);

Questi ambiti illustrano il principio secondo il quale l'integrità intesa come credibilità, affidabilità, è un concetto centrale nel corretto funzionamento di un sistema.
Un sistema può prevedere meccanismi di prevenzione di violazioni dell'integrità e meccanismi per la rilevazione di tali violazioni. *I meccanismi di prevenzione* cercano di mantenere l'integrità dei dati bloccando ogni tentativo non autorizzato di modifica dei dati od ogni tentativo di cambiare i dati in maniera diversa da quella autorizzata.
La distinzione tra questi due casi è molto importante. Il primo caso si verifica quando un utente tenta di cambiare dei dati ma non ha i diritti per poterlo fare.Il secondo, invece, quando un utente autorizzato a manipolare certi dati, cerca di modificarli in modi diversi da quelli previsti dai suoi diritti.
Generalmente, adeguati controlli di autenticazione ed accesso possono bloccare i tentativi di intrusione esterni, tuttavia i tentativi "interni" richiedono altre contromisure.

I meccanismi di rilevamento, invece, non tentano di prevenire le violazioni di integrità ma riportano solamente che l'integrità dei dati non è più garantita. Questi meccanismi possono analizzare gli eventi di sistema, ad esempio le azioni degli utenti o del sistema stesso, per rilevare problemi o possono analizzare i dati per controllare se i vincoli richiesti al sistema sono soddisfatti. I meccanismi che implementano tali controlli potranno anche segnalare la causa della violazione d'integrità.
I problemi posti dall'integrità sono diversi da quelli posti dalla confidenzialità. Infatti, quest'ultima garantisce solo che i dati possano essere noti a chi ne ha diritto, mentre un controllo di integrità include sia la correttezza che l'affidabilità. L'origine dei dati, la protezione durante la comunicazione dall'origine alla destinazione e la protezione del dato sul sistema in cui è memorizzato sono tutti fattori che afferiscono all'integrità.
Questo dato, come illustrato nell'introduzione al capitolo, fa comprendere come la valutazione dell'integrità sia un processo complesso, essa infatti, si basa principalmente sulle assunzioni fatte per l'entità origine dei dati e sulla fiducia in tale origine, due fondamenti della sicurezza informatica spesso trascurati.

Minacce all'integrità La minaccia principale, come detto in precedenza, riguarda la modifica, creazione, cancellazione dell'informazione da chi non ha le credenziali per farlo. Le operazioni od i processi che mettono a repentaglio l'integrità quindi, in generale, possono essere:

- Errori ed omissioni provocati dall'introduzione errata di dati;
- alterazione dei dati da parte di software come virus, malware;
- Attacchi informatici interni od esterni all'azienda, ad esempio il *website defacement*;

2.1.3 Disponibilità

La ***disponibilità*** si riferisce alla capacità di poter usare le informazioni e le risorse desiderate. Il concetto di disponibilità è un importante aspetto della integrità di un sistema, infatti un sistema non disponibile non può essere un sistema integro. L'aspetto della disponibilità che è rilevante per la sicurezza è che un servizio o dei dati possano essere resi indisponibili(*Denial of Service*) da qualche attaccante.Questi tentativi di bloccare la disponibilità di risorse possono essere i più difficili da rilevare, in quanto è necessario determinare se i pattern di accesso non corretti rilevati su un sistema sono attribuibili a manipolazioni di risorse e programmi nell'ambiente di elaborazione.
Un tentativo di rendere una risorsa non disponibile è un evento atipico, ma esso può essere non atipico in certi sistemi e questo complica il suo rilevamento.

Minacce alla disponibilità La perdita della disponibilità delle risorse può essere dovuta ad eventi accidentali oppure intenzionali. I primi sono eventi casuali, non voluti, come disastri naturali od errori umani. I secondi, invece, sono attacchi deliberati verso le risorse condivise. Tra le minacce più significative si evidenziano:

- Denial of service;
- Furti di apparecchiature o di informazioni;
- Malfunzionamenti;
- Virus, worms o bot quando influenzano le prestazioni di un sistema;
- Sabotaggi;
- Attacchi dns, website defacement;

2.2 Integrità di un sistema e dei dati

Se un programma eseguito in un ambiente, gestisce i dati e le comunicazioni in maniera sicura, anche l'ambiente circostante deve garantire che i servizi essenziali di sicurezza siano garantiti.
Tale contesto deve poi far sì che i dati trasmessi siano comunicati al partner della comunicazione corretto e non possano essere trasmessi o acceduti da utenti non autorizzati.
Inoltre, lo stesso ambiente deve anche verificare che tale programma esegua solo le funzioni attese, e, che sia le funzioni del programma sia i dati su cui opera non siano alterati.
L'integrità, dal punto di vista informatico, può essere intesa con il seguente enunciato:"il concetto di integrità di un sistema non dà garanzie riguardo al corretto funzionamento, in assoluto, di un sistema[...], consideriamo quindi un sistema con proprietà di integrità se esso può essere considerato fidato(*trusted*) in quanto opera secondo un ben definito codice comportamentale."[46].
Per potere valutare la correttezza e l'affidabilità di un sistema, la prima assunzione necessaria è che, inizialmente, sia stato certificato il suo corretto funzionamento. In questo caso, è possibile nel seguito assicurarsi che il funzionamento e l'operatività del sistema non sia sovvertita in maniera dolosa.
Per affrontare il problema dell'integrità come descritto in [46], è quindi fondamentale la formulazione di politiche di controllo degli accessi e meccanismi che forniscano quanto necessario al sistema per la protezione della sua originale operatività.
Anche se il documento citato risale al 1976, fino a poco tempo fa gli argomenti tradizionali della sicurezza informatica non hanno trattato spesso in dettaglio il problema dell'integrità, poichè

spesso è stata ritenuta prioritaria la protezione dei dati da accessi non autorizzati.
Negli ultimi anni, invece, è finalmente stato dato molto risalto all'impatto che un programma modificato dolosamente può avere su altri programmi o su dati presenti in un sistema od in una rete informatica. Buona parte dell'attenzione è dovuta alla rapida diffusione del web e di applicazioni sempre più complesse e operativamente critiche. La sicurezza, infatti, non può essere garantita senza la verifica dell'integrità delle funzioni critiche di un sistema come l'autenticazione, i meccanismi di controllo degli accessi, l'affidabilità dei componenti.

2.2.1 Analisi degli impatti

Una definizione generale "astratta" di impatto sull'integrità di un sistema, può essere espressa come l'alterazione completa, od in parte di suoi componenti, non preventivabile nella iniziale verifica e certificazione del corretto comportamento. Il nostro punto di vista, in questo capitolo, è focalizzato su un sottoinsieme di soggetti ed oggetti, individuati in base alla loro funzione od ai loro privilegi.
Per poter valutare l'impatto sul sistema di possibili comportamenti non corretti e non previsti dobbiamo utilizzare dei criteri di selezione degli impatti, o danni, e delle minacce per poter descriverle in maniera precisa ed accurata. In [46] sono considerate due dimensioni per classificare le minacce d'integrità:

- Sorgenti di minaccia;
- Tipi di minaccia.

La sorgente della minaccia è dove la minaccia è stata generata, mentre il tipo di minaccia identifica come la minaccia è stata creata e come può colpire.

2.2.1.1 Sorgenti delle minacce

Consideriamo essenzialmente due tipologie di sorgenti di minaccia:

- interne;
- esterne;

In generale, una minaccia è data da un sistema o sottosistema che tenta di cambiare(illegalmente) il comportamento del componente considerato ai fini dell'analisi fornendo dati falsi, invocando funzioni in maniera non corretta, o modificando direttamente il suo funzionamento.
Entrambe le tipologie, generano diversi rischi, dal funzionamento non conforme alle sue specifiche fino alla sua non usabilità parziale o totale.

Le due tipologie si differenziano rispetto alla modalità ed ai vettori utilizzati per l'attacco. Le minacce esterne, in generale, utilizzano le interfacce che permettono di collegare il componente attaccato ad altri componenti. In questo caso il vantaggio che si può avere dal punto di vista della sicurezza è dato dal fatto che su tali interfacce spesso è possibile applicare o già possiedono un elevato monitoraggio delle attività eseguite. Il secondo tipo di minaccia, invece, avendo una visione più granulare del sistema e dei suoi componenti, è abilitata ad accedere ad un maggior numero di risorse ed interfacce e, in generale, sfruttare loro vulnerabilità in un ambiente che è meno monitorato.
A partire da queste due *dimensioni* è possibile definire una ulteriore tassonomia, infatti per entrambi gli insiemi vanno differenziate due ***tipi*** di minaccia: (1)tipo diretto, (2) tipo indiretto. Le minacce dirette sono quelle in cui il danno prospettato può colpire direttamente i dati o gli oggetti utilizzati dal componente. Ad esempio la scrittura su un database, o su un file protetto. Le minacce indirette sottendono una classe di scenari più ampia. In generale, le minacce indirette si riferiscono a modifiche improprie risultanti però dall'uso di dati e procedure inserite nel componente da un programma *ostile* e che portano al suo malfunzionamento.
E' così possibile correlare le due classificazioni:

- *Minaccia esterna diretta*: un programma od un'altro sistema corrompe un'insieme necessario di informazioni, ad esempio codice eseguibile di sistema;
- *Minaccia esterna indiretta*: un programma od un'altro sistema invoca del codice ostile su un'altro programma o sistema;
- *Minaccia interna diretta*: codice di tipo *self-modifying*;
- *Minaccia interna indiretta*:codice di tipo *self-modifying*, ma attivato da errori di programmazione;

Minacce interne ed esterne sono mostrate in figura 2.1. Il soggetto *S1* modifica l'oggetto *O1*(esempio (a)), causando il cambiamento di comportamento del soggetto *S2*. Nell'esempio (b), il soggetto modifica alcuni suoi dati in una maniera non preventivata e conseguentemente cambia il suo comportamento. In un lavoro più recente, Shirey[22] divide la precedente classificazione in 4 tipologie più specifiche:

- *Disclosure*, cioè l'accesso non autorizzato all'informazione;
- *Deception*, cioè l'accettazione di dati falsi;
- *Disruption*, interruzione dell'esecuzione di operazioni corrette;
- *Usurpation*, controllo non autorizzato di alcune parti del sistema.

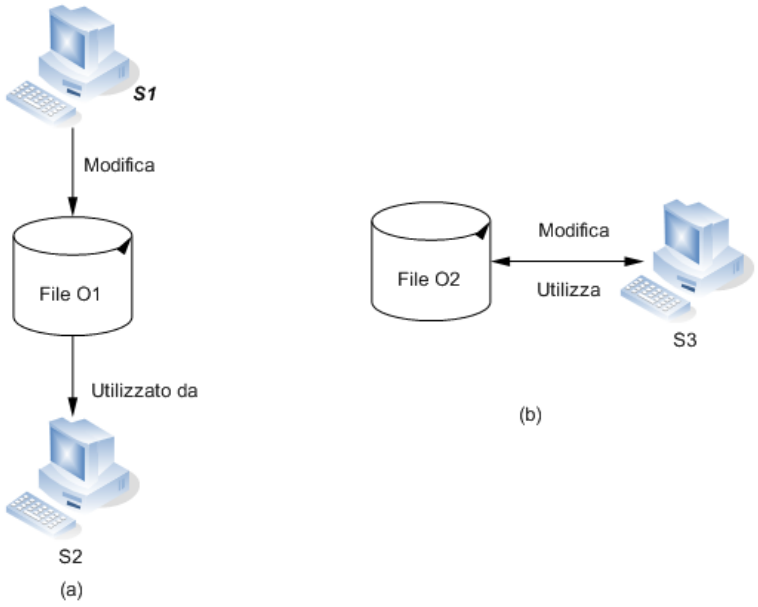


Figura 2.1: Minacce Interne ed esterne, modello Biba[46]

Esempi di attacchi all'integrità che sono assimilabili a questi tipi di minaccia sono molteplici, e si hanno sostanzialmente negli ultimi tre casi.Ad esempio, per la classe *deception*, un attacco molto frequente si ha quando l'attaccante ha conoscenza che alcune entità del sistema utilizzano i dati modificati per decidere le azioni da intraprendere in un particolare momento.Se i dati modificati controllano operazioni critiche del sistema, si presentano anche le classi *disruption* ed *usurpation*. Un'esempio nel caso delle comunicazioni dati su rete è il cosiddetto "*active wiretapping*", una forma di intercettazione dei dati trasparente ai partner della comunicazione in cui i dati sono modificati e spediti al destinatario. Il relativo attacco è il famoso *man in the middle attack*.

2.2.2 Politiche d'integrità

In ognuno dei casi precedenti, il principale obiettivo è l'identificazione di tutte quelle modifiche che preservano la validità (in termini di proprietà) degli elementi del sistema informatico. Le politiche di integrità hanno l'obiettivo di verificare, mediante l'utilizzo di meccanismi di controllo le *regole*, che l'entità che voglia utilizzare un certo tipo di informazione possieda i diritti che ne determinano l'autorizzazione all'accesso.
Una politica di sicurezza è normalmente dichiarata in termini di *oggetti* e *soggetti*, dati i due

insiemi il sistema di controllo utilizza un insieme di regole per determinare quando un dato soggetto può accedere ad uno specifico oggetto.
Come descritto in [46], due questioni regolano il tipo di politica d'integrità che un dato sistema può supportare:

- la disponibilità di meccanismi di controllo da applicare;
- i problemi di integrità che il sistema di controllo deve affrontare;

Biba[46] identifica due dimensioni per le politiche d'integrità:(1) la frequenza dell'applicazione dei controlli e (2) la granularità. La frequenza è funzione dell'istante in cui si applicano i meccanismi di controllo degli accessi.Se il controllo è effettuato solo una volta, ad esempio, all'avvio od alla creazione del programma, la frequenza è detta **statica**.
Se invece è effettuato ad ogni accesso di un programma ad un oggetto, è detta **dinamica**. La granularità è invece funzione della dimensione dell'oggetto protetto del sistema. Per un'efficace applicazione della politica d'integrità, la granularità del meccanismo di controllo deve corrispondere a quella della politica. Ad esempio, se una politica di protezione verifica l'accesso a parti di un file, un meccanismo di controllo che verifichi solo l'accesso all'intero file non è conforme alla politica richiesta.
Questo esempio evidenzia come le politiche che stabiliscono i controlli di protezione per la modifica degli attributi di soggetti ed oggetti devono essere di tipo dinamico e con alta granularità per implementare un effettivo controllo degli accessi.

L'integrità delle informazioni può quindi essere garantita se soggetti con opportuni diritti su oggetti possono eseguire solo modifiche regolate dalle politiche di sicurezza scelte.
Come indicato in precedenza, questo obiettivo può essere ottenuto in modi diversi, che vanno da controllo degli accessi a *tempo di caricamento* alla verifica statica del codice del programma.

2.2.2.1 Politiche MAC,DAC e RBAC

Due delle classi di politiche di sicurezza più importanti e discusse sono le politiche di tipo "*obbligatorio*"(mandatory) e quelle "*discrezionali*"(discretionary), da cui le sigle MAC(Mandatory access control) e DAC(Discretionary access control).
Esse differiscono sostanzialmente per come la politica di protezione, una volta creata, può essere cambiata.
La politica MAC fa riferimento ad una politica di protezione non scelta dall'utente, in cui alcuni vincoli, una volta definiti, devono essere sempre soddisfatti per tutti gli stati del sistema almeno finchè l'oggetto esiste.

La politica di tipo MAC ha una natura statica, e questo si ripercuote sulla gestione mediante una serie di funzioni delegate ad una autorità centrale, ad esempio un amministratore.
La politica di protezione DAC può essere completamente definita dall'utente in base alle proprie scelte, anche dinamiche. Infine, esiste un'ulteriore tipo di politica, detta *Role-Based*(RBAC) che assegna privilegi non agli utenti, ma alla funzione, il **ruolo**, che essi hanno nel contesto di una certa organizzazione. Nei seguenti paragrafi verranno illustrate brevemente le politiche, con particolare enfasi sulla politica MAC, che è ampiamente utilizzata nei modelli di integrità.

Mandatory Access Control La politica MAC definisce un tipo di controllo degli accessi attraverso il quale un sistema fissa dei vincoli ad un *soggetto* anche detto *initiator*[77] [22] che vuole accedere o che esegue una qualsiasi operazione su un *oggetto*.
In pratica, nei sistemi informatici, un soggetto di solito è o un processo o un thread; gli oggetti sono risorse come files, directories, porte TCP/UDP, segmenti di memoria, etc. Soggetti ed oggetti hanno entrambi una serie di attributi di sicurezza. Quando un soggetto cerca di accedere ad un oggetto, una regola applicata dal sistema esamina questi attributi di sicurezza e decide se l'accesso può avere luogo.Per ogni accesso effettuato da un soggetto su un oggetto, occorre applicare l'insieme di regole di autorizzazione, le **politiche**, per determinare se l' operazione richiesta può essere eseguita.
Con un controllo degli accessi di tipo MAC, la politica di sicurezza è controllata centralmente da un'amministratore. Sistemi di tipo MAC permettono agli amministratori di implementare politiche dettagliate per organizzazione e non solo per utente, e tutto ciò in maniera centralizzata e valida per tutti gli utenti.
Storicamente, la politica MAC è stata sempre associata ai sistemi Multilevel Secure(MLS)[77]. Il *Trusted Computer System Evaluation Criteria*[77] anche detto "Orange Book", definisce le politiche MAC "un modo di restringere l'accesso agli oggetti in base alla confidenzialità dell'informazione contenuta in essi [..]". In generale, l'utilizzo delle politiche MAC e di sistemi MLS sono state relegate ad una nicchia principalmente militare.
Di recente, tuttavia, l'avvento di implementazioni come SELinux[9](presenti nel kernel *Linux* dalla versione 2.6) e il Mandatory Integrity Control[62] incorporato in *Windows Vista*, ha portato le politiche MAC a diventare il principale attore nella protezione dei sistemi. In dettaglio, il funzionamento delle politiche MAC ha diversi aspetti, legati al tipo di verifica e di sicurezza da implementare.Ad ogni soggetto ed ad ogni risorsa viene assegnata una *classe d'accesso* costituita da:

- **un livello di sicurezza**: insieme totalmente ordinato che rispecchia i differenti gradi di sicurezza del sistema(es. top secret > secret > Confidential > Unclassified);

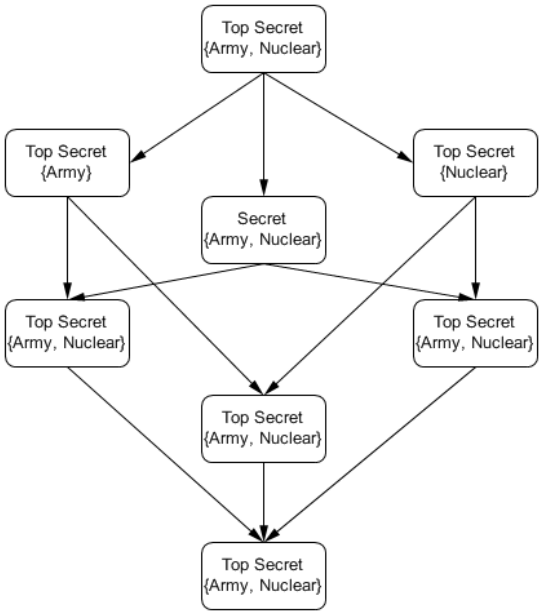


Figura 2.2: Reticolo MAC

- **un insieme di categorie**: una categoria è un insieme non ordinato che riflette le aree funzionali di un sistema;

Una delle proprietà della politica MAC è un ordinamento parziale della relazione di dominanza. E' possibile definire una **relazione di dominanza** tra classi. Date due classi d'accesso $C1$ e $C2$

- $C1$ domina $C2$($C1 \geq C2$) se e solo se il livello di sicurezza di $C1$ è maggiore a quello di $C2$ e le categorie di $C1$ includono quelle di $C2$;
- $C1$ e $C2$ sono non confrontabili se $C1$ non domina $C2$, nè $C2$ domina $C1$.

Per questo spesso si dice che le classi formino un *reticolo* e le politiche MAC vengono indicate per sistemi di tipo MLS, che già definiscono in maniera chiara questa gerarchia. L'uso delle classi di accesso assegnate a soggetti e risorse è diverso in base alla caratteristica di sicurezza che è più importante preservare: riservatezza o integrità, e, come vedremo, i modelli che preservano l'integrità utilizzano queste proprietà offerte dalla politica di sicurezza.
Ci soffermiamo sul secondo caso enunciando i principi basilari a cui si ispirano due modelli diversi per il controllo sulle scritture, cioè sulle operazioni che modificano un dato:

- **No read down**: un utente può accedere in lettura ad una risorsa se e solo se la classe d'accesso dell'utente *è dominata* dalla classe d'accesso della risorsa;

- **No write up**:un utente può accedere in scrittura ad una risorsa se e solo se la classe d'accesso dell'utente *domina* la classe d'accesso della risorsa;

Per esemplificare, formuliamo un caso relativo alla figura 2.2 con i seguenti soggetti ed oggetti e relative classi di appartenenza:

- Carlo (TopSecret, Nuclear)
- Documento A (Confidential, Nuclear)
- Documento B (TopSecret,Nuclear)
- Documento C (TopSecret,Army,Nuclear)

L'utente *Carlo* accede in lettura al *Documento A* ma non in scrittura, la classe di accesso dell'utente domina quella del documento ma non viceversa. Egli invece accede in lettura e scrittura al *Documento B*, la classe di accesso dell'utente domina quella del documento e viceversa.Ultimo caso, *Carlo* accede in scrittura a *Documento C* ma non in lettura, la sua classe è dominata da quella del documento.

Discretionary Access Control Le politiche di tipo DAC sono basate essenzialmente sull'identità del soggetto che richiede l'accesso alla risorsa e sul tipo di accesso richiesto.
Il modello di base di queste politiche è quello della *matrice di controllo d'accesso* che ha una struttura dinamica. Soggetti ed oggetti, infatti, possono essere creati dinamicamente ed i diritti che un soggetto ha possono essere esercitati solo ad un certo istante dopo altre operazioni.

	testo.doc	**stampa.txt**	**ciao.jpeg**
Tizio	lettura	lettura	lettura,esecuzione
Caio	scrittura	lettura,scrittura	

Tabella 2.1: Matrice di controllo degli accessi

Per qualsiasi risorsa possono essere specificate diverse operazioni, le più importanti e valide per qualsiasi soggetto sono :

- **Control**
- **Control***

La prima è utilizzata per trasmettere qualsiasi sottoinsieme delle proprie operazioni sulla risorsa ad altri oggetti, ma non l'operazione *control*. La seconda, invece, permette di trasmettere qualsiasi sottoinsieme, compresa l'operazione *control*.
E' da notare come inizialmente vanno comunque effettuate delle operazioni centralizzate da parte dell'amministratore, come dividere gli utenti in sottogruppi logici ed assegnare i diritti minimi ad ogni gruppo. Un esempio di questo tipo di implementazione è il file system di Unix.
Il problema principale delle politiche DAC è il seguente:"Data una matrice iniziale nota, esiste una sequenza di comandi che porta un utente a poter avere privilegi su una risorsa che normalmente non avrebbe?". In generale, questo problema è indecidibile, diventa decidibile se il numero soggetti ed oggetti è deciso staticamente e fissato.

Role Based Access Control Il *Role-based access control*[35] è un ulteriore approccio per restringere l'accesso al sistema ed alle sue risorse solo agli utenti autorizzati.
I sistemi di tipo RBAC assegnano i privilegi non agli utenti, ma alla funzione che questi possono svolgere nel contesto di una certa organizzazione. L'utente quindi *acquista* privilegi assumendo un ruolo ad un certo istante.
RBAC è "*policy neutral*", questo gli consente di supportare facilmente i due ben conosciuti principi della sicurezza:

- Principio del privilegio minimo;
- Separazione dei compiti: utilizzo di ruoli diversi tra chi controlla e chi deve essere controllato;

Inoltre, per questa caratteristica di duttilità, è anche possibile implementare politiche DAC e MAC attraverso una politica RBAC.

Per capire le politiche RBAC è fondamentale il concetto di **ruolo**. All'interno di una organizzazione, i ruoli sono creati per svolgere varie funzioni ed i permessi per eseguire particolari operazioni sono assegnati ad essi seguendo questa classificazione. Non essendo assegnati direttamente i permessi agli utenti, che acquisiscono mediante il loro ruolo(o ruoli), la gestione dei diritti dei singoli utenti diviene molto semplice perchè si assegna il ruolo più adeguato all'utente. Ciò semplifica molte operazioni come l'aggiunta di un'utente o lo spostamento di un'utente da un gruppo ad un'altro. In RBAC il ruolo è visto come un costrutto semantico in funzione del quale vengono formulate le politiche di controllo d'accesso. Il concetto di ruolo ha sia il significato di "*competenza*" nel compiere una determinata attività, sia di *autorità* e *responsabilità* su un determinato insieme di risorse.
Questo tipo di politica è implementata in molti sistemi come Microsoft Active Directory, SELinux, OpenSolaris, FreeBSD.

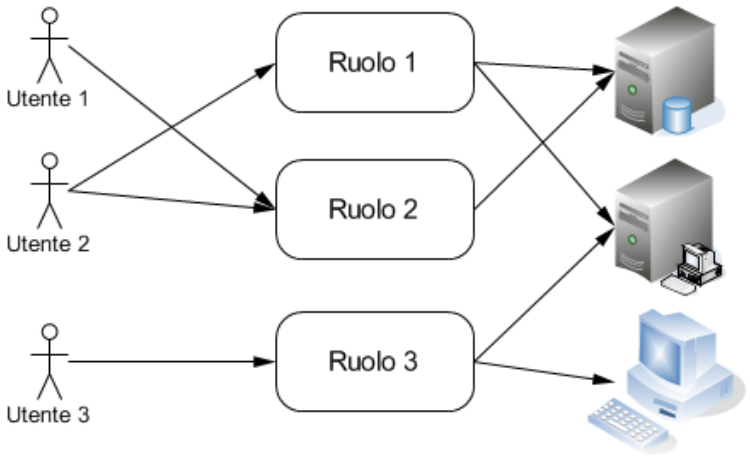


Figura 2.3: Ruoli nella politica Rbac.

2.2.3 Modelli per la sicurezza dell'integrità

Un modello di sicurezza è un approccio formale od informale per definire le regole di una politica di sicurezza.
I modelli di sicurezza sono un concetto importante del progetto di un sistema, la sua implementazione, infatti, deve far riferimento ad un modello di base per garantire la sicurezza di dati ed operazioni.
Esistono diversi modelli di sicurezza, ed ognuno di essi si riferisce a particolari aspetti della sicurezza delle informazioni.
I modelli più importanti possono essere classificati come segue:

- Politiche per la garanzia della confidenzialità
  - Bell - La Padula[20]
- Politiche per la garanzia di integrità
  - Biba[46]
  - Clark - Wilson[28]
- Politiche ibride
  - Chinese Wall
  - ORCON(Originator Controlled)

Dal nostro punto di vista, considereremo solo i modelli per la garanzia dell'integrità dei sistemi e dati.
Il problema che questi modelli affrontano si presenta principalmente in ambienti interconnessi e variegati come quelli industriali e commerciali, in cui, negli ultimi anni, il problema dell'integrità è diventato la priorità per la garanzia di sicurezza dei dati.

2.2.3.1 Modello Biba

Il modello Biba, o **Biba Integrity model**[46], sviluppato da Kenneth J. Biba nel 1977, è pensato per una politica MAC ed è una descrizione formale di un sistema di transizione di una politica di sicurezza. Utilizza un insieme di regole di controllo degli accessi creato per garantire l'integrità dei dati.
Soggetti ed oggetti sono raggruppati in livelli ordinati di integrità. Il modello è progettato in maniera tale che i soggetti non possano corrompere i dati in un livello superiore al proprio, e che un dato di un livello inferiore non possa corrompere il soggetto ad un livello superiore.
Questo modello, eredita le proprietà della politica MAC, è quindi caratterizzato dalle regole "*no write down, no read up*"[22].
Nel modello Biba gli utenti possono creare un qualsiasi contenuto ad un livello di integrità uguale od inferiore al proprio. Di conseguenza, possono leggere solo al proprio livello ed ai livelli superiori.
Il modello Biba definisce un insieme di regole duali a quelle definite, ad esempio, dal modello Bell - La Padula[20]:

- *Simple Integrity Axiom*, indica che un soggetto ad un dato livello di integrità non possa leggere un oggetto con un livello inferiore(**no read down**);
- $\star$-*Integrity Axiom*, indica che un soggetto ad un dato livello di integrità non possa scrivere su alcun oggetto con un livello di integrità più elevato(**no write up**);

Politica Low-watermark Quando un soggetto accede ad un oggetto, la politica cambia il livello di integrità del soggetto e dell'oggetto assegnando ad entrambi il minimo tra i due:

- Proprietà Low-watermark del soggetto: il soggetto s può leggere un oggetto o a qualsiasi livello di integrità;il nuovo livello di integrità del soggetto è $inf(f_s(s), f_o(o))$ dove $f_s(s)$ ed $f_o(o)$ sono i livelli di integrità prima dell'operazione;
- Proprietà Low-watermark dell'oggetto: il soggetto s può modificare un oggetto o ad un

qualsiasi livello di integrità. Il nuovo livello di integrità dell'oggetto sarà $inf(f_s(s), f_o(o))$ dove $f_s(s)$ ed $f_o(o)$ sono i livelli di integrità prima dell'operazione;

Questa politica genera il problema che il livello dei soggetti non cresce, ciò porta il soggetto a raggiungere in breve livelli troppo bassi per poter accedere ad oggetti a livelli d'integrità elevata.

Politiche per l'invocazione di oggetti Il modello Biba può essere esteso per includere un'operazione di accesso detta *invoke*, cioè un soggetto può invocare un'altro soggetto per accedere ad un terzo oggetto. Una diversa politica deve quindi poter essere applicata per usare questa proprietà. ***Invoke property*** è definita come segue: il soggetto $S1$ può invocare il soggetto $S2$ se e solo se $f_s(S2) \leq f_s(S1)$.
I soggetti sono abilitati ad invocare solo altri soggetti ad un livello inferiore. Si evita così che un soggetto di basso livello invochi un soggetto di alto livello e contamini un oggetto di alto livello violando le proprietà del modello. Nel caso che questo comportamento sia voluto, i soggetti di basso livello saranno abilitati ad accedere ad oggetti di alto livello solo se invocano soggetti di alto livello. Il soggetto invocato dovrà effettuare, innanzitutto, una serie di controlli di consistenza per assicurarsi che l'oggetto sia integro. In questo scenario ci è di aiuto una seconda proprietà, la ***Ring property.*** che permette ad un soggetto *S1* di leggere a qualsiasi livello di integrità. Esso potrà tuttavia modificare solamente oggetti con $f_o(o) \leq f_s(S1)$ e potrà invocare un soggetto S2 se e solo se $f_s(S1) \leq f_s(S2)$.

2.2.3.2 Modello Clark - Wilson

Il modello Clark-Wilson[28] è alternativo al modello Biba e stabilisce l'integrità mediante un insieme di vincoli. Lo stato di un oggetto è quindi consistente, se soddisfa i vincoli del modello. Questi vincoli, o requisiti, sono principalmente utilizzati per garantire l'integrità dei dati.
I requisiti di integrità possono essere introdotti per ragioni diverse:

- *Consistenza interna*: fa riferimento alle proprietà dello stato interno di un sistema e deve essere applicata dal sistema stesso.
- *Consistenza esterna*: considera la relazione tra lo stato interno del sistema e l'ambiente, composto da altri sistemi, in cui è inserito. Deve essere applicata dall'ambiente stesso.

Il modello Clark-Wilson identifica due meccanismi principali per garantire l'integrità ed il controllo degli accessi:

- *Transazioni ben formate*, una serie di operazioni che portano un sistema da uno stato consistente ad un'altro anch'esso consistente.

- *Separazione dei compiti*, assegnamenti diversi di una operazione ad utenti diversi, ciò limita il problema della modifica non autorizzata, perchè gli utenti dovranno accordarsi per portare a termine un attacco.

In linea generale, il modello definisce i dati su cui applicare i controlli di integrità come *constrained data items*, o `CDI`. I dati non vincolati dai controlli sono detti *unconstrained data items*, `UID`. Ad esempio in una banca, i dati dei conti correnti sono `CDI`, in quanto la loro integrità è cruciale per l'operato della banca, invece, i regali che l'operatore bancario consegna all'apertura del conto corrente sono `UID`. L'insieme dei `CDI` e degli `UID` partiziona l'insieme totale dei dati nel sistema modellato.
Un insieme di vincoli di integrità stabilisce i valori possibili per i `CDI`. Il modello, inoltre, definisce due insiemi di procedure:

- Le *procedure di verifica di integrità* o `IVP`, che verificano che i `CDI` siano conformi ai vincoli di integrità quando la procedura è stata eseguita.In questo caso, il sistema si dice in uno *stato valido*.

- Le *Procedure di Trasformazione*(`TP`), invece, cambiano lo stato del dato nel sistema da uno stato valido ad un'altro ed implementano le *transazioni ben formate*.

L'innovazione del modello Clark-Wilson rispetto al modello Biba sta nelle *regole di certificazione* e nelle *regole di applicazione*, che indicano dei vincoli sulle relazioni tra gli oggetti e sull'utilizzo delle `IVP` e `TP` oltre che regolare alcuni aspetti legati agli utenti ed alle azioni sui dati.
Questo modello aggiunge due concetti ai modelli di integrità. Il primo è l'associazione tra requisiti di ambito "commerciale" con i dati da essi trattati, rispettando le proprietà del modello stesso. Le aziende non classificano i dati usando schemi di tipo multilivello come ad esempio fa il modello Biba, ma applicano la separazione dei compiti. La seconda è la nozione di certificazione, diversa da quella di applicazione, ognuna con il proprio insieme di regole.
I contributi del modello Clark-Wilson sono molti, e per evidenziare queste caratteristiche aggiuntive lo confrontiamo con il modello Biba.
Il modello Biba, come detto in precedenza, associa i livelli di integrità agli oggetti ed ai soggetti. In un senso più ampio, il modello Clark-Wilson associa ad ogni oggetto due livelli: vincolato o alto(*CDI*) e non vincolato o basso(*UDI*). In modo simile, i soggetti hanno anch'essi due livelli: certificato(*TP*) e non certificato. La distinzione critica è rispetto alle regole di certificazione,il modello Biba non le prevede e non prevede nemmeno meccanismi o procedure per la verifica delle entità affidabili(*trusted*) e delle loro operazioni. Il modello Clark-Wilson introduce, invece,

espliciti requisiti che le entità e le operazioni da esse eseguite devono rispettare. Inoltre, nei sistemi che ricevono input da sistemi esterni e che possono modificare i livelli di integrità dei dati, il modello Clark-Wilson richiede che una entità *trusted* certifichi il metodo di aggiornamento dei dati al livello superiore.
In generale, si può affermare che il modello Clark-Wilson estende il modello Biba e può quindi anche emularlo nelle sue funzioni principali.

2.3 Verifica dell'integrità

Abbiamo già detto che, i servizi forniti da un sistema informatico "*sicuro*"(integrità, confidenzialità e disponibilità), interagiscono tra di loro, e non sono indipendenti. Ad esempio, l'integrità è una assoluta necessità per garantire la confidenzialità, indicando di fatto il problema dell'integrità come un prerequisito a garanzia degli altri servizi di sicurezza. Si consideri, ad esempio, un attaccante che cerchi di aggirare i controlli per la confidenzialità modificando il sistema operativo o la tabella di controllo degli accessi del sistema operativo. Quindi, garantire l'integrità del sistema operativo o della tabella di controllo accessi permette di garantire la confidenzialità.
L'integrità dei dati in un sistema informatico dipende però da come essi sono gestiti dal software e non solamente dal loro contenuto. I dati sono modificati dagli utenti mediante programmi software, e la loro integrità dipende dalla correttezza di tali programmi e la corretta associazione tra i programmi ed i dati sui quali operano. In questo contesto, la *Trusted Computing Base*(TCB), cioè l'insieme del'hardware, firmware e software di un sistema critici per la sua sicurezza, è formata dall'insieme di processi. E' perciò l'insieme del software che garantisce un corretto utilizzo dei dati su cui esso opera, proteggendoli dalla compromissione di utenti o processi non autorizzati.

2.3.1 Assurance ed affidabilità

Attualmente, non è possibile definire un sistema completamente sicuro, poichè non sempre è possibile contenere od eliminare tutte le vulnerabilità. Di conseguenza, non possiamo sviluppare sistemi che possono essere considerati sicuri nel tempo. Il concetto di sicurezza dipende sia dalla corretta specifica che dalla corretta implementazione di determinati requisiti, ma la corretta realizzazione di tali sistemi può non essere sufficiente per fornire affidabilità (*trust*) al sistema. Intuitivamente la ***fiducia*** è la supposizione che un'unità di elaborazione faccia ciò che è necessario per proteggere le proprie risorse e che sia difeso da attacchi. Tuttavia nel dominio della sicurezza informatica, il concetto di entità affidabile, in cui cioè si può riporre fiducia, ha un

significato specifico.
Bishop[22] definisce la fiducia in un sistema in termini di un concetto collegato:

> "Una entità si dice *attendibile*(trustworthy) se esiste una sufficiente e credibile evidenza che permette di assumere che essa soddisfi un insieme di requisiti. La fiducia è una misura dell'attendibilità, legata all'evidenza fornita."

Per determinare l'attendibilità di un sistema, ci si deve concentrare sulle metodologie e le metriche che permettono di misurare il grado di confidenza che possiamo porre nel sistema in considerazione. Un ulteriore definizione comprende questa nozione:

> " *Assurance* è la fiducia che un'entità soddisfi i suoi requisiti di sicurezza, basato sulla evidenza fornita dall'applicazione delle tecniche di assurance".

Un termine attinente, *assurance dell'informazione*, fa riferimento alla capacità di accedere alle informazioni e di preservarne la qualità e la sicurezza. Esso differisce dalla *assurance della sicurezza*, in quanto il focus è sulle minacce alle informazioni ed ai meccanismi utilizzati per proteggere tali informazioni, e non sulla correttezza, consistenza o completezza dei requisiti e delle implementazioni di tali meccanismi. Nel seguito usiamo la parola *assurance* per indicare *assurance della sicurezza*.

Siamo ora in grado di definire un *sistema fidato*[22]:

> "Un *sistema fidato*(trusted system) è un sistema che dimostra di soddisfare requisiti ben definiti secondo una valutazione di un insieme credibile di esperti, certificati per questo compito, e quindi anch'essi affidabili;"

Esistono specifiche metodologie per strutturare queste prove di assurance ed i risultati delle prove sono interpretati per l'assegnazione di *livelli di affidabilità* o *livelli di assurance*.

2.3.2 Fondamenti matematici

Normalmente il problema dell'integrità viene spesso suddiviso, in linea teorica, in due sottoproblemi, la garanzia dell'integrità in una comunicazione e quella dell'integrità di un sistema e dei suoi dati. Entrambi considerano la modifica non autorizzata di dati e la differenza è solo nel mezzo che deve garantire questa proprietà.
L'esempio seguente illustra come l'integrità sia strettamente legata all'autenticazione.

2.3.2.1 Firma Digitale

Consideriamo il problema dell'autenticazione di un mittente da parte del destinatario di un messaggio. Ad esempio, ci poniamo il problema di Alice, che riceve un messaggio da parte di Bob, suo socio d'affari, che indica di inviargli la combinazione della loro cassaforte. Il problema riguarda la certezza che il messaggio provenga da Bob se esiste una modalità per autenticare l'origine del messaggio. Un problema simile è ad esempio il famoso e temuto *phishing*.
Il problema sorge perchè Tom potrebbe avere conoscenza di cosa sa Bob, ed avere la stessa capacità di Bob di ottenere informazioni. Di conseguenza, se Bob può usare un protocollo per convincere Alice di essere Bob, Tom può fare altrettanto fingendosi Bob.Il problema può essere facilmente risolto mediante la firma digitale del messaggio le cui caratteristiche sono descritte in tabella 2.2.
In questo caso è da notare come l'enfasi non è sulla privacy del messaggio(quindi sulla confidenzialità), ma sull'autenticità del mittente legata alla ricezione del messaggio originale, quindi integro.

Uno schema di firma digitale è composto da due funzioni: una funzione privata S_α che permette al suo possessore α di *firmare* i messaggi che crea mediante una *firma s* ed una funzione pubblica V_α che permette a chiunque di validare la firma del firmatario per determinare la sua autenticità. **Proprietà**: • Validità universale; • Resistente alla contraffazione.

Tabella 2.2: Firma Digitale

2.3.2.2 Tamper detection

Come illustrato nel paragrafo precedente, l'integrità può essere associata all'autenticazione. In particolare i metodi utilizzati per l'autenticazione possono essere anche usati per rilevare la contraffazione e/o modifica dei dati. Tutto ruota intorno all'utilizzo di funzioni con particolari proprietà, le *funzioni hash*. Il problema del *tamper detection*, concerne con la determinazione di modifiche non autorizzate ai dati. Tornando al precedente esempio, se Alice invia a Bob un messaggio, allora Bob deve determinare se il messaggio che ha ricevuto è lo stesso che Alice ha inviato.
La soluzione è un *Integrity Detection Scheme*(IDS)[72], che è comunemente implementato usando le *firme digitali*. Un IDS è essenzialmente simile ad uno schema di firma digitale, ma

il suo obiettivo principale è la protezione dell'integrità del dato o documento piuttosto che l'autenticità.

2.3.3 Funzioni hash ed integrità dei dati

L'hash è una funzione unidirezionale, cioè che non può essere invertita, che permette la trasformazione di una stringa di lunghezza arbitraria in una di lunghezza fissa, relativamente ridotta. Tale stringa rappresenta una sorta di "impronta digitale" della stringa iniziale, e viene detta valore di hash, checksum crittografico o message digest.

Processo di verifica dell'integrità Il processo di verifica dell'integrità di un dato consta di due sotto-processi, generazione del checksum e verifica dell'integrità [30].

Generazione checksum Il processo di generazione del checksum è responsabile della creazione del checksum e della memorizzazione per verifiche successive(ad esempio in un database). Il processo avviene in due passi:

1. Il dato di input è ridotto in una stringa minima detta *checksum*,
2. Il checksum è memorizzato per riferimenti futuri.

Verifica dell'integrità Il processo di verifica dell'integrità è responsabile del controllo dell'integrità di un dato di input rispetto ad un checksum precedentemente generato. Questo processo è effettuato in due passi.

1. E' calcolato il checksum dal dato di input(come passo 1 del processo di generazione del checksum);
2. Il checksum memorizzato in precedenza è confrontato con quello generato e viene valutata la modifica o meno del dato;

Tabella 2.3: Verifica integrità mediante hash

Le funzioni hash sono correlate ai *data checksum*, ai *data fingerprint*, ai *codici correttori di errori*(CRC) ed alle funzioni crittografiche hash, le quali tuttavia differiscono nei requisiti e negli obiettivi.
L'idea di base delle funzioni hash è che un valore hash può servire come una *immagine rappresentativa compatta* (impronta digitale o anche message digest) di una stringa di input, e può essere usato come se fosse univocamente identificativo di quella stringa. La funzione di trasformazione che genera l'hash opera sui bit di una stringa qualsiasi, restituendo una stringa di bit di lunghezza predefinita. Spesso il nome della funzione di hash include il numero di bit

che questa genera: ad esempio, SHA-256 genera una stringa di 256 bit.
Non esiste una corrispondenza biunivoca tra l'hash ed il dato. Visto che i dati possibili, con dimensione finita maggiore dell'hash, sono più degli hash possibili, per il *principio dei cassetti*[39] ad almeno un hash corrisponderanno più dati possibili. Quando due dati producono lo stesso hash, si parla di *collisione*, e la qualità di una funzione di hash è misurata direttamente in base alla difficoltà nell'individuare due stringhe che generino una collisione. Per sconsigliare l'utilizzo di algoritmi di hashing è infatti sufficiente riuscire a generare una collisione. Questo è quello che è avvenuto ad esempio per gli algoritmi `SNEFRU`, `MD2`, `MD4` ed `MD5`. Più in dettaglio, una funzione hash *h*, mappa stringhe di bit di lunghezza arbitraria e finita a stringhe di lunghezza fissata, *n*.

2.3.3.1 Tipi di verifica hash dell'integrità

I metodi per la verifica dell'integrità mediante funzioni hash si distinguono in due classi, riassunte nella seguente tabella:

Tipo	**Algoritmo**	**Protezione azioni non intenzionali**	**Protezione azioni intenzionali**
Non Crittografico	CRC-16, CRC-32	SI	NO
Crittografico	MD4, MD5, SHA-1, HMAC	SI	SI

Tabella 2.4: Tipi di verifica hash

Metodi non crittografici La verifica dell'integrità dei dati mediante metodi non crittografici è un buon metodo per individuare modifiche non intenzionali, ma tuttavia non protegge contro modifiche intenzionali ed è inoltre poco utile per proteggere da eventuali attacchi.
Un esempio di metodo non crittografico è **l'algoritmo CRC** (Cyclic Redundancy Code).
CRC è un semplice algoritmo che usa cicli ripetituti per produrre il checksum di output. Esempi di algoritmi non crittografici sono il CRC-16 che produce un checksum a 16 bit e CRC-32 che invece produce checksum da 32 bit. L'algoritmo CRC-32 è usato, ad esempio, in popolari tools di compressione. Kim e Spafford [**?**] non considerano i checksum CRC adatti per il controllo dell'integrità. Infatti essi sono stati pensati per il controllo degli errori dei device hardware, e quindi effettuare il reverse di questi algoritmi è un processo ben conosciuto. Inoltre esistono diversi tools che aiutano un potenziale attaccante a compierlo in breve tempo.

CRC error detection Il seguente testo, preso dal primo canto della Divina Commedia di Dante Alighieri mostra come sia possibile avere due testi differenti che generano il medesimo checksum:

Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura, ché la diritta via era smarrita. Ahi quanto a dir qual era è cosa dura esta selva selvaggia e aspra e forte che nel pensier rinova la paura! Tant'è amara che poco è più morte; ma per trattar del ben ch'i' vi trovai, dirò de l'altre cose ch'i' v'ho scorte. Io non so ben ridir com'i' v'intrai, tant'era pien di sonno a quel punto che la verace via abbandonai. Filename: commedia.txt Date: 19 – 12 – 2008 Time: 17 : 03 Size: 88 CRC-16: $0x1537$	Nel mezzo del cammin di nostra vita mi ritrovai per una selva oscura, ché la diritta via era smarrita. Ahi quanto a dir qual era è cosa dura esta selva selvaggia e aspra e forte che nel pensier rinova la paura! Tant'è amara che poco è più morte; ma per trattar del ben ch'i' vi trovai, dirò de l'altre cose ch'i' v'ho scorte. Io non so ben ridir com'i' v'intrai, tant'era pien di sonno a quel punto che la verace *altered text* strada lasciai. Filename: commedia.txt Date: 19 – 12 – 2008 Time: 17 : 03 Size: 88 CRC-16: $0x1537$

Per proteggere realmente i dati da modifiche intenzionali deve essere usata la verifica dell'integrità crittografica.

Metodi crittografici Le funzioni hash hanno un ruolo essenziale per verificare l'integrità di un messaggio, poiché l'esecuzione dell'algoritmo su un testo anche minimamente modificato fornisce un digest completamente differente rispetto a quello calcolato sul testo originale, rivelando la modifica.
Le funzioni hash possono essere anche utilizzate per la creazione di firme digitali, in quanto permettono la rapida creazione della firma anche per file di grosse dimensioni, senza richiedere calcoli lunghi e complessi. E' infatti computazionalmente più conveniente eseguire con rapidità un hashing del testo da firmare, e poi autenticare solo quello. Infatti in questo modo si evita l'applicazione dei complessi algoritmi di crittografia asimmetrica a moli di dati molto grandi.
La firma digitale, è definita come il *digest* di un documento cifrato con chiave privata. In questo caso l'uso delle chiavi è invertito: la chiave pubblica serve a decifrare la firma e trovare il digest iniziale, mentre quella privata serve a cifrare una stringa anziché a rivelarla.

Utilizzo Hash per firma digitale La strategia che potrebbe essere adottata da un utente A che intendesse firmare un proprio messaggio x di lunghezza arbitraria è la seguente:

1. Calcola z=$h(x)$;
2. Firma il valore hash z costruendo $y = sig_{S_A}$ tramite la sua chiave privata S_A.
3. Trasmette all'utente B la coppia (x,y).

Una volta che il messaggio firmato è stato trasmesso, sia il destinatario B, che chiunque altro risultasse interessato alla verifica della firma effettua le seguenti operazioni:

1. Calcola z=$h(x)$, mediante $h()$ che è pubblica;
2. Verifica che y è la firma di z : $ver_{P_A}(z,y)$=$true$, mediante la chiave pubblica P_A di A.

La scelta di una funzione hash, che permetta di realizzare schemi di firme digitali per documenti di qualsiasi dimensione è un'operazione critica. Infatti, occorre assicurarsi che la funzione individuata non indebolisca la sicurezza dello schema di firma, fenomeno che si verifica quando la funzione produce facilmente *collisioni*. In tal caso è facile intuire che l'attaccante può trovare agevolmente più messaggi x, tali che $h(x)$=z per un certo z di cui conosce la firma, allora può costruire contraffazioni.

Tabella 2.5: Hash e firma digitale

Un hash crittograficamente sicuro non deve quindi permettere di risalire, in un tempo congruo con la dimensione dell'hash, ad un testo che possa generarlo.

Classi di algoritmi crittografici hash Esistono due classi di algoritmi crittografici basati sul calcolo dell'hash dei dati:

- *Unkeyed cryptographic algorithm* hanno in input solo il messaggio. I checksum risultanti sono soddisfacenti se l'origine ed il mezzo su cui sono trasmessi sono **affidabili**.
- *Keyed cryptographic algorithm* hanno due inputs, un messaggio ed una chiave segreta. La chiave è utilizzata per l'autenticazione del mittente, perchè si deve conoscere la chiave per riprodurre l'hash del messaggio. Questi algortimi sono detti "*message authentication codes*" (MAC). Gli algoritmi MAC possono essere visti come funzioni hash che prendono due input funzionalmente distinti, un messaggio ed una chiave segreta simmetrica e producono un output di dimensione fissata, in modo che non sia possibile produrre lo stesso output senza la conoscenza della chiave segreta. Un MAC può essere usato per garantire l'integrità e l'origine dei dati.

Tra gli algoritmi della prima categoria troviamo:

- Algoritmi MD4, MD5 creati da Ron Rivest, creano un hash di 128 bit.
- Algoritmi SHA(Secure hash Algorithm), sviluppati dal NIST. SHA1 si basa su MD4 e produce un checksum di 160 bit, mentre SHA-2, una versione modificata dell'algoritmo genera checksums di 256, 384 e 512 bit, rendendo molto più complesso la possibilità di collisioni(praticamente impossibili).
- Algoritmo RIPE realizzato per sostituire gli algoritmi MD* offre checksums di $160, 256$, 320 bits.

Nella seconda categoria, la classe di algoritmi più famosa è rappresentata dall'algoritmo HMAC(*Hashed Message authentication Code*) che descrive un meccanismo per utilizzare una chiave di cifratura con gli algoritmi precedenti.Infatti esso può essere usato con MD5 o SHA-1 in combinazione con una chiave segreta condivisa. Tali algoritmi forniscono sicurezza pari alla garanzia di integrità e non collisione che forniscono gli algoritmi associati.

2.3.3.2 Attacchi agli algoritmi

Un possibile attaccante ha a disposizione diverse strategie per attaccare o aggirare i controlli che implementano gli algoritmi di verifica d'integrità.
Infatti, non solo gli algoritmi possono possedere qualche debolezza, ma anche gli strumenti che li utilizzano possono non garantire la sicurezza dei dati che manipolano. Alcuni esempi possono essere gli strumenti software che implementano gli algoritmi, i database che contengono i checksums oppure il sistema operativo di supporto ad entrambi.
Vediamo di seguito alcune tipologie di attacco.

Ricerca brute force Una ricerca di tipo "forza bruta" è una ricerca esaustiva che cerca di individuare una collisione che permetta di modificare un dato in modo tale da non essere scoperto mediante verifica hash.
Per l'algoritmo MD5 con hash a 128 bit questo significa avere uno spazio di ricerca ampio 2^{128-1}. Con una tale mole di dati da analizzare non è detto che l'attacco sia fattibile, e per questo non è corretto dire che questo sia esattamente un attacco.

Birthday attack Il birthday attack è anch'esso un attacco verso l'algoritmo di crittografia. L'obiettivo di questo attacco è recuperare due dati che risultano avere lo stesso checksum.
Questo è molto più semplice che trovare il file di input a partire dal checksum. Per un hash di

tipo MD5 trovare due files che posseggono lo stesso checksum crittografico ha una complessità dell'ordine di $2^{128/2}$.

Compromissione del software di verifica integrità Un ulteriore minaccia è data dalla compromissione del software che verifica i dati applicando gli algoritmi di integrità. In quel caso, la corrispondenza o la non corrispondenza dei checksum non sarebbe corretta.
In [30] viene inoltre suggerito o di eseguire verifica del codice delle applicazioni a run-time o deputare la verifica di tali software ad un'altro sistema operativo affidabile. Vedremo nella sezione 2.4 che questo problema è determinante nell'ambito del trusted computing.

Kernel hack Come nel paragrafo precedente, un attaccante può sovvertire il funzionamento del sistema operativo per fornire informazioni errate sui dati da proteggere. Anche in questo caso valgono le raccomandazioni precedenti.

Modifica del Database o del supporto di memorizzazione Alterare un database con i checksum corretti può essere l'opzione più semplice se esso non è protetto adeguatamente. Per nascondere un attacco, l'intruso deve solo sostituire i checksum corretti con dei falsi corrispondenti ad i files alterati. E' fondamentale quindi proteggere il database mediante la sua memorizzazione su storage read-only o su supporti removibili. Un'altra soluzione è applicare una protezione crittografica al database stesso verificandone l'integrità attraverso il calcolo del checksum.

2.4 Trusted Computing

Due grandi tendenze hanno influenzato enormemente l'informatica negli ultimi anni.La prima è la *convergenza*. Possiamo infatti effettuare computazioni usando qualsiasi cosa: dai cellulari ai palmari fino ad i pc desktop ed i rack server - e questi devono poter interoperare. Tutto ciò ha portato ad una ampia consapevolezza ed accettazione di protocolli "*open*" per la comunicazione e linguaggi di programmazione indipendenti dalla piattaforma hardware sottostante per l'esecuzione di programmi come Java o .Net.
La seconda tendenza è la mobilità. Non solo tutti i diversi dispositivi informatici devono poter comunicare ed interoperare, ma devono essere sempre in grado di far accedere i propri utenti alla maggior parte dei programmi e dati che normalmente utilizzano.
Entrambi le tendenze hanno aumentato significativamente l'importanza di una "*piattaforma comune*". Essa è rappresentata al giorno d'oggi dal linguaggio di programmazione che esegue il codice eseguibile *trasportabile* indipendentemente dalla piattaforma. Divenendo completamente

dipendenti da questa infrastruttura informatica, i suoi problemi e le sue debolezze sono emerse in maniera prepotente. Da un lato si hanno danni causati periodicamente da virus e worms come *Nimda* o *SQL Slammer*. Dall'altro, i produttori e distributori di contenuti digitali vogliono sempre più controllare la diffusione delle loro creazioni. Considerando la rete come un luogo per natura ostile, abbiamo la necessità di assicurarci quindi che il nostro partner, appartenente alla piattaforma di servizi condivisa, non sia anch'esso ostile o non possieda software malevolo. C'è bisogno quindi di porre determinati vincoli sulla sua integrità a garanzia della fiducia riposta in esso durante la comunicazione.

Concetto di fiducia Un modo standard e poco flessibile di ottenere *assurance* è, come abbiamo visto nel capitolo 1 l'utilizzo di *sistemi chiusi*. In genere, essi sono realizzati da un singolo produttore in base a specifiche molto rigide e gli sviluppatori interagendo con un sistema dedicato possono facilmente soddisfare i requisiti di sicurezza. Quando un sistema chiuso comunica con un'altro simile, sa esattamente quale può essere il comportamento del partner remoto. Esempi di sistemi chiusi sono le console di gioco proprietarie od i sistemi industriali.
I *sistemi aperti*, come i normali pc od i cellulari o i palmari, possono essere facilmente modificati, e per questo soggetti a malintenzionati che possono provocarne facilmente il funzionamento scorretto. Due sistemi aperti che comunicano non possono assumere niente circa il comportamento del proprio partner.
I principali problemi posti dai meccanismi di sicurezza delle attuali tecnologie sono legati principalmente alla natura della loro implementazione, che è completamente realizzata in software. Una volta elusi questi meccanismi, non si è in grado di rilevare se la piattaforma è stata compromessa, venendo meno alla proprietà di fiducia. Questi problemi hanno portato alla consapevolezza della necessità di un nuovo concetto di architettura *trusted*. La creazione di un ambiente *trusted* è di solito realizzato tramite meccanismi hardware, che non possono essere sovvertiti da applicazioni software.
Un esempio di meccanismi hardware può essere preso dai moderni sistemi operativi. Esistono, infatti, controlli hardware che cooperano con controlli software per poter garantire alcune proprietà. Un'esempio, presente nella gestione della memoria[55], è quello che permette di assegnare ad ogni processo una zona di memoria particolare e di bloccarlo nel caso tenti di accedere a zone esterne. Per implementare questo controllo si utilizzano due meccanismi, uno software e uno hardware; il sistema operativo definisce a livello software delle tabelle, le *tabelle delle pagine*, che contengono le zone di memoria virtuale del processo. Nell'architettura hardware è presente un componente MMU (Memory Management Unit) che traduce gli indirizzi logici in fisici, controlla e blocca il processo che tenta di violare il proprio spazio di indirizzamento.
Lo sforzo di portare alcune caratteristiche e proprietà dei sistemi chiusi verso i sistemi aperti e

di implementare ad un livello sempre più basso dell'architettura il componente garante dell'integrità, è rappresentato dal ***Trusted Computing***(TC) che introduce meccanismi e componenti sia nell'hardware che nel software per verificare l'**integrità del sistema** e che permettono l'autenticazione verso un'altro sistema remoto.

2.4.1 Caratteristiche

Le piattaforme trusted garantiscono delle proprietà che vengono mantenute attraverso le componenti funzionali, i blocchi fondamentali su cui si poggiano i vari livelli dell'architettura; I controlli utilizzati nel TC richiedono tre meccanismi fondamentali:

- ***Boot Sicuro***: garantisce che il sistema sia caricato senza compromissioni in modo che il sistema operativo preveda determinate politiche di sicurezza. Se questo meccanismo non è implementato, un attaccante potrebbe eseguire un sistema operativo diverso che non consideri tali politiche.

- ***Isolamento Forte***: garantisce la prevenzione da compromissioni del sistema dopo l'avvio.

- ***Attestazione Remota***: necessaria per la certificazione dell'autenticità del software che è eseguito verso il partner di comunicazione.

2.4.2 L'architettura della Piattaforma Trusted

Alla base del processo di attestazione c'è un componente hardware detto *trusted module*(TM) [11]. Questo modulo memorizza la parte privata di una coppia di chiavi asimmetriche. La coppia di chiavi è certificata da un'autorità di certificazione(CA). Il TM possiede anche una piccola quantità di memoria persistente. L'hash del bios, ad esempio, è firmato utilizzando la chiave privata del TM e memorizzato in tale memoria.
All'avvio della macchina, il controllo viene passato, tramite una procedura firmware(detta CRTM) al TM, che ricalcola l'hash del BIOS e lo confronta con quello memorizzato. Se sono gli stessi, allora abbiamo la prova che il BIOS non è stato manomesso,ed il controllo quindi passa al BIOS.
Il BIOS, che consideriamo trusted, applica lo stesso controllo sul *bootloader*, passandogli successivamente il controllo e così il boot loader sul sistema operativo.
Tutti i componenti di questa catena hanno una propria coppia di chiavi pubbliche e private, che è certificata, cioè firmata con la chiave privata, dal livello immediatamente precedente nell'ordine della catena. Questa chiave, al turno successivo, è utilizzata per certificare il livello superiore. Più precisamente, ogni livello firma due dati del livello superiore:

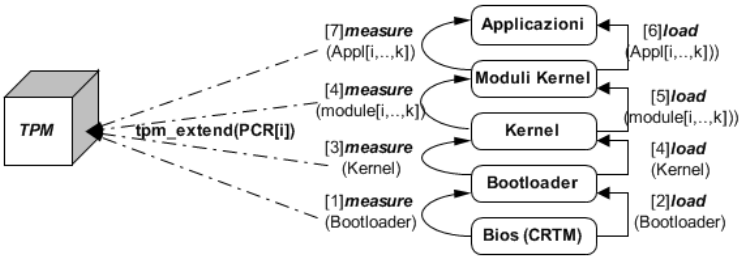


Figura 2.4: Trusted chain

- l'hash della sua immagine eseguibile;
- la sua chiave pubblica.

Questo processo lega la chiave pubblica al software eseguito da quel livello. Occorre anche notare che la chiave privata di ogni livello lungo questa catena, detta *chain of trust*, deve essere mantenuta segreta al livello superiore. Come illustrato in figura 2.4 ogni livello calcola per prima cosa il digest del livello successivo e lo confronta con quello presente nel TPM. Successivamente carica il livello successivo e gli passa il controllo. Al completamento di questa catena di verifiche, il sistema può dirsi, secondo le specifiche date, in uno stato sicuro ed affidabile, e l'insieme di componenti "*certificati*" è detto **Trusted Computing Base**. Quindi, una combinazione di hardware, il TM, e software, usato per il boot e l'isolamento, permette di garantire integrità ed autenticità dei sistemi trusted. Utilizzando il processo illustrato, si può dedurre facilmente lo stato del sistema e, con questo dato, è possibile certificarlo ad un partner remoto.

2.4.2.1 Root of trust

Una normale piattaforma informatica sia essa fornita o no di meccanismi di sicurezza software, viene trasformata in una piattaforma trusted, TP, con l'aggiunta delle *root of trust*. Una TP consente alle entità cooperanti, siano esse utenti o altre piattaforme, di determinare lo stato in cui essa si trova, ovvero i software caricati nell'istante della rilevazione, e di memorizzare dati al suo interno in maniera tale da consentirne l'utilizzo esclusivamente in presenza della medesima configurazione.
L'entità cooperante con una TP, sarà in grado quindi di richiedere informazioni in merito alla configurazione software e, sulla base dei dati che riceverà, valutare se è accettabile o no.
Nel caso in cui essa giudichi lo stato come fidato, potrà eseguire transazioni e memorizzare informazioni utilizzando le funzionalità crittografiche offerte dalla TP. Per consentire questa attestazione, la Trusted Platform deve essere in grado di fornire informazioni dalle quali sia

possibile dedurre deterministicamente lo stato del software operante sulla piattaforma, ed in un secondo tempo, abilitare l'utilizzo delle chiavi necessarie alla decifrazione delle informazioni precedentemente memorizzate.
Ambedue le funzionalità descritte richiedono opportune metriche per la misura d'integrità del sistema, ovvero dati in grado di riflettere univocamente lo stato del software in esecuzione sulla TP.
La misura, memorizzazione e presentazione in un formato idoneo di queste metriche d'integrità è realizzabile attraverso *tre* Root of Trust, note come *Root of Trust for Measurement* , RTM, *Root of Trust for Storing*, RTS, e *Root of Trust for Reporting*, RTR. Esse permettono di realizzare le misure e si occupano inoltre della memorizzazione e della gestione dei risultati prodotti.

Root of trust for measurement La RTM è in grado di fornire misure d'integrità affidabili, tipicamente realizzate mediante i normali self-tests eseguiti dalla macchina in fase di avvio e controllati dal *Core Root of Trust for Measurement*, CRTM. Il CRTM formato dal codice eseguibile che implementa le funzionalità della RTM ed è eseguito dalla piattaforma come primo passo nel processo di misurazione. Pertanto, esso rappresenta il punto di partenza dal quale è possibile estendere la fiducia.
Il nucleo delle misure d'integrità implementa le funzionalità in grado di generare quelli che vengono definiti *measurement events*. Questi *measurement events* rappresentano le misure effettuate sulle singole componenti hardware o software e si dividono in due classi:

- **misure per valore**: rappresentazione dei dati del componente o del codice del software,
- **misure di tipo digest**:rappresentazione dei dati misurati mediante funzioni crittografiche, utilizzate per identificare lo stato in cui si trova la macchina.

I digest vengono memorizzati all'interno del TM, in registri detti Platform Configuration Registers(PCRs). Tale processo avviene in maniera da identificare lo stato della piattaforma, in base non solo alle misure delle componenti, ma anche in base all'ordine con cui le misure vengono effettuate.

Root of Trust for Storing La RTS consente la memorizzazione nella memoria non volatile gestita dal TM, delle misure effettuate dalla RTM. Contestualmente, essa realizza un'indicizzazione della sequenza con la quale sono state effettuate.
Come vedremo, l'ordine con il quale vengono effettuate le misure risulta di fondamentale importanza in tutti quei processi che richiedono una verifica della configurazione di sistema, primo fra tutti quello di *attestazione remota*.

Root of Trust for Reporting Questa Root of Trust si occupa della formattazione delle misure effettuate per mezzo della RTM e dell'attestazione di autenticità di tali valori. I report d'integrità vengono firmati digitalmente attraverso una chiave, detta l'Attestation Identity Key (AIK). Questa operazione è in grado di garantire l'autenticità della configurazione poichè la AIK, come vedremo, viene generata ed utilizzata esclusivamente dal TM.

2.4.2.2 Funzionalità principali delle Trusted platforms

I processi e le componenti descritte permettono di illustrare le funzionalità introdotte dalle TP. Esse permettono di definire servizi di attestazione remota della configurazione (*Remote Attestation*), protezione dei dati (*Protected storage*) e la rilevazione dello stato della piattaforma.

Protected storage Per motivazioni economiche, il TPM, comprende un numero limitato di registri, quindi la sua architettura non è idonea alla memorizzazione di grandi quantità di dati. Il protected storage consente di ovviare questo problema, proteggendo i dati residenti all'esterno del dispositivo. Esso di fatto, consente la protezione di una quantità teoricamente illimitata di dati. Le informazioni protette dal TM consistono in chiavi e dati arbitrari. Mentre i dati vengono tipicamente svelati all'utente, le chiavi rappresentano spesso un'informazione il cui utilizzo è riservato al TM. Si consideri ad esempio, ad una chiave privata utilizzata per la firma digitale.

Binding La proprietà di binding consiste nella protezione di dati sensibili, come le chiavi, attraverso la loro cifratura mediante l'utilizzo di chiave pubblica, un meccanismo in grado di garantire confidenzialità ed integrità delle informazioni protette.
Le operazioni di binding effettuano inoltre un controllo sugli accessi ai dati protetti, in modo da poter garantirne l'integrità, al momento della decifrazione di un'oggetto. Esso infatti contiene un digest del segreto di autorizzazione, inserito al momento della cifratura. Durante il processo di decifratura il segreto è confrontato con quello presentato al TM dall'utilizzatore. Se il confronto dà esito positivo, l'utente sarà in grado di effettuare modifiche ai dati mentre in caso contrario verrà negato l'accesso alle informazioni richieste.

Sealing La proprietà di *sealing* offre le medesime funzionalità e sicurezze garantite dal binding ma introduce un'ulteriore condizione nell'accesso ai dati cifrati. Infatti, è possibile specificare all'atto della cifratura, lo stato della piattaforma mediante l'ausilio dei registri PCR, condizionando di fatto l'utilizzo delle informazioni protette ad una specifica configurazione.
Questa peculiarità fornisce la garanzia che i dati vengano utilizzati esclusivamente sulla piattaforma con la quale sono stati protetti. Durante le operazioni di *unseal*, il TM verifica lo

stato in cui si trova la piattaforma calcolando un digest sulla lista e relativi valori contenuti nei PCR. Il risultato di questa operazione viene successivamente confrontato con quello presente nell'oggetto protetto e l'esito comporta l'acquisizione o meno, dei diritti di accesso alle informazioni.

Key management Come abbiamo visto, gli oggetti protetti dal TM vengono cifrati attraverso chiavi di crittografia memorizzate all'esterno del dispositivo. Tali chiavi risultano a loro volta oggetti protetti. Gli oggetti chiave possono essere caratterizzati da proprietà di migrabilità cioè possono essere creati all'esterno del TPM o all'interno ed ad esse è consentita l'esportazione da una TP ad un'altra.
La struttura degli *oggetti protetti* è realizzata attraverso un'albero in cui ciascun figlio rappresenta un oggetto protetto per mezzo del nodo padre. La creazione e l'aggiornamento dell'albero sono operazioni soggette ad alcuni vincoli:

- Una chiave dedicata alle operazioni di firma (*signature key*) non può essere padre di una dedicata esclusivamente alle operazioni di cifratura e decifratura (*storage key*) e viceversa;
- Una chiave caratterizzata da attributi di migrabilità non può essere padre di una non migrabile, mentre il caso inverso è consentito.

Tutti i nodi aventi figli all'interno dell'albero consistono in storage keys mentre i nodi foglia sono rappresentati da dati arbitrari e signature keys. La radice della gerarchia è rappresentata da una chiave non migrabile, detta *Storage Root Key* (SRK), dal quale è possibile stabilire un cammino in grado di portare l'utilizzatore alla chiave che deve utilizzare.

La figura 2.5 mostra la struttura della gerarchia di oggetti protetti, utilizzata dalle TP. Qui di seguito i diversi oggetti indicati:

- *Storage keys*, chiavi in grado di cifrare e decifrare dati arbitrari ed altre chiavi. Consentono inoltre l'utilizzo delle operazioni di *seal* ed *unseal*, definite dal TCG.
- *Signature keys*, chiavi il cui utilizzo è dedicato esclusivamente all'implementazione della firma digitale.
- *Identity keys*(AIK), chiavi utilizzate da applicazioni aderenti alle specifiche del TCG. Sono in grado di attestare l'integrità dei dati prodotti e la genuinità del TM che li ha elaborati.
- *Bind keys*, consistono in chiavi generiche, utilizzabili da applicazioni non aderenti alle specifiche.

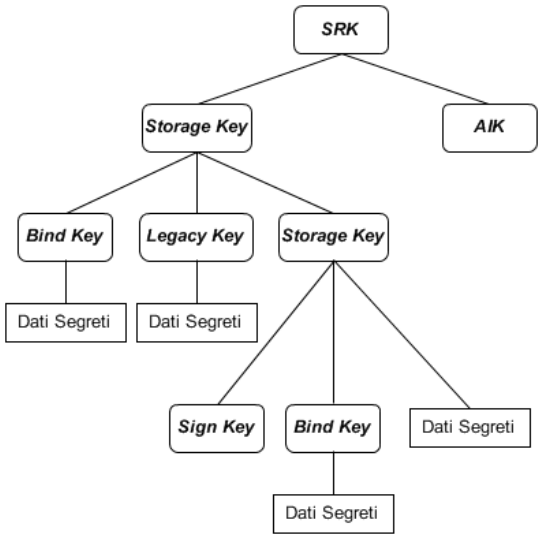


Figura 2.5: Gerarchia chiavi nelle specifiche TCG

- *Legacy keys*, chiavi utilizzate mutualmente per operazioni di firma e (de)cifratura. Anch'esse non risultano conformi alle specifiche.

Ciascuna TP è in grado di creare e gestire chiavi simmetriche ed asimmetriche. Per quanto concerne la crittografia asimmetrica, il TM adotta l'algoritmo RSA ed è in grado di generare e gestire le chiavi in maniera autonoma, senza incidere sulle prestazioni dell'intera piattaforma. Ogni chiave può assumere molteplici attributi che ne caratterizzeranno le funzionalità di utilizzo. All'interno del TM si distinguono diversi tipi di chiave, le più importanti sono:

- *Endorsement key*(EK), la EK è generata ed inserita nel dispositivo TM al momento della sua produzione. Essa è utilizzata nel processo di autenticazione del proprietario come chiave di decifratura dei dati di accesso e nel processo di creazione delle Attestation Identity Key. Il suo utilizzo è limitato alla decifrazione dei dati. Le specifiche sconsigliano caldamente l'uso di questa chiave nei colloqui in rete, poichè il suo utilizzo comporterebbe l'identificazione univoca del calcolatore configurando di fatto, un problema di privacy.

- *Storage Root Key*(SRK), Una volta autenticato, il proprietario può creare questa chiave. Essa consente la gestione di tutte le chiavi che saranno utilizzate per la cifratura di oggetti memorizzati all'interno ed all'esterno della memoria non volatile, ed è utilizzata a sua volta per cifrare le chiavi figlie.Il processo mediante il quale è possibile vincolare una chiave ad un'altra, è detto *wrapping* mentre l'inverso *unwrapping*.

- *Attestation Identity Key*(AIK), sono definite come alias della EK, infatti la EK non è uti-

lizzabile per operazioni di firma in modo da garantire l'impossibilità dell'identificazione univoca attraverso la rete. Queste chiavi possono essere generate in qualsiasi momento purchè vengano dimostrate le credenziali del TM proprietario, ovvero il legittimo utilizzatore della piattaforma. Il numero di AIK generabili è illimitato. Il proprietario del TM controlla tutti gli aspetti relativi alla generazione ed attivazione delle AIK, compresi i dati associati a tali chiavi. Le AIK sono chiavi asimmetriche RSA di 2048 bit. L'utente che intende utilizzarle dovrà fornire la prova di legittimità. Il meccanismo di generazione delle AIK fa sì che esse possano essere utilizzate esclusivamente per la firma, impedendo completamente di usarle per la cifratura.

2.4.3 Trusted Platform Module

I componenti principali che garantiscono la fiducia delle piattaforme trusted sono le *roots of trust*. Questi componenti sono rappresentati dal punto di vista fisico da un chip definito dalle specifiche del Trusted Computing Group(TCG)[11], il TPM [19] [41] e da un firmware, il CRTM, e dal punto di vista logico da tre oggetti rappresentati da RTS (Root of Trust for Storing), RTR (Root of Trust for Reporting) e RTM (Root of Trust for Measurement) descritte in precedenza. Le prime due roots of trust logiche sono associate al TPM e svolgono il compito di registrazione e reporting delle metriche, rispettivamente RTS e RTR. L'altra componente, l'RTM è associata al CRTM e il suo compito è quello di effettuare la misura degli oggetti della piattaforma. La fiducia nella piattaforma è garantita da questi tre principali blocchi funzionali, roots of trust logiche, che per questo devono essere attestati durante il processo di produzione della piattaforma. In particolare, durante la fase di attestazione devono essere prodotte le seguenti certificazioni:

- **Trusted Platform Module Entity** (TPME): questo certificato attesta la genuinità del TPM. Viene creata la endorsement key, la cui parte privata della chiave è inserita all'interno del TPM. La parte pubblica invece viene inserita nel certificato TPME, che viene firmato dal produttore.

- **Validation Entity** (VE): questo certificato attesta il corretto funzionamento del software che implementa le metriche nella piattaforma contenuto all'interno del componente fisico CRTM.

- **Conformance Entity** (CE): questo certificato attesta che il progetto del sottosistema TCG è conforme alle specifica rilasciata. In particolare il CE attesta che sia la progettazione che l'inserimento nella piattaforma del TPM seguono le specifiche TCG.

- **Platform Entity** (PE): Questo certificato attesta che un certo tipo di piattaforma contiene uno specifico TPM. Di solito, questo tipo di attestazione viene eseguita dal produttore della piattaforma, che firma il certificato attestando che la piattaforma è corretta rispetto alle specifiche TCG.
- **Privacy Certification Authority**(Privacy-CA): Questa CA viene scelta dal proprietario della piattaforma per attestare che un certo pseudonimo appartiene ad una certa Piattaforma. Nella fase di attestazione dell'identità della piattaforma la CA deve ricevere i certificati TPME, PE, e CE e fidarsi della loro attestazione;

2.4.3.1 CRTM

Dal punto di vista fisico, il CRTM è rappresentato da una memoria non volatile dove è contenuto il primo codice che deve essere eseguito sulla piattaforma, il cui compito è quello di effettuare le misure d'integrità sugli oggetti che compongono il sistema.

Per questo componente, la specifica TCG definisce diverse proprietà che devono essere garantite per un corretto funzionamento, tra queste:

- resistere a qualsiasi tipo di attacco sia software che hardware;
- implementare misure precise attraverso delle funzioni di metrica d'integrità, che riescono ad assegnare un valore univoco ai diversi oggetti della piattaforma misurati;
- fornire funzionalità di registrazione accurata all'interno del TPM delle misure delle metriche d'integrità calcolate, ;
- fornire funzionalità di registrazione accurata delle azioni coinvolte nel processo di registrazione delle metriche.

Inizialmente, questo componente era memorizzato nello spazio di memoria dove risiede il BIOS ed era separato dal TPM, perchè le specifiche TCG non impongono che esso sia tamper-proof. Ma questa specifica può essere considerata una debolezza, infatti il Bios può essere accessibile via software, ed è ormai superata nelle nuove piattaforme che includono il CRTM nel TPM stesso.

2.4.3.2 TPM

La seconda root of trust della piattaforma è il TPM. Esso rappresenta il cuore dell'architettura e insieme al CRTM completa i blocchi fondamentali che garantiscono la fiducia. Il TPM è

paragonabile ad un ambiente di esecuzione fidato ed è composto da un proprio processore e da altri blocchi funzionali
I componenti più importanti sono:

- un generatore di numeri casuali;
- un componente che fornisce la funzionalità dell'HMAC;
- un generatore di chiavi crittografiche assimetriche RSA;
- un motore di cifratura e firma RSA;
- un motore per il calcolo delle funzioni digest SHA1

Oltre a questi componenti funzionali, hanno una grande importanza le zone di memoria all'interno del TPM dove sono presenti due tipologie di registri la cui funzionalità è legata alla conservazione delle misure calcolate, i PCR e i DIR. I PCR(*Platform Configuration Register*) sono 16 registri in una zona di memoria volatile all'interno del TPM. Essi vengono utilizzati per contenere le misure di integrità calcolate dai vari agenti durante il ciclo di vita del sistema. I primi 8 registri vengono utilizzati nella fase di boot della piattaforma in cui vengono misurati i firmware dei componenti hardware, il codice del BIOS etc., mentre gli altri 8 non vengono utilizzati dalla specifica e vengono lasciati a disposizione dello sviluppatore di applicazione per piattaforme trusted. La seconda tipologia di registri, i DIR, *Data Integrity Register*, sono situati nella memoria non volatile del TPM e sono utilizzati per poter conservare le misure effettuate dal TPM durante l'intero ciclo di vita del sistema. Tuttavia essi scompaiono dalla v.1.2 delle specifiche TCG.

2.4.3.3 Controllo d'integrità

La seconda funzionalità offerta dalle piattaforme trusted è quella del controllo d'integrità. Questa funzionalità permette di misurare gli oggetti del sistema e, insieme alla roots of trust del TPM, registrare e riportare le misure in modo fidato. Il concetto di misura che è utilizzato da questa funzionalità ha come scopo quello di rilevare un'*impronta* univoca dei dati e dei programmi che una piattaforma contiene. In questo modo eventuali modifiche illegali di questi oggetti, per esempio effettuate da virus o worms, possono essere immediatamente scoperte da un verificatore remoto che vuole controllare lo stato della piattaforma, prima di avviare una comunicazione contenente dati sensibili.
Il controllo d'integrità, è affidato a componenti che collaborano per garantire la correttezza dell'attività. Tra questi troviamo innanzitutto le due roots of trust, CRTM e TPM,e gli agenti

che applicano le misure, che vengono creati nella fase di boot del sistema e che ereditano la funzionalità di RTM (root of trust for measurement) dal CRTM. Altro componente è il database TPMS, che raccoglie la lista cronologica delle misure effettuate sui componenti con i relativi risultati, esso utile in fase di attestazione remota per avere la storia della validità dei componenti.

PCR (Platform Configuration Register) La zona di memoria adibita alla registrazione delle misure degli oggetti della piattaforma è rappresentata da una zona protetta del TPM dove sono presenti i 16 registri PCR. La dimensione dei registri PCR è di 20 byte, la scelta di questa grandezza è data dal fatto che i progettisti dell'architettura associano la misura degli oggetti della piattaforma ad una funzione d'integrità SHA1 (160 bit). La dimensione di questi registri è quindi dipendente da questa dimensione.
Un problema che i progettisti dell'architettura hanno dovuto risolvere è quella di numero di registri finiti e infinite misure, o comunque un numero di registrazioni non definibile a priori. Per questo problema è stato creato il seguente sistema di misura dei componenti, chiamato "*misurazione a catena*". Ogni volta che viene effettuata una misura tramite la funzione d'integrità di qualche oggetto (x), viene recuperato il valore (v) presente nel registro destinazione e viene concatenato (*operazione* "||") con la nuova misura da registrare(c), infine viene applicata sul risultato ottenuto la funzione d'integrità *SHA1*. Il risultato di quest'ultima operazione viene infine registrata nel registro specificato (*pcr[i]*) come illustrato dalle seguenti regole:

$$\begin{aligned} n &= SHA1(x); \\ c &= SHA1(v||n); \\ pcr[i] &= c; \end{aligned}$$

In questo modo si possono registrare infinite misure con un numero finito di registri. L'unico vincolo di questo metodo si ha nella fase di verifica della misura. Infatti per ottenere lo stesso valore presente nel registro, occorre riprodurre le operazioni nello stesso ordine cronologico della fase di misura, per questo l'architettura specifica l'esistenza di un database esterno che contenga l'ordine delle misure con i relativi oggetti.
L'architettura, inoltre, non pone vincoli di sicurezza su questo database che può essere considerato non fidato. Una sua eventuale compromissione è facilmente rilevabile dal valore custodito all'interno di uno dei registri PCR, che per sua natura è invece considerato fidato. Anche per i comandi che permettono di scrivere all'interno dei PCR, la specifica non prevede nessun tipo di autenticazione. Infatti, la scrittura nei registri PCR non è distruttiva ma cumulativa, una volta che un programma ostile volesse inserire delle misure fasulle all'interno dei registri, il sistema le rileverebbe immediatamente nella fase di verifica delle misure, poichè per raggiungere il valore

contenuto nel registro PCR dovrebbe considerare anche l'oggetto che porta a quella misura falsata.
Le uniche funzioni che permettono di scrivere all'interno dei registri sono:

- il comando TPM *extend*;
- tramite il reboot della piattaforma che nella fase di *inizializzazione* del TPM pone tutti i registri PCR al valore 0.

DIR (Data Integrity Registers) Oltre ai registri PCR, utilizzati prevalentemente per contenere le misure degli oggetti del sistema durante il ciclo di vita della piattaforma, all'interno del TPM troviamo altri registri della stessa dimensione dei PCR denominati registri DIR (Data Integrity Register). Questi registri sono contenuti nella memoria nonvolatile del TPM e sono utilizzati per mantenere i valori delle misure quando la piattaforma è spenta. Essi vengono utilizzati principalmente nella fase di boot sicuro, quando si ha bisogno di un parametro di confronto calcolato in una fase precedente.
La scrittura in questi registri, a differenza dei registri PCR, è non cumulativa, per questo motivo una loro sovrascrittura potrebbe eludere il sistema di integrità della piattaforma. La specifica impone quindi un comando con autenticazione per poter operare su di essi.
I registri DIR e PCR vengono utilizzati in quattro compiti correlati al controllo d'integrità:

- Il Boot sicuro è usato per avere un fase di boot fidato della piattaforma, vengono in questo caso confrontati valori contenuti nei PCR, con valori contenuti nei registri DIR;
- Il Boot autenticato viene usato per poter misurare i componenti che operano nella fase di iniziale del sistema;
- Protezione dei segreti, questa funzionalità permette di rilasciare segreti solo in particolari condizioni della piattaforma, associate a metriche particolari contenute nei registri PCR;
- L'integrity challenge, usato da un eventuale verificatore remoto (*challenger*) che vuole assicurarsi, prima di effettuare delle transazione con la piattaforma, che essa sia in uno stato fidato;

2.4.4 Remote Attestation

L'*attestazione remota*(Remote Attestation) è un metodo mediante il quale un host(client) autentica la propria configurazione hardware e software verso un host remoto(server).
L'obiettivo dell'attestazione è di abilitare un sistema remoto a determinare il livello di affidabilità

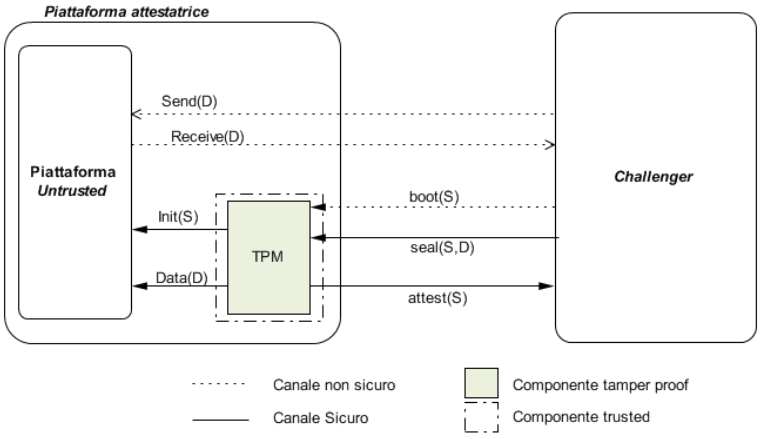


Figura 2.6: Attestazione, modello TCG.

basandosi sull'integrità della piattaforma di un'altro sistema.
In generale, l'architettura per l'attestazione remota consiste in due componenti primari: l'*architettura di misura dell'integrità* ed il *protocollo di attestazione*. In breve, nelle seguenti sezioni verranno illustrati alcuni tra i modelli di architettura con relativi protocolli di attestazione più citati in letteratura.

2.4.4.1 Modello TCG

Il modello di remote attestation iniziale, su cui poi sono stati basati i successivi è stato elaborato dal *Trusted computing group* ed è basato su due componenti fondamentali:

- Il componente hardware TPM
- La PKI realizzata mediante una *terza parte fidata*(TTP)

L'attestazione remota diviene quindi realizzabile solo dopo la corretta combinazione ed operatività di questi due componenti. Il modello formulato dal TCG, è mostrato in figura 2.6.

I sistemi che si scambiano i messaggi sono connesse mediante canali sicuri ed insicuri. `C` può comunicare con `U`, per eseguire normali operazioni, usando il canale di output `send()` e ricevere dal canale `receive()`, entrambi non sicuri. Il TPM fornisce un canale insicuro di inizializzazione del protocollo `boot()`, con cui comunicare con eventuali partners remoti(challenger `C`). Il canale ha come parametro uno stato S_0, che viene memorizzato localmente e poi inviato verso `U` usando il canale sicuro `init(S)`. Tale stato rappresenta la configurazione iniziale da eseguire su `U` al boot del sistema, che rappresenta anche la catena

di hash. Mediante il canale sicuro `attest(S)`, il TPM restituisce a `C` lo stato utilizzato per inizializzare `U`, quindi un canale sicuro detto `seal()` è utilizzato per ricevere il dato `D` ed una configurazione S_i.Se S_0 è uguale ad S_i, `D` è inviato ad `U` mediante il canale sicuro `data(D)`.

Uno dei concetti sui quali si basa l'attestazione remota è costituito dallo *stato di sistema S*. Con stato si intende l'insieme che comprende le misure hardware e software rilevate durante l'avvio della piattaforma, memorizzate nei registri PCR, unitamente al registro degli eventi misurati, memorizzati nello *Stored Measurement Log* (SML).
L'insieme dei *digests* contenuti nei registri PCR, rappresenta lo stato corrente della piattaforma, mentre lo SML consente la ricostruzione di tali valori.
La funzione dello SML è proprio quella di consentire al partner di calcolare nuovamente le misure che hanno caratterizzato l'avvio della parte remota ed il loro ordine, in maniera tale da poterdecidere se essa è fidata. Nella piattaforma così configurata, l'integrità dei componenti è *misurata* nel momento in cui sono eseguiti, e memorizzata nei PCR. La piattaforma, inoltre, mantiene tali informazioni sotto forma di "eventi" mediante l'SML.L'integrità dei componenti misurati è elencata nell'SML, e l'integrità dell'SML è mantenuta attraverso i PCR e le misure dell'architettura TCG.
L'implementazione dell'attestazione remota è possibile direttamente tra le due TP coinvolte, e per poter far ciò è necessario firmare le informazioni, per poter verificare l'autenticità delle stesse, con chiavi uniche e segretamente custodite dalle due piattaforme.
Le specifiche risolvono questo problema mediante le *AIK*, che, come abbiamo già visto, sono chiavi la cui generazione è riservata unicamente al proprietario del TPM. Esse rappresentano le identità della TP, nonchè degli pseudonimi della EK. Il loro utilizzo non consente l'identificazione diretta della TP, risolvendo a priori il problema della privacy. L'utilizzo della identità della piattaforma è reso possibile da una **certificazione** in grado di garantire l'autenticità delle chiavi *AIK* e la loro effettiva corrispondenza ad un TPM genuino. L'entità in grado di fornire tale certificazione, è descritta nelle specifiche come **Privacy CA** o *Trusted Third Party*(TTP), e per poter svolgere il proprio compito, entra a far parte del processo di creazione delle chiavi d'identità.

Si assume che la Privacy-CA conosca la parte pubblica delle *EK* dei TPM non corrotti. In questa maniera, nel momento in cui un TPM voglia abilitare l'autenticazione con un partner remoto, genera le AIK, una coppia di chiavi RSA, e invia la parte pubblica alla CA, che la associa alla EK e genera un certificato. Tale certificato è poi utilizzato dal TPM per autenticare i propri dati di integrità al partner. La Privacy CA rappresenta quindi l'ente certificatore preposto a garanzia delle identità di ciascuna piattaforma.
Una volta create le chiavi AIK e certificate dalla Privacy-CA le due piattaforme possono scambiare i dati per la verifica dell'integrità.

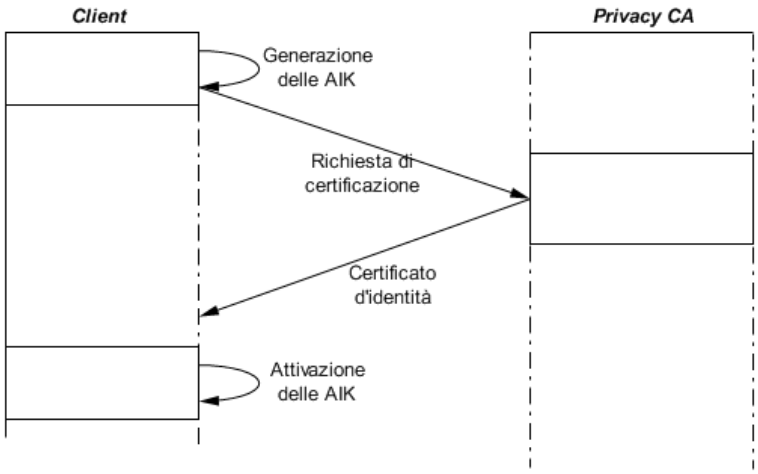


Figura 2.7: Attestazione, trusted CA.

La parte attestatrice invia alla parte remota le misure della configurazione della macchina o di una o più applicazioni, così come memorizzate nel TPM. Queste ultime, insieme ad un valore *Nonce* che garantisce la "*freschezza*" dei dati vengono firmate mediante la AIK. La Privacy-CA permette di garantire la chiave ed il suo reale proprietario.
Al termine della comunicazione, il partner remoto può verificare lo stato della piattaforma attestatrice ed applicare la propria logica nella gestione della comunicazione.

2.4.4.2 Modello IBM

Nel modello di misurazione formulato da IBM, chiamato Integrity Measurement Architecture (IMA) [66] [73] il modello del TCG è esteso, attraverso una estensione del kernel caricata dopo la fase di trusted boot. Mediante questa estensione, è possibile la misura granulare dei diversi componenti caricati durante l'esecuzione del sistema operativo.
Il meccanismo di misura consiste in una base di misure permanente, che viene estesa nel momento in cui un qualche processo o libreria viene caricato.
La parte iniziale, all'avvio della piattaforma, rimane invariata, e le misure effettuate su BIOS, bootloader e librerie caricate all'avvio del sistema operativo vengono utilizzate come base di affidabilità.
L'estensione abilita il kernel ad automisurarsi ed a misurare il codice di qualunque altro processo debba essere eseguito, **prima** di mandarlo in esecuzione.

Gli hash calcolati sui componenti, vengono memorizzati in una lista di misure, non modificabile, che definisce la storia dell'integrità del sistema dal suo ultimo avvio. Tale integrità è garantita mediante misure del kernel. Esse sono memorizzate mediante la funzione `TPM_extend`,

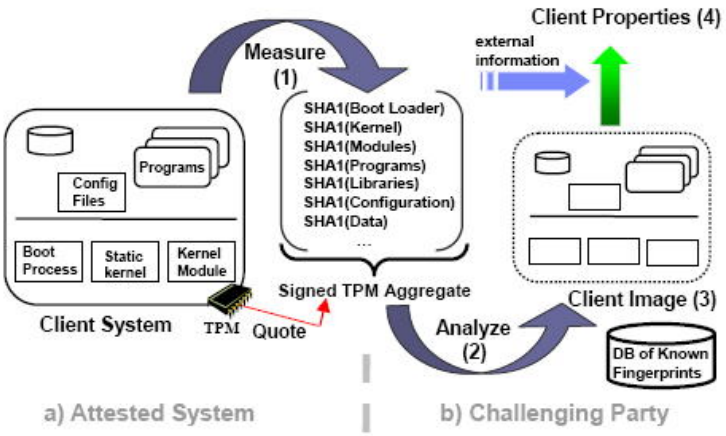


Figure 1: Attestation Architecture Overview

Figura 2.8: Architettura IMA [66]

che ha come parametro n, un intero di 160 bit, ed un intero i. Le misure effettuate vengono aggregate nei vari PCR con l'elaborazione della funzione:

$$PCR[i] = SHA1(PCR[i]||n); \tag{2.1}$$

In questo modo, il contenuto dei PCR mantiene una storia che non può essere modificata da alcun software ostile. Infatti, escludendo il caso del riavvio della macchina che resetta i PCR, ai registri può essere solo aggiunto un dato mediante la funzione hash SHA-1. L'architettura realizzata da IBM permette anche di inserire, all'interno dell'architettura, controlli ad-hoc, per la verifica della consistenza di files di input letti all'avvio, ad esempio files di configurazione, mediante le applicazioni stesse precedentemente misurate.
Tutto il procedimento di misura dei processi e dei files è implementato attraverso il sistema di hooks LSM(*Linux Secure Module*) dei kernel linux che l'architettura sfrutta. Ad ogni richiesta di esecuzione o di modifica, il modulo intercetta la richiesta e fa eseguire la misura del processo e del dato al kernel che mediante chiamata alla libreria TPM, controlla i digest calcolati con quelli misurati all'avvio del sistema. Rispetto all'architettura nativa del TCG, IMA permette di estendere e realizzare un sistema automatico di controlli nel sistema operativo stesso. Un problema è come realizzare la memoria che possa memorizzare i log delle misure e le misure stesse essendo i componenti da misurare in un numero molto più elevato che in precedenza. La risposta è la struttura della stessa architettura. Infatti, IBM ha predisposto uno storage sicuro, misurato ad ogni richiesta di modifica che implementa l'estensione dello storage TPM e

mediante il quale è possibile accedere alle misure effettuate. Il procedimento di attestazione non cambia, infatti tutte le misure effettuate vengono "cumulate" all'interno dei PCR del TPM, come indica la specifica. Il passaggio ulteriore che avviene riguarda il messaggio di attestazione inviato al partner. All'interno del messaggio, infatti, sarà inserita la serie di singole misure ed il contenuto dei PCR. Il partner verificherà la corrispondenza di entrambe. La garanzia sul mittente è data dalla certificazione e firma digitale apposta, come di consueto, dallo stesso TPM. Inoltre, lato remoto, i singoli digest contenuti nel file SML delle misure, vengono confrontati con un database di hash conosciuti e validi, permettendo non solo di rilevare che il prodotto hash dei PCR sia valido, ma anche che l'integrità del sistema sia garantita.

2.4.5 Analisi dell'architettura

I meccanismi di attestazione indicati in precedenza permettono di definire strategie di protezione ma possiedono anche alcuni difetti sostanziali che riguardano sia l'architettura che il protocollo.
Nel modello TCG, ad esempio, se ne individuano almeno tre:

- Configurazioni dei sistemi clients troppo variegate;
- Possibilità di conoscere la configurazione esatta di una macchina;
- Problemi di concorrenza.

Il primo è un fattore importante.Infatti in un ambiente libero e flessibile come l'informatica imporrebbe una rigidità impossibile da implementare in generale. Infatti, la conoscenza degli hash di software introduce dei vincoli su tutto ciò che è lecito eseguire su di una macchina per poter essere definita affidabile. Ma l'esistenza di patch, di software personale o di strumenti poco conosciuti impone un vincolo molto forte alla diffusione di tale metodo, infatti la base dati contente gli hash crescerebbe in maniera esponenziale.
Legato a questo è il problema della concorrenza.Imponendo questo standard, i grandi produttori potrebbero imporre le loro distribuzioni ed il loro software, e per questo tale tecnologia è critica nella comunità informatica [40].
Per ultimo, un problema legato alla sicurezza. La possibile conoscenza della configurazione esatta della macchina nel momento in cui tale informazione sia accessibile da malintenzionati porterebbe grossi rischi.

Dal punto di vista architetturale, esistono ulteriori problemi aperti.
Ad esempio l'utilizzo della Privacy-CA nell'algoritmo introduce fattori non ottimi nella gestione delle transazioni tra attestatore ed attestante.

Il primo problema è che per ogni transazione la CA è inclusa nella comunicazione, quindi da un lato deve avere proprietà di disponibilità elevata e dall'altro deve essere sicura. Infatti il punto più debole dell'algoritmo risiede proprio nella sicurezza dei dati contenuti dalla CA. Per risolvere questo problema, IBM, HP ed Intel hanno sviluppato una variante dell'algoritmo detta *Direct Anonymous Attestation*(DAA) capace di ottenere le stesse garanzie mediante algoritmi di crittografia come l'*Ateniese et al. group signature scheme* ed il *Camenisch-Lysyanskaya anonymous credential system*[23].
IMA eredita molti di questi problemi, anche se ottimizza l'utilizzo dell'attestazione remota permettendo di estendere la trusted chain fino al livello applicativo.Un'altra difficoltà insita nell'architettura è la *one-time-measure*, cioè IMA misura il file caricato una sola volta finchè il binario non è modificato, ciò comporta la possibilità di corruzioni a run-time. A partire da questa considerazione si evidenzia come uno dei problemi più significativi di queste architetture è la sola attestazione della correttezza dei file binari ma non del comportamento del software.
I controlli di integrità, sono una parte consistente della sicurezza di un sistema e della sua affidabilità, ma il problema fondamentale che trascurano è relativo alla verifica della correttezza di un processo durante la sua esecuzione.

Capitolo 3

Xen ed il controllo d'integrità delle Macchine Virtuali

Indice

Negli ultimi anni stiamo assistendo alla evoluzione ed all'utilizzo sempre più diffuso delle tecnologie di virtualizzazione come metodologia di consolidamento dei server e dei datacenter. Possiamo considerare la virtualizzazione anche come un'opportunità per aumentare la sicurezza degli ambienti di esecuzione. Anche nell'informatica, quindi, va applicata una difesa basata su differenti livelli di protezione (security layers). Questa filosofia è chiamata,come abbiamo visto, *defense in depth*. In breve, questa strategia prevede l'uso di tecniche di sicurezza per mitigare i rischi che un eventuale livello venga compromesso o aggirato, e la virtualizzazione fornisce questa suddivisione per definizione.

Un software di virtualizzazione è uno strumento che implementa una astrazione delle risorse a livello hardware, interponendo un ulteriore livello software tra il livello hardware ed il livello del sistema operativo.

Il nuovo livello, Virtual Machine Monitor (VMM), astrae ogni componente della macchina fisica, quali il processore, la memoria e i dispositivi di I/O, creando degli ambienti di esecuzione software che emulano il comportamento dell'hardware della macchina. Il VMM, detto anche Hypervisor, deve creare e gestire l'esecuzione concorrente delle VM sulla stessa macchina reale, garantendo: (1) la compatibilità di tutte le applicazioni che possono essere eseguite sulla macchina reale, (2) l'isolamento tra le VM, per motivi di correttezza e sicurezza e (3) un buon grado di efficienza rispetto a quella del sistema reale.

Di conseguenza, tre sono i requisiti dell'architettura di un VMM:

a) isolamento tra i domini: l'esecuzione di un dominio non deve influire sulle prestazioni di un altro dominio;

b) possibilità di eseguire la maggior parte dei sistemi operativi;

c) l'overhead introdotto dalla virtualizzazione deve essere estremamente ridotto.

3.1 Virtualizzazione: Concetti Generali

Il concetto che sottende a tutte le tecniche di virtualizzazione è l'isolamento dello strato "fisico" attraverso una sorta di incapsulamento. La virtualizzazione crea una interfaccia esterna che nasconde ed incapsula le risorse sottostanti e permette l'accesso concorrente alle stesse risorse da parte di più istanze.

3.1.1 Virtual Machine Monitor

Il Virtual Machine Monitor, è un livello software di dimensioni ridotte che permette a più sistemi operativi di essere eseguiti concorrentemente sulla stessa macchina fisica. Ogni sistema operativo ha a disposizione una Macchina Virtuale(Virtual Machine), cioè un'ambiente che emula le risorse della macchina fisica, per poter essere eseguito. Si possono individuare due tipi di VMM presenti in letteratura:

- **Type-I VMM** o *Unhosted VMM*: gli hypervisor dialogano con l'hardware senza l'utilizzo di un sistema operativo host. VMM di questo tipo richiedono un supporto hardware potente e sono, di solito, utilizzati in ambienti client-server.

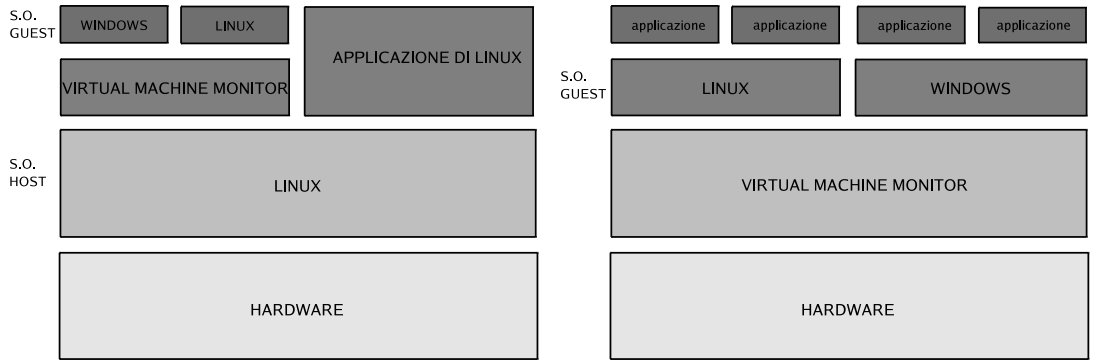


Figura 3.1: Type-I (a); Type-II (b)

- **Type-II VMM** o *Hosted VMM*: sono i più utilizzati su workstation e sono implementati al livello superiore del sistema operativo della macchina. In altre parole si tratta di veri e propri programmi che lavorano,ad esempio, in ambiente Windows.

L'architettura tipica dei Type-I e Type-II VMM è mostrata in Fig. 3.1(b) e 3.1(a).

Nel caso dei Type-I VMM, la performance effettiva di un sistema operativo eseguito su una macchina virtuale, detto GuestOS, è molto vicina a quella nativa, cioè a quella che si può ottenere senza utilizzare il sofware di virtualizzazione.
L'interfaccia virtuale è identica, o molto simile, a quella reale e questo permette di ridurre l'overhead dovuto al trasferimento delle richieste fra le due interfacce. Il VMM interviene solamente quando deve prendere controllo della macchina virtuale, ad esempio per mantenere l'isolamento tramite la gestione degli spazi d'indirizzamento del GuestOS e dei processi in esecuzione. Negli altri casi, la VM utilizza le risorse in modo diretto poiché le istruzioni non privilegiate possono essere eseguite direttamente sulla macchina reale.

Inoltre, la dimensione ridotta, in termini di numero di istruzioni, di un VMM rispetto ad un sistema operativo moderno, rende il VMM stesso più efficiente nell'esecuzione delle operazioni e permette di verificarne formalmente la correttezza.

3.1.2 Full virtualization e Paravirtualization

Si possono distinguere due modalità di virtualizzazione:

1. Full Virtualization (virtualizzazione completa): il sistema operativo ospite può essere eseguito all'interno di una VM creata dall'Hypervisor senza alcuna modifica al Kernel.

2. Para-virtualization (para-virtualizzazione): il sistema operativo ospite deve essere modificato per poter essere eseguito da una VM.

Nella **full virtualization**, il VMM esporta un'interfaccia virtuale del tutto identica all'hardware sottostante. In questo caso, il VMM è più complesso rispetto alla seconda modalità e deve adottare tecniche quali la *dynamic recompilation* [48] o la *binary translation* [69], per catturare ed eseguire le operazioni privilegiate richieste dal GuestOS. Nella **para-virtualization**, il VMM esporta un'interfaccia simile, ma non identica, all'hardware sottostante. Di conseguenza, è necessario modificare il codice del GuestOS per effettuare la comunicazione con l'Hypervisor, ma non è richiesta alcuna modifica al codice delle applicazioni eseguiti. Il vantaggio principale di questa modalità è che permette di ottenere performance molto elevate, generalmente superiori a quelle ottenibili dalla virtualizzazione completa. Di contro, il limite di questo approccio è il numero di sistemi operativi che possono essere eseguiti poiché si possono utilizzare solo gli OS di cui è possibile modificare il codice.
Considerati i limiti della virtualizzazione completa e della para-virtualizzazione, AMD ed Intel hanno sviluppato propri supporti hardware/firmware a questa tecnologia che sono denominati, rispettivamente, Intel Virtualization Technology (Intel-VT) [5] e AMD Pacifica Virtualization Technology (AMD-VT) [1]. Queste tecnologie introducono nuove istruzioni che permettono di non modificare il codice del GuestOS o eseguire dynamic/binary recompilation/translation. L'introduzione di nuove istruzioni nel set d'istruzioni del processore permette, di fatto, di eseguire un ampio numero di sistemi operativi sul VMM ottenendo elevate prestazioni. Se si adottano queste tecnologie, si fa riferimento alle macchine virtuali con il termine di Hardware Virtual Machine (HVM).

3.2 Xen

Xen è un Hypervisor di tipo I nato come progetto di ricerca all'Università di Cambridge [13]. Attualmente è giunto alla versione 3.3.0 ed è sviluppato con i contributi da parte di importanti aziende del settore informatico quali Intel, IBM, Hp e Novell. Nell'ambiente di esecuzione di Xen, le macchine virtuali sono chiamate *domini* ed il software su di esse eseguito si può comportare come se fosse eseguito direttamente sull'architettura fisica e non deve essere modificato per essere eseguito sulle VM. Inoltre Xen, come tutti i software di virtualizzazione, fornisce degli ambienti di esecuzione isolati per i domini, che garantiscono che le attività di un dominio non interferiscano con le attività di un altro.

3.2.1 Architettura

L'hypervisor implementa delle API che permettono di controllare i GuestOS che possono interagire direttamente o indirettamente con il livello hardware/firmware sottostante. In Xen, le

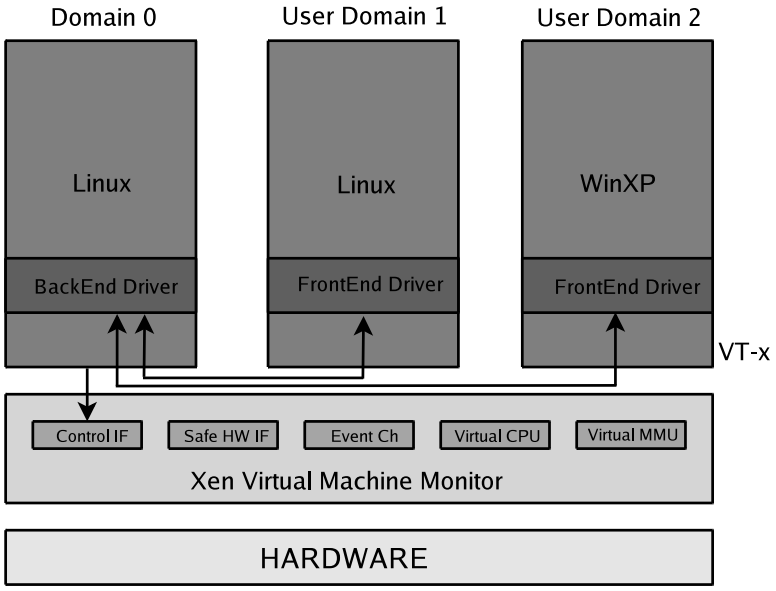


Figura 3.2: L'architettura di Xen 3.0

macchine virtuali, i domini, e possono essere di due tipi:

- *Domain-0* (Dom0) o Control Domain: rappresenta una Virtual Machine privilegiata che gestisce e controlla gli altri domini utilizzando le Management API e il Management Code.
- *Domain-U* (DomU) o XenVM: sono le virtual machine in cui vengono eseguiti i GuestOS.

Dom0 viene avviato automaticamente al boot della macchina ed è abilitato a gestire la schedulazione delle CPU virtuali assegnate ai domU, l'allocazione della memoria e l'accesso ai device fisici come, ad esempio, i dischi e le schede di rete. La Virtual Hardware API permette di gestire le risorse hardware e definisce le seguenti risorse virtuali:

- Virtual Network Interfaces (VIFs)
- Virtual Firewall e Routers (VFRs)
- Virtual Block Devices (VBDs)

Ad ogni risorsa è associata un'Access Control List (ACL) che specifica se un domU può accedere alla risorsa e quali sono le operazioni che può invocare. Tramite strumenti di controllo implementati a livello utente, Dom0 può invocare diverse operazioni, ad esempio creare, distruggere o riavviare i domini utente od i device di I/O, monitorare le macchine o la rete.

3.2.2 Hypercalls

L'interazione fra i domU e l'hypervisor avviene tramite le hypercalls, un meccanismo di comunicazione sincrono che permette di generare una eccezione gestita da Xen quando i domini tentano di eseguire operazioni privilegiate. L'hypercalls interface è definita nelle Virtual Hardware API e permette di modificare il livello di privilegio corrente, da Ring-1 a Ring-0, tramite un meccanismo simile a quello delle system calls. L'hypervisor interagisce con i domini in modo asincrono, emulando un meccanismo simile a quello delle interruzioni. In questo modo, Xen può rispondere ad una richiesta, segnalata da un domU, via hypercall o inviare notifiche asincrone di controllo alle xenVM come, ad esempio, l'arresto o la pausa.

3.2.3 Virtualizzazione della CPU

Nella *x*86, come in molte altre architetture, la protezione è basata sul concetto dei livelli di privilegio, detti anche *rings*. Le istruzioni che un modulo può eseguire dipendono dal ring in cui viene eseguito. In questo modo, gli anelli permettono di limitare l'esecuzione di alcune operazioni da parte, ad esempio, delle applicazioni. Nell'architettura *x*86 i rings sono 4, lo 0 è quello con più privilegi, il 3 quello con minor privilegi; solitamente l'OS viene eseguito nel ring 0 e le applicazioni sul 3. Xen richiede che gli OS vengano modificati per essere eseguiti sul ring 1. In questo modo tutte le operazioni privilegiate possono essere eseguite solo da Xen, eseguito al livello 0. Quindi, Xen para-virtualizza i sistemi operativi guest in modo tale che quando viene richiesta un'operazione privilegiata, il processore generi un'eccezione che trasferisce il controllo a Xen. I valori dei registri della CPU sono salvati in un *Virtual CPU-Context*(VCPUC) associato a ciascun dominio. Quando viene schedulato un domU, i valori correnti vengono salvati all'interno del VCPU-C che, dunque, viene utilizzato per ripristinare lo stato del processore nel momento in cui il VMM deve tornare a gestire il dominio.

3.2.4 Gestione della memoria

La gestione della memoria è cruciale per l'efficienza. Xen implementa due diversi tipi di spazio di indirizzamento: uno contiene gli indirizzi della machine-memory, l'altro quelli della pseudo-physical memory. La "machine memory" è la memoria fisica presente sulla macchina e comprende tutta la memoria associata ai domini, quella usata da Xen e quella non allocata. La "pseudo-physical memory" è un'astrazione che permette di offrire ai sistemi operativi uno spazio di memoria logico contiguo, anche se esso in realtà non lo è. È stata adottata questa astrazione perché molti sistemi operativi richiedono che le pagine fisiche siano contigue. Per implementare questa astrazione, si utilizza una tabella globale, che mappa gli indirizzi appartenenti alla

"machine memory" in quelli della "pseudo-physical memory", e una tabella in ogni dominio per implementare la mappatura inversa.
Dal punto di vista della memoria virtuale, Xen occupa i primi 64MB dello spazio di indirizzamento virtuale di ogni processo.Questa scelta è stata fatta per evitare una perdita elevata di prestazioni dovuta alla gestione hardware del *Translation Lookaside Buffer* (TLB) per architetture *x*86.

3.2.5 Virtualizzazione dei dispositivi di I/O

Xen implementa due tipi differenti di driver che cooperano per fornire l'astrazione dei dispositivi virtuali:

- il *frontend driver*, che viene eseguito nel dominio non privilegiato, quello guest;
- il *backend driver*, che viene eseguito sul dominio 0, oppure da un driver specifico che ha accesso diretto all'hardware.

Il "frontend driver" permette all'OS guest di accedere al dispositivo mediante un'interfaccia uguale a quella reale, cioè a quella definita dal driver originale. Il driver riceve le richieste dal sistema operativo e, visto che non ha accesso al dispositivo, invia una richiesta al "backend" driver. Il "backend" driver ha la responsabilità di soddisfare le richieste di I/O. Esso deve controllare che le richieste rispettino la politica di sicurezza, e trasmette le richieste corrette al dispositivo hardware. Quando il dispositivo ha terminato il lavoro di I/O, il "backend" driver segnala al "frontend" che i dati sono pronti e, successivamente, il "frontend" lo segnala al sistema operativo.

3.3 Controlli d'integrità in ambienti virtualizzati

In questo capitolo viene affrontato il problema del controllo d'integrità a partire da un'ambiente di esecuzione virtualizzato. A partire dal ruolo svolto da strumenti come Network IDS od Host IDS per l'analisi d'integrità di un host e dai problemi di sicurezza che essi non riescono a risolvere, discutiamo due metodi molto utilizzati negli ultimi tempi nel campo della verifica d'integrità e che si integrano bene con la virtualizzazione e le sue caratteristiche.

3.3.1 Introspezione delle macchine virtuali

L'introspezione è una tecnica di rilevazione delle intrusioni basata sulla valutazione di un insieme di condizioni logiche, espresse in funzione del valore di posizioni della memoria di un

sistema. Queste condizioni possono essere dedotte, in maniera automatica, da uno strumento che analizza il software in esecuzione oppure stabilite dall'amministratore in base a proprie conoscenze. In generale, sistemi che offrono queste tipologie di controllo implementati in maniera però diversa esistono sul mercato e sono ben consolidate da anni: *Host IDS* e *Network IDS*. Le loro funzionalità ed i contesti in cui possono essere adottati per eseguire controlli di consistenza sono già stati discussi in precedenza.
Molti degli attacchi possibili a sistemi di rilevamento delle intrusioni si basano sulla tecnica del consumo di risorse: l'attaccante identifica alcune operazioni nell'attività di rete dell'IDS che richiedono l'allocazione di risorse e le sfrutta fino a farle terminare. Altri tipi di attacco più dannosi perchè non direttamente individuabili sono l'*attacco diretto*, in cui i componenti dell'IDS sono manomessi in modo che non riconoscano o non registrino e le attività sospette e l'*evasione*, in cui gli attacchi sono eseguiti in modo da ingannare l'IDS che non li riconosce come tentativi di intrusione [47]. Per tenere conto di questi tipi di attacco, sono state formulate diverse contromisure che richiedono l'aumento della visibilità dell'IDS sul sistema monitorato. Il corrispondente grande svantaggio è che l'IDS deve essere più integrato nel sistema, e quindi è meno isolato e può perciò essere attaccato. La Virtual Machine Introspection (VMI) è una strategia che può permettere si soddisfare questi due requisiti apparentemente contrastanti, visibilità degli eventi e resistenza agli attacchi [37]. L'idea centrale è quella di utilizzare la virtualizzazione per spostare l'IDS fuori dall'host che deve controllare e installarlo all'interno del Virtual Machine Monitor o di un'altra macchina virtuale sullo stesso host.In questo modo è possibile sfruttare la possibilità offerta dal VMM di ispezionare in maniera diretta lo stato della macchina virtuale che esegue l'host che si vuole monitorare. Si possono così di ottenere sia i benefici propri di un Host IDS, la visibilità dell'host, che quelli di un Network IDS, la maggiore resistenza agli attacchi 3.3. Inoltre, dato che un IDS tradizionale si affida ai controlli di sicurezza offerti da un sistema operativo per garantire la propria protezione e l'isolamento rispetto alle applicazioni, l'IDS è suscettibile di manomissioni se un host viene attaccato, perché il sistema operativo è stato compromesso, tali controlli possono essere facilmente disabilitati [61]. Invece, un IDS realizzato tramite la Virtual Machine Introspection, in caso di compromissione del sistema operativo dell'host, può ugualmente osservare lo stato della macchina virtuale su cui è in esecuzione il sistema operativo compromesso ed effettuare eventualmente azioni di ripristino. Inoltre, nei casi in cui alcuni dei controlli di sicurezza adottati dal sistema operativo siano stati disabilitati, l'IDS con la tecnologia VMI può effettuare tali controlli di protezione per conto del sistema operativo. Ad esempio, esso può controllare che le interfacce di rete dell'host non operino in modalità promiscua).

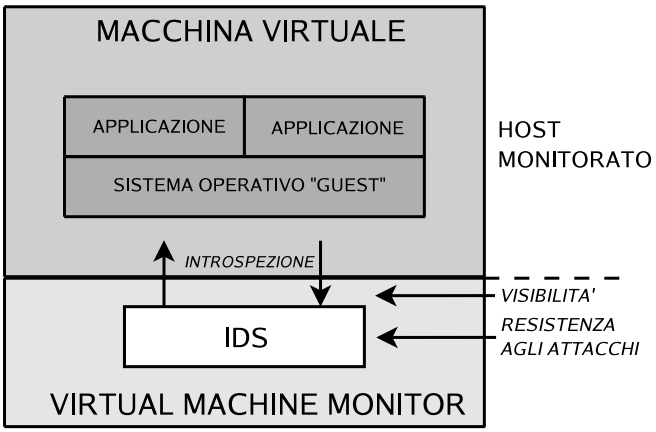


Figura 3.3: Introspezione

3.3.1.1 Psyco Virt

Il progetto *Psyco-Virt* [18] integra la tecnologia della virtualizzazione con le normali metodologie di rilevamento delle intrusioni su sistemi distribuiti, per ottenere un'architettura in grado di rilevare e prevenire le intrusioni monitorando i sistemi a più livelli e che sfrutta così la passibilità offerta dal VMM di ispezionare in maniera diretta lo stato dell'host da monitorare. In questo modo, l'IDS ha la stessa visibilità che avrebbe se fosse installato sull'host, come avviene per gli Host-IDS, ma è però più resistente agli attacchi, poiché non fa parte dell'host, ma si trova al livello inferiore, nel VMM. Ovviamente, in questa architettura, l'ipotesi di una maggior sicurezza è giustificata se si assume che un VMM sia più difficile da attaccare e compromettere rispetto ad un host normale.
L'architettura del sistema è distribuita e prevede un insieme di macchine virtuali allocate su un insieme di nodi reali. Ogni nodo reale esegue un VMM, che applica l'introspezione sulle VM "locali". Per semplicità di implementazione l'IDS viene eseguito dal dominio 0 di Xen che può essere considerato come un'estensione del VMM. Ogni VM monitorata esegue un insieme di *agent* (o agenti), coordinati da un manager, che implementano le funzionalità di un Network-IDS ed ha accesso all'interfaccia di controllo esportata dal Virtual Machine Monitor. Al manager sono affidati due compiti:

- esaminare lo stato delle macchine virtuali tramite introspezione;
- ricevere ed analizzare tutti i messaggi relativi a tentativi di intrusione dalle componenti del sistema installate sulle macchine virtuali (agenti).

Quando il manager rileva delle intrusioni su un host monitorato, esegue delle azioni in risposta a tali eventi, modificando lo stato di esecuzione del dominio attaccato. Alcune delle

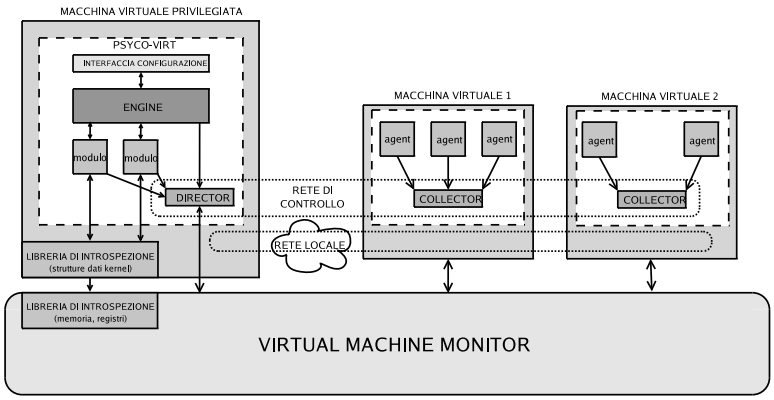


Figura 3.4: L'architettura di PsycoVirt

azioni possibili sono:

- bloccare l'esecuzione del domino attaccato e salvare il suo stato su un file;
- terminare i processi eseguiti dall'attaccante;
- chiudere la connessione della VM con la rete locale;
- disconnettere l'attaccante connesso con la VM;
- inviare un segnale alla console di amministrazione.

Dal dominio 0, il manager può esaminare lo stato di ogni macchina virtuale, ed in particolare le parti critiche per la sicurezza dei sistemi. Questo esame è semplificato dalla libreria di introspezione di "alto livello", che permette di interpretare ed utilizzare i dati di "basso livello", cioè quelli forniti attraverso la libreria di controllo di Xen. Questa libreria permette di accedere al contenuto della memoria, dei blocchi del disco e dei registri del processore. I controlli in introspezione devono verificare la consistenza delle parti critiche del nucleo (kernel) del sistema e garantire la corretta esecuzione degli agent, i quali, a loro volta, applicano un insieme di controlli sui processi a livello utente.

In questo scenario, gli agent si comportano come dei normali Host-IDS che rilevano intrusioni e attacchi sul sistema e informano il loro coordinatore, il manager, che deciderà cosa fare.

Le comunicazioni tra manager e agent sfruttano su una rete "virtuale", creata dal dominio 0, tramite OpenVPN [54], e detta rete di controllo. Il manager, oltre a ricevere le comunicazioni dai propri agent, può trasmettere loro delle richieste, per ottenere altre informazioni, nel caso si presentino situazioni che non riesce a valutare immediatamente. Inoltre, i manager possono comunicare per scambiare informazioni di controllo, per combattere meglio gli attacchi

distribuiti. La rete di controllo può essere utilizzata solo dalle componenti del sistema Psyco-Virt, il manager e gli agenti, e non è raggiungibile da nessun host. Un opportuno controllo anti-spoofing è stato introdotto per verificare che gli indirizzi sorgenti dei pacchetti, inviati sulla rete, siano solo quelli autorizzati.
In sintesi, si può affermare che la sicurezza del sistema è garantita da un approccio a due livelli:

- verifica, tramite introspezione, dell'integrità delle parti critiche del sistema eseguito sulle VM monitorate;
- estensione del singolo sistema operativo con un insieme di funzioni di sicurezza, tra cui gli agenti, che eseguono una serie di controlli sull'integrità di altre parti del sistema e delle componenti a livello utente, come processi e file.

Esistono alcuni sistemi operativi che implementano in modo nativo il secondo step, ad esempio SELinux [9]. Il vantaggio di Psyco-Virt è che utilizza le normali tecniche di rilevamento delle intrusioni sulle macchine virtuali, sulle quali può essere applicata la tecnica della VMI, che permette di effettuare un insieme di controlli non eludibili dagli attaccanti, poiché si assume che l'introspezione faccia parte del Trusted Computing Base(TCB). Sotto queste ipotesi l'approccio considerato riesce a garantire l'integrità di tutte le componenti critiche del sistema, effettuando i controlli a più livelli a partire dal VMM. L'introspezione verifica l'integrità delle parti del sistema che sono cruciali per garantire la corretta esecuzione delle funzioni di sicurezza (tra cui gli agenti). Grazie a questi controlli, il TCB viene esteso e comprende anche le componenti eseguite sulle VM monitorate e, di conseguenza, lo spazio di azione degli attaccanti si riduce sensibilmente.

3.3.1.2 Libreria Xen-VMI

Per rendere più efficace il sistema Psyco-Virt introducendo nuovi controlli ed estendendo quelli già presenti presso il Dipartimento di Informatica di Pisa è stata implementata una libreria di introspezione ad alto livello, *Xen-VMI*, il cui scopo principale è l'interpretazione dei dati di "basso livello" in modo più semplice e strutturato rispetto alla precedente libreria Psyco-Virt. La libreria Xen-VMI, comprende un insieme di funzioni che implementano dei controlli su un insieme di dati sensibili del kernel degli OS. I controlli di introspezione, che utilizzavano la libreria precedente, sono stati integrati nella libreria Xen-VMI. I controlli possono essere partizionati in due categorie:

- Quelli che analizzano i valori dei dati ottenuti mediante introspezione e controllano che siano uguali a quelli che indicano uno stato corretto per il sistema;

- Quelli che confrontano i dati ottenuti mediante introspezione con quelli forniti dal sistema monitorato, per scoprire eventuali differenze tra i due dati. I dati dal sistema sono ottenuti invocando una funzione di livello più alto rispetto all'introspezione, tipicamente di sistema operativo.

Tutti i controlli implementati, una volta attivati, sono eseguiti ad intervalli di tempo regolare e possono essere interrotti e riavviati in qualsiasi momento, senza dover riattivare il sistema del dominio privilegiato o quelli dei domini guest. I controlli sono stati implementati in due tipologie, che garantiscono lo stesso livello di sicurezza:

- Controlli eseguiti sui dati ottenuti tramite introspezione e raggruppati nella libreria di alto livello Xen-VMI ;

- Controlli eseguiti sui dati forniti del sistema, a livello utente o a livello kernel, dagli agenti in esecuzione sulle VM monitorate.

Nel secondo caso, bisogna verificare la corretta esecuzione dell'agent: ciò può avvenire in uno dei seguenti modi:

- Controllando che l'agent non sia manomesso, ad esempio controllando le pagine di memoria relative alla text area del processo associato;

- Controllando che l'output prodotto dall'agent sia uguale all'output prodotto da un controllo dello stesso tipo ma implementato come una funzione di introspezione.

Il *Trusted Computing Base* del sistema include il processo di introspezione, che verifica: (1) l'integrità delle parti critiche del nucleo (kernel) dei sistemi monitorati e (2) la corretta esecuzione degli agent, eseguiti sui sistemi monitorati. Per questo motivo, anche gli agent fanno parte del *TCB*. In questo modo si sfruttano pienamente, e nel modo più efficiente, i vantaggi permessi dalla virtualizzazione e quelli offerti dalle tecniche per la rilevazione di intrusioni. Infatti, la soluzione adottata minimizza la complessità del processo di introspezione e dell'apposita libreria che fornisce la necessaria visione di alto livello per elaborare i dati, poiché essa deve "ricostruire" solo le strutture dati del kernel. Gli agenti e le altre funzioni di sicurezza, installate sulle VM monitorate, verificano la consistenza di altre parti degli OS e dei processi a livello utente rendendo il sistema molto flessibile all'inserimento di controlli a livello utente variegati e modulari. La libreria è in grado di implementare un vasto insieme di controlli:

- modalità promiscua dei device di rete;

- integrità dei processi;

- integrità dei moduli del kernel;
- presenza di redirezioni nella Syscall Table del kernel;
- consistenza della tabella delle interruzioni;
- integrità della text area del Kernel;
- consistenza sella lista processi attivi;
- consistenza della lista file aperti;

Le relative funzioni sono state implementate con il linguaggio C e sono state compilate in modo da includere gli header del kernel relativi agli OS monitorati. Gli header del kernel sono lo strumento che consente di eliminare la differenza tra quella che è la visione del sistema dall'interno e quella ottenuta attraverso l'introspezione. L'implementazione di ogni funzione è stata strutturata in due fasi:

- accesso mediante introspezione alle specifiche informazioni;
- controllo dei dati ottenuti nella fase precedente.

Gli header del kernel sono utilizzati inizialmente per individuare la descrizione di tutti i tipi di strutture dati che memorizzano le informazioni utilizzate dal kernel. In generale, le strutture sono molto complesse e variano da una versione all'altra del kernel, quindi è fondamentale interfacciarsi con gli header del kernel. I controlli locali, implementati come moduli del kernel , sono installati ed eseguiti nel kernel del dominio monitorato ed hanno accesso diretto alle strutture dati del kernel, dichiarate dai relativi header. Il manager, invece, è un processo eseguito in user space e non può accedere direttamente alle strutture dati del kernel delle VM che, in generale, è diverso da quello del dominio 0 che esegue il manager. La soluzione utilizzabile è quella che prevede di copiare la zona di memoria che contiene i dati che interessano nello spazio di indirizzamento del processo che implementa il manager. Poichè il manager è stato compilato in modo da includere gli header del kernel delle VM monitorate, esso può ricostruire le strutture dati che descrivono il contenuto della memoria a livello fisico, evitando di doverle ridefinire "manualmente". Xen-VMI utilizza le funzioni di interfaccia di XenAccess [14], un insieme di funzioni che implementano una interfaccia verso la libreria di controllo di Xen, che ricostruisce le strutture dati del kernel utilizzando le dichiarazioni contenute negli header del kernel. Inoltre, essa utilizza le funzioni della libreria di controllo di Xen per modificare lo stato di esecuzione delle VM, ad esempio, per bloccare l'esecuzione di un dominio.

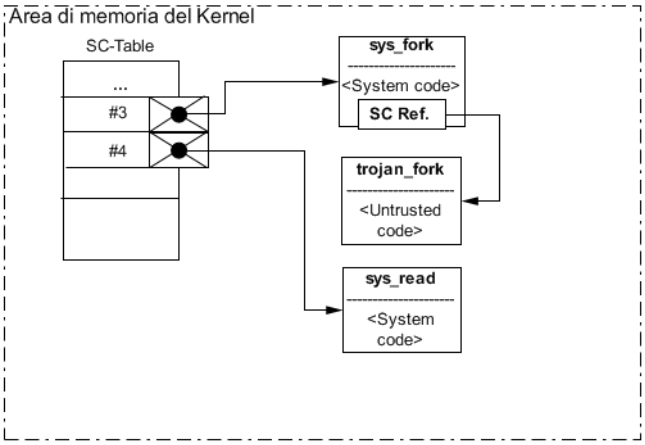


Figura 3.5: Stealth di una system call

Esempio di controllo: Kernel Text Area La text area del kernel è quella compresa tra l'indirizzo *_text* e l'indirizzo *_etext* del kernel. Questa sezione del nucleo contiene il codice statico delle funzioni del kernel, in particolare:

- il codice delle system calls;
- degli handlers che gestiscono le eccezioni e le interruzioni di I/O;

Il controllo implementato ha l'obiettivo di rilevare il cosiddetto stealth delle SC o degli handler, che consente agli attaccanti di eseguire codice maligno nello spazio kernel, modificando il codice eseguibile delle procedure (vedi Fig. 3.5).

Anche in questo caso, la consistenza è controllata calcolando l'hash delle informazioni, in particolare, dell'intera text area del kernel. Qualsiasi tentativo di "stealth" nell'area è rilevato e classificato come un'intrusione. La funzione di introspezione è `get_kernel_text()`.

3.3.2 Virtual TPM

Per estendere le funzionalità del TPM in ambienti di esecuzione virtualizzato, IBM ha proposto e realizzato una libreria che, sfruttando le capacità di generazioni di chiavi asimmetriche del TPM potesse implementare il componente su più macchine virtuali, il vTPM.
Questa libreria "emula" il comportamento del TPM Hardware(così come l'hardware è emulato dal VMM verso le VMs) permettendo la condivisione tra i vari ambienti vrituali dell'unico chip dell'host.
La figura 3.6[21] mostra l'architettura della libreria in riferimento all'hypervisor Xen.

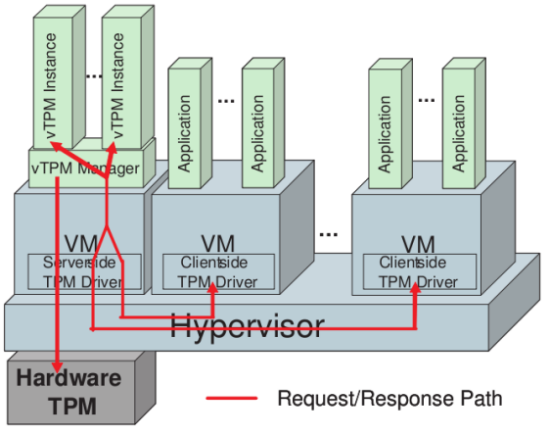


Figura 3.6: Architettura vTPM

Ogni istanza del vTPM implementa le specifiche del TPM v1.2 . Ad ogni macchina virtuale che richiede le funzionalità del TPM, è assegnata una istanza vTPM. Il manager, presente sul VMM o su di una macchina virtuale appositamente instanziata, è deputato all'esecuzione delle funzionalità di creazione, distruzione, migrazione delle istanze legate alle macchine assegnatarie, oltre che, naturalmente, all'instradamento dei comandi e dei dati tra le varie istanze ed il device fisico. Le macchine virtuali comunicano con il vTPM mediante il cosiddetto modello "*split device-driver*", dove il driver client-side è eseguito nella macchina virtuale, il driver server-side sul VMM o sulla macchina virtuale che ospita il vTPM manager. Ad ogni macchina virtuale che comunica con il vTPM, il manager assegna un identificatore di 4 byte che viene utilizzato durante le comunicazioni per instradare correttamente le richieste e le risposte tra i vari componenti.

Nei moderni hypervisor, la possibilità di sospensione e migrazione delle macchine virtuali permette un'elevata "mobilità" di questi sistemi, e, di conseguenza, l'implementazione del vTPM ha dovuto tenere conto anche di queste proprietà.
Il core del vTPM è la macchina manager, detta *Root vTPM Instance*, che si occupa di preservare le misure effettuate dall'istanza vTPM in caso di sospensione e migrazione della VM, e permette una grande flessibilità e buone performance per la generazione delle chiavi asimmetriche da associare alle diverse istanze. La capacità principale di questa entità è la gestione sia delle coppie di chiavi asimmetriche e sia dello stato delle VM quando queste vengono migrate verso un'altro host con un'altro manager. Essa fornisce un ottimo livello di affidabilità e sicurezza mediante la cifratura dello stato della macchina. Inoltre, per garantire una completa conformità alle specifiche TPM v1.2, ad ogni istanza vTPM è associata una gerarchia indipendente di chiavi crittografiche, a partire dalla *Endorsement Key*, che abilita completamente l'interfaccia

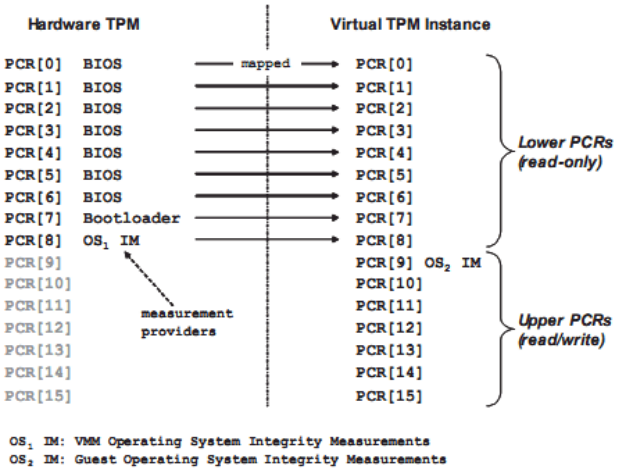


Figura 3.7: Mapping dei registri del vTPM

dell'istanza vTPM all'utilizzo dei comandi forniti dal TPM fisico. Le chiavi di ogni istanza sono memorizzate in memoria condivisa, e la loro integrità è garantita dalla cifratura mediante le chiavi crittografiche del chip TPM fisico.

Collegamento del vTPM con la TCB dell'host Perchè possa svolgere funzionalità equivalenti a quelle del TPM, l'ambiente in cui ogni vTPM è istanziato deve poter fornire un insieme di controlli d'integrità e memorizzare le misure dei componenti della macchina virtuale nello spazio di indirizzamento del processo che esegue il vTPM. Tuttavia, è necessario avere la possibilità, così come in un host non virtualizzato e con un TPM hardware, di poter fornire le credenziali di affidabilità ad un partner remoto in un ambiente che comprende però più componenti rispetto alla sola macchina virtuale. Infatti per poter attestare il proprio stato mediante un protocollo di attestazione remota, la macchina virtuale deve poter conoscere le misure effettuate "al di fuori" di essa, ad esempio dell'hypervisor o del processo di boot del sistema. Per garantire ciò, l'architettura del vTPM, unisce la visione dell'ambiente locale alla VM a quella dell'ambiente globale dell'host, mediante i registri PCR virtuali del vTPM.
Come mostrato in figura 3.7 [21], l'insieme dei registri PCR di indice $[0-8]$ del vTPM sono riservati e di sola lettura, scritti dal manager all'avvio dell'istanza del vTPM. Essi conservano lo stato d'integrità dell'hardware, BIOS, del boot process dell'hypervisor e dell'hypervisor stesso, del vTPM Manager.

L'insieme di registri con indice maggiore di 8 vengono utilizzati, secondo le specifiche TCG, come normali registri PCR. In questa maniera, l'eventuale partner remoto ha la possibilità di verificare un insieme completo di misure.

Infine, è da notare che, per gestire il caso in cui il partner non riponga fiducia sul certificato fornito dal vTPM, che è diverso da quello del TPM, la libreria vTPM implementa una speciale procedura che permette di risolvere questo problema abilitando il vTPM all'invio del certificato originale.

Capitolo 4

Architettura VIMS

Indice

Questo capitolo descrive il progetto e la realizzazione di un framework per il calcolo dell'integrità di un host e la sua attestazione remota, chiamato *Virtual machine Integrity Measurement System*(VIMS). Il framework utilizza tecniche di *introspezione*, per la verifica semantica del software eseguito su una macchina virtuale, e di verifica statiche dell'integrità dei livelli sottostanti. Completa il framework un protocollo di comunicazione sicura per la dimostrazione dell'affidabilità del sistema ad un partner remoto.

4.1 Obiettivi

Come abbiamo visto nel capitolo 1, uno dei problemi principali da affrontare in ambito SCADA è la possibilità di infezione di una delle macchine interne alla rete di controllo o di gestione che possa effettuare operazioni critiche sugli impianti industriali. In particolare,

partendo dal presupposto che un sistema SCADA possa essere corrotto da accessi *esterni*, il principale vettore di minaccia sono le connessioni con la rete esterna, come le connessioni VPN o per il controllo da remoto di dispositivi e dati.
Questa tesi ha quindi elaborato il concetto di analisi d'integrità di un sistema client che vuole connettersi ad un sistema server, che fornisce un insieme di servizi, dove i servizi sono considerati in senso lato. Lo scopo dell'architettura definita è quello di fornire uno strumento per la misura dell'integrità di un sistema remoto client in maniera tale che l'host server possa essere sicuro del proprio stato e di quello del software eseguito. Nelle diverse architetture per la verifica dell'integrità presenti in letteratura, si nota, in particolare, l'utilizzo esclusivo di soluzioni basate su un componente TPM, legando l'affidabilità del sistema al solo concetto di root of trust hardware. Considerando alcuni dei problemi che tali soluzioni non risolvono completamente, come la rilevazione di software corrotto a run-time mediante controlli non solo al caricamento ma anche durante l'esecuzione dei processi, si è definito un modello che potesse individuare i controlli necessari e garantire l'effettiva implementabilità dell'architettura su un prototipo reale. Per poter avere la necessaria visibilità ad un livello che garantisce il necessario isolamento tra chi deve certificare e chi certifica, si è utilizzata la virtualizzazione dei componenti sfruttando le proprietà di isolamento offerte da tale tecnologia.
Per poter comunque garantire un buon livello di affidabilità delle misure considerando anche i livelli inferiori quello di esecuzione delle VM, si è prevista l'estensione della propria TCB mediante l'utilizzo del TPM. L'integrità del client è quindi misurata e memorizzata in due fasi: (a) durante la fase di bootstrap del sistema[70] fino al caricamento del VMM mediante l'uso del TPM, (b) al caricamento del sistema operativo sulle macchine virtuali e dei singoli processi mediante i meccanismi implementati da VIMS.
Un'altro punto critico intorno al quale è stata definita e realizzata l'architettura è stato la possibilità di definire una serie di politiche e controlli che potessero essere dipendenti dalle verifiche a run-time effettuate, quindi dinamiche. Si vuole così permettere il monitoraggio continuo ed effettivo del sistema. Alle azioni che il client compie, o richiede al lato server, corrisponde così la possibilità di reazione immediata in caso di rilevamento di incorretezze, non solo alla richiesta d'accesso ma anche durante la sessione autenticata.

4.2 Modello Formale

Questa sezione descrive un possibile modello teorico di *trust* su cui basare VIMS, ed utilizzato per provare, successivamente, la reale sicurezza offerta dall'architettura proposta.
Il sistema che descriviamo ha l'obiettivo di determinare quali *chain of trust* possano esistere in un sistema in base alle diverse relazioni di *trust* tra i soggetti, detti *componenti*.

Definizione 1 (Componente) *Un componente è un elemento di un sistema che:*

a) definisce delle operazioni che possono essere invocate da altri componenti;

b) può applicare dei controlli od essere controllato da un'altro componente.

Il livello di granularità dei componenti a cui è possibile applicare il modello non è volutamente specificata per permetterne l'applicazione al livello di dettaglio di interesse, ad esempio al livello di host, di macchine virtuali o di processi. Il modello definisce un insieme parzialmente ordinato di *test* e di *politiche*, dove una politica è definita come coppia formata da un **insieme di test** ed una **frequenza** con cui applicare tali test.

Definizione 2 (Test) *Definiamo un test, T, come una procedura di verifica di una specifica proprietà di un componente c. Dati due test T_1 e T_2, definiamo un ordinamento parziale $\sqsubseteq_t$, dove $T_1 \sqsubseteq_t T_2$ se ogni componente che supera T_2 supera sempre T_1.*

Possiamo considerare un test T come una funzione che partiziona il dominio di applicazione in 3 sottoinsiemi: i valori che superano il test, T_S, quelli che non lo superano, T_F, e quelli a cui non interessa applicare il test, T_D. Quindi $T_1 \sqsubseteq_t T_2$ se $T_{S2} \sqsubseteq_t T_{S1}$ e $T_{F1} \sqsubseteq_t T_{F2}$.

Definizione 3 (Insieme di Test) *Possiamo estendere l'ordinamento parziale $\sqsubseteq_t$ ad un insieme di test. Dati due sottoinsiemi S_1 ed S_2 di T, si ha:*

$$S_1 \sqsubseteq_s S_2 \Leftrightarrow \forall t_1 \in S_1 \; \exists t_2 \in S_2 | t_1 \sqsubseteq_t t_2$$

Intuitivamente, S_2 è più severo di S_1 perchè, per ogni test in S_1, ne esiste uno più severo in S_2. Consideriamo ad esempio l'insieme S_1 come un insieme di test dove il file viene decomposto in frammenti, ogni test considera un frammento del file e verifica che il frammento non sia stato modificato, applicando ad esempio, una funzione di hash. Consideriamo ora un insieme S_2 che è definito in modo simile, ma dove il numero di frammenti è strettamente minore e dove inoltre ogni frammento di S_1 appartiene ad un unico frammento di S_2. In questo caso, per ogni test di S_1 ne esiste uno più severo di S_2.

Definizione 4 (Politica) *Una politica p è rappresentata da una coppia $< S, f >$, dove S è un insieme di test da applicare ed f è la frequenza con cui tali test dovranno essere applicati. Definiamo $p_1 =< S_1, f_1 >, p_2 =< S_2, f_2 >$, allora si ha che:*

$$p_1 \sqsubseteq_p p_2 \Leftrightarrow S_1 \sqsubseteq_s S_2 \wedge f_1 \leq f_2$$

La relazione $\sqsubseteq_p$ stabilisce un ordinamento parziale tra politiche, dove $p_1 \sqsubseteq_p p_2$ se p_1 è meno vincolante di p_2. Questa definizione afferma che per garantire un livello di sicurezza più elevato, la politica deve applicare un insieme di controlli più severo e applicarli con una frequenza almeno uguale a quella meno severa.
Considerando come esempio gli insiemi di controlli S_1 ed S_2 definiti in precedenza, e definendo due frequenze f_1 ed f_2 da associare ad essi tali per cui $p_1 =< S_1, f_1 >$ e $p_2 =< S_2, f_2 >$, allora $p_1 \sqsubseteq_p p_2 \Leftrightarrow f_1 \leq f_2$. Se, invece, $f_1 > f_2$, le due politiche non sono confrontabili.

Per ogni componente c del modello sono definiti diversi insiemi, i cui elementi sono di tipo *componente* oppure di tipo *politica*. Inoltre, un componente può richiedere l'esecuzione di operazioni op o servire richieste provenienti interagendo con alcuni componenti. Gli insiemi con elementi di tipo *politica* sono definiti nel modello come *statici*, gli insiemi di tipo *componente* sono invece *dinamici*. Per ogni componente c definiamo i seguenti insiemi di componenti:

a) $trusted(c)$, un insieme di componenti che c ritiene affidabili;

b) $closeapply(c)$, insieme di componenti che possono chiedere a c di applicare una politica, ad es. dei controlli, su altri componenti;

c) $closeinvoke(c, op)$, insieme di componenti che possono invocare l'operazione op di c.

d) $closetrust(c)$, insieme di componenti che possono applicare dei test al posto di c, e per questo *affidabili*.

Un vincolo importante sugli insiemi precedenti è che, per ogni componente, $trusted(c) \cup closetrust(c) \neq 0$.
Se un componente non soddisfa questo vincolo può essere trascurato perchè non contribuisce alle *chain of trust* successive. Gli insiemi di politiche che c definisce sono :

a) $trustedpolicy(c)$: la politica che c applica su un componente per poterlo includere nell'insieme $closetrust(c)$;

b) $test(c, c_i)$, l'insieme delle politiche che c può applicare al componente c_i. Se non può applicare nessuna politica, è restituito l'insieme vuoto;

c) $assurance(c)$: la politica che c deve applicare ad un componente c_1, che non appartiene a $closeapply(c)$, perchè c si fidi di esso e gli deleghi i controlli.

d) $policy(c, op_i)$: la politica che definisce i controlli che c deve applicare ad un componente c_j perchè possa invocare l'operazione op_i.

La definizione di $test(c, c_i)$ dipende dal livello di astrazione considerato e dalle relazioni di vicinanza tra componenti. Ad esempio, un componente potrà testarne un altro solo se sono allocati allo stesso nodo di una rete.

Inoltre, definiamo **applied(x, y, p, sp)**, una funzione che indica l'applicazione con successo sul componente *y*, da parte del componente *x*, di una politica *mp*, scelta tra quelle *meno vincolanti* dell'insieme *sp* che sono maggiori o uguali di *p*. In altri termini, se $p_a = \{a \mid a \in sp, p \sqsubseteq_p a\}$ allora *mp* è uno dei minimi di p_a. A sua volta, questo sottoinsieme delle politiche è tale che:

$$\neg\exists b \in p_a \mid b \sqsubseteq_p mp; \tag{4.1}$$

Possono esistere più politiche che soddisfano i vincoli di *mp*, poichè l'ordinamento tra politiche è parziale.

Per ogni componente *c*, è possibile calcolare l'insieme dei componenti per esso *fidati* (trusted). Questo insieme è calcolato come il *punto fisso* di un sistema di equazioni, presentate di seguito, che operano sull'insieme dei componenti di cui *c* si fida e su quello dei componenti che *c* può invocare per applicare delle politiche.
Nel seguito supponiamo che un componente *x* possa comunicare in modo sicuro ed affidabile ad un altro componente, *c*, il risultato di un test su un ulteriore componente *y*, dove $a \neq b, a \neq bec \neq a$.

Le equazioni che definiscono le possibili estensioni della *chain of trust* tra i componenti sono le seguenti:

a) *y* è inserito in $closetrust(c)$ se *c* può richiedere al componente $x \in closetrust(c)$ di applicare la politica $trustedpolicy(c)$ per controllare se y è affidabile. *c* può richiedere l'operazione perchè appartiene all'insieme $closeapply(x)$:

$$x \in closetrust(c) \wedge c \in closeapply(x) \wedge \ (\{trustedpolicy(c)\} \sqsubseteq_P test(x,y))$$
$$\wedge\ applied(x, y, trustedpolicy(c), test(x,y)) \Rightarrow y \in closetrust(c)$$

b) se *c* non appartiene a $closeapply(x)$, allora deve esistere un componente *k*, con i requisiti che gli permettono di invocare per conto di *x* la politica $assurance(x,y)$ su *c*. In questo modo, *k* dimostra ad *x* che *c* è in uno stato corretto e che quindi si può fidare e permettere a *c* di invocare uno dei test che *X* può applicare:

$$x \in closetrust(c) \wedge\ c \notin closeapply(x) \wedge\ \{assurance(x) \sqsubseteq_p test(x,c)\}$$
$$\wedge\ [\exists k : k \in closetrust(x) \wedge x \in closeapply(k) \wedge\ applied(k, c, assurance(x), test(x,c))]$$

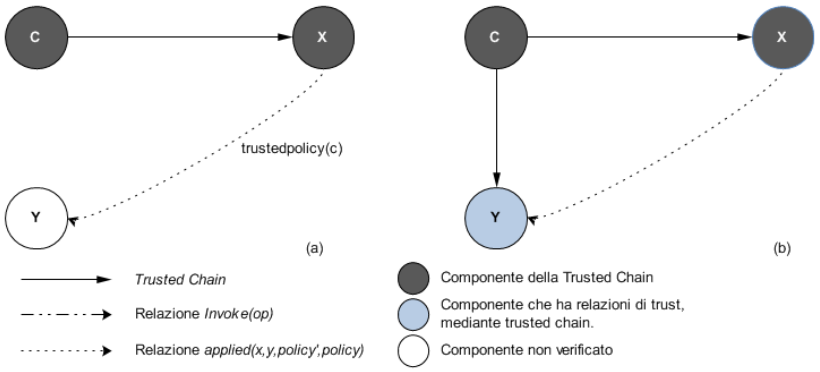


Figura 4.1: Regola a).

$\Rightarrow c \in closeapply(x)$

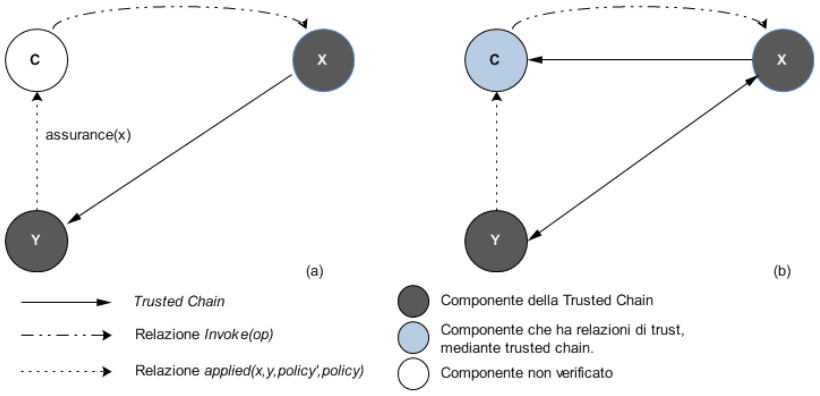


Figura 4.2: Regola b).

c) c invoca x per sapere se y è affidabile e può invocare su c l'operazione op_i. c chiede ad x di applicare su y una politica relativa all'operazione op_i:

$$x \in (closetrust(c)) \wedge\ c \in closeapply(x) \wedge\ \{p(c,op) \sqsubseteq_p test(x,y)\}$$
$$\wedge\ applied(x,y,p(c,op),test(x,y)) \Rightarrow y \in closeinvoke(c,op)$$

d) y deve potersi certificare per poter invocare op_i a c. y ha i requisiti per poter chiedere ad x, di cui c si fida, l'applicazione di una politica, $p(c,op_i)$, tale che y possa certificare il proprio stato a c. In questo modo, si estende l'insieme $closeinvoke(c,op)$;

$$x \in (closetrust(c)) \wedge\ y \in closeapply(x) \wedge\ \{p(c,op)) \sqsubseteq_p test(x,y)\}$$

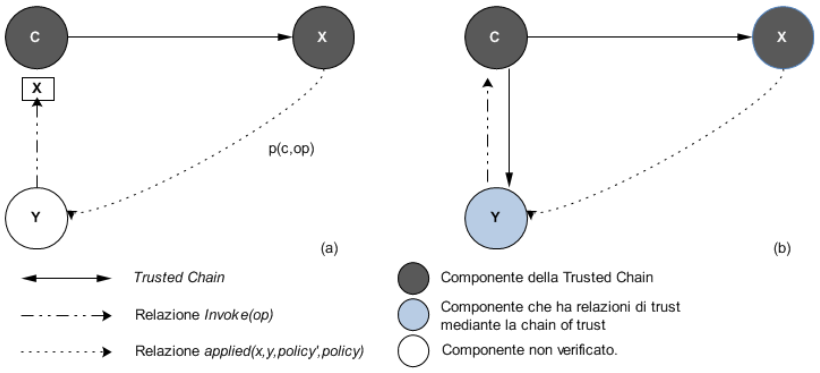


Figura 4.3: Regola c).

$$\wedge\, applied(x,y,p(c,op),test(x,y)) \Rightarrow y \in closeinvoke(c,op)$$

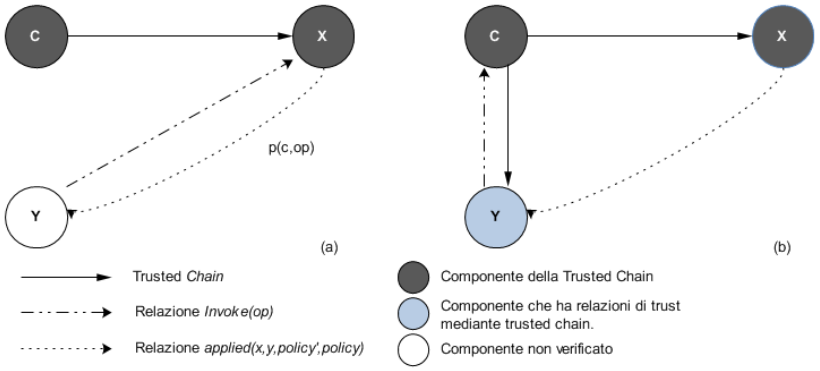


Figura 4.4: Regola d).

e) Il componente y, ha i requisiti per richiedere ad x, componente di cui c si fida, di eseguire dei test di tipo $trustedpolicy(c)$ su esso, perchè y possa essere ritenuto affidabile, successivamente, da c.

$$x \in (closetrust(c)) \wedge\ c \in closeapply(x) \wedge\ \{trustedpolicy(c) \sqsubseteq_p test(x,y)\}$$
$$\wedge\, applied(x,y,trustedpolicy(c),test(x,y)) \Rightarrow y \in closetrust(c)$$

f) y richiede ad x l'applicazione di una politica, $assurance(c)$, che possa certificarlo presso c; in tal maniera y può richiedere a c, successivamente, di effettuare dei controlli su un componente terzo, k. I controlli relativi all'invocazione su di c, $p(c,op)$, devono essere

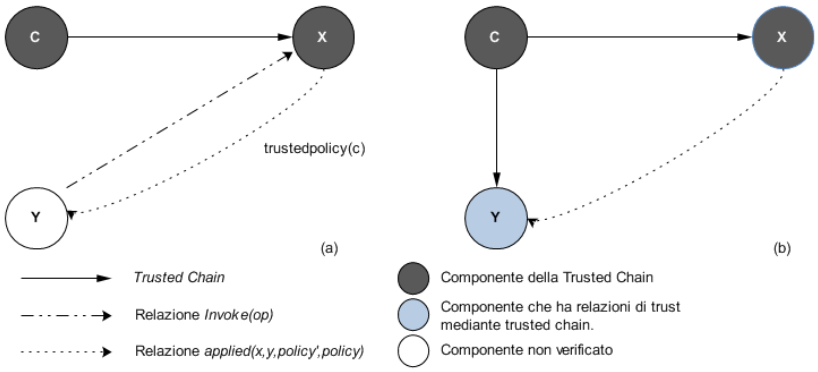


Figura 4.5: Regola e).

presenti nell'insieme $test(c,k)$ di c, cioè c deve poter eseguire tali controlli.

$$x \in (closetrust(c)) \wedge\ y \in closeapply(x) \wedge\ [\exists p(c,op) : p(c,op) \sqsubseteq_p test(c,k)]$$
$$\wedge \{assurance(c))\} \sqsubseteq_p test(x,y)$$
$$\wedge applied(x,y,assurance(c),test(x,y)) \Rightarrow y \in closeapply(c)$$

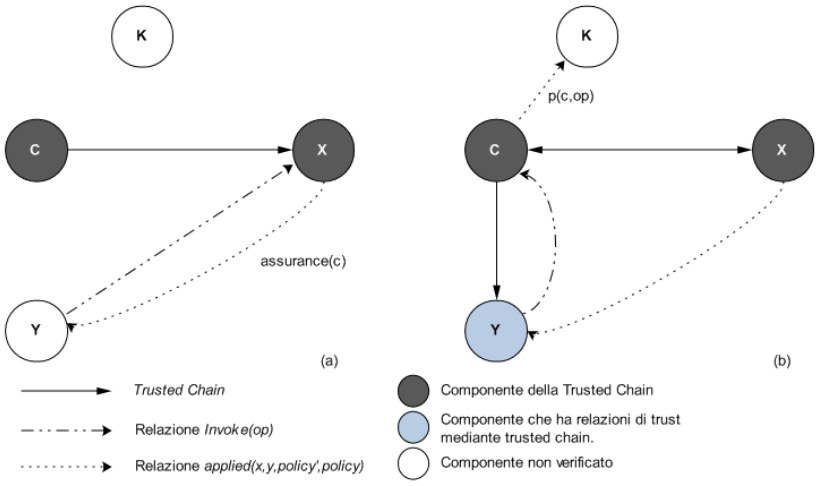


Figura 4.6: Regola f).

Le equazioni sopra esposte indicano due modalità con cui i controlli vengono effettuati, che definiscono una prima classificazione delle verifiche effettuabili. Le proposizioni a), b), c), permettono la verifica di determinati componenti mediante richiesta ad un terzo soggetto *fidato*, che ha la visibilità adeguata per poter *applicare* le politiche indicate dal componente *richiedente*. Per gli insiemi $closeapply(c)$, $closetrust(c)$, $closeinvoke(c)$, definiti in precedenza, è descritta

quindi una equazione che specifica i vincoli che un componente deve soddisfare per poterli estendere.
Le proposizioni e), f), d), invece, descrivono i medesimi controlli effettuati nel caso in cui:

- la visibilità del componente certificatore nei confronti di chi richiede la verifica d'integrità è *diretta*;

- il componente y richiede di essere **esso stesso** certificato. Non appartenendo ad alcuno degli insiemi definiti per un terzo componente c, richiede di essere verificato da un soggetto *fidato* per c;

Inoltre, l'appartenenza di un componente *richiedente* C_1 all'insieme $closeapply$ di un componente *certificatore* C_2 e del componente *certificatore* C_2 all'insieme $closetrust$ del *richiedente* C_1, indicano una relazione di *trust* tra C_1 e C_2.

Il calcolo del punto fisso corrisponde alla chiusura transitiva dei vari insiemi calcolata per tutti i componenti che interagiscono nel modello. In questo modo si calcola, per ogni componente, l'insieme di soggetti che esso può ritenere affidabili mediante verifica diretta o mediante una *trusted chain*.

Un esempio di applicazione del modello formale, nel caso di una richiesta di esecuzione di un'operazione da un componente, può essere descritto a partire da uno **stato iniziale**, nei seguenti passi:

1. Il componente A richiede di invocare una operazione op_i al componente B;

2. B verifica se A può invocare op_i, cioè se $A \in closeinvoke(B)$;

3. se $A \in closeinvoke(B)$ viene eseguita op_i, altrimenti B può:

   - controllare se esiste un test che esso stesso può applicare ad A(caso di *visibilità diretta*)
   - richiedere di eseguire il test ad uno dei componenti x in $closetrust(B)$ che può applicarlo, cioè tale per cui $p(B,op)) \sqsubseteq_p test(x,A)$ di effettuare la verifica per conto di B;

4. se $B \notin closeinvoke(x)$, x può richiedere l'applicazione di una politica di assurance su B da parte di un terzo soggetto che appartiene a $closetrust(x)$;

Il precedente esempio mostra la caratteristica fondamentale di questo modello, cioè che ogni politica che un componente può applicare dipende dal livello di sicurezza che esso richiede. Infatti, se prendiamo come esempio le operazioni di estensione dei diversi insiemi $closeinvoke(c)$, $closeapply(c)$, $closetrust(c)$, notiamo che, per poter inserire un componente in un insieme, viene applicata al componente stesso una politica tanto rigida quanto i requisiti che l'insieme impone. Inoltre, il componente deputato all'applicazione di tale politica deve avere un determinato livello di affidabilità, che dipende dall'insieme che si vuole estendere. Ad esempio, per estendere $closetrust(c)$ è necessario un componente che possa applicare una politica vincolante almeno quanto $trustedpolicy(c)$ e che appartenga a $closetrust(c)$. Questo dato ci permette di definire meglio l'ordinamento parziale tra le tre tipologie di politiche definite in precedenza.
Definiamo quindi le politiche $trustedpolicy(c) =< S1, f1 >$, $assurance(c) =< S2, f2 >$ e $p(c,op) =< S3, f3 >$, allora

$$p(c,op) \sqsubseteq_p assurance(c) \sqsubseteq_p trustedpolicy(c) \Leftrightarrow [S_3 \sqsubseteq_s S_2 \sqsubseteq_s S_1] \wedge [f_3 \leq f_2 \leq f_1]; \quad (4.2)$$

Di conseguenza, un componente che è capace di applicare la politica più rigida può applicare le altre, ma non viceversa. Inoltre, tale caratteristica descrive la *chain of trust* che si stabilisce tra i vari componenti. Ad esempio, il componente che può applicare la politica $assurance(c)$ su un componente che invoca c, può eseguire l'operazione perchè ritenuto affidabile da c, in quanto ad esso è stata applicata la politica $trustedpolicy(c)$. Questo vale anche per i componenti che eseguono $p(c,op)$ per conto di c.

Nel mapping del modello astratto in una architettura reale, è quindi necessario definire **come** un componente possa applicare determinate politiche per verificare altri componenti. Il modello formale, infatti, non può stabilire a priori quali siano le politiche, cioè gli insiemi di controlli, che determinano le proprietà di affidabilità dei componenti.
Il modello permette così di calcolare a priori quali componenti possono essere certificati in base alla catena che si crea mediante componenti che hanno superato test sempre più rigidi, fino ad arrivare all'origine dell'affidabilità che è il componente o l'insieme di componenti *radice*, le *root of trust*.

Come detto in precedenza, è possibile formulare diverse versioni del modello dipendenti dalla granularità prescelta per la descrizione di componente e dai tipi di politiche che i componenti applicano o richiedono, come vedremo nella sezione 4.3.

4.3 Architettura

Il framework descritto in questo lavoro ha il proposito di implementare, a partire dal modello formale definito nella sezione 4.2 la verifica dell'integrità di un sistema, dalla sua configurazione statica fino a controlli più granulari, come l'ispezione delle proprietà semantiche del software eseguito al momento dei controlli.
Inoltre, il framework sfrutta un protocollo di attestazione remota definito per poter comunicare le misure d'integrità effettuate sul sistema client al partner remoto a cui esso vuole collegarsi. Nelle seguenti sezioni illustreremo i vari componenti dell'architettura, i diversi modelli di architettura realizzati, i moduli implementati, il protocollo di attestazione remota.

4.3.1 Componenti del framework

VIMS, nella sua concezione generale, include diversi componenti ognuno dei quali con un ruolo ben definito. I componenti del framework sono i seguenti:

- Il server remoto di servizi(Remote-S);
- L'host client esegue un hypervisor ed i componenti:
  - Macchina virtuale di assurance(Assurance-VM);
  - Macchina virtuale client(Client-VM);
  - Trusted Platform Module, se presente sull'host.

Descrizione generale delle interazioni Le interazioni tra i vari componenti partono da una richiesta di accesso effettuata da parte di uno dei processi eseguiti dalla *Client-VM* verso un server remoto *Remote-S*, che può essere l'interfaccia d'accesso a servizi o ad una rete privata. La Client-VM è una macchina virtuale che simula un sistema general purpose con tutti i principali servizi che un utente può avere e/o configurare. Non è introdotto un vincolo sul software presente e, nel prototipo, essa viene fornita con un minimo insieme di processi già presenti che è estendibile dall'utente. Remote-S, prima di accordare l'accesso al client, utilizza un canale di controllo verso il componente *Assurance-VM*, una macchina virtuale appositamente istanziata sullo stesso host fisico di Client-VM ma non visibile ad essa, che possiede le librerie ed i moduli software necessari per l'applicazione dei controlli. Tali controlli, effettuati a run-time sui processi e strutture dati di Client-VM, vengono comunicati a Remote-S mediante un protocollo di attestazione remota, per permettere di applicare politiche adeguate riguardo l'accesso ed i diritti da concedere a Client-VM sui servizi e risorse da esso esposti verso l'esterno. Mediante il canale di controllo, il Remote-S e la Assurance-VM si scambiano informazioni riguardo:

- La politica di sicurezza da applicare;
- Il risultato dei controlli periodici su Client-VM.

Remote-S è perciò effettivamente in grado di monitorare lo stato del client e poter prendere decisioni durante tutto l'arco della connessione, estendendo o restringendo i diritti di accesso fino a chiudere la connessione in caso di rilevamento di gravi minacce.

4.3.1.1 Il Server remoto(Remote-S)

La piattaforma server prevede due configurazioni, che corrispondono a due diverse tipologie di architettura:(i)server di accesso normale,(ii) server di accesso virtualizzato. La prima configurazione e la seconda differiscono, dal punto di vista funzionale, per alcune proprietà. La principale è la possibilità di implementare funzionalità di mutua autenticazione e/o mutua attestazione tra Remote-S e Client, realizzando il server con la stessa architettura del client e modificando il protocollo di attestazione.

Server non virtualizzato Il server esegue dei moduli, realizzati per l'utilizzo del framework, che interagiscono tra di loro per la gestione del flusso dei dati, e con la Assurance-VM per effettuare i controlli e definire altre informazioni che controllano la loro applicazione periodica. Il sistema che interagisce con la macchina di assurance fornisce diverse funzionalità:

- comunicazione cifrata e delle relative chiavi;
- gestione di un database di hash di processi, strutture dati, moduli e componenti (BIOS, Bootloader) validi;
- gestione di un database di politiche applicabili;
- implementazione del protocollo di attestazione.

Nella fase iniziale della comunicazione con Assurance-VM, Remote-S può effettuare richieste alternative utili alla definizione del livello di affidabilità iniziale di Client-VM.
Tali richieste dipendono dalla configurazione iniziale del sistema installato sul server.
I messaggi che può inviare per l'esecuzione dei controlli si dividono in due tipologie:

- richieste esplicite di controlli;
- definizione di livelli di assurance.

Ogni richiesta è associata ad una politica che Assurance-VM dovrà applicare. La politica indicata infatti, è necessaria per definire i controlli e la loro frequenza di applicazione.
Le politiche previste nel prototipo sono:

- on-demand, cioè con controlli eseguiti su richiesta;
- timeout, periodo di tempo che intercorre tra l'esecuzione di due insiemi di controlli;

In generale, una richiesta avrà la forma:

$$\begin{aligned} Request &= < IDProcesso(0).....IDProcesso(i), policy > \\ Request &= < Assurancelevel, policy > \end{aligned}$$

Il primo tipo è eseguito su richesta e permette un controllo più mirato e granulare di determinati insiemi di processi e/o strutture dati, ad esempio potremmo dover valutare l'integrità dell'antivirus o di un HIDS o del kernel stesso per poter reputare affidabile un sistema che accede alla intranet aziendale. Il secondo tipo definisce invece un mapping tra insiemi di controlli e determinati livelli di affidabilità decisi in fase di configurazione. Il livello di assurance è una funzione che associa ad un valore intero un insieme di misure:

$$level : Int \rightarrow Measures* \tag{4.3}$$

In ogni caso, l'Assurance-VM possiede un insieme minimo di controlli definito alla configurazione e che sono eseguiti dal sistema che ospita la Client-VM, indipendentemente dalla presenza di Remote-S. Nella fase di handshake del protocollo, invece, il Remote-S comunica il livello di assurance e la temporizzazione che la Assurance-VM utilizzerà come canoni per effettuare i controlli, estendendo l'insieme minimo.

Server virtualizzato Partendo dal modello teorico, sono state analizzate le modalità che permettono di applicare i medesimi controlli effettuati sul client anche sulla macchina server. E' stato possibile incapsulare le precedenti funzionalità all'interno del server remoto implementato mediante due macchine virtuali, RemoteS-VM, che utilizza i componenti indicati per la precedente soluzione e AssuranceServer-VM, la cui configurazione è simile a quella dell'Assurance-VM descritta nel paragrafo successivo. Questa configurazione permette il monitoraggio del server da parte di un'altro host. L'host con le due macchine virtuali RemoteS-VM ed Assurance-VMs potrà quindi essere contemporaneamente client verso un host verificatore e server verso gli host che vogliono accedere ai suoi servizi.
In tal caso, il protocollo di attestazione remota, dovrà essere leggermente modificato rispetto

alla versione senza mutua attestazione, basata sulla richiesta ricevuta da una VM o dall'altra come descritto in 4.3.2.

4.3.1.2 L'Assurance VM

La *assurance-VM* è una macchina virtuale privilegiata, configurata per effettuare le verifiche sulla macchina Client-VM associata. Essa ha il vincolo di eseguire un insieme minimo di software, senza possibilità di aggiungere altri servizi oltre quello di attestazione remota e di non essere visibile all'utente della macchina Client-VM. Partendo da questo presupposto, la stabilità della Assurance-VM permette di effettuare test con un certo livello di affidabilità su diversi processi eseguiti nella macchina client. La macchina virtuale di assurance sfrutta due diverse tecniche per poter valutare l'integrità software della Client-VM:

- introspezione delle macchine virtuali;
- misura d'integrità mediante TPM;

Mediante l'introspezione della Client-VM, la Assurance-VM ottiene le strutture dati richieste dai controlli direttamente dalla memoria della macchina, *vista* dal livello dell'hypervisor. Una volta estrapolate le strutture dati, il framework VIMS estrae informazioni sui componenti a cui quei dati corrispondono. Ad esempio, potremmo voler controllare lo stato delle interfacce di rete di Client-VM per verificare se sono in modalità promiscua. Come vedremo nel paragrafo 4.3.1.2, le possibilità offerte dalla libreria di introspezione utilizzata ci permettono di poter scegliere la configurazione opportuna a seconda delle circostanze e della granularità dei controlli desiderata. In particolare, la Assurance-VM valuta degli insiemi di *asserzioni* dalle strutture dati recuperate in memoria. Ogni asserzione viene definita come un **invariante** calcolato sulla applicazione originale che vincola i dati, e mediante il quale è possibile rilevare se essa è stata attaccata o corrotta. Valutando tali asserzioni, il framework può ad esempio garantire l'integrità delle strutture dati del kernel o controllare che un processo esegua solo un predefinito insieme di operazioni senza compromettere il sistema.
Questa è in realtà una forma di attestazione semantica dell'integrità, infatti attraverso il monitoraggio del comportamento corrente della Client-VM, la Assurance-VM controlla non solo i file binari dei processi prima del loro caricamento nel sistema operativo, ma estende tale verifica anche allo spazio di memoria utilizzato dal programma nell'intervallo di tempo relativo all'esecuzione.
Questa estensione ha il pregio di **mitigare** le limitazioni del controllo di assurance basato unicamente sul controllo dell'integrità effettuato mediante il solo chip TPM e le librerie ad esso

associate. Questi componenti hanno il particolare svantaggio di poter misurare l'integrità solo prima o dopo l'esecuzione del codice oggetto dei processi.

Quindi, le asserzioni che VIMS permette di valutare sono definite a partire da asserzioni che considerano:

- Hash dei processi;
- Configurazioni e strutture dati del kernel;
- Componenti dell'ambiente di esecuzione (es. interfacce di rete, lista processi)
- Strutture del kernel riguardanti le system-calls.

La Assurance-VM, esegue i controlli in base a due tipi di politiche, che, in ogni caso, sono estese con un insieme di verifiche minime stabilito dalla configurazione iniziale. Le politiche, come descritto nella sezione 4.3.1.1, sono inviate dal Remote-S mediante un canale di controllo cifrato ed un protocollo di attestazione remota d'integrità. Le politiche vengono interpretate dalla Assurance-VM che memorizzerà la richiesta ed applicherà le verifiche con la frequenza voluta. La macchina virtuale di assurance, in VIMS, possiede diversi componenti, che sono attivati a seconda della configurazione scelta sull'host in fase di inizializzazione.
Tali componenti dipendono principalmente da:

- configurazione della libreria di introspezione;
- presenza di vTPM e TPM hardware;

Configurazioni Architettura L'estrema flessibilità della libreria di introspezione utilizzata ha permesso di ottenere due configurazioni diverse per componenti utilizzati e per test effettuabili. Il nome assegnato alle due architetture è esplicativo delle proprietà che implementano:

- Architettura trasparente;
- Architettura non trasparente;

La prima, implementata mediante le librerie Xen-VMI come descritto nella sezione 3.3.1.2, permette un isolamento completo della macchina che effettua i controlli rispetto alla macchina che è controllata. Il passaggio delle informazioni tra gli spazi di indirizzamento dei due sistemi operativi concorrenti avviene mediante le chiamate alle API dell'hypervisor Xen eseguite dalla macchina di assurance. Come descritto in precedenza, questa libreria permette al framework VIMS di implementare un ampio insieme di controlli [18]:

- Dispositivi di rete;
- Lista dei processi;
- Lista dei file aperti;
- Moduli del kernel;
- Tabella delle system call;
- Tabella delle interruzioni;
- Text area del kernel.

L'implementazione dell'accesso alle funzionalità di introspezione "trasparente" sono descritti nella sezione 4.3.1.2. La seconda architettura, invece, sfrutta la possibilità fornita dalla libreria

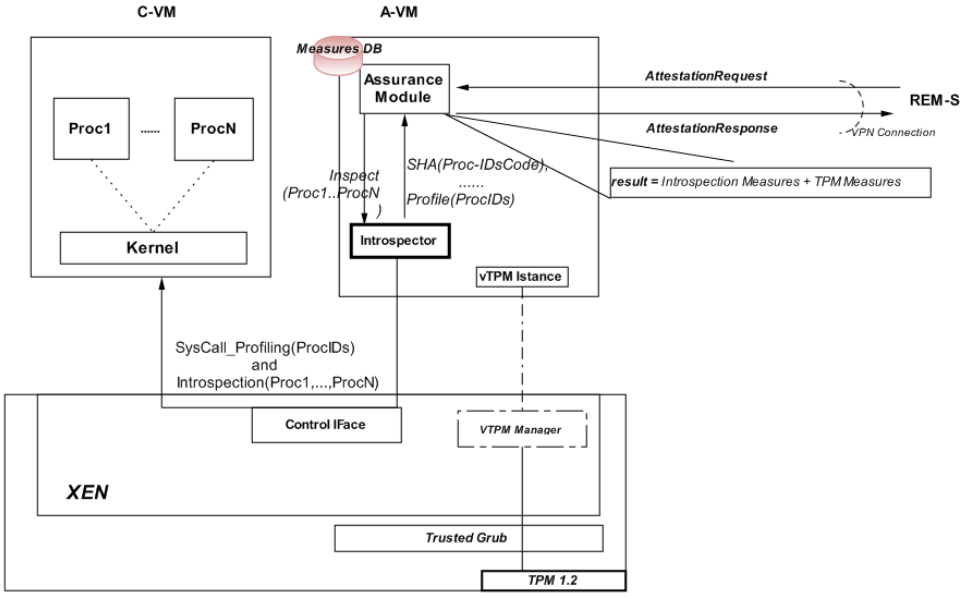


Figura 4.7: Architettura *trasparente*.

di introspezione di poter effettuare controlli più granulari e specifici al costo di dover rivelare la presenza della macchina virtuale che effettua i controlli. Infatti, come mostrato in figura 4.8, il framework offerto da VIMS è esteso mediante dei moduli implementati e configurati sulla macchina Client-VM che implementano la verifica di determinate proprietà. Questi moduli, detti *agent*, possono essere, ad esempio, antivirus, HostIDS, parser di log di sistema. Gli agenti definiscono una interfaccia ad-hoc mediante il quale comunicano le informazione raccolte ad un processo, eseguito sulla stessa macchina, il **Collettore**, delegato a comunicare verso la

Assurance-VM tali segnalazioni, gli *allarmi*. Sul lato della Assurance-VM, gli allarmi vengono ricevuti da un apposito modulo abilitato in fase di configurazione, il **Director**, che comunica all'interfaccia di VIMS su Assurance-VM il codice dell'allarme.

Il funzionamento del framework, in questo caso, è esteso dalla funzionalità di ricezione di segnalazioni che è attivata dalla Client-VM. Assurance-VM, una volta ricevute le informazioni ed interpretate, esegue, autonomamente, un insieme di controlli associato all'allarme ricevuto e quindi memorizza lo stato della Client-VM per comunicarlo nel caso sia richiesta una attestazione remota.

$$alarm : Int \rightarrow Measures* \tag{4.4}$$

La funzione *allarme* è interpretata in maniera simile alla funzione *livello di assurance*. Essa, infatti, associa un valore intero ad un insieme di misure da effettuare. Nel caso del *livello di assurance*, però i controlli sono effettuati in base ad una logica dettata dal partner che offre i servizi, il Remote-S. Tale la politica di sicurezza può non corrispondere a quella applicata su Assurance-VM e definita inizialmente dall'insieme minimo di controlli. Grazie all'estensione fornita dalla libreria di introspezione, invece, la Assurance-VM può anche eseguire controlli in base alle politiche definite sui moduli in Client-VM, che potrà essere diversa da quelle adottate sia su Assurance-VM che da quelle ricevute da Remote-S, e che non sono legate strettamente all'accesso remoto a servizi.
Il vantaggio di questa configurazione è quello di ridurre il numero di messaggi scambiati tra i due partner remoti. Infatti, tale modalità di controllo permette che un insieme significativo di verifiche siano già state effettuate alla richiesta di attestazione. Quindi all'handshake sarà già possibile determinare l'integrità dei principali componenti del sistema. Questo procedimento rappresenta a tutti gli effetti una politica locale di determinazione dell'integrità del sistema. Il protocollo di attestazione non è modificato nelle due configurazioni.

Estensione dell'affidabilità mediante TPM L'insieme di controlli effettuati mediante VIMS presenta un grosso inconveniente dal punto di vista del modello di architettura. Infatti, il modello può garantire la sicurezza del sistema perchè si possa assicurare che il VMM e Assurance-VM non siano stati oggetto di compromissione. La minimizzazione del software eseguito sul VMM e dei servizi esposti verso la rete di Assurance-VM, chiaramente non permette di garantire che esse siano sicure. Si è resa quindi necessaria la ricerca di un meccanismo che permettesse di estendere la sicurezza di questi componenti. A partire dal modello formale, si è adottata una strategia basata sulla constatazione che per garantire che un componente *x* sia sicuro, è necessario un'altro componente *y* che lo certifichi, e così anche *y* richiede la stessa certificazione effettuata da un terzo componente. Da questo concetto, e quello di *root of*

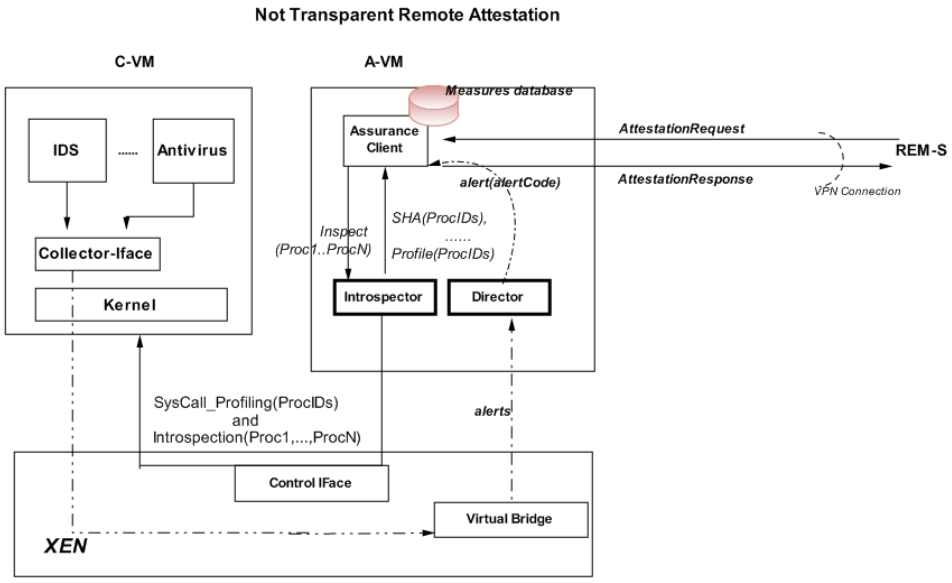


Figura 4.8: Architettura *non trasparente*.

trust, la soluzione più ragionevole da applicare è stata l'utilizzo del *Trusted Platform Module*. Diversamente da altre soluzioni presenti in letteratura [36], [66], [45], [38] e [73], l'utilizzo di tale componente in VIMS è limitato alla validazione della sequenza di boot fino all'esecuzione del VMM, estendendo così la Trusted Computing Base fino al livello hardware. Come descritto nel Cap.2.4.3.2, all'avvio del sistema, vengono effettuate una serie di misure dai livelli più bassi, a partire dal BIOS fino all'hypervisor. Tali misure sono abilitate mediante l'implementazione dello stack TCG da parte dei livelli che effettuano i controlli.
Le verifiche con il sussidio della tecnologia TPM si fermano al controllo che il VMM effettua sulla Assurance-VM, l'esito di tali controlli è ottenuto poi da VIMS mediante lo stack e le librerie di alto livello di accesso al TPM, completando la trusted chain dalla root of trust fino al software eseguito nella Assurance-VM.

Le misure, necessarie per la fase di attestazione remota, vengono acquisite dai registri del TPM mediante i comandi esportati dal framework VIMS. Questi comandi sono legati alle API definite dai componenti hardware.
I dati così ottenuti vengono usati come base mediante la quale costruire la *trusted chain* dei vari componenti software eseguiti sulla macchina.
Il framework consente una buona flessibilità, abilitando le modalità chiamate `TPM` e `noTPM` a seconda della presenza sull'host fisico del trusted module. Chiaramente, nel caso del flag `noTPM` la trusted chain, e di conseguenza la TCB, non avranno le stesse caratteristiche di affidabilità e,

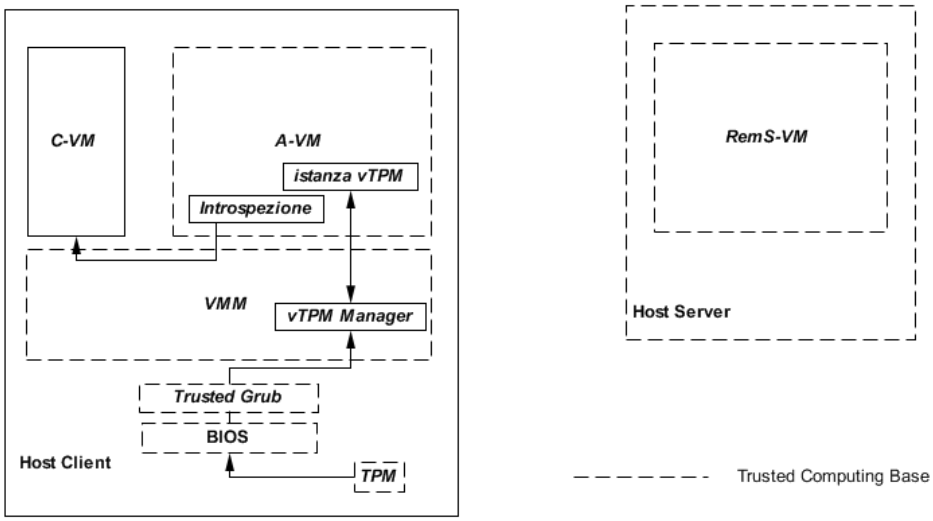


Figura 4.9: Trusted Computing Base ottenuta mediante tpm.

in base alla politica adottata dal *Remote-S*, potranno eseguire solo alcune operazioni.

4.3.1.3 Misure

Le misure d'integrità effettuate sui componenti dell'host hanno lo scopo di rilevare eventuali modifiche illegali del software per poter comunicare ad un secondo sistema il livello di assurance. Infatti, se VIMS rileva una qualche incongruenza nella configurazione o nel codice eseguito, non effettua altre azioni oltre quella di memorizzare ed acquisire le informazioni necessarie per ricostruire in remoto lo stato e la storia del sistema **dal suo ultimo riavvio**. Si è così scelto di fare aderire il framework alle specifiche rilasciate dal TCG per il TPM v1.2 riguardo il trattamento delle misure, rendendo omogenee le misure effettuate mediante il chip e quella mediante introspezione. Il problema che si è posto circa la procedura di attestazione riguarda la modalità con cui le misure effettuate dal TPM devono essere trattate prima di essere spedite ad un sistema che attesti l'integrità dell'host. Come illustrato nel paragrafo 2.4.3.2, il TPM estende le sue funzionalità ai livelli superiori dell'architettura software mediante due funzioni: `tpm_quote` e `tpm_extend`.
La `tpm_extend` permette di calcolare l'hash di un valore con il contenuto di uno dei PCR memorizzando il risultato nel medesimo registro come illustrato nel paragrafo 2.4.3.2. Il framework utilizza questa funzione per implementare i controlli che permettono di generare una trusted chain. La `tpm_quote` invece è utilizzata per la remote attestation. Infatti, essa permette di garantire l'autenticità dei dati comunicati all'host attestatore, firmando il messaggio mediante una delle chiavi custodite dal TPM, la *AIK*.

L'esistenza di questa chiave e la modalità con cui le specifiche TCG definiscono il trattamento del contenuto dei PCR per il protocollo di attestazione hanno portato all'utilizzo di *due chiavi* diverse per la cifratura delle misure effettuate, rispettivamente, mediante introspezione e mediante TPM. Ciò rispettando anche il fatto che non esistono relazioni tra i dati ottenuti mediante le due tecniche. Si può dire, in generale, che il messaggio di risposta verso il server remoto sia così composto:

$$AttestationResponse = [MisureIntrospezione]_{Sign_{AVM-key}} + [MisureTPM]_{Sign_{AIK}} \quad (4.5)$$

La sicurezza della chiave privata utilizzata nella Assurance-VM è garantita dalla TCB e dalla configurazione del protocollo di attestazione mediante un sistema di certificazione garantito da una *certification autority* terza, implementata come modulo in Remote-S o come entità a sè stante. Se il TPM non è presente sulla macchina fisica, la procedura chiaramente non inserirà le misure del TPM.

4.3.2 Protocollo di attestazione

L'attestazione remota [73], come illustrato nel capitolo 2, è usata per attestare la configurazione di un host ad un altro host. Nel nostro caso, il protocollo è applicato in un contesto di comunicazione tra sistemi remoti che simulano un ambiente di comunicazione SCADA, in particolare tra una macchina client che vuole accedere in VPN a risorse di rete critiche.

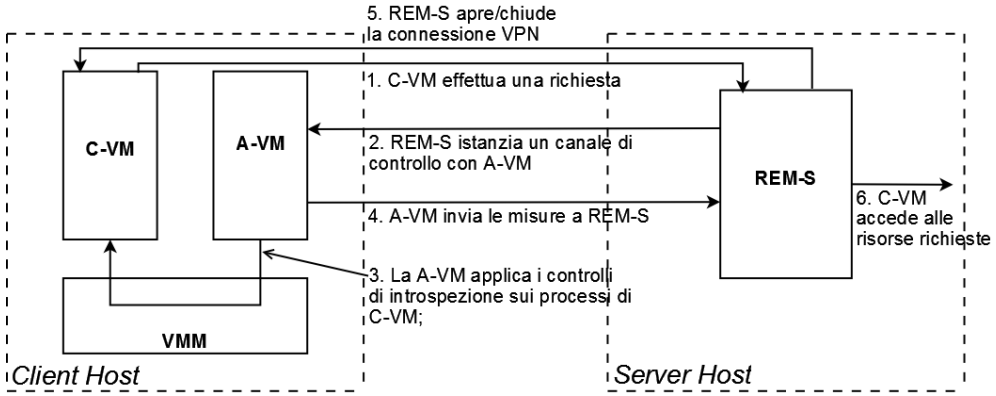


Figura 4.10: Protocollo di attestazione.

Il protocollo di attestazione remota che presentiamo, in una visione ad alto livello, è illustrato nella figura 4.10 ed i passi base del protocollo tra Remote-S, Assurance-VM e Client-VM sono i seguenti:

- Client-VM richiede a Remote-S di stabilire una connessione alla rete Intranet di cui è interfaccia esterna;
- Remote-S, dopo averlo autenticato mediante le procedure del protocollo VPN, stabilisce un canale di controllo con la Assurance-VM presente sullo stesso host del client;
- Remote-S inizia l'*handshake* del protocollo di attestazione con la Assurance-VM e trasmette una lista di parametri iniziali;
- la Assurance-VM applica i controlli concordati secondo la politica indicata da Remote-S. Successivamente, verifica l'effettiva corrispondenza tra le misure ed il risultato atteso degli oggetti misurati;
- la Assurance-VM restituisce al Remote-S, sul canale di controllo, il risultato delle misure calcolate;
- in caso di riscontro positivo, il Remote-S comunica ad Assurance-VM la politica da applicare durante la sessione di comunicazione e viene accordato l'accesso alla Client-VM alla Intranet;

Il protocollo, esclusa la fase di handshake, viene eseguito nuovamente secondo la frequenza stabilita dalla politica scelta dal Remote-S. In questo modo, come precedentemente illustrato, i controlli possono essere periodici e/o on-demand in base alla configurazione che l'amministratore di rete vuol utilizzare per il gateway di accesso. In generale, il Remote-S sarà quindi in attesa di risultati secondo la frequenza prestabilita dalla politica scelta.

Nel caso in cui i riscontri inviati dalla Assurance-VM fossero negativi, la logica del Remote-S del prototipo prevede diverse possibilità. Ad esempio, l'invio di specifiche richieste di attestazione dei componenti rilevati malfunzionanti, un cambio di politica ad un livello più restrittivo, la possibilità di ignorare il risultato (caso di componenti non critici), la chiusura della connessione del client.
In questo contesto, è evidente che è possibile estendere il framework con funzionalità interessanti ed utili per la gestione di una rete aziendale, come vedremo nel capitolo5.

Nel caso dell'implementazione con virtualizzazione del componente Remote-S ed utilizzo del protocollo di *mutua attestazione*, un esempio di interazione tra due host che verificano la propria integrità è illustrato in figura 4.11.

Come illustrato in figura 4.11, la modifica del protocollo di attestazione coinvolge l'ulteriore passaggio effettuato tra la Assurance-VM residente nell'host client e quella residente sull'host

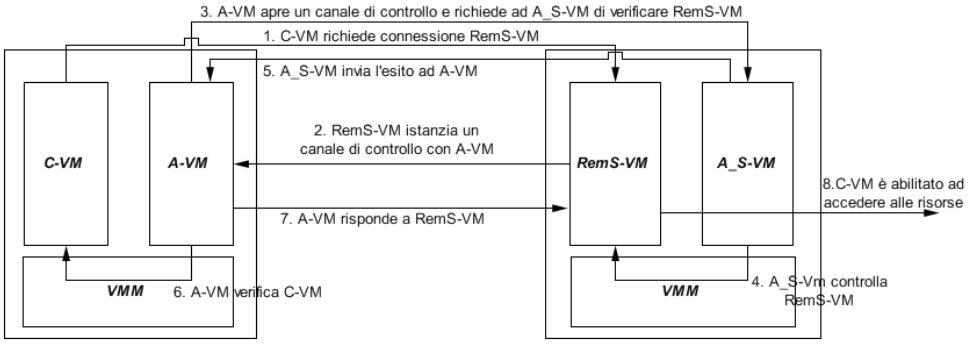


Figura 4.11: Visione generale del protocollo di mutua attestazione.

server. I passi sono estesi mediante la verifica dello stato della macchina virtuale RemS-VM, che avviene in maniera speculare alla macchina Client-VM.

4.4 Implementazione

Questa sezione presenta una descrizione più dettagliata dell'implementazione del framework VIMS. Per quanto concerne l'architettura ed il modello di rischio ad essa associato, sono necessarie alcune ipotesi.
La proprietà più importante, dal punto di vista della sicurezza del modello, è data dalla presenza del TPM. Se il TPM non è disponibile sul sistema su cui sono eseguite la Client-VM e la Assurance-VM, non è possibile garantire una root of trust trasparente e completamente affidabile. In questo caso, la credibilità ed affidabilità delle misure d'integrità può essere garantita se e solo se:

- il VMM è fidato. Una ipotesi ragionevole per garantire queste proprietà può essere che non esistano interfacce remote sull'hypervisor e che il suo stato possa essere modificato solo mediante accessi fisici al sistema e che tali accessi siano rilevabili;
- la coppia di chiavi crittografiche della Assurance-VM non può essere ottenuta o rivelata a terzi;
- il Remote-S è fidato; ipotesi ragionevole se assumiamo che il server risieda in un ambiente controllato o che sia implementata l'architettura di mutua attestazione.

Se, invece, sul sistema che esegue il lato client di VIMS è presente il TPM, allora le principali ipotesi per garantire la sicurezza sono:

- TPM e la CPU sono affidabili;

- il BIOS che implementa il CRTM e che inizia la catena di misure sui componenti fino alla Assurance-VM è affidabile;
- la memoria non può essere corrotta, ad esempio mediante DMA.

Ovviamente, in entrambi i casi, la Client-VM non è affidabile e quindi può essere infetta da malware e virus. Infatti, VIMS non previene la possibile corruzione della Client-VM, ma permette di valutare in maniera sicura ed affidabile il suo stato.
Il Remote-S e la Assurance-VM, inizialmente stabiliscono la comunicazione sicura mediante certificati firmati da una Certification Authority fidata.
L'implementazione corrente fa uso dell'hypervisor Xen v.3.1.0 [64] [32] per la creazione ed il controllo delle macchine virtuali, ognuna dotata di sistema operativo Debian Etch 4.0 con kernel Linux v.2.6.18. Si sono adottati i tool e librerie Xen-VMI [18] e Psyco-Virt [18] per implementare la fase di introspezione su macchine virtuali, lo stack TCG fornito dalla libreria TrouSers [10], le librerie TPM/J [76] e vTPM [21] per utilizzare i dati ed effettuare misure mediante il chip TPM sull'hypervisor e la Assurance-VM.
Non avendo a disposizione una macchina fisica fornita del chip si è utilizzato un emulatore [50], eseguito al livello del VMM di Xen realizzato sulle specifiche TCG dall'IBM la cui interfaccia e comportamento corrisponde a quella esposta da un chip TPM di tipo Infineon che ha permesso la fase di test dell'architettura con prestazioni confrontabili.
I moduli del framework sono stati realizzati in Java. La scelta del linguaggio è dovuta a diverse ragioni. La prima è quella di avere una certa agilità nel descrivere una applicazione essenzialmente multithread ed ad alto livello. Un'altra ragione è che il funzionamento del framework richiede l'integrazione con librerie Java come le TPM/J.
L'interfaccia con le librerie di introspezione utilizza un semplice wrapper Java che avvia i controlli e legge i risultati ricevuti in formato testuale mediante apposito parser.

4.4.1 Remote-S

L'implementazione del Remote-S ha portato a due alternative, che rappresentano due casi d'uso valutati in questo lavoro:

- Server Gateway VPN;
- Generico Webserver http/https.

L'implementazione del framework nelle due architetture è molto simile, ma, mentre nella prima architettura i test effettuati sono basati principalmente sull'affidabilità, nel secondo caso si è considerata anche la valutazione delle prestazioni. La descrizione dei test è nel capitolo 5.

Il software utilizzato nel caso del Server VPN è OpenVPN [54], noto software open-source utile alla realizzazione di infrastrutture VPN. L'utilizzo di OpenVPN ha due motivazioni: (1)la licenza, (2)la possibilità di estensione mediante plug-in. Lo sviluppo del plug-in è stato molto complesso per la mancanza di supporto allo sviluppo di tali estensioni sul sito principale e nelle mailing-list degli sviluppatori.
Per effettuare test funzionali del framework, si è scelto di sviluppare un semplice modulo per OpenVPN che scrive i dati del sistema che si vuole autenticare su di un file secondo un certo formato. Su tale file, una classe chiamata **Starter**, eseguita sul server implementa un funzionamento di tipo *polling* attendendo modifiche, e, nel caso siano presenti, inizia la procedura di verifica di VIMS. In ogni caso, il meccanismo a plug-in di OpenVPN permette di mettere in attesa il client prima di permettere l'accesso alla rete, implementando la procedura di handshake coerentemente con il modello architetturale. Nel secondo caso, il webserver è stato realizzato mediante una classe chiamata JHttp che funge da webserver minimale. Tale WebServer, prima di inviare la pagina web cifrata mediante connessione OpenSSL, attiva VIMS per effettuare i controlli di consistenza. Il risultato è passato alla classe del webserver come semplice booleano che indica se inviare la pagina richiesta od un messaggio di errore. Lo *snippet* di codice della classe che implementa la chiamata a VIMS è riportato qui di seguito.

```
/*Call al framework VIMS*/
//misuriamo in base alla richiesta http del client
boolean resultHttp = vims.measure(connection.getInetAddress(),httprequest);
if(resultHttp){
    /*Invio pagina*/
    [.....]
}
else{ /*Error:403*/}
```

Un particolare interessante è capire come Remote-S possa derivare l'IP della Assurance-VM per poter richiedere i controlli. Nel nostro prototipo, l'IP è associato *staticamente*, ma si possono considerare diverse ipotesi di implementazione, come ad esempio calcolare l'IP di Assurance-VM rispetto all'IP di Client-VM partendo dal vincolo che entrambe le macchine risiedono sullo stesso host. L'assegnazione dell'IP alle macchine virtuali è comunque effettuata mediante il VMM di Xen, quindi in ambiente isolato da possibili manomissioni.

I moduli di VIMS usati dal Remote-S sono:

- Manager del database;
- Manager delle politiche;
- Modulo di attestazione;

- CoreEngine;

Il **manager del database** gestisce le chiamate al mini database realizzato con SQLite [29] che contiene le configurazioni inserite dall'amministratore mediante il manager delle policies, quali la tabella delle *misure*, la tabella dei *livelli di assurance*, la tabella delle *politiche*, la tabella degli *hash* dei processi e strutture dati. Il **manager delle politiche** gestisce tutta la logica delle azioni da intraprendere in base alle misure effettuate e interagisce con il modulo di attestazione per gestire i timer necessari per i controlli da effettuare. Il **CoreEngine** è il componente che integra tutti i moduli implementati, gestisce il flusso delle informazioni ed opera per soddisfare la logica delle interazioni. Il **modulo di attestazione** è il core del protocollo di attestazione del framework. Esso si occupa di tutte le procedure che riguardano la comunicazione con Assurance-VM, dalla cifratura/decifratura dei dati fino alla formattazione ed interpretazione dei messaggi. L'implementazione di questo modulo dipende dal protocollo. Ad esempio, esso deve essere modificato per inserire la comunicazione mediante il modulo di certification authority. Se consideriamo un livello più dettagliato, il protocollo implementa le interazioni mostrate nella figura 4.12 e figura 4.13. La figura 4.12 illustra l'interazione dei componenti principali di Assurance-VM e Remote-S, la fig.4.13 è invece focalizzata sulla Client-VM.

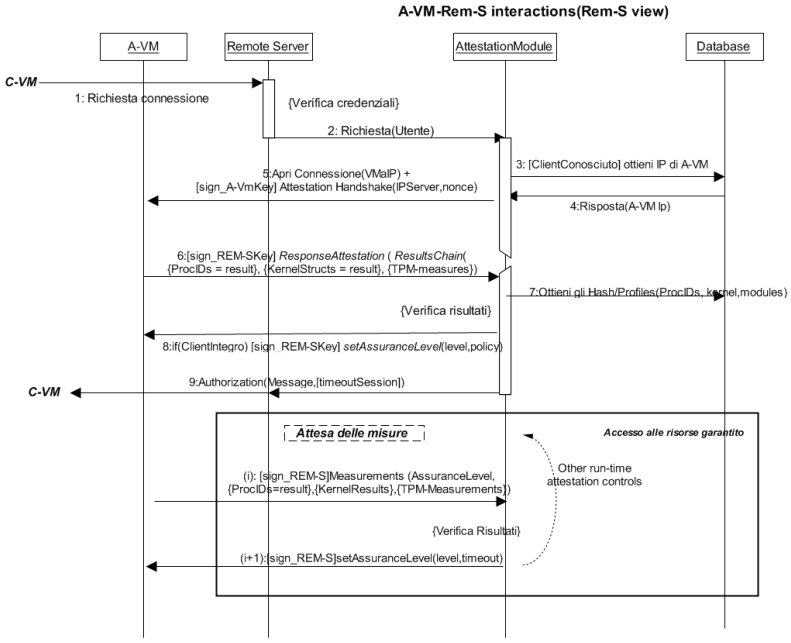


Figura 4.12: Protocollo di attestazione, interazioni Assurance-Vm e Remote-S.

La componente del protocollo più importante è quella che descrive le comunicazioni tra la Assurance-Vm e il Remote-S. I passi principali sono i seguenti:

1. $RichiestaConnessione(username, password)$:

   - apertura connessione tra la Client-VM ed il Remote-S;
   - verifica credenziali utente;

2. $Richiesta(Username, IP)$, il modulo di attestazione è attivato;

3. se nel precedente passo il client è stato autenticato, il modulo di attestazione richiede i dati della macchina virtuale di assurance al modulo gestore del database;

4. il modulo di attestazione riceve l'IP della macchina virtuale di assurance mediante il risultato della query;

5. $ApriConnessione(Assurance-VMIP)$, il modulo di attestazione crea un canale di controllo con la Assurance-VM;

6. $[AttestationHandshake(Ip\_Remote-S, nonce)]_{sign-AVM-Key}$, il modulo di attestazione invia il messaggio di handshake con alcuni parametri, firmato con la chiave pubblica della Assurance-VM;

7. $[AttestationResponse(ResultsChain(\{ProcIDs = results\}, \{KernelStructures = results\}, \{TPM\_measures\}), nonce)]_{sign-RemS-Key}$, la Assurance-VM effettua le misure delle applicazioni eseguite dalla Client-VM e le invia al Remote-S insieme alle misure dei componenti sottostanti al livello delle macchine virtuali effettuato dal TPM;

8. $Check-Results$: il modulo di attestazione verifica la catena di misurazioni inviata dalla Assurance-VM con quelle attese, contenute nel suo database;

9. $[setAssurancePolicy(level, policy)]_{sign-AVM-Key}$, il modulo di attestazione invia la *politica ed il livello di assurance* che la Assurance-VM dovrà applicare,
   ad es.: `(AssuranceLevel = 1, timeout);`

10. $Authorization(Message, [timeoutSession])$: il Remote-S accorda o chiude la connessione con la Client-VM stabilendo un timeout di sessione legato all'attesa dei risultati dalla Assurance-VM. Il campo `Message` è la risposta all'attestazione, ad esempio potrebbe essere "HTTP:403" nel caso di una Client-VM non integra.

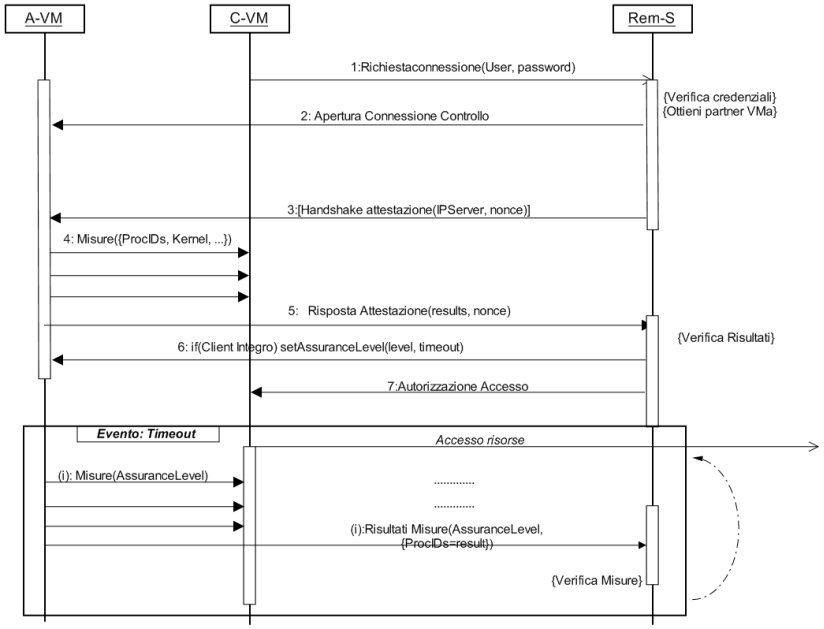


Figura 4.13: Protocollo di attestazione, interazioni Client-VM ed Assurance-VM.

4.4.2 Assurance-VM

Alcuni componenti dell'Assurance-VM implementano sia i controlli che la parte di comunicazione, e, per questo, risultano la parte più complessa dal punto di vista realizzativo. Sfruttando le due librerie di introspezione, Psyco-Virt e Xen-VMI, si sono potute realizzare due tipi di architetture, *non trasparente* e *trasparente* che differiscono per come l'introspezione è applicata sulla client-VM. I moduli base implementati sulla Assurance-VM sono:

- CoreEngine: gestisce il flusso di informazioni e richiede le misure;
- Database Manager: gestisce l'accesso al mini-database;
- Modulo di attestazione: implementa il protocollo di attestazione lato client;
- Manager delle politiche: utilizza le politiche ricevute da Remote-S per istanziare dei thread con timeout e recupera dal database le corrispondenti misure da effettuare.
- Introspection Manager: gestisce le chiamate alle librerie di introspezione;
- TPM Manager: wrapper e gestore della libreria TPM/J.

In base all'architettura scelta, all'interno dell'Introspector Manager sono abilitati dei moduli e dei corrispettivi wrapper che interpretano i risultati dei controlli ed inviano i comandi da

eseguire, rendendo **trasparente** a VIMS la gestione delle misure. In particolare, la soluzione non trasparente fa uso di un modulo, **Director**, che attende eventuali *allarmi* da moduli installati sulla macchina Client-VM, come ad esempio un IDS od un firewall, come si vede in figura 4.8. Un componente **Collettore**, istanziato sulla macchina Client-VM, invia l'output dei moduli alla macchina di assurance attraverso l'utilizzo delle API di Xen. Mediante il modulo Director, questo modulo interpreterà i valori ricevuti e attiverà delle verifiche di introspezione ad-hoc sui componenti segnalati.
Tali verifiche sono attivate mediante una logica inizializzata nel database durante la configurazione iniziale del prototipo, che, nel nostro caso, rimane *statica* durante l'esecuzione del sistema. Inoltre, a differenza dell'architettura trasparente, i controlli e le verifiche sono effettuate in maniera trasparente al Remote-S ed i loro risultati sono memorizzati per poter essere inviati alla richiesta od alla scadenza del timeout successivo.

In questo caso, il protocollo di attestazione utilizzato dall'attestation module sulla Assurance-VM, non subisce delle modifiche per sfruttare la conoscenza più granulare introdotta dalla libreria *non trasparente*. Infatti, il modulo definisce una interfaccia che rende indipendente la parte dei controlli del framework da quella di comunicazione e riceve le misure da inviare direttamente dal modulo *CoreEngine*. Per quanto riguarda l'implementazione del modulo CoreEngine, esso è stato dotato di una struttura dati, una coda con priorità che implementa una coda di richieste, che, non deterministicamente, possono provenire da:

- Allarmi libreria di Introspezione;
- Scadenza timeout controlli(caso policy temporizzata);
- Remote-S(se presente una policy on-demand).

Le richieste sono gestite mediante un thread che si occupa di smistarle mediante una logica opportuna e seguendo l'ordinamento stabilito dalle priorità.

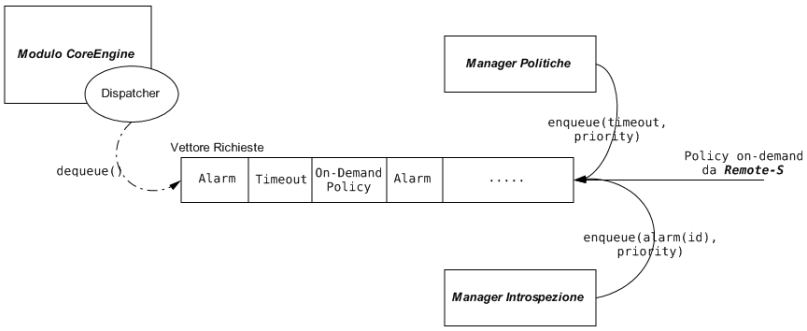


Figura 4.14: Thread dispatcher.

4.4.3 TPM e misura del sistema

La presenza sulla macchina fisica di un TPM abilita l'architettura lato client ad una estensione delle misure, inserendo tutti i livelli inferiori alle macchine virtuali nella trusted computing base. L'implementazione del modulo *Manager TPM* ha previsto una classe che utilizzasse le librerie TPM/J e IAIK jTSS messe a disposizione per l'elaborazione e l'accesso ad alto livello delle funzionalità della tecnologia TCG. Inoltre, per poter abilitare l'uso del componente hardware si sono dovute applicare delle patch a:

- kernel del VMM;
- macchina virtuale di Assurance;

Non avendo a disposizione un chip TPM reale ma di un emulatore realizzato presso l'ETH Zurich per effettuare test equivalenti, non si è dovuto applicare la patch al bootloader, prevista dalle specifiche. Per poter accedere alle funzionalità del TPM, si è utilizzata l'implementazione dello stack TCG *TroUsers Software Suite*, con licenza open source.
Lo stack fornisce una libreria che permette di utilizzare i seguenti servizi disponibili sul TPM: generazione chiavi asimmetriche RSA, cifratura e decifratura RSA, firma e verifica RSA, estensione dati nei PCR e log degli eventi, funzioni di Seal ed Unseal sui PCR, generazione numeri casuali. Un esempio di chiamata all'emulatore TPM per ottenere la parte pubblica dell'*endorsement key*:

```
\$sudo tpm_getpubek
    Tspi_TPM_GetPubEndorsementKey failed: 0x0000008 - layer=tpm, code=0008 (8),
        The TPM target command has been disabled
    Enter owner password:
    Public Endorsement Key:
       Version:   0101100
       Usage:     0x0002 (Unknown)
       Flags:     0x00000000 (!VOLATILE, !MIGRATABLE, !REDIRECTION)
       AuthUsage: 0x00 (Never)
       Algorithm:         0x00000020 (Unknown)
       Encryption Scheme: 0x00000201 (Unknown)
       Signature Scheme:  0x00000110 (Unknown)
       Public Key:
       c18c3d93 1bac9e9c e1c242c5 b11d5ce5 26b6fce8 b6fce7cf f1c242c5 e0e594d8
       f1c242c5 5a3b1d5c fda2e201 b1d5c201 5a3b1d5c b6fcbb7e e1851eab c8cbcbe5
       fda2e201 c93e4bb7 e0e55779 1bac9e9c 8208f203 f1477705 0317fe2b ca983d4d
       a71def6a f09b39cb a71def6a f09b39cb 26ca2278 1a5804ad f1e6dfc5 e0e55779
               [...]
```

Per abilitare i controlli sulla macchina virtuale di assurance,invece, si è esteso Xen mediante le librerie vTPM di IBM, presenti nella versione dell'hypervisor utilizzata.

La trusted chain realizzata mediante l'utilizzo del TPM e della libreria Trousers permette a VIMS di tenere traccia dell'integrità dei livelli inferiori alle macchine virtuali. Le patch che vengono applicate al kernel ed al bootloader, invece, permettono ai singoli livelli di implementare delle misure dei livelli superiori memorizzando i risultati nei PCR.
L'accesso alle informazioni da VIMS avviene mediante le librerie Java.
Un esempio del contenuto dei registri PCR a run-time è il seguente:

```
\$sudo cat /sys/devices/platform/tpm0/pcrs
PCR-00: 87 31 03 3F DD DB 3F DA 05 F5 07 63 21 50 B9 9B BF 6D 45 15
PCR-01: 13 02 B2 55 28 E4 B7 06 35 04 F2 6A 74 6E D2 E1 1D 17 24 3B
PCR-02: EB B3 BA AE E7 57 4B B6 37 AA AB 67 0F 9A C1 BC EB 6F 80 F3
PCR-03: 04 FD EC DD 50 1D AF 0F 62 4C 1F 99 60 12 CF 30 44 FF 46 10
PCR-04: 11 AF B4 D2 CC 62 91 18 80 D3 3E 51 BA 18 8A EE E9 E8 0A D3
PCR-05: 3D D4 62 3E 57 34 78 B1 EB A3 89 FE 24 5B 44 DB D0 79 70 52
PCR-06: 04 FD EC DD 50 1D AF 0F 62 4C 1F 99 60 12 CF 30 44 FF 46 10
PCR-07: 04 FD EC DD 50 1D AF 0F 62 4C 1F 99 60 12 CF 30 44 FF 46 10
PCR-08: 78 F7 B8 1C 1B 69 DD 24 10 4E 0C 81 59 70 FE 66 1B DD 93 46
```

Eseguite le misure sui componenti, esse possono essere memorizzate sul TPM mediante la funzione `tpm_extend`. Nella architettura ottenuta, queste funzioni non sono implementate al livello delle macchine virtuali, ma vengono eseguite fino al livello VMM, che verificherà lo stato della Assurance-VM controllando i file di configurazione ed i file immagine del kernel, ad esempio i file `xen_assurance.cfg`, `vmlinuz-2.6.18-xen` e `initrd.img-2.6.18-xen`. In questo caso, il controllo d'integrità statico risulta affidabile se si può assumere che la configurazione ed i componenti del DomU utilizzati da Assurance-VM siano fissati.
Le librerie Java saranno utilizzate per comporre le misure dell'host effettuate dal TPM con quelle effettuate mediante introspezione.

Durante l'esecuzione del sistema, la procedura per ottenere il file di misure completo da inviare durante il protocollo di attestazione remota prevede i seguenti passi:

- Accesso alle informazioni dei PCR con la funzione `lowLevel.TPM_PCRRead(i);`, cifrate mediante AIK del TPM;
- Accesso alle ultime misure effettuate con introspezione memorizzate sul database, cifrate mediante chiave privata di Assurance-VM;
- Composizione file XML con le misure effettuate;
- Invio del file XML, delle misure cifrate e dei relativi certificati sul tunnel SSL tra Assurance-VM e Remote-S

Se è presente una CA condivisa, Remote-S sottomette le informazioni necessarie per ottenere la chiave pubblica relativa alle componenti cifrate dal TPM ed dalla Assurance-VM. In questo modo, esso autentica anche i componenti, viceversa, si assume che Remote-S possieda tali informazioni, fungendo esso stesso da CA. Un esempio di attestation response per politica di tipo timeout è il seguente:

```
<attestation>
    <response>
        <overallresult>suspicious</overallresult>
        <nonceid>89892345</nonceid>
        <AssuranceIp>192.168.1.1</AssuranceIp>
        <reqpolicy>timeout</reqpolicy>
        <timemillis>20000</timemillis>
        <assurancelevel>4</assurancelevel>
        <measuresaggregate>
            <measureschain>
                <measure>
                    <idmeasure>TPM-Boot</idmeasure>
                    <hash>7844e409c39fad83bb65f7dac4c8a53e</hash>
                    <status>trusted</status>
                    <object>TPM.Measure.aggregate</object>
                    <name>boot-sequence</name>
                </measure>
                <measure>
                    <idmeasure>0</idmeasure>
                    <hash>7844e409c39fad83bb65f7dac4c8a53e</hash>
                    <status>trusted</status>
                    <object>kernel.struct</object>
                    <name>idt_table</name>
                </measure>
                ..................
                <measure>
                    <idmeasure>78</idmeasure>
                    <hash>0000e409c39fad83bb65f7dac4c8a53e</hash>
                    <status>trusted</status>
                    <statusprofile>wrong</statusprofile>
                    <object>process</object>
                    <name>modprobe</name>
                </measure>
            </measureschain>
        </measuresaggregate>
    </response>
</attestation>
```

Infine, la verifica della presenza del TPM sulla macchina fisica, che abilita il procedimento di aggregazione dei risultati delle misure mediante i flag `TPM` / `noTPM`, viene eseguita tramite la chiamata `takeOwnership(authKey, srkKey)` della libreria java di accesso al device.

4.4.4 Certification authority

La comunicazione implementata dal framework ed il protocollo di attestazione utilizzano la crittografia asimmetrica ed i certificati per lo scambio di messaggi tra i partner remoti e la loro mutua autenticazione. Il prototipo, inizialmente, ha utilizzato certificati generati e dislocati staticamente sulle macchine di test. Tale procedura, nel momento in cui il framework è stato provato con più di due partners remoti è divenuta poco flessibile. Si è scelto, quindi, di inizializzare dei certificati mediante un piccolo modulo che realizzasse semplici funzionalità di Certification Authority(CA).
Il modulo è stato anche utile per poter utilizzare, in maniera sicura ed in un caso reale, la parte crittografica delle procedure esposte dallo stack TCG per il TPM. Il certificato che è utilizzato per firmare le misure del TPM, infatti, è normalmente validato da una CA, a cui i partner remoti del protocollo si rivolgono per avere riscontro sull'autenticità dell'identità. Nel caso di utilizzo di una certification authority autonoma, e dislocabile quindi su una macchina diversa, è stata modificata ogni fase del protocollo di attestazione che prevedeva la ricezione di messaggio firmato. La modifica introduce una richiesta di validazione mediante CA fidata e condivisa tra le macchine del prototipo.
La messa in sicurezza e la verifica dell'integrità della macchina che esegue il *modulo CA* non è stata implementata, ma, chiaramente, il modello e l'architettura presentato in questo lavoro può essere applicato anche in tale host, rendendo l'architettura ed il protocollo omogeneo tra tutte le entità in gioco.

Capitolo 5

Test e conclusioni

Indice

Questo capitolo illustra i test sull'architettura di VIMS. In particolare, i test hanno l'obiettivo di valutare sia l'effettivo livello di sicurezza garantito, sia le prestazioni del protocollo di attestazione in casi reali.

5.1 Test

5.1.1 Test di sicurezza

I test di sicurezza sono stati realizzati per verificare l'efficacia di VIMS nella rilevazione di tentativi di intrusione o di manomissione e nella loro segnalazione mediante un procedimento consistente e tamper-proof. Sono stati effettuati test con due domini utente eseguiti come macchine virtuali su hypervisor Xen v.3.1.0. Il sistema utilizzato per i test ha le seguenti caratteristiche:

- Intel Centrino Core Duo con frequenza di clock 1,6 GHz;
- dimensione memoria fisica disponibile 2GB;
- dimensione memoria virtuale associata a dominio utente 128MB;

- immagine kernel distribuzione Debian: 2.6.18-etch4
- immagine kernel dom0 di Xen: 2.6.18-xen0;
- immagine kernel domU compilata dai sorgenti: vmlinuz-2.6.18-xenU
- versione di OpenVPN utilizzata: openvpn-2.1-rc7

L'architettura di riferimento per i test, tra quelle descritte in questo lavoro è quella "trasparente".

5.1.1.1 Rilevamento intrusioni

Per poter applicare i test di efficacia al dominio utente che rappresenta la Client-VM, si è utilizzata la batteria di test già presente nelle librerie di introspezione e si è valutato l'effettivo riconoscimento dei dati da parte del Remote-S. I controlli della libreria vengono eseguiti invocando il comando `xen_check_dom`, il cui output è elaborato dal modulo *Introspection Manager* eseguito da VIMS. Un esempio di output del comando della libreria, che elenca i controlli eseguiti sui processi eseguiti e su alcune strutture dati del kernel, è mostrato nella figura seguente.

```
cr4zy:/xen/xen_vmi# ./xen_check_dom -d 2 -p -s -k -i
-------START---------
System call table digest is:
83d3b0eceab0dbca652f740680edbe9178df16d9        correct
Kernel text area digest is:
60ad45c8b0aaf1b52595ce5fbf1003894b93bdfb        correct
PROCESS LIST:
PID     UID     COMMAND
1       0       init
2       0       migration/0
3       0       ksoftirqd/0
4       0       watchdog/0
5       0       events/0
6       0       khelper
7       0       kthread
9       0       xenwatch
10      0       xenbus
16      0       kblockd/0
18      0       kseriod
49      0       pdflush
50      0       pdflush
51      0       kswapd0
52      0       aio/0
739     0       udevd
1646    0       syslogd
1652    0       klogd
1670    0       sshd
```

```
1684    0       cron
1707    0       getty
1708    0       getty
1709    0       getty
1710    0       getty
1711    0       getty
1718    0       getty
INTERFACES:
eth0: PROMISC
-------FINISH---------
```

La libreria attiva i controlli sul dominio trasmesso come argomento, e, ad ogni verifica, confronta i risultati prodotti dall'host eseguito con i processi attesi e quelli ottenuti mediante introspezione. Il risultato della funzione viene restituito al chiamante che la visualizza come visto in precedenza.

I test di valutazione dell'efficacia dei controlli presenti con la libreria permettono di eseguire diversi tipi di attacco contro il dominio utente e di verificare l'effettivo rilevamento sul lato server. Nelle nostre prove si sono utilizzati i seguenti:

- Rootkit per la modifica del puntatore alla tabella *sys_call_table*;
- Rootkit per la modifica della *text area* di un processo;
- Sostituzione di uno dei binari di linux (*ps*) con una versione modificata per nascondere alcuni processi;
- Esecuzione del comando di cambio modalità della scheda di rete;

Mostriamo qui di seguito, come esempio, l'esecuzione dell'attacco che modifica delle aree di memoria del kernel che contengono la tabella delle system calls e le corrispondenti azioni di VIMS nella fase di attestazione.
La politica stabilita da Remote-S è di tipo `timeout` e le connessioni tra Client-VM / Remote-S e Remote-S / Assurance-VM sono già stabilite. Abbiamo simulato un attacco eseguendo un rootkit simile a `kad` [57], che modifica a run-time la tabella delle system calls. Utilizzando la funzione `get_syscall_table()`, la libreria di introspezione verifica che la tabella delle syscall sia integra mediante l'applicazione della funzione hash all'area di memoria individuata dalla libreria:

```
-------START---------
System call table digest is:
63c8a909cef0dacb80a3e067409abed98266d9df1      NOT correct
[...]
```

In VIMS, l'output riconosciuto mediante un parser è interpretato e memorizzato nel database controllato dal Database Manager. I dati ottenuti dall'Introspection Manager sono "taggati" con un identificativo del controllo effettuato ed un timestamp, in modo tale da poter mantenere uno storico di controlli all'interno del database.
Quando Remote-S riceve il risultato di controlli memorizzati, viene scelto l'insieme di dati legato all'ultimo controllo effettuato e spedito sul canale di controllo. Nel caso del controllo descritto in precedenza il file delle misure era composto come illustrato di seguito.Vengono omesse alcune parti non significative.

```
<attestation>
    <response>
        <overallresult>suspicious</overallresult>
        <nonceid>59852768</nonceid>
        <AssuranceIp>192.168.1.1</AssuranceIp>
        <reqpolicy>timeout</reqpolicy>
        <timemillis>20000</timemillis>
        <assurancelevel>4</assurancelevel>
        <measuresaggregate>
            <measureschain>
                <measure>
                    <idmeasure>TPM-Boot</idmeasure>
                    <hash>7844e409c39fad83bb65f7dac4c8a53e</hash>
                    <status>trusted</status>
                    <object>TPM.Measure.aggregate</object>
                    <name>boot-sequence</name>
                </measure>
                [.................]
                <measure>
                    <idmeasure>0</idmeasure>
                    <hash>63c8a909cef0dacb80a3e067409abed98266d9df1</hash>
                    <status>wrong</status>
                    <object>kernel.struct</object>
                    <name>sysc_table</name>
                </measure>
                [.................]
            </measureschain>
        </measuresaggregate>
    </response>
</attestation>
```

Remote-S rileva lo stato del client e aumenta il livello di assurance inviando una policy di tipo timeout:

```
Attestation Response received. Analyzing....done.
Structure Kernel-Text : correct;
[......]
Structure Syscall Table: not correct. Taking action.
```

```
Action: Update assurancelevel on Assurance-VM to : 5;
```

A seconda della configurazione scelta, il server può comportarsi anche in maniera differente e meno permissiva. Ad esempio, se nella stessa ipotesi di attacco la politica locale a Remote-S è definita come immediata chiusura della connessione, il risultato, sulla console di Remote-S è il seguente:

```
[......]
Structure Syscall Table: not correct. Taking action.
Action: Close client connection;
Aborting Client-VM connection....done;
Closing Assurance-VM control channel....done;
```

Sulla Client-VM, invece viene visualizzato il messaggio di errore di OpenVPN:

```
[.....]
Fri Jan 09 12:30:13 2009 us=863324 route ADD 192.168.1.0 MASK 255.255.255.0
                                                                  192.168.2.1
Fri Jan 09 12:30:13 2009 us=865174 Route addition via IPAPI succeeded
Fri Jan 09 12:30:13 2009 us=865252 Initialization Sequence Completed
Fri Jan 09 12:32:32 2009 us=865725 read UDPv4: Connection reset by peer
                                              (WSAECONNRESET) (code=10054)
```

Si è testata la medesima situazione nel caso di Remote-S implementato come WebServer che espone una semplice pagina html, e, partendo dal caso del precedente attacco, l'esito dei controlli ha portato alla nota schermata `403:Forbidden`.

5.1.1.2 Protocollo di attestazione

Per testare la sicurezza del protocollo di attestazione si sono analizzati i seguenti casi:

- *Replay Attack*, un attaccante invia valori corretti di misure, firmate mediante la funzione `quote` del TPM del sistema attaccato, ma effettuate in un momento precedente, ad esempio, quando il sistema non era ancora corrotto;
- *Mascheramento*, un attaccante invia valori di un sistema valido per mascherarne uno corrotto;
- *Tampering*, un attaccante manomette la lista di misure o la funzione `TPM_Quote`;
- *Framework VIMS ostile*, il framework di misurazione è stato manomesso e riporta misure non valide;
- *Attacchi Hardware*, l'attaccante ottiene le chiavi AIK del TPM o riesce ad effettuare il reset dei PCR e memorizzare dati su di essi.

Si assume che il Server esegua fedelmente le azioni definite nel protocollo. Il client fa lo stesso, finchè non è corrotto da software ostile.
Sia il client che il server si fidano di una certification authority per la verifica dell'identità. Si assume che l'attacco sia lanciato da un terzo host capace di accedere alla comunicazione sulla rete tra client e server, memorizzare i pacchetti ed effettuare un replay di tali dati. Nei casi analizzati, può assumere il ruolo di client o server ostile all'interno del protocollo di attestazione, e, naturalmente, i dati da esso inviati non possono essere considerati affidabili. Di seguito, viene descritta l'analisi della resistenza del protocollo rispetto agli attacchi.

Replay Attack Il caso dell'attacco di tipo *replay* è mitigato mediante l'utilizzo di un valore casuale *nonce* inviato per ogni messaggio cifrato scambiato tra client e server. L'attaccante non può replicare tali messaggi perchè sono protetti dal *nonce*. Il vincolo cade se l'attaccante è capace di predire il *nonce* del server.

Mascheramento Il server stabilisce un tunnel cifrato con la macchina di assurance mediante SSL, oppure con TLS se il tunnel è una VPN, e protegge i dati da trasmettere mediante crittografia a chiave pubblica certificata da una certification authority. Questo garantisce la sicurezza sull'identità dei partner e l'impossibilità di un attacco *man in the middle*, con il vincolo che la CA sia affidabile.

Tampering La manomissione delle misure richiede la conoscenza delle chiavi con cui sono cifrate, sia per quelle effettuate mediante introspezione, sia per quelle del TPM. La crittografia a chiave pubblica ci garantisce, anche in questo caso, di rilevare ventuali manipolazioni dei dati. Essi, infatti, vengono inviate su tunnel cifrato ed associate alla firma applicata mediante chiavi private del TPM e della Assurance-VM, tale firma viene poi validata dalla CA. Questa assunzione non è applicabile se è avvenuta una manomissione del TPM, facendo cadere le ipotesi di affidabilità sull'intero sistema.

Framework VIMS ostile L'attacco prevede la presenza di un programma software ostile sulla Assurance-VM. Rispetto alle condizioni date nel capitolo 4, cioè che l'Assurance-VM sia misurata all'avvio della macchina mediante TPM, che possieda un insieme minimale e conosciuto di software e che sia raggiungibile dall'esterno solo Remote-S, tale possibilità è considerabile come mitigata.

Attacco hardware al TPM Un attaccante che possa corrompere la logica del TPM ed accedere in scrittura ai registri e/o leggere la parte privata della coppia di chiavi AIK, può corrompere qualsiasi messaggio inviato verso il server, facendo cadere tutte le assunzioni di affidabilità fatte. In questo caso, il server non potrebbe capire se il client è integro o no.

L'analisi del protocollo effettuata ne convalida la sua sicurezza grazie all'utilizzo di strumenti come la crittografia a chiave pubblica e ad il concetto di *root of trust* offerto dal TPM. Un sistema client corrotto non può quindi essere attestato a meno che una o più tra le seguenti condizioni sia vera:

- E' possibile contraffare i certificati;
- Sono possibili attacchi hardware mediante DMA o sul TPM;
- Il TPM è stato manomesso e le chiavi private sono state scoperte;
- E' possibile predire il *nonce* del server.

Concludiamo quindi che il protocollo è sicuro se le seguenti assunzioni sono vere:

- CRTM(Bios) e TPM sono trusted;
- il TPM è conforme alle specifiche TCG;
- VIMS non è manomesso;
- il comportamento di un processo non è modificato mediante DMA.

5.1.2 Performance

I test di performance sono stati effettuati per verificare l'efficienza di VIMS, valutata in termini di overhead introdotto nel sistema.
I test sono divisi in due classi:

- Implementazione dei controlli applicati a Client e Server;
- Tempi di risposta dei servizi di Remote-S.

Per valutare le prestazioni dei controlli di introspezione è stato utilizzato `IOZone` [42], un benchmark che misura le prestazioni di un sistema mediante la valutazione delle prestazioni delle richieste di scrittura e lettura I/O. Per le performance di rete è stato usato il tool `httperf` [52]. Entrambi sono eseguiti in diverse batterie di test e in maniera concorrente ai controlli.

5.1.2.1 Test di performance dei controlli

Le macchine obiettivo dei test sono state la macchina virtuale Client-VM ed il Remote-S e le macchine fisiche sono state collegate su rete LAN con banda 100 Mbps.
Il primo test è stato effettuato per valutare l'impatto dei controlli di introspezione sulla normale

esecuzione del sistema client, il secondo, per valutare l'overhead dei controlli sul tempo di servizio delle richieste di operazioni di lettura e scrittura sul server. Per entrambi i test le prestazioni sono rilevate rispetto al partner remoto ed i valori dei grafici sono espressi in kb/s. Il periodo che intercorre tra le invocazioni della libreria, per entrambi i test, è stato variato tra i 2 ed i 60 secondi, ed è stato confrontato allo stesso benchmark con i controlli disabilitati.

Nella figura 5.1 è mostrato l'overhead delle diverse batterie di test più significative al fine dell'analisi.

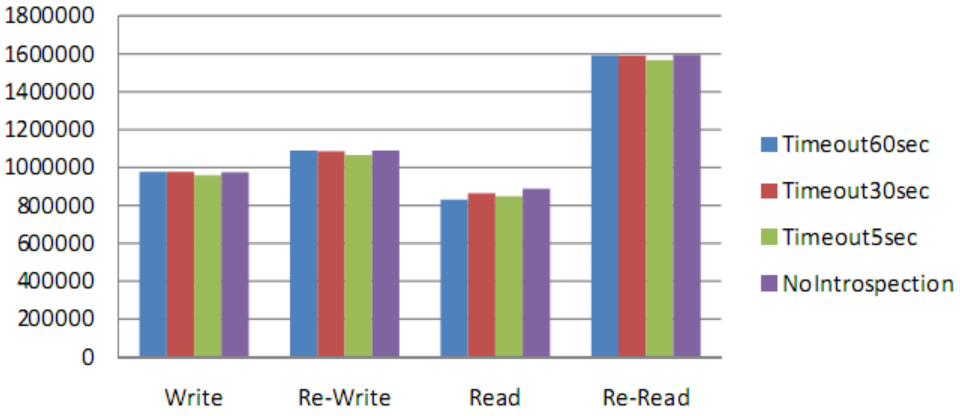


Figura 5.1: Test overhead sul client.

Dal grafico si evince chiaramente che l'impatto dell'overhead indotto dai controlli rispetto alle prestazioni rilevate durante l'esecuzione normale della Client-VM è minimo. Questo risultato mostra due dati significativi, il primo riguarda l'esecuzione, in cui si nota che i controlli vengono eseguiti in parallelo alla normale esecuzione. Quindi, da remoto, l'overhead non è rilevabile. Il secondo, riguarda invece le misure di introspezione che sono dell'ordine di grandezza dei μsec [74] mentre l'overhead di rete è, in generale, dell'ordine dei *msec* e quindi le misure non possono cambiare tale ordine.

Il secondo test è stato effettuato invece sul Remote-S virtualizzato su cui sono applicati i controlli. E' stato calcolato il tempo di risposta rilevato su 3 client collegati al Remote-S, rispetto ad operazioni di lettura, scrittura, rilettura, riscrittura. Il risultato è mostrato in fig.5.2.

In questo caso, il Remote-S ha un calo di prestazioni proporzionale alla frequenza dei controlli ed inversamente proporzionale al valore del timeout. Con un valore di timeout pari a 5 secondi, l'overhead è del 25%.

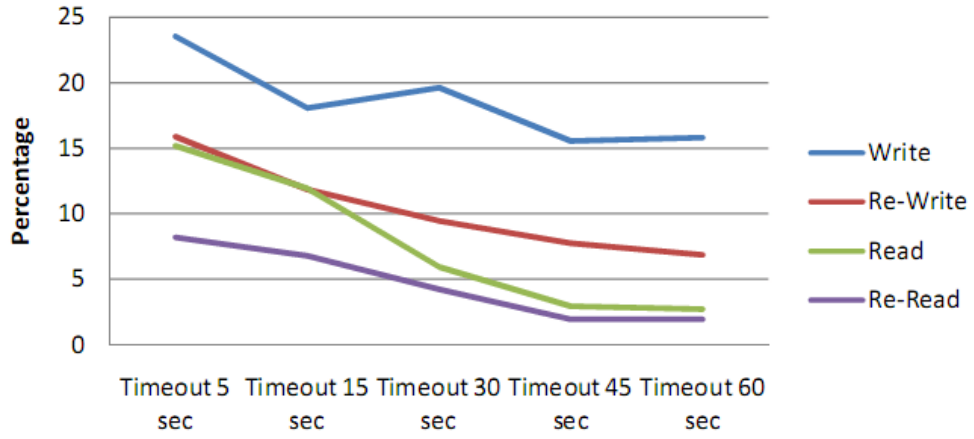


Figura 5.2: Test overhead sul server.

Ovviamente anche Xen introduce un overhead, se pure minimo, ma esso va preso in considerazione solo quando le misure siano confrontate con controlli effettuati mediante device hardware.
Per quanto riguarda l'overhead del protocollo di attestazione remota, la parte misurabile delle prestazioni riguarda la richiesta di avvio connessione tra Remote-S e Client-VM, che, rispetto al funzionamento complessivo dei componenti, applica un meccanismo di comunicazione di tipo *domanda-risposta*.

Per completezza, e per poter effettuare un confronto delle prestazioni, si è scelto di misurare le prestazioni tenendo conto dei tempi di risposta del server nel caso di una richiesta di connessione. L'overhead è generato dalle 4 fasi della comunicazione iniziale del protocollo come mostrato in figura 5.3.

Le misure sulla fase iniziale del protocollo sono state valutate in diversi test, al variare del livello di assurance scelto per i controlli. In figura 5.4 è mostrata la differenza di prestazioni nel tempo di risposta calcolata durante l'handshake.
Questi test evidenziano ulteriormente che, nella valutazione delle prestazioni del nostro framework, il ritardo di rete è preponderante. In questo caso, ad esempio, si può notare che al variare dell'insieme dei controlli effettuati dalla Assurance-VM, dall'insieme più severo a quello minimo, l'overhead subisce solo piccole variazioni.

Sono state inoltre effettuate diverse batterie di stress test verso il webserver *SSL* minimale che è abilitato a ricevere richieste sulla porta 443 della macchina server.
Il test è stato effettuato per valutare se esiste, e quanto influisce sul funzionamento del webserver, l'overhead generato dalle funzioni di libreria, nel caso che esso sia realizzato mediante

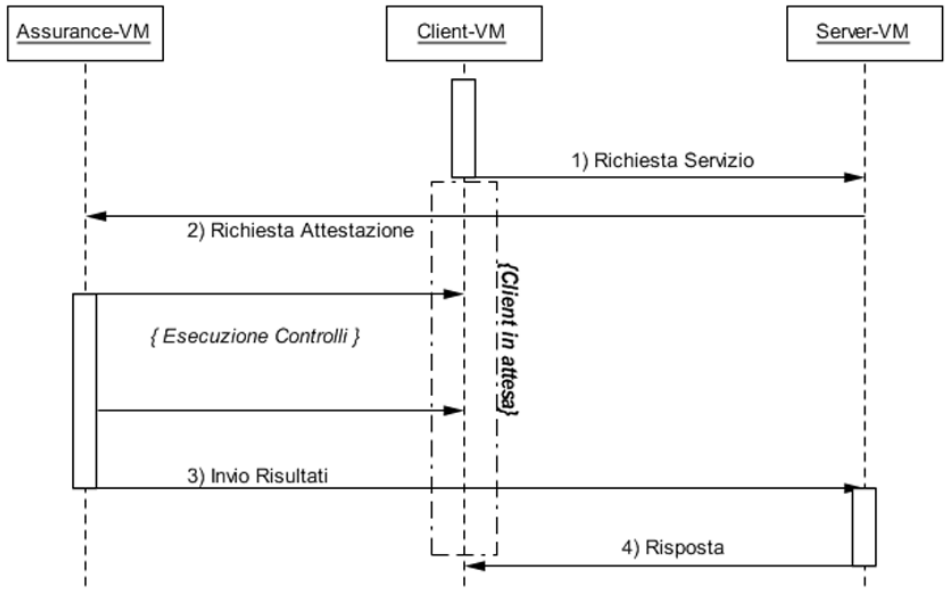


Figura 5.3: Fasi dell'handshake del protocollo.

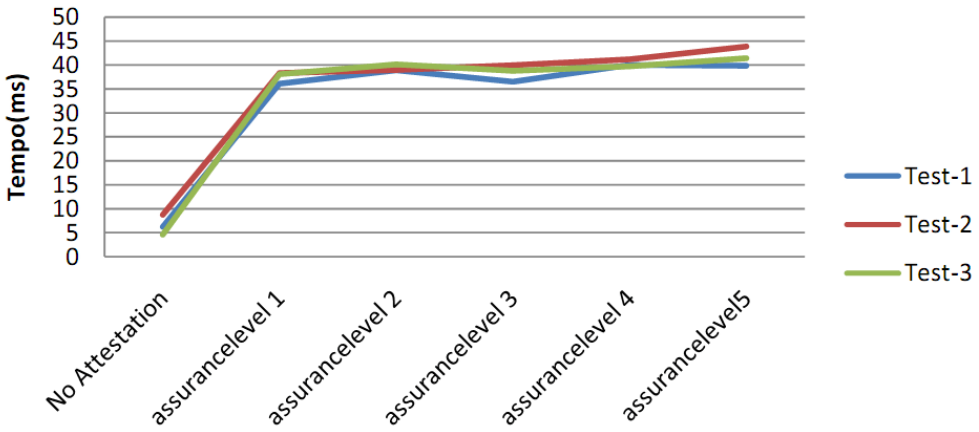


Figura 5.4: Overhead dell'handshake del protocollo.

virtualizzazione.
Per realizzare lo stress test è stato utilizzato il tool `httperf`[52] configurato con i seguenti parametri:

- numero massimo di connessioni = 5000;
- numero massimo di connessioni al secondo = 200;

- numero di richieste http per connessione = 20;

Una volta eseguito `httperf` dalla Client-VM, si è cercato di valutare, relativamente alla implementazione descritta, quanto carico, cioè quanto stress fosse capace di sostenere il WebServer con l'applicazione delle misure o senza. I risultati sono mostrati nei grafici seguenti in fig.5.5 ed in fig.5.6.

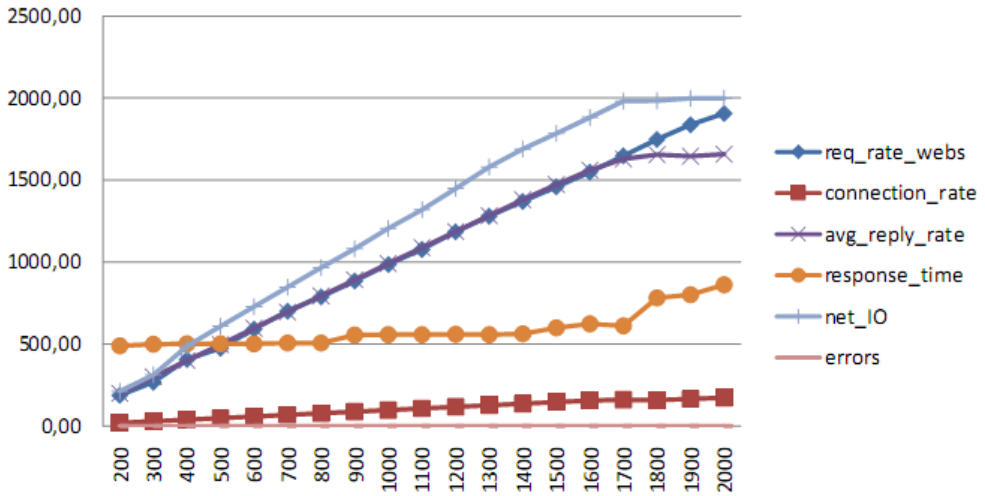


Figura 5.5: Performance del webserver con controlli.

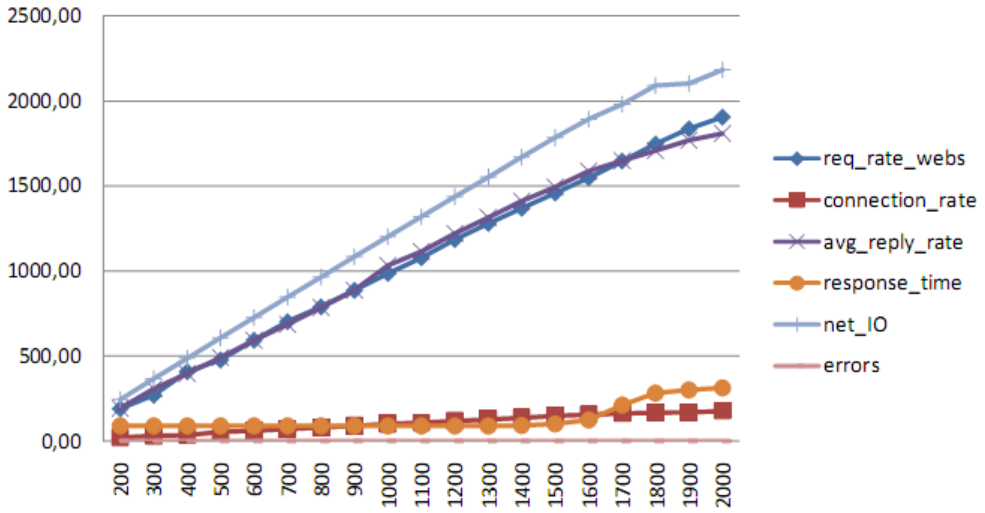


Figura 5.6: Performance del webserver senza controlli.

I risultati evidenziano come, con un tasso di 1700 richieste/sec, il webserver subisce un crollo di prestazioni, non riuscendo più a servire il client che esegue `httperf`.

Il test mostra una inefficienza sostanziale del semplice webserver utilizzato per eseguire i test. Esso, infatti, non utilizza alcun tipo di ottimizzazione nella gestione di multi richieste come quelle effettuate nello stress-test. Per poter estendere la valutazione di questo tipo di controlli, una soluzione possibile è quella di adottare webservers che ottimizzano le risorse disponibili, come i noti Tomcat webserver[3] od Apache [2], tale confronto non è trattato in questo lavoro.

5.2 Conclusioni

Una gestione sicura di informazioni critiche richiede un continuo monitoraggio dell'integrità dei sistemi che trattano l'accesso a tali informazioni. Di conseguenza, è necessario che questi sistemi siano altamente affidabili e robusti rispetto ad eventuali attacchi, come accessi remoti non autorizzati o infezioni da parte di virus o malware eseguito su di essi.
Come detto in precedenza, il contesto in cui è ultimamente urgente e necessaria una garanzia di affidabilità è quello delle infrastrutture critiche, proprio per il grado di criticità dei dati e delle risorse utilizzate. Per questo, a partire dall'analisi fatta in questo lavoro, si è scelto di realizzare una piattaforma software delegata all'applicazione di controlli di integrità su un sistema client, per poter verificare che esso soddisfi determinati standard di sicurezza e con il vincolo che anche tale piattaforma debba essere integra.
A partire da questo concetto, per la realizzazione dell'architettura e la dimostrazione delle proprietà di affidabilità dei componenti, è stato formalizzato un modello matematico che definisce rigorosamente le relazioni di affidabilità (*trust*) tra i componenti dimostrando formalmente come possa crearsi una *chain of trust* tra di essi.
Per illustrare una possibile implementazione del modello teorico è stato descritto Virtual Machine Integrity Measurement System(VIMS), un insieme di architetture che permettono ad un sistema connesso in rete, di ottenere una garanzia di affidabilità circa l'integrità di un partner remoto prima di eventuali interazioni. Come abbiamo visto, un esempio può essere la richiesta di pagine web o la richiesta di accesso ad una rete VPN. A questo scopo, VIMS, sfrutta i concetti di virtualizzazione ed introspezione di macchine virtuali per ottenere una verifica dell'integrità dei componenti software che il client remoto esegue, ed un protocollo, per attestare la loro validità al server di accesso alle risorse.
L'adozione di queste tecnologie, permette al sistema client di eseguire una macchina virtuale nascosta, che applica i controlli di integrità mediante l'introspezione in maniera trasparente alla macchina che richiede l'accesso remoto.
Un aspetto originale di VIMS è l'integrazione del concetto di *root of trust* con tecniche sia statiche che dinamiche di verifica dell'integrità. Infatti, rispetto ad altri lavori presenti in letteratura [40] [58] [33] [63], VIMS ha la possibilità di poter effettuare controlli anche a tempo

di esecuzione su processi e strutture dati di un sistema, permettendo di definire insiemi di controlli più ampi e granulari rispetto alle soluzioni basate solo sul TPM. Inoltre, permette anche la verifica dell'integrità dei componenti sottostanti le macchine virtuali. Esso crea quindi un ambiente che fornisce una adeguata affidabilità per la protezione di risorse critiche a tutti i livelli del sistema.
Così, la rilevazione di misure differenti da quelle attese sul sistema monitorato, permette al server remoto di reagire in maniera immediata coerentemente con lo standard di sicurezza di interesse. Ad esempio, può decidere di innalzare il livello di guardia o eseguire azioni più severe come la chiusura della connessione del client.
Il prototipo ha anche messo in evidenza i vantaggi che si ottengono utilizzando le tecniche di verifica di integrità, introspezione e misure TPM, in combinazione con un protocollo di attestazione remota.
Si può dire che l'adozione di VIMS può garantire agli amministratori di rete una adeguata protezione di dati e risorse critiche da sistemi remoti ostili e portatori di rischi quali rootkit, virus o malware. Questo, specialmente nel contesto SCADA, può essere un fattore determinante per la sicurezza.

5.3 Sviluppi futuri

L'architettura di VIMS ha diverse possibilità di estensione che riguardano principalmente due ambiti:

- misure effettuabili;
- politiche da applicare lato server;

Nel primo caso occorre considerare che l'insieme di misure che si possono applicare ai processi dipendono dalla libreria di introspezione e dai suoi limiti. In particolare, la possibilità di valutare il comportamento semantico dei processi è vincolato alla sola verifica, mediante accesso in memoria, delle informazioni presenti nell'area *text* dei processi ed alla verifica delle strutture dati del kernel del sistema operativo e dei suoi moduli.
Nel secondo caso, invece, abbiamo visto come la reazione del Remote-S rispetto ad una segnalazione di compromissione corrisponde alla richiesta di ulteriori controlli, all'innalzamento del livello di guardia, oppure alla chiusura della connessione.

Per quanto riguarda la possibilità di analisi ancora più granulare dell'esecuzione dei processi, una possibilità la fornisce il tool **PsycoTrace** [67], una libreria di introspezione che, effettuando

il *profiling* del codice sorgente del processo da monitorare verifica, durante l'esecuzione, se la sequenza di system calls chiamate dal processo corrisponde a quella dedotta dall'analisi statica del codice sorgente. Questo tipo di tool permette quindi valutare una serie di asserzioni molto specifiche sul comportamento di un insieme di processi, garantendo un livello di affidabilità molto elevato anche al livello utente.
L'architettura potrebbe quindi essere estesa mediante questo tool opportunamente interfacciato al modulo *Introspection Manager*.
Anche dal punto di vista della gestione della connessione sul Remote-S e delle politiche che esso può adottare alla macchina Client-VM, si potrebbero realizzare alcune estensioni.
Le politiche definite in VIMS ed applicate dalla Assurance-VM sono politiche che associano un insieme di controlli ad una frequenza con cui essi devono essere eseguiti. Nel caso del Remote-S, prendendo come spunto precedenti lavori [65] e [44], sarebbe interessante estendere le funzionalità del modulo *Manager delle politiche* per poter scegliere le azioni da effettuare sulla connessione in base ad un insieme di valutazioni, che tengano conto dei diritti dell'utente collegato, delle operazioni richieste e dello stato della macchina da cui è connesso. A partire da questo concetto, si potrebbe poi estendere l'insieme di azioni che Remote-S può effettuare rispetto ai diversi casi possibili, ad esempio si potrebbe interfacciare con un firewall od un IDS, abilitati a monitorare la connessione secondo predefinite regole.
In questo caso il concetto di politica verrebbe anch'esso esteso divenendo più complesso da gestire.

Un'ultima considerazione riguarda l'implementazione dell'architettura. Essa, infatti, per poter essere utilizzata anche in contesti dove il TPM non è presente, potrebbe essere implementata con una interfaccia verso un componente USB usato come *root of trust* per la verifica del VMM e della Assurance-VM. Su di esso, con garanzia di integrità hardware, dovrebbe essere eseguita la libreria che effettua le verifiche. In questo senso, l'architettura sarebbe portabile anche in contesti dove l'utilizzo del TPM potrebbe portare a problemi di privacy.

Bibliografia

[1] AMD's Virtualization Solutions. http://enterprise.amd.com/us-en/Solutions/Consolidation/virtualization.aspx.

[2] The Apache Software Foundation. http://www.apache.org/.

[3] The Apache Software Foundation - Tomcat. http://tomcat.apache.org/.

[4] Distributed Network protocol. http://www.dnp.org/About/Default.aspx.

[5] Intel Virtualizazion Technology. http://www.intel.com/technology/computing/vptech/.

[6] The Modbus protocol. http://www.modbus.com/.

[7] The Opc protocol for SCADA. http://www.opcfoundation.org/.

[8] QEMU: open source processor emulator. http://fabrice.bellard.free.fr/qemu/.

[9] Security-Enhanced Linux. http://www.nsa.gov/selinux/.

[10] TrouSerS - The open-source TCG software stack. http://trousers.sourceforge.net/.

[11] Trusted computing group: TCG TPM Specification Version 1.2. http://www.trustedcomputinggroup.org..

[12] VMware Workstation. http://www.vmware.com/products/ws/.

[13] The Xen virtual machine monitor. http://www.cl.cam.ac.uk/Research/SRG/netos/xen/.

[14] XenAccess Library. `http://xenaccess.sourceforge.net/`.

[15] 1164, A. S. Pipeline SCADA Security. Tech. rep., American Petroleum Institute(API), 2004.

[16] 800-82, N. S. Guide to Industrial Control Systems Security. Tech. rep., National Institute of Standards and Technology, Washington, DC, September 2008.

[17] AXELSSON, S. The base-rate fallacy and its implications for the difficulty of intrusion detection. In *CCS '99: Proceedings of the 6th ACM conference on Computer and communications security* (New York, NY, USA, 1999), ACM Press, pp. 1–7.

[18] BAIARDI, F., AND SGANDURRA, D. Building trustworthy intrusion detection through vm introspection. *Information Assurance and Security, 2007. IAS 2007. Third International Symposium on* (Aug. 2007), 209–214.

[19] BAJIKAR, S. Trusted Platform Module (TPM) based Security on Notebook PCs-White Paper. *Mobile Platforms Group, Intel Corporation, June 20* (2002).

[20] BELL, D. E., AND PADULA, L. J. L. Secure computer system: Unified exposition and multics interpretation. Tech. rep., MITRE, Corp., 1976.

[21] BERGER, S., CÁCERES, R., GOLDMAN, K. A., PEREZ, R., SAILER, R., AND VAN DOORN, L. vtpm: virtualizing the trusted platform module. In *USENIX-SS'06: Proceedings of the 15th conference on USENIX Security Symposium* (Berkeley, CA, USA, 2006), USENIX Association, pp. 21–21.

[22] BISHOP, M. *Computer Security: Art and Science*. Addison-Wesley, 2003.

[23] BRICKELL, E., CAMENISCH, J., AND CHEN, L. Direct anonymous attestation. In *Proceedings of 11th ACM Conference on Computer and Communications Security* (2004), ACM Press.

[24] BYRES, E., KARSCH, J., AND CARTER, J. Good practice guide on firewall deployment for SCADA and Process control networks. *National Infrastructure Security Co-ordination Centre* (2005).

[25] BYRES, E., LEVERSAGE, D., AND KUBE, N. Security incidents and trends in the scada and process industries. *Symantec Corp. White Paper* (2007).

[26] CHEN, L., LANDFERMANN, R., LÖHR, H., ROHE, M., SADEGHI, A., AND STÜBLE, C. A protocol for property-based attestation. In *Proceedings of the first ACM workshop on Scalable trusted computing* (2006), ACM New York, NY, USA, pp. 7–16.

[27] CIP-008-1, AND CIP-009-1. Cyber Security Standards. Tech. rep., North American Electric Reliability Council(NERC), 2006.

[28] CLARK, D. D., AND WILSON, D. R. A Comparison of Commercial and Military Computer Security Policies. In *IEEE Symposium on Security and Privacy* (1987), IEEE, p. 184.

[29] D. RICHARD HIPP. SQLite. `http://www.sqlite.org/`.

[30] DEN ENGELSEN, D. Budget File and System Integrity Verification for Windows . Tech. rep., SANS Institute, Washington, DC, January 2004.

[31] DENNING, D. E. An intrusion-detection model. *IEEE Trans. Softw. Eng. 13*, 2 (1987), 222–232.

[32] DRAGOVIC, B., FRASER, K., HAND, S., HARRIS, T., HO, A., PRATT, I., WARFIELD, A., BARHAM, P., AND NEUGEBAUER, R. Xen and the art of virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles* (October 2003).

[33] DUNLAP, G., KING, S., CINAR, S., BASRAI, M., AND CHEN, P. ReVirt: enabling intrusion analysis through virtual-machine logging and replay. *ACM SIGOPS Operating Systems Review 36* (2002), 211–224.

[34] ENGLAND, P., AND LOESER, J. Para-Virtualized TPM Sharing. In *Trusted Computing Challenges and Applications: First International Conference on Trusted Computing and Trust in Information Technologies, Trust 2008 Villach, Austria, March 11-12, 2008 Proceedings* (2008), Springer, p. 119.

[35] FERRAIOLO, D. F., AND KUHN, D. R. Role-based access controls. *15th National Computer Security Conference* (1992), 554–563.

[36] GARFINKEL, T., PFAFF, B., CHOW, J., ROSENBLUM, M., AND BONEH, D. Terra: A virtual machine-based platform for trusted computing. In *Proceedings of the 19th Symposium on Operating System Principles(SOSP 2003)* (October 2003).

[37] GARFINKEL, T., AND ROSENBLUM, M. A virtual machine introspection based architecture for intrusion detection. In *Proc. Network and Distributed Systems Security Symposium* (February 2003).

[38] GRIFFIN, J., JAEGER, T., PEREZ, R., SAILER, R., VAN DOORN, L., AND CACERES, R. Trusted Virtual Domains: Toward secure distributed services. *Proc. of 1st IEEE Workshop on Hot Topics in System Dependability (HotDep)* (2005).

[39] GRIMALDI, R. P. *Discrete and Combinatorial Mathematics: an Applied Introduction 4th ed.* Sams, 1998.

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

[41] INTEL. Trusted Execution Technology. `http://www.intel.com/technology/security`.

[42] IOZONE. Filesystem Benchmark. `http://www.iozone.org/`.

[43] ISA-TR99.00.01-2004. Security Technologies for Manufacturing and Control Systems. Tech. rep., Instrumentation, Systems and Automation Society (ISA), 2004.

[44] JAEGER, T., SAILER, R., AND SHANKAR, U. PRIMA: policy-reduced integrity measurement architecture. In *Proceedings of the eleventh ACM symposium on Access control models and technologies* (2006), ACM New York, NY, USA, pp. 19–28.

[45] JANSEN, B., RAMASAMY, H., AND SCHUNTER, M. Policy enforcement and compliance proofs for Xen virtual machines. *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments* (2008), 101–110.

[46] J.BIBA, K. Integrity considerations for secure computer systems. Tech. rep., MITRE Corp., Bedford, MA, 1976.

[47] KENT, K., AND MELL, P. Guide to Intrusion Detection and Prevention (IDP) systems (DRAFT). *National Institute of Standards and Technology* (2006).

[48] KULKARNI, P., ARNOLD, M., AND HIND, M. Dynamic Compilation: The Benefits of Early Investing. In *ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments* (2007), ACM, p. 119.

[49] LOSCOCCO, P. A., WILSON, P. W., PENDERGRASS, J. A., AND MCDONELL, C. D. Linux kernel integrity measurement using contextual inspection. In *STC '07: Proceedings of the 2007 ACM workshop on Scalable trusted computing* (New York, NY, USA, 2007), ACM, pp. 21–29.

[50] MARIO STRASSER, ET AL. Software-based TPM Emulator for Linux. http://tpm-emulator.berlios.de/.

[51] MAYNOR, D., AND GRAHAM, R. Scada security and terrorism: We're not crying wolf! *Black Hat Federal Briefings and Training* (2006).

[52] MOSBERGER, D., AND JIN, T. httperf: a tool for measuring web server performance. *ACM SIGMETRICS Performance Evaluation Review 26*, 3 (1998), 31–37.

[53] OFFICE, G. A. GAO Technical Assessment,Cybersecurity for Critical Infrastructure Protection. Tech. rep., General Accounting Office(GAO), 2004.

[54] OPENVPN. An Open Source SSL VPN Solution. http://openvpn.net/.

[55] PATTERSON, D., AND HENNESSY, J. *Computer Organization and Design, The Hardware/Software Interface*. Morgan Kaufmann Publishers, 1997.

[56] PEARSON, S. Trusted Computing Platforms, the Next Security Solution. *Beaverton, USA: Trusted Computing Group Administration* (2002).

[57] PHRACK. kad. Handling Interrupt Descriptor Table for fun and profit. *Phrack* (2002).

[58] PORITZ, J., SCHUNTER, M., VAN HERREWEGHEN, E., AND WAIDNER, M. Property attestation: scalable and privacy-friendly security assessment of peer computers. *Research Report RZ3548, IBM Corporation, May* (2004).

[59] PRESIDENT'S CRITICAL INFRASTRUCTURE PROTECTION BOARD AND THE OFFICE OF ENERGY ASSURANCE. 21 steps to improve cyber security of SCADA network. Tech. rep., US Department of Energy, Washington, DC, September 2002.

[60] PROCESSENGINEERING WEB MAGAZINE. Beating the hackers. http://www.processengineering.co.uk/Articles/298007/Beating+the+hackers.htm.

[61] PTACEK, T. H., AND NEWSHAM, T. N. Insertion, evasion, and denial of service: Eluding network intrusion detection. Tech. rep., Secure Networks, Inc., Suite 330, 1201 5th Street S.W, Calgary, Alberta, Canada, T2R-0Y6, 1998.

[62] RILEY, S. Mandatory integrity control. http://blogs.technet.com/steriley/archive/2006/07/21/442870.aspx.

[63] SADEGHI, A.-R., AND STÜBLE, C. Property-based attestation for computing platforms: caring about properties, not mechanisms. In *NSPW '04: Proceedings of the 2004 workshop on New security paradigms* (New York, NY, USA, 2004), ACM, pp. 67–77.

[64] SAILER, R., JAEGER, T., VALDEZ, E., CACERES, R., PEREZ, R., BERGER, S., GRIFFIN, J. L., AND VAN DOORN, L. Building a MAC-based security architecture for the xen open-source hypervisor. In *ACSAC '05: Proceedings of the 21st Annual Computer Security Applications Conference* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 276–285.

[65] SAILER, R., VALDEZ, E., JAEGER, T., PEREZ, R., VAN DOORN, L., GRIFFIN, J. L., AND BERGER, S. sHype: A secure hypervisor approach to trusted virtualized systems. IBM Research Report, 2005.

[66] SAILER, R., ZHANG, X., AND JAEGER, T. Design and implementation of a TCG-based integrity measurement architecture. *Proceedings of the 13th conference on USENIX Security Symposium-Volume 13 table of contents* (2004), 16–16.

[67] SGANDURRA, D., BAIARDI, F., MAGGIARI, D., AND TAMBERI, F. Transparent Process Monitoring in a Virtual Environment. In *Proceedings of the Third International Workshop on Views On Designing Complex Architectures (VODCA 2008), Bertinoro* (2008), To appear in ENTCS, Elsevier ScienceDirect.

[68] SINGH, A. An introduction to virtualization. `http://www.kernelthread.com/publications/virtualization/`.

[69] SITES, R., AND AL. Binary Translation. *Communications of the ACM* (1993), 36–50.

[70] SOURCEFORGE.NET. Trusted Boot. `http://sourceforge.net/projects/tboot`.

[71] SPARKS, S., AND BUTLER, J. Shadow Walker: raising the bar for rootkit detection. `hwww.blackhat.com/presentations/bh-jp-05/bh-jp-05-sparks-butler.pdf`.

[72] STALLINGS, W. *Cryptography and Network Security, Third Edition*. Prentice Hall, 1999.

[73] STUMPF, F., ECKERT, C., AND BALFE, S. Towards secure e-commerce based on virtualization and attestation techniques. In *ARES '08: Proceedings of the 2008 Third International Conference on Availability, Reliability and Security* (Washington, DC, USA, 2008), IEEE Computer Society, pp. 376–382.

[74] TAMBERI, F., MAGGIARI, D., SGANDURRA, D., AND BAIARDI, F. Semantics-Driven Introspection in a Virtual Environment. In *Proceedings of the Fourth International Conference on Information Assurance and Security, (IAS 2008)* (2008), pp. 299–302.

[75] TIEGHI, E. M. *Introduzione alla protezione di reti e sistemi di controllo e automazione*. Clusit, 2007.

[76] TPM/J. Java-based API for the Trusted Platform Module (TPM). `http://projects.csail.mit.edu/tc/tpmj/`.

[77] UNITED STATES DEPARTMENT OF DEFENSE. Trusted computer system evaluation criteria. *DoD Standard 5200.28-STD* (1985).

[78] ZIEGLER, R. *Linux firewalls (2nd Edition)*. Sams, 2001.

Ringraziamenti

La pagina dei ringraziamenti è sempre quella più complessa da scrivere. Infatti mentre scrivevo questa pagina cercavo di pensare chi ringraziare, come ringraziare, come organizzare tutto. E meditando sul numero immenso di persone che avrei voluto ringraziare per un attimo, un momento passato insieme, un qualcosa di insignificante ma che è rimasto indelebile in me, mi sono reso conto di quanto l'esperienza fatta in questi anni fosse stata qualcosa di indescrivibile, fatta di tanti incontri eccezionali che mi hanno cambiato radicalmente in tutto. La fine di un percorso così (ho sentito dire di un'*era*...addirittura!), porta con se un insieme confuso di timori ed anche di grandi attese, con tanti interrogativi e tante speranze, come andrà? Che ne sarà di me e dei miei amici? San Gregorio da Nissa diceva *[...] si va nella vita di inizio in inizio attraverso inizi che non hanno mai fine[...]*, e penso che in questo momento, dopo tanti anni, mi sento proprio in un nuovo inizio, nuovamente in gioco, teso alla scoperta del destino che mi è riservato.

Ma andiamo con ordine.

Il primo pensiero va alla mia famiglia, che con grandi sacrifici ha fatto il tifo per me in questi anni. Pensando a quel giorno d'agosto di sette anni fa, dove se avessi potuto scegliere sarei rimasto a Reggio, capisco che è una grande grazia aver ricevuto un tale tesoro.

Vorrei esprimere la mia gratitudine al prof. Fabrizio Baiardi, per la stima e la fiducia che mi ha dimostrato durante il periodo di tesi e per la sua pazienza e precisione, che ha contribuito alla mia formazione accademica e scientifica ed alla realizzazione di questo lavoro.

Inoltre vorrei ringraziare Daniele Sgandurra, grande amico e *sostegno scientifico* nei momenti più importanti del periodo di tesi, Salvo Ferraro, un amico di grande spessore e compagno di ventura dell'ultimo anno universitario, e tutti gli *amici di facoltà* che hanno condiviso con me l'esperienza di rappresentanza studentesca, l'esperienza dell'associazione @System e le lezioni ed esami (scusate non basterebbero 2 pagine per ringraziarvi tutti!).

Ho sempre pensato che un'elenco smisurato di persone amiche fa sempre piacere farlo oppure

leggerlo, vai sempre a vedere se ci sei o no, tra gli amici, e, scorrendo di pagina in pagina, ritrovarti lì, tra le righe, inconsciamente ti fa sempre dire:Ecco! Sono un amico!. Durante una laurea mi piace osservare questa dinamica, tutti vanno a leggere quelle pagine (e non la tesi!) alla ricerca del proprio nome.
Per cui, così iniziate a ritrovarvi, partiamo con un ringraziamento di cuore a tutti i miei amici del Movimento, compagni fedeli che mi hanno fatto e mi fanno sempre scoprire quali sono i talenti che Dio mi ha donato, e me li fanno giocare in ogni istante. Senza di voi non avrei mai potuto vivere la mia vita all'altezza dei miei desideri in questi anni.
In particolare volevo ringraziare alcuni che, come Virgilio e Beatrice per Dante, sono stati una guida inesauribile per carisma e tenacia, anche nei miei momenti più abissali e critici, quelle persone che con una parola od uno sguardo ti comprendono più di te (incredibile!).
Per questo ringrazio Ciccio, Alessandro *Rom* G. e Luca S., tre rocce salde per la mia vita con tre temperamenti diversi, dei grandi amici che mi guardano avendo a cuore il mio destino.
Poi vorrei ringraziare Vincenzo M., Juan, Samuele, Maria Laura e Patrizio, esempi diversi e concreti di umanità e di affezione all'Ideale, per me dei *cardini* fondamentali. Quelli che....l'appartamento *Principi*, di cui sono l'ultimo "sopravvissuto" tra gli studenti(era ora ti levassi di...!), come dire "Il primo **Incontro** non si scorda mai!", e Quelli che...l'appartamento *Patriarchi*, compagni di mille avventure e grande sostegno, una continua educazione e richiamo a vivere la vita da cristiano, secondo tutti i suoi fattori. Andrea M., Emanuele, Alessandro P., Mauro, Francesca B., Matteo, Virginia, Chiara N., Giulio, Federico, Peppe B., Michele N., Ziggy, Cinzia e Rocco, una salda e reale amicizia e Valentina la *cugggina*, se sono qui è anche per causa sua ;-). Margherita ed Antonello, due incredibili scoperte che continuamente riaprono lo sguardo che ho sulla mia vita.
Poi vorrei ringraziare Marco, Federica, la *fuffa* Valentina, Nice e Paolo, una seconda famiglia, sorprendente ed inaspettata.

No, non mi sono dimenticato di te, lo sai che all'ultimo ci sono sempre i fuochi d'artificio migliori no? La tua presenza, la tua gratuità e la tua costanza mi risvegliano sempre la domanda sulla misteriosità della vita: ma con uno così chi te la fa fare? Eppure ci sei, Sara, sempre e più di prima, e questo mi basta per ringraziarti, perchè questa presenza sa di eternità.
Infine vorrei ringraziare Don Luigi Giussani, mediante il quale Cristo misteriosamente mi ha acciuffato, e mi ha messo in questa compagnia che è il movimento di Comunione e Liberazione, senza di Lui tutto quello che ho scritto non sarebbe mai accaduto.

VSSVPM

D.