POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Supervisione dello stato di sicurezza di una rete softwarizzata

Relatori
prof. Antonio Lioy
dott. Marco De Benedictis

Candidato
Giovanni Trivigno

Anno accademico 2017-2018

Sommario

Lo scenario in cui si svolge la tesi proposta è il Cloud Computing che negli ultimi anni ha mutato il modo in cui i servizi IT vengono progettati e impiegati all'interno della rete. Il cambiamento è stato possibile mediante il concetto di virtualizzazione, che permette un'elevata scalabilità e flessibilità ai servizi di rete che si vanno ad utilizzare. Per garantire questi aspetti la virtualizzazione consente di condividere risorse hardware presenti all'interno di un dato host fisico tra più ambienti virtuali isolati e indipendenti tra loro. Tramite il concetto di virtualizzazione le attuali reti di comunicazione tradizionali vengono sostituite da funzioni di rete virtualizzate che possono essere in esecuzione all'interno di un unico host fisico. La tecnologia a cui la tesi proposta fa riferimento prende il nome di Network Function Virtualization (NFV) che fornisce soluzioni per migliorare le attuali reti di comunicazione attraverso la virtualizzazione. Lo scenario offerto dalla tecnologia NFV porta a sostanziali miglioramenti nell'ottimizzazione delle risorse in uso all'interno della rete. La tecnologia NFV offre la possibilità di istanziare servizi di rete virtuali tramite delle funzioni di rete chiamate Virtual Network Function (VNF).

L'utilizzo dei servizi IT negli ultimi anni è cresciuto notevolmente aumentando di conseguenza i possibili tipi di attacchi informatici all'interno della rete. Al fine di gestire questa situazione la tecnologia NFV fornisce delle funzioni di sicurezza di rete chiamate Virtual Network Security Function (vNSF). Le vNSF istanziate all'interno dell'infrastruttura NFV permettono di tutelare l'intera infrastruttura da vari tipi di attacchi informatici, sono però esposte al mondo quindi possono subire diversi tipo di attacchi, in particolare un attaccante potrebbe manipolare un nodo per produrre un comportamento diverso da quello atteso. Al fine di evitare quest'ultima problematica occorre che venga introdotto all'interno dell'infrastruttura NFV un meccanismo che permetta di verificare lo stato di integrità di ogni funzione di rete presente. L'insieme delle tecnologie che ci permettono di determinare il livello di fiducia di nodi di rete tramite un analisi della sua integrità afferiscono al Trusted Computing.

Al fine di garantire l'integrità del sistema è necessario fidarsi di un componente hardware affidabile, che nel contesto di questa tesi risulta essere il Trusted Platform Module (TPM). Il processo di verifica di integrità dei sistemi che utilizzano il TPM prende il nome di Remote Attestation (RA). Nel contesto di questa tesi si è lavorato in uno scenario rivolto alla virtualizzazione sfruttando il concetto di "virtualizzazione leggera". A differenza del meccanismo di virtualizzazione standard dove ogni macchina virtualizzata viene astratta da un sistema intermedio di virtualizzazione chiamato hypervisor, nella virtualizzazione leggera i nodi virtualizzati condividono parte del sistema operativo dell'host. In questa tesi il sistema che permette di svolgere questo tipo di virtualizzazione è Docker. Docker permette di eseguire funzioni di rete all'interno di container Docker isolati tra loro, la sua grande potenzialità è che si tratta di un modulo installato all'interno dell'host fisico e per tale ragione quest'ultimo è a conoscenza di tutte le componenti software in esecuzione all'interno dei container. Il processo di RA consiste nello scambio di informazioni tra un host da attestare e il componente che si occupa di verificare lo stato di fiducia del nodo. Le informazioni scambiate tra

queste due entità sono rappresentate dal risultato dell'output, chiamato digest, di una funzione di hash applicata alle componenti software eseguite all'interno delle VNF dunque all'interno dei container Docker. Il componente che permette di ottenere l'elenco dei digest associate alle componenti software è l'Integrity Measurement Architecture (IMA) presente nel modulo kernel Linux.

La tesi proposta ha come obiettivo quello di tentare di estendere il processo di RA per determinare l'integrità dei nodi presenti all'interno dell'infrastruttura NFV. ETSI NFV ha definito, tramite un elevato numero di standard, tutti i componenti necessari al mondo NFV per funzionare e per gestire il ciclo di vita della funzioni di rete che possono essere istanziate all'interno dell'infrastruttura NFV. ETSI NFV ha anche proposto come potrebbe avvenire il processo di RA delle VNF ed in particolare ha definito che per verificare l'integrità di una VNF è necessario inserire all'interno dell'architettura NFV un modulo chiamato "Trust Manager". Questo modulo permette di verificare l'integrità delle VNF in esecuzione in un dato istante di tempo all'interno dell'infrastruttura NFV. Allo stato attuale ETSI fornisce solo una definizione dell'utilizzo di questa entità, ma non è presente nessun componente che permette di svolgere tale processo. Mediante lo sviluppo di questa tesi vogliamo tentare di compare questa mancanza implementando un modulo che si occupi di attestare l'integrità delle VNF e che sia anche in grado di comunicare con gli altri componenti presenti nell'architettura NFV, il modulo in esame prende il nome di Trust Monitor. La tesi ha come obiettivo quello di realizzare un modulo più generico possibile, che possa adattarsi a diversi framework di attestazione. Al fine di riadattarlo occorre realizzare un driver di attestazione specifico per ogni meccanismo di attestazione.

Il framework di attestazione utilizzato in questa tesi per eseguire il processo di RA è Open Attestation (OAT). Si è utilizzato OAT in modo da partire da un sistema già funzionante ed esteso sviluppato nell'ambito del progetto europeo SECURED realizzato dal gruppo TORSEC. Il sistema durante il progetto è stato esteso per permettere di eseguire il processo di RA anche per i nodi virtuali rappresentati da container Docker. Il Trust Monitor per svolgere il processo di attestazione al fine di determinare lo stato di fiducia delle VNF deve necessariamente interfacciarsi con le altre entità presenti all'interno dell'architettura NFV. I digest relative alle componenti software devono essere valutati durante il processo di attestazione in modo da determinare se il nodo contente le VNF risulta essere fidato o meno. Per fidato si intende che le componenti in esecuzione all'interno del nodo e all'interno delle VNF sono contenute all'interno di un database di riferimento chiamato Whitelist Database.

In conclusione con la realizzazione di questa tesi è stato possibile eseguire il processo di verifica di integrità di ogni funzione di rete e di ogni nodo fisico presente all'interno dell'infrastruttura NFV al fine di determinare possibili manomissioni. Il modulo sviluppato in questa tesi conduce a miglioramenti in termini di rilevamento di attacchi informatici all'interno del Cloud Computing, portando dunque al rilevamento di manomissioni di funzioni di rete virtuali.

Indice

Capitolo 1

Introduzione

La protezione contro la criminalità informatica è diventata centrale per tutta l'economia online. Con l'aumentare della criminalità informatica e con l'utilizzo massivo di servizi IT, gli attacchi si evolvono e diventano più complessi, questo costringe il mondo economico ad aggiornare continuamente le tecniche di cybersecurity. Si tratta di un processo costoso che causa alle aziende un esborso economico ed un uso di risorse umane molto elevato. Questo perché molte aziende fanno uso di reti tradizionali, nelle quali ogni funzione di sicurezza è implementata su un host fisico, questo porta la rete ad essere poco flessibile. Per contrastare lo scenario appena descritto e evitare perdite di risorse economiche, occorre indirizzare il mondo economico ad utilizzare un contesto basato sul cloud. Negli ultimi anni, il cloud computing ha sicuramente rivoluzionato il modo in cui i servizi IT vengono progettati e impiegati all'interno della rete. Il cloud computing ha portato un miglioramento in termini di flessibilità e di disponibilità del servizio di rete che si va ad utilizzare. Mediante il suo utilizzo abbiamo anche un significativo abbattimento dei costi, questo perché sulla stessa macchina possono essere eseguite più funzionalità di rete nello stesso istante. I vari servizi di rete implementati condivideranno l'hardware presente nell'host fisico sul quale sono istanziati. La tecnologia Network Function Virtualization (NFV) fornisce soluzioni specifiche per combattere questo problema. NFV introduce un sostanziale cambio di paradigma nel modo in cui vengono realizzate le reti di telecomunicazioni. Questo nuovo scenario porta a sostanziali miglioramenti riguardanti l'ottimizzazione delle risorse in uso all'interno della rete. La grande potenzialità è che in uno scenario del genere su una stessa macchina possono essere istanziate diverse funzionalità di rete, dunque Virtual Network Function (VNF) con scopi differenti. Le VNF portano anche a gestire in modo più flessibile la rete stessa, andando a inserire/rimuovere/spostare all'occorrenza un nodo, più precisamente una funzione virtuale all'interno della rete.

Per gestire la sicurezza, tale paradigma introduce il concetto di Virtual Network Security Function (vNSF) per virtualizzare funzioni di sicurezza di rete. L'idea principale consiste nella costruzione di Servizi Security as a Service (Secaas) [18] avanzati che sfruttano le tecnologie NFV.

La presenza di vNSF consente di avere delle funzioni di sicurezza flessibili, che permettono all'infrastruttura NFV di proteggersi da determinati tipi di attacchi. Le vNSF sono esposte al mondo quindi possono subire diversi tipi di attacchi, in particolare è possibile manipolare un nodo per produrre un comportamento diverso da quello atteso. Per tale ragione occorre introdurre all'interno dell'infrastruttura NFV meccanismi che permettono di verificare lo stato di integrità di ogni funzione di rete presente. La tecnologia basata su questo tipo di controllo viene chiamata Trusted Computing [2]. L'idea della tesi è quella di andare a colmare questo gap, andando a realizzare un modulo standalone che possa fornire garanzie di sicurezza all'infrastruttura NFV e allo stesso tempo sia in grado di comunicare con gli altri componenti NFV, quali Orchestatore, NFV Store e DARE. Il modulo svolge quindi un meccanismo di attestazione mirato a reti virtualizzate.

Per garantire l'integrità del sistema, è necessario fidarsi di un componente hardware affidabile, che nel nostro contesto risulta essere il Trusted Platform Module (TPM) [5], le cui specifiche sono definite dal Trusted Computing Group (TCG) [3]. Il processo di verifica d'integrità dei sistemi che utilizzano il TPM prende il nome di Remote Attestation (RA). Il processo di RA è rappresentato dell'invio di dati tra un host da attestare tipicamente definito come *Requestor*, rappresenta il

componente che richiede di essere attestato e il componente che verifica l'attestazione definito come *Verifier*. Le specifiche del TCG non definiscono come debbano essere le informazioni scambiate tra *Requestor* e *Verifier*, ma parlano in generale di "misure" in modo da adattare lo scenario della RA a diversi tipi di piattaforme. La RA richiede che il *Verifier* sia in grado di verificare l'integrità sulla base di un elenco di misure inviatogli dal *Requestor* firmato digitalmente attraverso l'hardware TPM. In questo modo siamo certi che, se il file contenente l'elenco delle misure venga alterato da terze parti durante il transito nella rete, tali alterazioni vengano rilevate tramite il TPM. Nell'ambito della tesi si farà riferimento alla *binary attestation*. Le misure nel nostro contesto non sono altro che il risultato dell'output, chiamato digest, relativo a una funzione di hash. Una funzione di hash è una funzione che mappa una stringa di lunghezza variabile, in una stringa a lunghezza fissa. La *binary attestation* valuta l'integrità dei componenti dell'infrastruttura da attestare a partire dal digest, più precisamente il risultato della funzione di hash SHA-1 relativo ai componenti software eseguiti all'interno delle varie VNF. Il componente che ci permette di ottenere l'elenco dei digest è l'Integrity Measurement Architecture (IMA) [1] presente all'interno del modulo kernel di Linux, tale modulo di default per ottenere l'elenco dei digest, misura tutti i programmi eseguibili, i file eseguibili mappati in memoria e tutti i file aperti per la lettura. Il processo di verifica richiede che il *Verifier* conservi un database aggiornato di componenti software ritenuti accettabili per valutare l'integrità di un nodo.

Lo scopo generale della mia tesi consiste nello sviluppo di un modulo di attestazione chiamato Trust Monitor, il cui compito è quello di attestare host fisici e/o i nodi virtuali nell'infrastruttura NFV. Si intende sviluppare questo modulo in quanto, allo stato attuale non risulta essere implementato un framework di attestazione direttamente applicabile allo scenario cloud NFV. Il tipico scenario in cui si svolge l'intero processo della mia tesi, consiste nell'eseguire un processo di attestazione verso un nodo virtuale per valutarne la sua integrità. Questo processo può generare due possibili risultati, il componente può risultare fidato oppure non fidato.

La tesi ha come obiettivo quello di realizzare un modulo più generico possibile, che si possa adattare a diversi framework di attestazione. Per riadattarlo, occorre realizzare un driver specifico per ogni meccanismo di attestazione, questo per far interfacciare in modo corretto il componente realizzato con il framework di attestazione utilizzato. Lo scopo di realizzare un driver specifico risiede nel fatto di non andare a modificare lo scheletro dell'applicativo. Nel contesto della tesi si utilizza un sistema di Remote Attestation basato su Open Attestation (OAT) [21], framework di Intel. Si è utilizzato OAT in modo da partire da un sistema già funzionante sviluppato nell'ambito del progetto europeo SECURED [22] realizzato dal gruppo TORSEC [20]. In principio il sistema di attestazione basato su OAT era mirato a verificare l'integrità esclusivamente del host fisico, ma per espandere il suo funzionamento a un mondo cloud, è stato necessario eseguire un estensione in modo da includere durante il processo di attestazione anche l'attestazione basata su nodi virtuali. Il vincolo su cui OAT è basato è che ogni host fisico che si intente attestare deve essere dotato necessariamente del TPM. I nodi virtuali a cui fa riferimento OAT sono espressi da container Docker [10]. Docker è un progetto open-source che permette di eseguire applicazioni software all'interno di container eseguiti sull'host. Si tratta di una "virtualizzazione leggera", si utilizzano container basati su Docker perché tali container fanno riferimento a un demone in esecuzione sull'host da attestare, tutte le operazioni svolte all'interno di tali nodi virtuali possono essere misurati da IMA. Tutto questo porta dunque a considerare in fase di attestazione di un nodo fisico anche tutti i container Docker istanziati su di esso. Per considerare i container durante il processo di attestazione occorre aggiungere una lista di container rappresentata dall'elenco degli ID dei Docker in esecuzione, in modo da informare OAT che vogliamo includere anche i container nel processo di verifica d'integrità.

Il Trust Monitor per andare a svolgere il processo di attestazione, deve interfacciarsi con i restanti componenti dell'architettura NFV. Questo aspetto risulta essere fondamentale, in modo da recuperare tutte le informazioni riguardanti i nodi virtuali, quali nome della VNF in esecuzione su un dato host e la lista dei digest SHA-1 utilizzati dalla stessa VNF. Tali digest devono essere presi in considerazione durante il processo di attestazione per valutare lo stato d'integrità in cui una VNF si trova. Per recuperare tali informazioni il componente sviluppato comunica tramite un connettore che si interfaccia con Open Source Mano (OSM) [19], un Orchestratore open source il quale ci permette di istanziare cioè mandare in esecuzione una o più VNF su un dato host, più precisamente su un dato Virtualized Infrastructure Manager (VIM). Il VIM è responsabile

di gestire l'infrastruttura NFV e istanziare le VNF, che poi potranno essere attestate dal Trust Monitor.

Nel corso degli altri capitoli, verrà presentato un resoconto dettagliato sullo stato dell'arte attuale relativo all'argomento trattato con una panoramica su tutti gli aspetti di background necessari a comprendere lo scenario in cui vogliamo portare miglioramenti. Inoltre verranno indicati quali strumenti vogliamo andare ad utilizzare per migliorare la sicurezza informatica in un contesto mirato al mondo cloud. Verranno anche presentate le novità introdotte descrivendo nel dettaglio le tecnologie utilizzate per realizzare tale architettura, fornendo anche un'implementazioni della stessa. Infine verranno definiti una serie di risultati che ci permettono di documentare il lavoro svolto.

Capitolo 2

Background

Oggigiorno l'uso del Cloud Computing è aumentato significativamente, perché questa tecnologia garantisce benefici economici alle aziende. In un contesto del genere assume sempre più importanza il concetto che un nodo sia in grado di fidarsi di un altro nodo all'interno della rete. I nodi nel mondo Cloud sono collegati direttamente tra loro, per questo motivo è importante riconoscere se un nodo sia stato manomesso in modo da modificare il suo comportamento originale. Se così fosse, il nodo non dovrebbe più fidarsi dell'altro. La fiducia è rappresentata dalla possibilità di un nodo di affidarsi ad un altro nodo per adempire ai compiti per i quali è stato creato. Per questa ragione la sicurezza informatica all'interno del cloud computing assume un ruolo cardine. In questo capitolo verranno presentati tutti gli argomenti necessari a comprendere l'architettura realizzata nella tesi. Più nel dettaglio all'interno della sezione 2.1 verrà introdotto il concetto del Trusted Computing, del Trusted Platform Module e della Remote Attestation, nella sezione 2.2 verrà descritto nel dettaglio in cosa consiste la Network Function Virtualizzation andando a rappresentare l'architettura e andando a descrivere il caso d'uso "Security as a Service" (SecaaS), che ci permette di introdurre dei componenti utilizzati nel contesto di questa tesi, mentre nella sezione 2.3 sarà definito il concetto di container più precisamente verranno trattati i container basati su Docker.

2.1 Trusted Computing

Il Trusted Computing (TC) [2], è stato introdotto nel 1990 per trattare l'attendibilità di una piattaforma, la Trusted Computing Group (TCG) [3] è l'organizzazione che promuove e realizza documentazioni e tools relativi alla TC. Il TC è l'insieme delle tecniche di progettazione e principi operativi per creare un ambiente di elaborazione di cui l'utente può fidarsi. Se l'ambiente si comporta in maniera attesa esso viene etichettato come "trusted". Il concetto di "trust" è molto complesso da definire la stessa TCG ha definito "trust" dicendo che la fiducia è quell'aspettativa che un dispositivo si comporterà come previsto per uno specifico scopo.

In un contesto relativo a un ambiente Cloud, poter determinare il livello di fiducia di un nodo all'interno della rete è molto importante, questo perché i nodi della rete non sono soltanto collegati tra loro ma vengono esposti anche al mondo esterno, per cui sono soggetti a differenti tipi di attacchi. I nodi hanno misure di sicurezza che gli permettono di avere una protezione del canale di comunicazione tramite protocollo TLS e certificato X.509, ma un attacco all'interno del canale di comunicazione instaurato tra due nodi non è l'unico tipo di problema che si può riscontrare su un nodo. Le piattaforme all'interno della rete possono essere manipolate ad esempio tramite alterazione del software, occorre dimostrare che tali macchine garantiscano integrità ed il loro corretto funzionamento. L'amministratore di rete dovrebbe essere in grado di capire se il nodo si comporta in maniera attesa oppure ha subito un'alterazione. Per tale ragione serve qualcosa che dimostri agli altri nodi della rete che la macchina in esame risulta integra prima di avviare il processo di comunicazione. Con questa affermazione vogliamo dire che il nodo si comporta in maniera attesa e il software utilizzato è conforme alle specifiche.

Il TCG ha pubblicato diverse specifiche che definiscono il concetto di piattaforma fidata [4], cosa più importante, propone un'implementazione ampiamente promossa dal mondo industriale, che si basa su un chip aggiuntivo da inserire all'interno della scheda madre del dispositivo di cui si vuole avere un analisi di integrità, esso prende il nome di Trusted Platform Module (TPM) [5]. Al giorno d'oggi questo chip è presente nella maggior parte dei dispositivi, questo permette di utilizzare la soluzione del TC basata sul TPM su larga scala in tutto il mondo.

Il concetto principale dell'architettura proposta dal TCG prende il nome di "Trusted platform" [4]. Secondo la visione del TCG una piattaforma computazionale affidabile deve fornire almeno tre caratteristiche fondamentali:

- capacità di memorizzazione sicura;
- capacità di produrre misure di integrità;
- capacità di effettuare un reporting delle misure di integrità presenti nel sistema.

La capacità di memorizzazione sicura è un'operazione di base che è fondamentale per avere la garanzia che tutto il sistema sia fidato. È strettamente legato al concetto di memoria protetta: regioni speciali sulla piattaforma in cui è sicuro archiviare e operare su dati sensibili. Le informazioni di integrità e le chiavi private utilizzate per operazioni crittografiche, sono esempi di dati sensibili presenti in questa area di memoria.

L'integrità della piattaforma è definita come l'insieme di metriche che identificano i componenti software, come il Sistema Operativo, le applicazioni e i relativi file di configurazione. Le misure vengono raccolte nell'area di memoria protetta per ottenere e archiviare le metriche delle caratteristiche della piattaforma in modo cumulativo al fine di poter in un secondo momento dichiarare se una piattaforma risulti fidata o meno.

Un dispositivo deve essere in grado di poter riportare il proprio stato di integrità ad una terza parte durante il processo di Remote Attestation (RA), in modo che tale entità possa verificare se il dispositivo risulti essere manipolato da un attaccante. Complessivamente una piattaforma affidabile deve essere capace di misurare la propria integrità, archiviare localmente le misure correlate e riportare questi valori alle entità remote in modo autentico e sicuro.

2.1.1 Trusted Platform Module

Il Trusted Platform Module (TPM) [5], come già introdotto precedentemente è un chip che può essere aggiunto sulla scheda madre di un dispositivo ed è il componente principale di una piattaforma TCG. Per accedere alle funzionalità offerte dal TPM è possibile, via software, utilizzare una serie di comandi predefiniti [6]. Il TPM offre al suo interno differenti funzionalità:

- Random Number Generator (RNG), si tratta di un generatore di numeri casuali utilizzato dall'hardware TPM per la creazione del nonce utilizzato durante il processo di Remote Attestation, per la generazione di chiavi e firme digitali.
- Algoritmo asimmetrico RSA utilizzato per l'operazione di firma digitale e per la crittografia.
- Crittografia simmetrica, utilizzata per crittografare le informazioni di autenticazione e proteggere i dati archiviati all'interno del TPM.
- Key Generation, crea coppie di chiavi RSA e chiavi simmetriche.
- HMAC Engine, il TPM lo utilizza per provare la conoscenza dell'*AuthData* e per garantire che la richiesta in arrivo sia autorizzata e che non sia alterata.
- SHA-1 Engine, rappresenta la funzione SHA-1 utilizzata dall'utente, viene utilizzata durante il processo di misurazione.

Il TPM dispone di una memoria non volatile resistente alla manomissione, questa memoria è principalmente utilizzata per la memorizzazione persistente delle chiavi. Possiamo notare che la zona di memoria protetta copre in pieno il concetto espresso prima relativo alla definizione di piattaforma affidabile. Il TPM esegue calcoli crittografici internamente, componenti hardware e software esterni al TPM non hanno accesso all'esecuzione delle funzioni crittografiche al suo interno.

Le misure di integrità relative alla configurazione dal dispositivo vengono mantenute all'interno di particolari registri chiamati Platform Configuration Registers (PCR). Il contenuto di questi registri viene popolato nel momento in cui avviene il boot del sistema e successivamente può essere modificato soltanto attraverso l'operazione di estensione, più precisamente tramite il comando extend mediante la formula sotto indicata (dove $\mathrm{PCR}_{\mathrm{old}}$ corrisponde al valore di registro precedente all'interno del PCR, $\mathrm{PCR}_{\mathrm{new}}$ rappresenta il nuovo valore del PCR, measured data è la nuova misura e SHA1 si riferisce al Secure Hash Algorithm, algoritmo di hashing crittografico):

$$PCR_{\mathrm{new}} = SHA1(PCR_{\mathrm{old}} \,||\, measured\_data)$$

Ogni TPM ha un set minimo pari a 24 registri PCR, ognuno dei quali è formato da 20 byte. Ogni registro PCR è progettato per contenere un numero illimitato di misurazioni al suo interno, questo viene fatto usando una funzione di hash che permette di ottenere in uscita una stringa di lunghezza fissa pari a 160 bit. Come detto precedentemente ogni PCR può essere soltanto esteso con una nuova misura, per tale ragione il TPM non supporta l'operazione di reset di un PCR. Ciascun PCR contiene la storia di tutte le misure estese presenti all'interno di una piattaforma. Il processo di estensione viene realizzato tramite una funzione di hash, per questo motivo derivare l'elenco dei valori memorizzati in ciascun PCR è un operazione impossibile. Se il TPM è disabilitato o disattivato, il comando che estende il PCR ritorna un valore di PCR pari a tutti zeri.

Per fidarsi delle operazioni di capacità protette, il TCG definisce tre *Root of Trust*, sono componenti pensati per essere considerati attendibili perché il loro comportamento scorretto potrebbe non essere rilevato. Ogni *root of trust* si occupa di realizzare una delle caratteristiche principali che una piattaforma affidabile deve fornire:

- La *root of trust for measurement* (RTM), implementa un motore capace di effettuare misure di integrità affidabili e costituisce anche la radice della catena di fiducia che si instaura in una piattaforma fidata;
- La *root of trust for storage* (RTS), è responsabile di mantenere all'interno di una memoria protetta esterna, le misure d'integrià e le chiavi crittografiche utilizzate dalla piattaforma;
- La *root of trust for reporting* (RTR), è responsabile della segnalazione affidabile a una terza parte delle informazioni di integrità protette dal RTS.

L'RTM può essere implementato in una piccola porzione del BIOS di un sistema quando esso viene avviato, questo perché il BIOS è il primo modulo software che viene attivato all'avvio della macchina, oppure direttamente integrato all'interno dei processori di ultima generazione. L'RTM deve essere attivato per primo quando viene avviata la piattaforma in modo che possa misurare il processo di avvio, questa procedura prende il nome di "trusted boot" o "measured boot". Il set di operazioni eseguite in questa fase costituisce il *Core Root of Trust of Measurement* (CRTM). RTS e RTR possono essere implementati utilizzando il TPM, in grado di eseguire operazioni crittografiche, per mantenere le misure di integrità e per comunicarle a una terza parte. Il TPM non è in grado di implementare l'RTM, perché l'RTM è solitamente dipendente dalla piattaforma in quanto fa riferimento alla prima parte del BIOS, mentre il TPM è un chip installabile all'interno dei dispositivi. Il TPM in combinazione con il CRTM, fornisce una base completa per una piattaforma affidabile perché include in pieno i tre principi che classificano una piattaforma come affidabile.

Il TPM al suo interno può memorizzare una serie di chiavi RSA, tra queste abbiamo due chiavi RSA speciali: l'Endorsement Key (EK) e la Storage Root Key (SRK). L'EK è la chiave che permette di identificare univocamente il chip TPM a livello globale, per questo motivo è unica per ogni TPM. Si tratta di una chiave non migrabile, cioè tale chiave non può essere spostata all'esterno del TPM. Questa chiave viene creata dal produttore del TPM, inoltre la specifica TCG

richiede che sia necessario fornire un certificato per garantire che la chiave appartenga a un dato TPM. La SRK è anch'essa una chiave non migrabile, l'utilizzo di questa chiave è vincolato nel proteggere altre chiavi utilizzate per funzioni crittografiche e che risiedono al di fuori del chip, non viene mai utilizzata per funzioni crittografiche. Associate ad ogni TPM possono esserci una serie di chiavi definite Attestation Identity Key (AIK), le quali sono create dal TPM. Queste chiavi vengono utilizzate per firmare digitalmente un sottoinsieme dei valori contenuti nei PCR insieme a un nonce di 160 bit, vengono utilizzate nel TPM_Quote durante il processo di attestazione. Per certificare una chiave AIK viene utilizzata una terza parte attendibile denominata Privacy Certification Authority (Privacy CA), la quale prima di rilasciare il certificato deve accertarsi che la chiave AIK sia stata creata da un TPM genuino, questo viene appurato tramite il certificato associato alla chiave EK. Durante il processo di attestazione remota, un TPM può utilizzare differenti chiavi AIK in modo riportare l'integrità del sistema a diversi verificatori.

La versione 1.2 della specifica TCG definisce un protocollo crittografico Direct Anonymous Attestation (DAA) [7] utilizzato per eliminare la necessità di utilizzare una Privacy CA, per determinare se una chiave appartiene a un dato TPM. Il DAA è un protocollo crittografico che permette di ottenere un'autenticazione remota del modulo TPM, preservando l'anonimato dell'utente che possiede il modulo. Per svolgere un'autenticazione corretta, il protocollo DAA richiede l'utilizzo di tre dispositivi, una piattaforma dotata di un TPM, un *issuer* e un *verifier*. Il protocollo DAA è composto da due sottoprotocolli: il protocollo join e il protocollo DAA-sign. Il primo consente a un TPM di ottenere un certificato dal *issuer*, mentre il secondo consente a un TPM di firmare digitalmente un messaggio per essere poi inviato al *verifier*. Il protocollo DAA-sign include al suo interno anche una procedura che impedisce a un TPM corrotto di ottenere certificati e di autenticare i messaggi. Il funzionamento del DAA è suddiviso in due fasi:

- nella prima fase, l'issuer deve verificare che l'hardware TPM non sia stato compromesso, al termine di questa fase fornisce al TPM le credenziali DAA. Tali credenziali saranno poi utilizzate per eseguire le firme;
- nella seconda fase, il verifier riceve la firma DAA dalla piattaforma e verifica le credenziali di appartenenza alla firma.

La verifica di integrità di un sistema consiste nel verificare l'integrità di ogni componente prima che venga eseguito. Oltre a verificare l'integrità in un dato momento temporale, dobbiamo anche essere certi che il sistema sia stato avviato correttamente e che si stia utilizzando il sistema operativo appropriato. Questo processo si ottiene creando una catena di fiducia a partire dal CRTM, che come introdotto precedentemente è un'estensione del BIOS che viene eseguita prima di misurare altre parti del BIOS. Il BIOS quindi nel processo di **Measured Boot** misura l'hardware e il bootloader e passa successivamente il controllo al bootloader il quale misura l'immagine del kernel del sistema operativo prima di caricalo in memoria e passa il controllo al sistema operativo. Il sistema operativo può misurare le varie applicazioni eseguite all'interno del sistema. Ogni fase del processo di avvio estende il valore del PCR appropriato nel TPM con le misurazioni effettuate in ogni passaggio, l'insieme di queste misurazione attestano l'integrità del sistema.

2.1.2 Integrity Measurement Architecture

L'Integrity Measurement Architecture (IMA) [1] è l'architettura utilizzata per ottenere l'elenco delle misure di integrità relative ad un dato host. La grande potenzialità di IMA è che per il suo funzionamento non richiede nessuna modifica a livello di Sistema Operativo in quanto tale modulo è presente all'interno del kernel Linux a partire dalla versione 2.6.30. L'architettura IMA è in grado di dare ottime garanzie di fiducia del host senza portare massicci cambiamenti all'hardware o al software sottostante, ma per il suo funzionamento corretto richiede l'utilizzo del hardware TPM. Oggigiorno questo non è più un problema perché il TPM è presente in larga scala in molti computer.

All'interno del processo di attestazione mirato al mondo cloud l'architettura IMA svolge un ruolo fondamentale perché tramite l'elenco delle misure fornire da IMA, l'amministratore di rete può accorgersi molto velocemente se un nodo risulta essere fidato o meno. In caso di un'eventuale

manomissione di un nodo all'interno della rete l'amministratore della rete può decidere le sorti del nodo in modo da garantire la stabilità dell'intera rete. In ambito di sicurezza l'architettura IMA ricopre un ruolo molto importante.

Il sottosistema IMA è formato da una serie di moduli che svolgono differenti funzioni di integrità, essi sono:

- **Collect:** operazione che consiste nella misurazione del file prima che esso venga aperto nel sistema in cui avviato ed acceduto;
- **Store:** operazione di aggiunta della misurazione all'interno di una lista residente a livello di kernel, se è presente l'hardware TPM nel sistema, estensione del relativo registro PCR con la misura in esame;
- **Attest:** operazione vincolata alla presenza dell'hardware TPM, infatti se è presente uso del TPM per firmare digitalmente il valore presente nel registro PCR gestito da IMA, in modo da consentire una validazione remota della lista di misure;
- **Appraise:** forzare una validazione locale di un valore considerato "buono" e memorizzato in un attributo di estensione al file misurato;
- **Protect** protezione da attacchi degli attributi di sicurezza di estensione dei file;
- **Audit:** verifica degli hash del file.

Le prime tre funzionalità sono state introdotte in IMA dalla prima versione, le funzioni di "appraise" e "protect" sono state inizialmente pubblicate come una singola patch applicata in IMA per poi essere successivamente scorporate, ora sono disponibili nel modulo kernel EVM (Extended Verification Module) quindi non incluse nel modulo IMA.

La lista delle misure calcolate da IMA, risiede all'interno del securityfs filesystem, in due formati differenti, in formato testuale codificato ASCII nel file `ascii_runtime_measurements` ed in formato binario all'interno del file `binary_runtime_measurements`. IMA durante la sua esecuzione mantiene anche al termine di ogni calcolo di una misura, l'aggregato delle misure all'interno dei registri del TPM (se presente), tipicamente all'interno del PCR10.

Nonostante IMA sia presente in tutti i kernel linux dal 2.6.30 in poi, occorre che venga abilitato andando a specificare una politica di misurazione. Per abilitare IMA occorre riavviare il kernel includendo il comando `ima_tcb`, in questo modo IMA verrà avviato per funzionare in maniera standard secondo la politica del Trusted Computing Base (TCB). Tale comando risulta essere deprecato ed è utilizzato soltanto per avviare IMA attraverso la politica di default. È possibile avviare IMA andando ad utilizzare una propria politica di misurazione, più precisamente occorre creare il file `ima-policy` specificando al suo interno la politica che si vuole applicare. All'avvio del sistema il modulo IMA andrà a verificare in un primo momento se è stata definita una propria politica di misurazione, se non è presente verificherà se il kernel è stato eseguito con il comando `ima_tcb` e andrà ad eseguire il processo di misurazione secondo le regole del TCB.

2.1.3 Remote Attestation

La Remote Attestation (RA) è il processo che ci permette di verificare se una piattaforma risulta essere affidabile o meno. Una possibile implementazione è quella definita della TCG Infrastructure Working Group (IWG), definisce un modello chiamato Integrity Management Model (IMM) [8].

In generale il temine "integrità" viene utilizzato per indicare lo stato di un componente all'interno di un piattaforma attendibile. Durante l'uso del componente, esso dovrebbe rimanere nel suo stato originario e non deve presentare manomissioni, eventuali stati anomali dovrebbero essere rilevati. IMM richiede l'implementazione di due funzioni fondamentali:

- *Integrity Measurement*: è utilizzata per consentire a un componente affidabile di eseguire misurazioni di integrità. Il termine misurazione ha due significati: la *runtime measurement* si riferisce all'operazione di hash–and-extend (all'interno dei PCR) eseguita sui componenti software durante il funzionamento di una piattaforma affidabile; invece, *reference measurement* si riferisce alla raccolta dei digest e alla registrazione di questi valori come parte di metadati informativi ad opera del produttore dei componenti non-hardware;

- *Integrity Reporting*: utilizzata per consentire alle misure di essere inviate al mondo esterno, più precisamente a un terzo componente che richiede di eseguire il processo di attestazione. La possibilità di riportare il proprio stato di integrità fornisce un elemento fondamentale alla base del TC. La sintassi del report delle misurazioni deve essere compresa chiaramente dalla macchina che lo realizza e dal server che attesta la macchina.

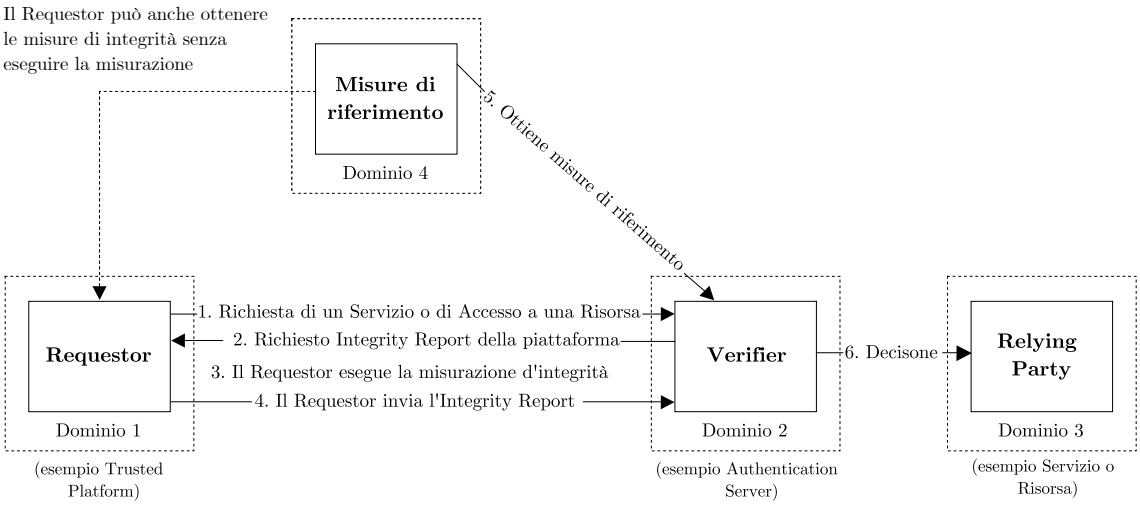

Figura 2.1. Modello per misurazione dell'integrità e per il reporting (immagine citata da [8])

La figura 2.1 mostra la comunicazione di base che avviene tra il Requestor e il Verifier, al fine di illustrare il modello utilizzato per la misurazione dell'integrità e il reporting. Nel primo passo il Requestor indica la sua richiesta al Verifier, il Verifier in risposta chiede al Requestor di eseguire una misurazione di integrità dell'intera piattaforma. Il Requestor al termine di questa misurazione (indicata nel terzo passaggio), restituisce un report di integrità che verrà trasmesso al Verifier, questo nel quarto passaggio. Il Requestor può anche ottenere le misure di integrità da un'altra entità.

Non si può essere certi che il Verifier abbia lo stesso sistema operativo della piattaforma da attestare, dunque il Verifier non può conoscere tutti i componenti utilizzati dal Requestor. Per questa ragione il Verifier deve cercare informazioni su ciascuno dei componenti della piattaforma del Requestor per essere in grado di prendere decisioni sulla affidabilità (nel quinto passaggio). Queste informazioni non sono altro che misure di riferimento che possono essere rese disponibili dei fornitori di metriche. Una volta che il Verifier è in grado di identificare ogni misura essa può essere confrontata con la lista di misure di riferimento, in questo modo il Verifier può valutare il livello di affidabilità della piattaforma del Requestor.

Gli attori che determinano questo modello sono essenzialmente tre: il Requestor raccoglie e invia l'integrity Report, il Verifier valuta le informazione ricevute sulla base di un set di misure di riferimento ed eventualmente comunica l'esito dell'attestazione ad un Relying Party. All'interno di questo processo vengono utilizzate due tipi di misure: *Runtime Measurement* sono inviate dal Requestor, rappresentano l'elenco di misure che descrivono lo stato del componente dalla fase del boot fino all'ultima operazione eseguita, tali misure vengono in un secondo momento inserite in una struttura chiamata Integrity Report; invece, le *Reference Measurement* sono utilizzate dal Verifier per valutare l'attendibilità delle misure ricevute.

Il linguaggio scelto per definire l'Integrity Report è l'XML questo per garantire le proprietà fondamentali di interoperabilità, estendibilità e maneggevolezza.

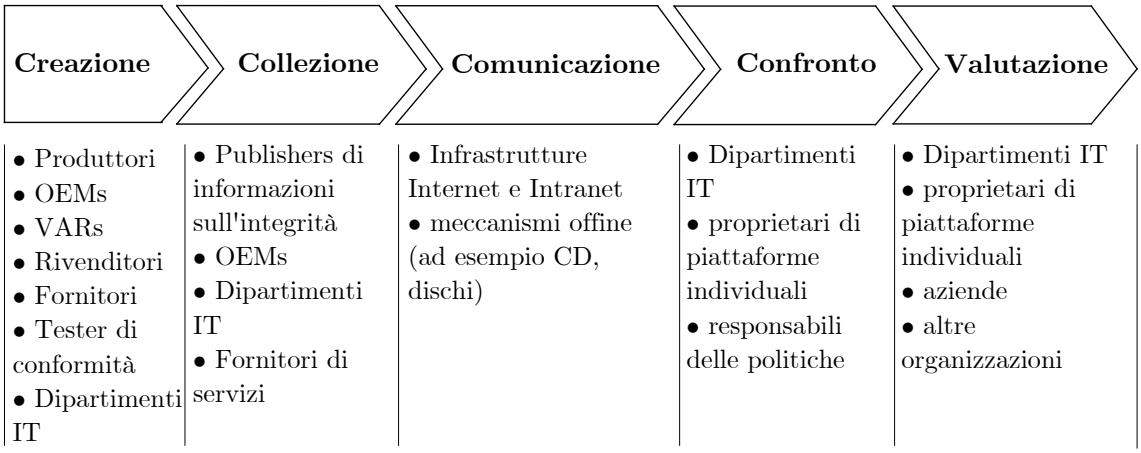

Figura 2.2. Immagine citata da [8] che descrive le fasi generali del IMM

Gli elementi principali dell'IMM sono l'affidabilità e autenticità delle misure, per garantirli vengono definite cinque fasi (Figura 2.2) per gestire le informazioni di integrità, esse sono:

1. *Creazione*: i dati di integrità o i valori relativi a un determinato componente devono essere creati o prodotti da un'entità fidata.

2. *Collezione*: le misure di integrità vengono raccolte per realizzare un set di misure che definiscono lo stato della piattaforma.

3. *Comunicazione*: la lista delle misure definite dalla piattaforma (Requestor) è inviato a una seconda entità che si occuperà di valutarle.

4. *Confronto*: consiste nell'atto di raccogliere l'insieme delle misure di integrità relative a una specifica piattaforma, per la successiva valutazione. Lo scopo del Confronto consiste nel poter verificare eventualmente le informazioni di integrità, mentre lo scopo della Collezione è di rendere disponibili le misure ad altri componenti.

5. *Valutazione*: è la fase che ci permette di accertare l'integrità di una piattaforma e quindi di accertare la correttezza dei dati di integrità. Un Verifier può eseguire valutazioni per propri scopi o per conti di un Relying Party.

2.1.4 Processo di Attestazione Remota

In questa sezione verrà trattato il flusso di attestazione [9] rappresentato in figura 2.3, che avviene tra il Requestor e il Verifier.

- *Identity Credential Enrollment* (freccia 1A): prima che una piattaforma affidabile (Requestor) possa comunicare con il mondo esterno, deve ottenere una credenziale di identità, come un certificato AIK, emesso da una Platform-CA. Per effettuare questa richiesta, l'entità deve essere in possesso di altre credenziali TCG (come la credenziale EK).

- *Identity Credential Publish* (freccia 1B): quando una piattaforma attendibile (Verifier) desidera verificare l'affidabilità di una'altra piattaforma (Requestor), deve necessariamente ottenere copie delle credenziali di identità (certificati AIK) delle piattaforme da attestare da una o più Platform-CA.

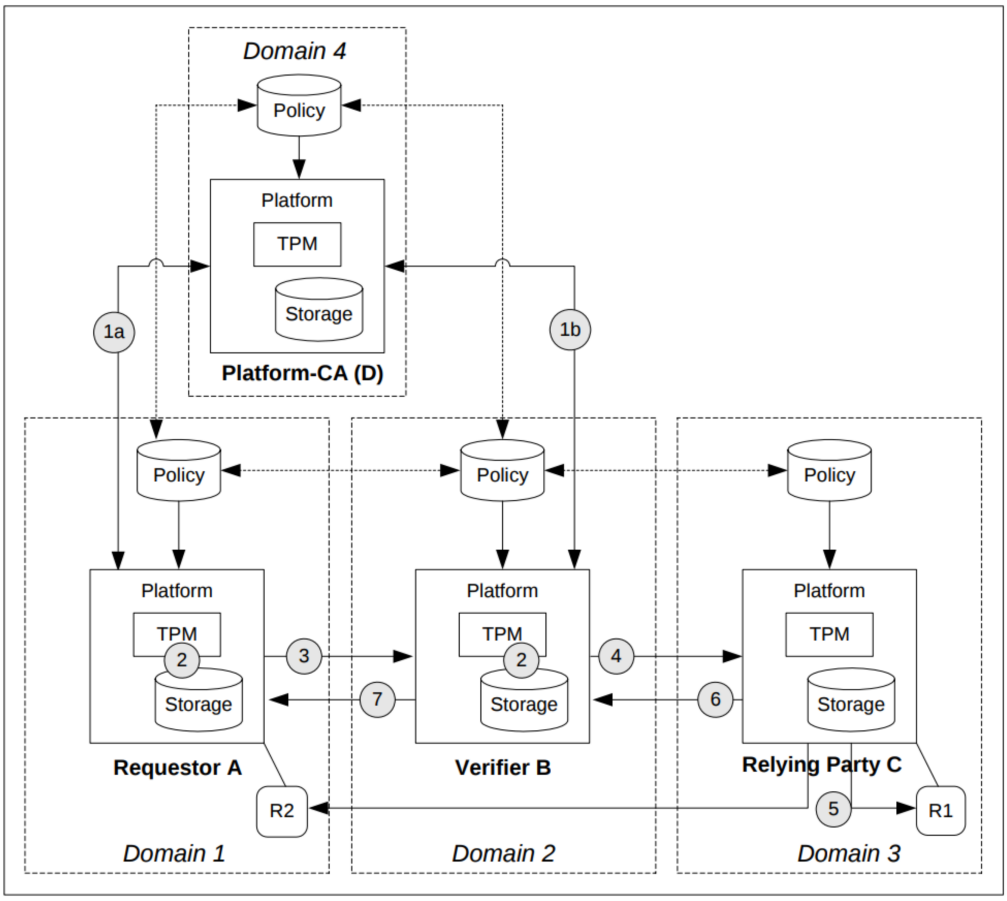

Figura 2.3. Immagine citata da [9], che descrivere il flusso di attestazione

- *Platform Integrity Measurement* (freccia 2): è il processo che permette ad una piattaforma di ottenere l'elenco delle sue metriche. Queste metriche sono memorizzate nei log (Stored Measurement Logs) e i digest (hash) relativi alle metriche vengono collocati nei PCR. Le misurazioni dell'integrità della piattaforma sono di primaria importanza per il Verifier per effettuare una valutazione sull'affidabilità del Requestor.

- *Platform Integrity Reporting* (freccia 3): le misurazioni di integrità relative alla piattaforma del Requestor vengono segnalate al Verifier per essere poi valutate dallo stesso. Due aspetti fondamentali di questo passaggio sono: la struttura e il formato nella quale le misurazioni vengono riportate (Integrity Report) e il protocollo di trasporto utilizzato per trasmetterle.

- *Evaluation Reporting* (freccia 4): il risultato della valutazione da parte del Verifier deve essere comunicato al Relying Party. Questo implica che è stato raggiunto un accordo tra il Relying Party e il Verifier, per quanto riguarda la metrica di valutazione che deve essere utilizzata dal Verifier. Tale valutazione dipende dal contesto o dal caso d'uso in cui il Requestor si trova, ad esempio le metriche di valutazione utilizzate in uno scenario VPN saranno diverse da uno scenario rivolto all'Online Banking.

- *Direct Response/Action* (freccia 5): il Relying Party può generare una Risposta e/o un'Azione per la valutazione del Requestor da parte del Verifier. Tale Risposta/Azione può essere locale (R1) oppure può interessare il Requestor (R2).

- *Indirect Response/Action* (freccia 6 e 7): questo flusso si instaura quando un Requestor chiede un servizio alla Relying Party, la risposta sarà trasmessa dal Verifier che opera come intermediario tra le entità.

2.2 Network Function Virtualization

Network Function Virtualization (NFV) [16] è un paradigma che offre un nuovo modo di progettare, implementare e gestire i servizi di rete. Attualmente le reti degli internet provider (fornitori dei servizi di rete), sono formate da un ampio numero di apparecchiature hardware (middle-box). Le funzioni di rete sono inserite all'interno di apparecchiature hardware, dunque nel caso in cui ci fosse la necessità di inserire all'interno della rete un nuovo servizio, questo potrebbe risultare complicato. L'aggiunta di un servizio di rete porta con se varie problematiche, come: gestione di costi elevati di manutenzione, necessita di personale altamente qualificato, un elevato costo energetico ed infine, le apparecchiature hardware hanno una mancanza di scalibilità nel soddisfare in modo dinamico le esigenze della rete. Questo ultimo punto porta complicanze nel gestire in modo dinamico la topologia della rete stessa. Un'altra grande problematica delle apparecchiature hardware è che sono proprietarie quindi ogni componente hardware viene realizzata secondo la polita di un internet provider.

NFV mira dunque ad affrontare le problematiche appena elencate sfruttando la tecnologie della virtualizzazione, disaccoppiando completamente le funzioni di rete dalle apparecchiature hardware sulle quali vengono eseguite. Nel contesto NFV, un servizio di rete può essere scomposto in una serie di applicazioni software, chiamate Virtual Network Function (VNF). Il paradigma NFV potrebbe potenzialmente offrire molti benefici ai fornitori di servizi di rete, alcuni dei quali sono elencati di seguito [16]:

- L'utilizzo delle funzioni di rete basate sulla virtualizzazione permette ai fornitori di avere una riduzione dei costi delle apparecchiature hardware ed anche una diminuzione del consumo di energia.
- Aumento della velocità del "Time to Market", ovvero la diminuzione del tempo che intercorre dall'ideazione alla messa in vendita delle funzioni di rete. Questo è possibile perché in un contesto NFV è sufficiente agire sul software. Gli operatori di rete per effettuare aggiornamenti e migliorie possono concentrarsi esclusivamente sullo sviluppo del software.
- Supporto al *multi-tenancy*, cioè la possibilità di fornire contemporaneamente a più utenti servizi di rete in esecuzione sullo stesso hardware. Questo è uno spunto molto importante perché consente agli operatori di rete di condividere le risorse fisiche. Nel contesto NFV si cerca quanto più possibile di limitare lo spreco di risorse.
- Il paradigma NFV potrebbe incoraggiare un cambio del sistema, in quanto non vengono più utilizzate componenti hardware proprietarie, ma viene utilizzato dell'hardware generico sul quale è possibile istanziare qualsiasi tipo di funzione di rete. Questo nuovo sistema apre il mercato delle apparecchiature virtuali a tutti, incoraggiando una maggiore innovazione che porta ad aggiungere all'interno della rete nuovi servizi in modo molto rapido.
- Possibilità di agire da remoto sui dispositivi di rete, andando ad aumentare notevolmente la velocità di assistenza.
- Migliore configurazione e topologia della rete in tempo reale, andando ad osservare il traffico e i servizi offerti nella rete in un determinato istante, questo permette di agire in modo dinamico per migliorare la qualità del servizio offerto.

Lo scenario NFV trasforma completamente il modo in cui gli operatori di rete architettano la rete, andando ad utilizzare tecnologie di virtualizzazione, per consolidare molti tipi di apparecchiature di rete su server, switch e storage, i quali potrebbero essere collocati in datacenter, in nodi di rete oppure direttamente nelle sedi dell'utente finale a cui il servizio è rivolto.

Il gruppo ETSI ha definito un'architettura ad alto livello [15] per rappresentare il contesto NFV, all'interno della figura 2.4 sono identificati i tre componenti architetturali principali, rappresentati da: VNF, NFV Infrastracture (NFVI) e NFV Management and Orchestration (NFV MANO).

All'interno dell'architettura, NFVI è rappresentata come la combinazione di risorse fisiche e virtuali, le quali sono suddivise in tre categorie: compute, storage e networking, l'unione di queste

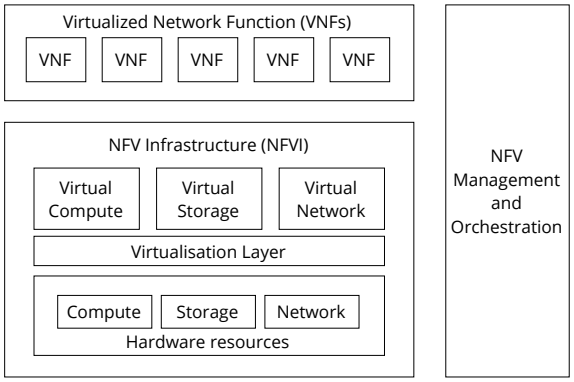

Figura 2.4. Architettura high-level rivolta allo schenario NFV [15].

risorse costituisce l'ambiente in cui le VNF sono gestite, eseguite e distribuite. NFVI è costituito da componenti hardware generici il cui compito è quello di fornire un pool di risorse fisiche. Queste risorse, vengono successivamente astratte da uno strato intermedio di virtualizzazione, basato su un hypervisor, che permette di realizzare un disaccoppiamento tra l'hardware fisico e il dominio in cui verranno dispiegate le VNF. Le funzioni esposte dallo strato di virtualizzazione sono dunque delle astrazioni delle risorse fisiche sottostanti, tali risorse virtuali possono, per esempio, essere rappresentate in temine di macchine virtuali (VM).

Come definito in precedenza, le VNF sono componenti che forniscono delle funzionalità di rete. Tali funzionalità vengono raggiunte attraverso l'implementazione di moduli software, che possono essere in esecuzione su una o più VM. Ogni VNF può svolgere compiti differenti (ad esempio, funzioni di firewall, NAT, routing) e può essere istanziata su uno o più server fisici.

L'ambiente NFV MANO, si occupa di gestire ed orchestrare le risorse fisiche e virtuali con lo scopo di istanziare e gestire i servizi di rete all'interno dell'infrastruttura NFV. Tale componente svolte un ruolo molto importante infatti si occupa di gestire anche il ciclo di vita delle varie VNF presenti all'interno dell'ambiente. Nella sezione successiva verrà approfondito il dominio NFV MANO, in modo da comprendere gli aspetti architetturali definiti nel contesto della tesi.

2.2.1 NFV MANO

Il dominio NFV MANO svolge un ruolo cardine all'interno dell'architettura NFV, si occupa di gestire l'infrastruttura NFV (NFVI) e di orchestrare l'allocazione delle risorse necessarie ai Network Service (NS, si basano su una combinazione di più VNF) e alle VNF. Il suo ruolo di coordinamento è fondamentale perché nello scenario NFV avviene un disaccoppiamento tra le funzioni di rete e l'infrastruttura.

Per quanto riguarda la gestione e l'orchestrazione delle NFVI, il dominio MANO si occupa di amministrare entrambe le risorse fisiche e virtuali. La prima è limitata principalmente agli aspetti di connettività tra risorse fisiche e virtualizzate, mentre le risorse virtualizzate sono mirate a gestire le risorse di un NFVI in *NFVI Point of Presence* (NFVI PoP), un elemento concreto dell'infrastruttura, come un datacenter o un server, su cui è possibile eseguire VNF. Le risorse virtualizzate vengono utilizzate per fornire alle VNF tutte le risorse di cui hanno bisogno per la loro esecuzione. La concessione delle risorse a una VNF è un'attività complessa, perché bisogna soddisfare tutte le esigenze di una VNF espresse sotto forma di molteplici vincoli e requisiti. Durante il suo ciclo di vita, una VNF deve essere monitorata in modo costante, in modo da garantire che i suoi vincoli vengano sempre soddisfatti. Il dominio MANO durante il ciclo di vita di una VNF per garantire che i vincoli imposti dalla VNF siano sempre rispettati, potrebbe interrompere, riprendere e generare nuove istanze della VNF.

Gli aspetti relativi alla gestione e orchestrazione delle VNF si concentrano principalmente alle operazioni riguardanti il ciclo di vita delle VNF, come:

- istanziazione di una nuova VNF, che comporta la creazione di una VNF e dunque l'allocazione delle risorse dell'infrastruttura NFV;
- scalabilità, che va ad aumentare o ridurre la capacità della VNF;
- update e upgrade, consiste principalmente in modifiche software di varia complessità;
- terminazione, che consiste nel rilasciare e restituire le risorse NFVI associate a una VNF.

Tutte le informazioni relative a una data VNF vengono acquisite in un modello di implementazione e memorizzate durante il processo di on-boarding della VNF in un catalogo. Durante la fase di creazione, le risorse NFVI vengono associate a una VNF in base ai requisiti presenti all'interno del modello di implementazione e a delle informazioni aggiuntive che possono essere presenti all'intero della richiesta di istanziazione. Durante il ciclo di vita di una VNF le sue capacità potrebbero essere aumentate o ridotte in base alle sue effettive prestazioni, questo scenario appena descritto rappresenta il concetto di scalabilità. Il processo di scalabilità può includere ad esempio, la modifica della configurazione delle risorse virtualizzate (ridimensionare o aumentare CPU) oppure aggiungere nuove risorse virtualizzate. Un altro aspetto importante delle VNF è legato all'utilizzo dei Key Performance Indicator (KPI), il cui uso è mirato a monitorare l'attività di una VNF principalmente per scopi di ridimensionamento.

Il Network Service Orchestrator si concentra sulla gestione del ciclo di vita dei NS. Svolte molteplici operazioni come: registrazione di un NS, descritto tramite un descriptor (NSD) che definisce il NS e le VNF a cui il NS fa riferimento, tale descriptor viene memorizzato in un catalogo in modo da istanziare il NS quando richiesto; istanziazione del NS; scalabilità (aumento o riduzione delle capacità di un NS); aggiornamento e terminazione.

Come possiamo osservare dalla figura 2.5 che mostra nel dettaglio l'architettura NFV MANO [17], sono presenti tre blocchi funzionali principali:

- Virtualised Infrastructure Manager (VIM);
- NFV Orchestrator (NFVO);
- VNF Manager (VNFM).

L'architettura NFV MANO presenta anche delle sezioni in cui includere i descriptor delle VNF e dei NS. Ora verranno descritti più nel dettaglio i tre componenti principali dell'architettura MANO.

NFV Orchestrator (NFVO) è il componente dell'architettura responsabile di organizzare le risorse NFVI su più VIM e di gestire il ciclo di vista dei Network Service. Queste due responsabilità soddisfano i due aspetti principali sulle quali è basato il NFVO:

- Resouce Orchestration viene soddisfatto attraverso funzioni che si occupano di gestire l'allocazione e il rilascio delle risorse di un NFVI.
- Network Service Orchestration viene soddisfatto attraverso funzioni che gestiscono il ciclo di vita di un NS, come on-boarding, l'istanziazione e la terminazione del servizio.

NFVO utilizza le funzioni espresse dal Network Service Orchestration per coordinare più VNF in modo da creare servizi di rete che implementino una funzione di rete complessa. Il Network Service Orchestration utilizza i servizi esposti dalla funzione di VNF Manager e dalla funzione di Resource Orchestration per fornire diverse funzionalità, le quali vengo esposte tramite delle interfacce utilizzabili dai vari blocchi NFV MANO o anche da entità esterne. Alcune funzionalità fornite da NFVO, tramite le funzioni del Network Service Orchestration sono elencate di seguito:

- Gestione dei modelli, cioè dei descriptior, con cui vengono definiti i NS e le VNF. È necessario effettuare un processo di convalida durante l'on-boarding dei NS e delle VNF che si occupa di verificare l'integrità e l'autenticità del modello di implementazione fornito, verificando anche che tutte le informazioni obbligatorie siano presenti e risultino coerenti. Durante il processo di on-boarding delle VNF, utilizzando il supporto del VIM, le immagini software fornite nel pacchetto VNF sono catalogate in uno o più NFVI-PoP.

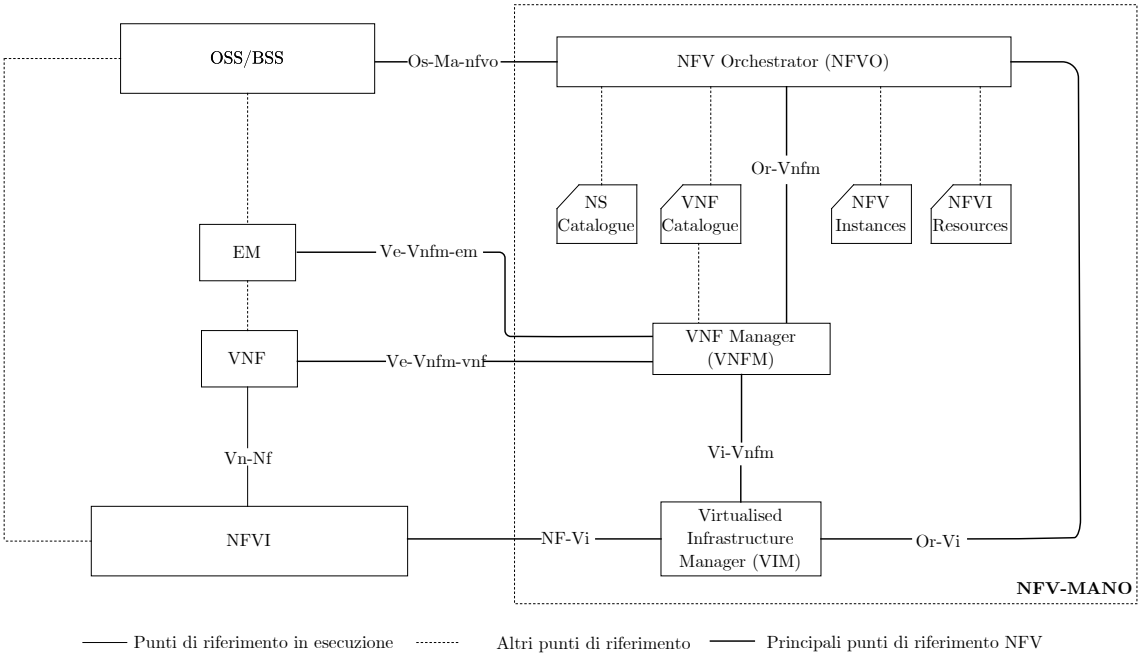

Figura 2.5. Architettura NFV MANO (immagine citata da [17]).

- Gestione del ciclo di vita e istanza del Network Service.
- Gestione dell'istanziazione dei VNF manager.
- Gestione dell'istanziazione dei VNF, in coordinamento con i VNF Manager.
- Convalida e autorizzazione delle richieste di risorse NFVI dai VNF Manager, perché potrebbero influire sul funzionamento dei NS.
- Gestione dell'integrità e della visibilità delle istanze dei NS durante il loro ciclo di vita.

NFVO utilizza le funzionalità del Resource Orchestration per fornire servizi che permettono un accesso astratto alle risorse NFVI, evitando così che possano dipendentemente da qualsiasi VIM. Alcune delle funzionalità più rilevanti sono espresse di seguito:

- Convalida e autorizzazione delle richieste di risorse NFVI da parte di VNF Manager, le risorse devono essere necessariamente autorizzate dal VNFM, perché potrebbero influire sul modo in cui le risorse vengono allocate all'interno di uno o più NFVI-PoP.
- Gestione della relazione tra le istanze VNF e le risorse NFVI ad esse assegnate, utilizzando il repository *NFVI Resources* e le informazioni ricevute dai VIM.
- Raccolta delle informazioni relative all'utilizzo delle risorse NFVI da istanziare per una o più VNF.

VNF Manager (VNFM) è una funzione NFV MANO che è responsabile di gestire il ciclo di vita delle VNF. Ogni VNF deve essere associata a un VNFM, il quale può essere associato a una o più VNF che possono svolgere lo stesso compito oppure compiti differenti. Le funzioni definite dal VNF Manager sono generiche, in modo che possano essere applicate a qualsiasi tipo di VNF. Queste funzionalità includono:

- istanziazione e se richiesta configurazione della VNF;
- verifica se l'istanziazione di una VNF è fattibile;

- update e upgrade del software contenuto in un'istanza di una VNF;
- modificare un'istanza di una VNF;
- hestire il ridimensionamento di un'istanza di una VNF;
- fornire assistenza, assistita o automatizzata ad una VNF;
- terminazione di una VNF;
- gestione dell'integrità di un'istanza di una VNF attraverso il suo ciclo di vita.

Ogni VNF è definita tramite un modello chiamato Virtualised Network Function Descriptor (VNFD), memorizzato nel *VNF catalogue*. Il VNFD serve a descrivere gli attributi e i requisiti necessari a una VNF per essere mandata in esecuzione. NFV MANO utilizza il VNFD per istanziare una VNF e per gestirne il ciclo di vita. All'interno del descriptor, le risorse hardware vengono espresse in modo astratto, questo per garantire la piena portabilità delle VNF in ambienti NFVI differenti. Le risorse NFVI vengono assegnate a una VNF in base alle specifiche definite nel suo descriptor.

Virtualized Infrastructure Manager (VIM) è responsabile del controllo e della gestione delle risorse di compute, storage e networking, contenute all'interno di una infrastruttura NFV. Un VIM si può occupare di gestire tutte le risorse contenute in un NFVI oppure può essere specializzato a gestire soltanto un certo tipo di risorsa, ad esempio solo le risorse di storage. Per gestire le risorse un VIM espone un'interfaccia verso altre funzioni. Per eseguire le funzionalità esposte attraverso la sua interfaccia il VIM comunica con uno o più hypervisor. Le operazioni eseguite dal VIM includono:

- Gestione delle risorse attraverso l'allocazione, l'aggiornamento e infine il rilascio delle risorse NFVI, gestendo l'associazione tra le risorse virtualizzate e quelle fisiche. Per svolgere questo compito il VIM mantiene al suo interno un inventario su come le risorse virtuali sono associate a quelle fisiche.
- Supporto per la gestione delle VNF organizzando collegamenti virtuali, reti, sottoreti e porte.
- Gestione di un repository di risorse hardware NFVI e risorse software, in modo da ottimizzare l'utilizzo di tali risorse.

Il VIM memorizza anche le immagini software che verranno associate successivamente alle VNF che saranno istanziate all'interno del sistema.

2.2.2 Security as a Service

In questa sezione verrà descritto un caso d'uso interessante per il contesto NFV, che prende il nome di "Security as a Service" (SecaaS) [18]. L'ETSI ha descritto il caso d'uso perché l'aumento della criminalità informatica ha come conseguenza quello di creare problemi importanti all'economia online. Oggigiorno gli attacchi informatici sono sempre più complessi e le aziende economiche per far fronte a questo fenomeno devono aggiornare continuamente le loro tecniche di cybersecurity. Tale aggiornamento è un processo costoso da sostenere e non si tratta di un'operazione veloce. Per questo motivo risulta difficile contrastare in modo efficace le nuove minacce informatiche. Si aspetta che questa situazione peggiorerà in futuro, perché l'utilizzo di internet e i servizi IoT all'interno di un'azienda, svolgono un ruolo cruciale e questo porta a una maggiore esposizione della stessa al mondo esterno. La tecnologia NFV può fornire protezione efficace e veloce in un ambiente ricco di minacce informatiche in rapida evoluzione. Per farlo può utilizzare tipi specifici di VNF che garantiscono funzioni di sicurezza, prendono il nome di Virtual Network Security Function (VNSF). L'utilizzo di una VNSF può fornire diversi benefici come:

- fornire una risposta dinamica e specifica per ogni minaccia alla sicurezza, attraverso l'utilizzo di una VNF specifica;

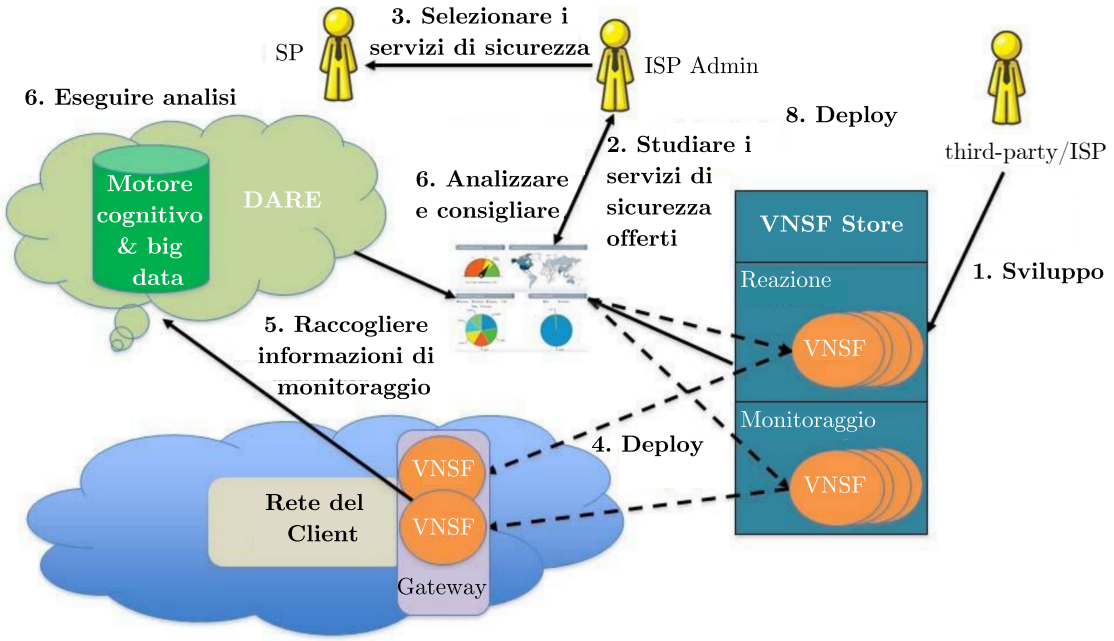

Figura 2.6. Rappresentazione grafica del caso d'uso SecaaS (immagine citata da [18]).

- possibilità di scalare le risorse;
- monitoraggio dei dati di sicurezza, in modo da analizzare la rete.

L'idea principale consiste nella creazione di servizi SecaaS avanzati, sfruttando la tecnologia NFV.

La figura 2.6 descrive nel dettaglio il caso d'uso SecaaS. Utilizzando un approccio del genere, la complessità dell'analisi di sicurezza, non è più di responsabilità dei clienti della rete, ma viene gestita direttamente dagli Internet Service Provider (ISP). Questo libera i clienti dalla necessità di acquistare, gestire e aggiornare le apparecchiature di sicurezza. In questo modo un ISP sarebbe in grado di andare ad inserire, tramite il gateway o l'infrastruttura di rete, una nuova funzionalità di rete orientata alla sicurezza direttamente nella rete dell'utente che ne ha fatto richiesta.

All'interno di questo scenario occorre definire tre identità, che sono Client, ISP e thirtd-party (terze parti). I Client, sono coloro che utilizzano la rete e richiedono particolari servizi di sicurezza definiti dal ISP oppure da una terza parte. Gli ISP utilizzano le risorse NFVI e forniscono servizi di rete, mentre i third-party, aziende con specifici accordi commerciali con gli ISP, possono pubblicare funzioni di sicurezza della rete virtuale. Le funzioni saranno presenti all'interno di un catalogo, in modo che possano essere selezionabili per l'implementazione da parte dei client.

I componenti principali del caso d'uso SecaaS sono:

- **VNFS**, sono VNF orientate alla sicurezza e si occupano del monitoraggio delle minacce e di agire sulla sicurezza. Ad esempio i VNSF possono funzionare come sonde di rete oppure honeypot per svolgere un ruolo di monitoraggio della rete, oppure possono essere anche utilizzate per agire sulla rete, con l'obiettivo di fermare e prevenire attacchi informatici;
- **Data Analysis and Remediation Engine (DARE)**, utilizza tecniche analitiche, di monitoraggio delle minacce e di intelligenza cognitiva per elaborare le informazioni derivanti dalle VNSF, utilizza un motore di analisi su big data andando a mettere in relazione enormi volumi di dati eterogenei. Il suo scopo principale è quello di identificare i comportamenti dannosi attuali all'interno dell'infrastruttura oppure di predire delle probabili minacce future;
- **VNSF Store**, il suo scopo è quello di fornire un repository e un catalogo, in modo da offrire la possibilità alle VNSF di essere selezionate per soddisfare un particolare obiettivo;

- **VNSF Orchestrator**, definisce una funzionalità associata al dominio NFV MANO, il suo compito è quello di orchestrare le politiche di sicurezza ed è in grado di gestire in modo efficiente le varie VNSF controllandone i loro cicli di vita;
- **VNSF and infrastructure attestation**, serve a dimostrare agli utenti della rete che il servizio è affidabile. Questo viene fatto instaurando un collegamento tra le VNSF e le loro configurazioni di rete e gli altri componenti all'interno dell'architettura NFV. Questo processo è fondamentale per verificare che tutti i servizi all'interno della rete si comportino secondo le specifiche per il quale essi sono stati implementati. Serve a dimostrare che le VNSF non siano state manipolate da terze parti.

2.2.3 Flusso Security as a Service

Il processo in cui tutto questo si concretizza è definito all'interno della figura 2.6:

- *Sviluppo* (flusso 1): Inizialmente abbiamo la fase di sviluppo, all'interno della quale le VNSF possono essere sviluppate dal ISP oppure da una terza parte. Nel momento in cui le VNSF vengono sviluppate e testate con esito positivo, vengono incluse all'interno del VNSF Store.
- *Studiare i servizi di sicurezza offerti* (flusso 2): in questa fase il client analizza quale VNSF risulti a lui più utile, in base al servizio di sicurezza che si vuole raggiungere.
- *Selezionare i servizi di sicurezza* (flusso 3): il client seleziona i servizi di sicurezza desiderati, definiti al passo precedente.
- *Deploy* (flusso 4): i servizi di sicurezza selezionati vengono distribuiti al client. Tali servizi che possono essere formati da una o più VNSF, istanziate su un NFV-PoP dell'infrastruttura NFVI del ISP, oppure se richiesto direttamente sull'hardware locale del client.
- *Raccogliere informazioni di monitoraggio* (flusso 5): le VNSF inviano le informazioni sul monitoraggio della rete al DARE, che prima verifica che le informazioni siano valide e successivamente le memorizza.
- *Eseguire analisi* (flusso 6): il DARE elabora le informazioni ricevute in base alle esigenze dei servizi di sicurezza erogati al client.
- *Analizzare e consigliare* (flusso 7): i dati di monitoraggio elaborati dal DARE vengono messi a disposizione del client tramite una dashboard. Oltre ai dati, sono presenti anche informazioni aggiuntive, tramite il quale il DARE può raccomandare di effettuare ulteriori azioni in base alle minacce analizzate.
- *Deploy* (flusso 8): in base all'esito del punto precedente il client può rendersi conto se ha bisogno di più VNSF per raggiungere il suo obiettivo di sicurezza.

Il processo si interrompe quando il client annulla la sottoscrizione al servizio offerto dal ISP, in caso contrario il processo di monitoraggio proseguirà.

2.3 Container

Negli ultimi tempi, il software basato sui containers [23] ha cambiato notevolmente il mondo industriale e il modo in cui il software viene distribuito e gestito. A differenza della virtualizzazione basata sull'hypervisor, nella quale una o più macchine virtuali indipendenti sono eseguite sull'hardware fisico tramite un livello che funge da intermediario, i containers vengono invece eseguiti nello spazio utente sopra il kernel del sistema operativo. La virtualizzazione basata su containers viene definita come virtualizzazione a livello di sistema operativo, proprio perché non ha il livello di hypervisor. La tecnologia basata sull'utilizzo dei containers permette di eseguire su una macchina fisica un grande numero di containers contemporaneamente, ognuno dei quali ha un proprio file

system e un insieme di applicazioni installate su esso. Ogni singola istanza si comporta come una macchina virtuale separata. A differenza delle tecnologie di virtualizzazione tradizionali, i containers non richiedono un livello di hypervisor da eseguire, utilizzano invece, una normale interfaccia di chiamata di sistema, realizzata a livello di sistema operativo, questo riduce il sovraccarico necessario per eseguire i containers.

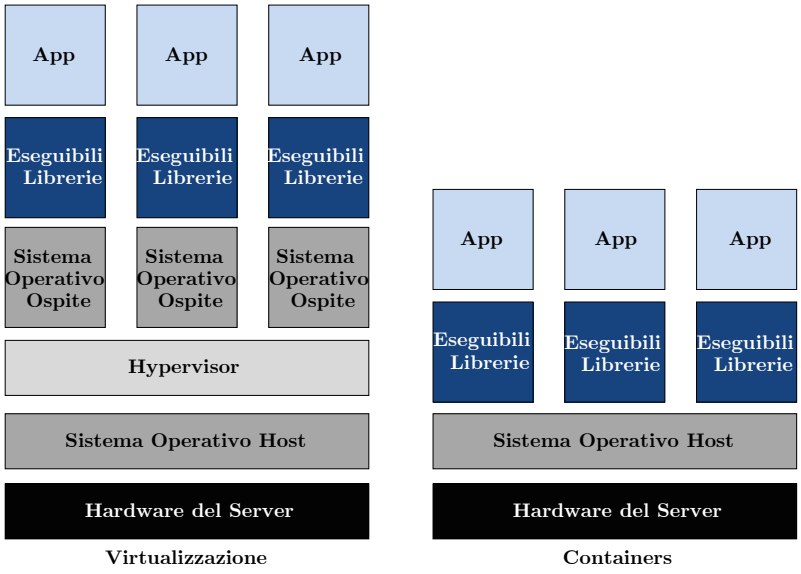

Figura 2.7. Differenza di Architettura tra Virtualization e Container

Alcune delle implementazioni più utilizzate che fanno uso di containers sono Linux Containers (LXC) e Docker. La virtualizzazione basata su Docker verrà analizzata nel dettaglio nella prossima sezione, perché all'interno della tesi andiamo ad utilizzare il processo di Remote Attestation basato su l'attestazione di container Docker.

2.3.1 Docker

Docker [10] [11] è un progetto open-source il quale si occupa di automatizzare il deployment di applicazioni all'interno di containers, introduce il concetto di virtualizzazione leggera, cioè permette di realizzare una virtualizzazione senza l'utilizzo di un livello intermedio, definito dall'hypervisor. È progettato per fornire un ambiente leggero in maniera molto rapida nel quale eseguire applicazioni. Il suo utilizzo è estremamente semplice, infatti per il suo funzionamento occorre utilizzare un kernel Linux compatibile e un binario Docker. I containers Docker vanno in esecuzione direttamente nell'"user space", dunque al di sopra del kernel e del sistema operativo.

All'interno di una stessa macchina come introdotto precedentemente possono essere eseguiti più container contemporaneamente, questo è possibile perché vi è una gestione dell'isolamento tra i vari container. L'isolamento tipico, applicato ai container Docker, avviene attraverso l'utilizzo di alcune funzionalità offerte dal kernel Linux, più precisamente `namespace` e `control groups`.

Docker utilizza la funzionalità offerta dal namespace per offrire la prima e più semplice forma di isolamento. I processi in esecuzione all'interno di un container non possono influire sul comportamento dei processi in esecuzione su un altro container o sull'host. Per creare questo isolamento, quando viene creato un container, Docker crea un set di namespace per quel container. Il namespace più importante è chiamato *PID namespace*, si occupa di fornire un isolamento tra processi di container differenti. Un altro importante namespace che viene creato è il *network namespace*, si occupa di fornire un isolamento a livello di rete. Ciascun container avrà il proprio stack di rete, in modo tale che il container possa avere accesso soltanto alle proprie socket e alle proprie interfacce di rete. Ad ogni container verrà inoltre attribuito un indirizzo IP. I vari container tra di loro possono comunque comunicare attraverso le proprie interfacce di rete. Attraverso l'esposizione di porte pubbliche è possibile avviare il traffico IP tra container differenti.

I control groups (cgroups) sono un'altra funzionalità presente nel kernel Linux, sono utilizzati da Docker per l'allocazione, la contabilità e la limitazione delle risorse tra processi appartenenti a diversi namespace. I cgroups garantiscono che ogni container ottenga una equa allocazione delle risorse come CPU e memoria. La caratteristica più importante è quella di garantire che un singolo container non esaurisca tutte le risorse presenti nel sistema. Questa ultima caratteristica permette a un container di non andare a saturare le risorse all'interno del host evitando così il suo collasso.

I contenitori Docker sono indipendenti dall'hardware e dalla piattaforma su cui sono in esecuzione, questo li rende davvero flessibili ed anche potabili da una piattaforma ad un altra, perché risultano essere più piccoli in dimensioni occupate di una macchina virtuale. La maggior parte dei container Docker impiegano meno di un secondo per essere caricati in memoria. Per questa ragione Docker risolta essere potente e molto utilizzato.

Docker è un'applicazione di tipo client-server. Il client Docker comunica con il server Docker, che a sua volta si occupa di eseguire tutto il lavoro di creazione e gestione dei contenitori in base ai comandi ricevuto dal client. Il server Docker è un demone che, se attivato, resta in esecuzione in background nel sistema. Docker mette a disposizione una serie di comandi tramite quali il client può definire richieste. È presente anche una API RESTful lato server tramite la quale è possibile interagire direttamente con il demone docker (dockerd). Docker è molto flessibile, infatti è possibile eseguire il demone Docker e il client direttamente sullo stesso host, oppure collegare il client Docker a un demone remoto in esecuzione su un altro host, ma questo implica che il demone Docker sia esposto al mondo.

2.3.2 Immagine Docker

Le immagini Docker sono alla base dei container. Un'immagine è una raccolta ordinata di modifiche al filesystem di root e include i parametri di esecuzione da utilizzare all'interno di un contenitore.

Un'immagine Docker è rappresentata da filesystem sovrapposti uno sull'altro, come è possibile osservare nella figura 2.11. Alla base è presente il boot file system, boots, che è simile al tipico filesystem boot di Linux/Unix. Nel momento in cui il container è stato creato ed avviato, il boots viene smontato per liberare RAM, in modo da sposare il container in memoria. Sopra di esso Docker esegue un nuovo livello rappresentato dal file system radice, rootfs, che al suo interno ospita il filesystem di un qualsiasi sistema operativo Linux, come Ubuntu oppure Debian. Tradizionalmente all'avvio di un sistema Linux, il filesystem radice viene mondato in modalità di sola lettura (read-only), successivamente viene passato in modalità lettura e scrittura dopo l'avvio. In Docker rootfs rimane in sola lettura. Il livello rootfs viene indicato come immagine base, a partire dalla quale Docker utilizza l'unione dei file system per aggiungere altri filesystem di sola lettura ad di sopra del roofs. Ognuno dei filesystem è chiamato nel mondo Docker immagine. L'unione dei file system è implementato tramite un servizio Linux chiamato "`Union filesystem`". L'Union filesystem permette di combinare in un unico filesystem file e cartelle provenienti da filesystem differenti, in modo da agglomerare i vari livelli in un'unica immagine.

Quando un container viene lanciato tramite l'utilizzo di un'immagine, Docker monta un filesystem in lettura e scrittura in cima alla struttura dell'immagine, qui è dove qualsiasi processo che si vuole lanciare all'interno del contenitore viene eseguito. Il layer di scrittura inizialmente risulta essere vuoto, nel momento in cui si verificano cambiamenti, questi vengono applicati a questo livello. La versione dell'immagine di sola lettura continuerà ancora ad esistere.

Questo è lo schema principale mediante il quale Docker gestisce le sue immagini, è basato su un pattern chiamato "copy on write", ovvero, ogni livello di immagine è in sola lettura e questa immagine non cambia mai. Quando viene creato un container, Docker aggiunge un livello di lettura e scrittura in cima all'immagine. La versione del file in sola lettura continua ad esistere e non viene mai modificata dal contenitore in esecuzione, ma rimane nascosta sotto la versione di copia. Il nuovo livello, combinato con i livelli sottostanti e alcuni dati di configurazione, danno vita al container.

Questo concetto rappresentato, indica il framework di layeding delle immagini, che ci consente di creare in maniera rapida immagini ed eseguire contenitori con varie applicazioni.

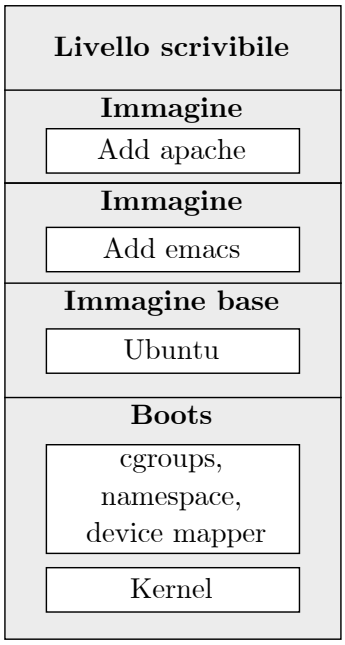

Figura 2.8. Rappresentazione dei livelli di un'immagine Docker

Le immagini Docker possono essere costruite seguendo un set di operazioni, che combinate tra loro danno vita a un'immagine, queste operazioni sono contenute all'interno di un file chiamato `Dockerfile`. Ogni operazione presente all'interno di questo file va ad aggiungere un nuovo livello all'immagine di base scelta. Un esempio di Dockerfile è indicato in figura 2.9, rappresenta la creazione di un'immagine Docker che contiene una applicazione web utilizzata nel contesto di questa tesi.

```
FROM python:2-alpine
MAINTAINER Giovanni Trivigno "<s231595@studenti.polito.it>"
RUN mkdir /logs
WORKDIR /usr/src/app
COPY dare.py .
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
EXPOSE 5000
ENTRYPOINT ["python"]
CMD ["dare.py"]
```

Figura 2.9. Esempio di Dockerfile utilizzato per la creazione di una componente utilizzata in questa tesi.

All'interno del Dockerfile sono presenti diverse righe ognuna delle quali contiene istruzioni accoppiate con argomenti, ogni istruzione viene scritta in maiuscolo ed è seguita da un argomento. Le istruzioni contenute nel Dockerfile vengono elaborate dall'alto verso il basso ed ogni istruzione aggiunge un nuovo livello all'immagine di base. Ciò significa che se la creazione dell'immagine partendo dal Dockerfile si interrompe per qualche motivo, come ad esempio un'istruzione non riesce a completarsi, la creazione dell'immagine non andrà a buon fine, ma i livelli già completati non saranno più processati in quanto già presenti. Questo è molto utile anche in fase di debug e riduce notevolmente i tempi di creazione, perché il processo di build non deve ripartire dall'inizio se le righe al di sopra del punto di interruzione non hanno subito modifiche. Analizziamo ora il contenuto del Dockerfile presente in figura 2.9:

- `FROM` è una parola chiave utilizzata per identificare l'immagine di base a partire dalla quale la nuova immagine viene creata, in questo caso si utilizza un'immagine basata su alpine;
- `MAINTAINER` identifica l'utente proprietario e manutentore dell'immagine, opzionalmente può essere specificato anche l'indirizzo email che può essere utilizzato per contattare l'utente;
- `RUN` rappresenta un comando che viene eseguito all'interno dell'immagine, nell'esempio riportato `RUN` viene utilizzato per creare una nuova directory e installare tramite il comando pip dei pacchetti per python;
- `WORKDIR` permette di impostare la directory di lavoro all'interno del container, tutte le operazioni eseguite successivamente avranno come punto di inizio la directory indicata nel `WORKDIR`, nel caso dell'esempio `/usr/src/app`;
- `COPY`, tale istruzione consente la copia di file locali, contenuti all'interno della directory di build, cioè la directory in cui è contenuto il Dockerfile, all'interno del container stesso. Il simbolo "." presente all'interno dell'esempio del Dockerfile, permette di copiare il file all'interno della directory definita da `WORKDIR`;
- `EXPOSE` specifica una porta da aprire, che verrà utilizzata dall'applicazione all'interno del contenitore realizzato;
- `ENTRYPOINT`, consente di eseguire un comando all'interno del container non appena lo stesso viene avviato, tale istruzione non aggiunge un nuovo layer all'immagine Docker;
- `CMD`, avrà il compito di fornire dei parametri di default al comando espresso da `ENTRYPOINT`, nel caso dell'esempio riportato tale comando avvierà l'applicativo python.

Come abbiamo visto, creare una propria immagine Docker, risulta essere molto semplice, partendo da un'immagine base è possibile aggiungere diverse operazioni necessarie alle applicazioni che verranno eseguite all'interno del container. Al termine della creazione del Dockerfile occorre creare la nuova immagine che sarà realizzata tramite il comando `docker build`. Al momento della creazione dell'immagine Docker è possibile specificare il nome dell'immagine e il nome di un repository. Il nome del repository è fondamentale perché le immagini Docker risiedono all'interno di repository. Nel mondo Docker un repository può essere pubblico o privato, il repository pubblico prende il nome di `Docker Hub` [12], al suo interno possiamo avere due tipo di repository:

- **Top-level repository**: sono repository gestiti da Docker, contengono immagini base utilizzate per creare altre immagini. Le immagini contenute al suo interno sono sempre aggiornate e siamo certi del loro corretto funzionamento perché sono sempre accertate dal team Docker;
- **User repository**: contiene immagini realizzate da utenti, iscritti a Docker Hub, il nome del repository è preceduto dal nome dell'utente. Per caricare nuove immagini nel proprio repository è possibile utilizzare il comando `docker push`.

L'utilizzo del repository pubblico Docker Hub è molto comodo per condividere le immagini con altri utenti. Se non esiste la necessità di avere immagine condivise, perché ad esempio un utente utilizza l'immagine soltanto localmente, è possibile utilizzare immagini private. Docker offre due possibilità: creare dei repository privati all'interno di Docker Hub oppure costruire un proprio registro privato.

Una volta creata la propria immagine Docker, essa può essere eseguita tramite il comando `docker run`, per eseguire il container il demone Docker esegue le seguenti operazioni:

- effettua una ricerca andando a ricercare prima localmente il nome dell'immagine a cui il comando run fa riferimento, in caso in cui non venga trovata verrà eseguita una ricerca sui repository pubblici di Docker Hub;
- crea un nuovo container e carica l'immagine al suo interno;

- aggiunge all'immagine il livello del filesystem che permette di eseguire operazioni di lettura e scrittura;
- crea un'interfaccia di rete, in modo che il container possa comunicare con la macchina locale sul quale è in esecuzione e assegna un indirizzo IP al container;
- carica uno specifico processo.

Ogni container è caricato in memoria allo stesso modo.

2.3.3 Docker Compose

Docker Compose [11] [13] permette di definire tramite un file YAML un insieme di container da avviare, specificando per ogni container alcune proprietà a runtime. All'interno di Docker Compose, ogni container prende il nome di "servizio" ed è definito come un container che interagisce con altri container in base a delle proprietà definite. La grande di potenzialità di Docker Compose è proprio la possibilità di far interagire i container tra loro, specificando una relazione che permette di creare una connettività tra i container senza specificare il loro indirizzo IP.

Un esempio di Docker Compose è riportato nella figura 2.10. Il file `docker-compose.yml` contiene le istruzioni per eseguire due container, nello specifico vogliamo eseguire due servizi: web e redis.

```
web:
  image: jamtur01/composeapp
  command: python app.py
  ports:
   - "5000:5000"
  volumes:
   - .:/composeapp
  links:
   - redis

redis:
  image: redis
```

Figura 2.10. Esempio di un file docker-compose.yml citato da [11].

Come possiamo osservare per la realizzazione di web vengono specificate delle operazioni in runtime, nello specifico andiamo ad indicare l'immagine alla quale facciamo riferimento indicando anche il comando da eseguire all'avvio del servizio. Successivamente viene specificata la porta alla quale il servizio può essere contattato. Come si può osservare il numero di porta è ripetuto per due volte, ciò significa che stiamo creando una relazione tra la porta 5000 all'interno del servizio e la porta 5000 sull'host. È indicato anche il volume per il servizio web, permette di montare tutto il contenuto della cartella nel quale`docker-compose.yml` è presente, all'interno del servizio sotto il percorso `/composeapp`. Il concetto del volume è molto potente, infatti se viene modificato il file sull'host e il container viene riavviato, le modifiche saranno presenti anche all'interno del container. L'uso dei volumi in Docker permette di mantenere i dati in modo persistente. Senza il volume se un file, sotto la directory `/composeapp`, venisse modificato e il container successivamente venisse stoppato, non resterebbe alcuna traccia della modifica apportata. Tramite la parola chiave `links`, andiamo a specificare un collegamento con il servizio redis.

Docker Compose è uno strumento molto utile che permette di eseguire più container contemporaneamente in un'unica operazione, senza andare ad eseguire il comando run per ogni container.

2.3.4 Storage Driver

Come già definito, un'immagine Docker è costituita da una serie di livelli, ognuno dei quali rappresenta un'istruzione dall'interno del Dockerfile. Ogni livello, tranne l'ultimo è di sola lettura. Ogni livello rispetto al livello precedente, è formato da un insieme aggiuntivo di operazioni che vanno quindi a definire delle differenze con il livello sottostante. I livelli sono posizionati uno sull'altro. Ogni volta che viene creato un contenitore, viene aggiunto un nuovo livello scrivibile al di sopra dei livelli sottostanti, tutte le modifiche apportate dal container durante la sua esecuzione, come la scrittura di un nuovo file, la modifica di un file esistente oppure l'eliminazione di un file, vengono realizzate all'interno dello strato scrivibile. Infatti la differenza tra un container e un'immagine, risiede proprio nell'aggiunta del livello scrivibile. Quando un container viene eliminato, viene eliminato anche il livello scrivibile, l'immagine, invece, rimane invariata. Ogni livello ha un suo identificativo univoco universale (UUID), è rappresentato da numeri random di 256 bit, Docker normalmente utilizza una versione ridotta a 12 cifre esadecimali (48 bit), per riferirsi a un'immagine o un contenitore.

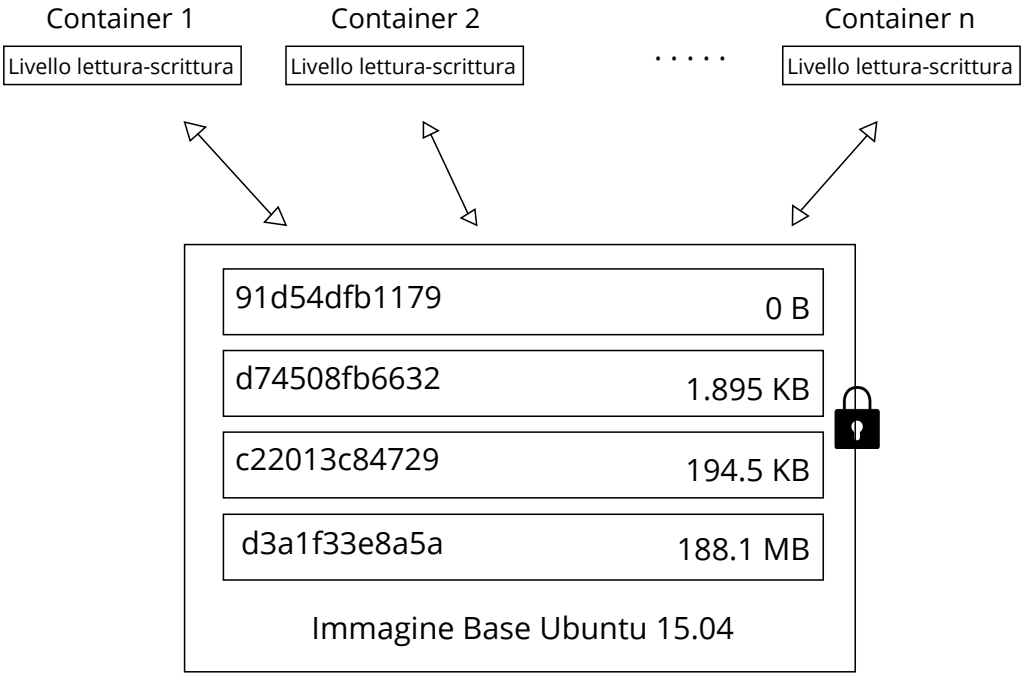

Figura 2.11. Condivisione di un'immagine Docker tra vari container [14].

Poiché ogni container ha un proprio livello scrivibile e tutte le modifiche apportate dal container risiedono in questo livello, più container possono condividere l'accesso alla stessa immagine, la figura 2.11 mostra lo senario appena descritto. Il concetto di condivisione delle immagini, permette a Docker di poter lanciare nuovi contenitori in tempi molto brevi, proprio perché deve soltanto creare un nuovo livello molto sottile che sia accessibile in scrittura e che naturalmente sia di proprietà del container appena lanciato. Oltre alla creazione anche l'operazione di eliminazione di un container è molto rapida perché viene rimosso soltanto il layer di scrittura, che è di proprietà del container, l'intera immagine resta invariata.

Docker utilizza uno storage driver [14] per gestire i livelli delle immagini, i quali risiedono all'interno di una specifica posizione nel filesystem dell'host, tipicamente all'interno della cartella `/var/lib/docker`. Uno storage driver, si occupa di svolgere molteplici compiti: consente di creare dati nel livello scrivibile del contenitore; gestisce i dettagli sul modo in cui i vari livelli interagiscono tra di loro; gestisce il contenuto dei livelli dell'immagine e del livello scrivibile aggiunto per ogni container. Uno storage driver si occupa di eseguire anche operazioni di "copy on write", ovvero viene creata una nuova copia dell'immagine base e viene posta nel livello di scrittura del contenitore che ha fatto richiesta.

Docker supporta un certo numero di storage driver, ognuno dei quali gestisce i vari livelli di immagine e contenitori secondo la propria implementazione. Se si vuole avviare Docker con uno specifico storage driver, occorre aggiungere il flag `--storage-driver`, specificando il nome del driver quando il demone Docker viene mandato in esecuzione all'interno del sistema host.

AUFS è stato il primo storage driver per il quale Docker ha offerto supporto, esso si basa sulla tecnologia del copy-on-write e lavora a livello di file. Quindi se un file viene modificato anche solo leggermente all'interno di un contenitore, esso viene copiato per intero nel livello di scrittura del container. Questa condizione crea un impatto molto negativo sulle prestazioni.

Analizzeremo lo storage driver Device Mapper, il quale viene utilizzato nell'ambito di questa tesi. Device Mapper è un framework basato sul kernel Linux e permette di mappare blocchi di dispositivi fisici in blocchi virtuali. È il secondo storage driver introdotto in docker, anche essendo presente all'interno del kernel Linux, è necessaria una configurazione specifica per utilizzarlo con Docker. Device Mapper memorizza tutte le immagini e i contenitori all'interno di dispositivi virtuali usando la tecnica copy-on-write, quindi ogni file, anche se disponibile nel sistema host, può essere visto come appartenente a un diverso dispositivo virtuale. Un dispositivo virtuale viene creato dallo storage driver a partire da un pool di base, cioè un insieme di risorse definite durante la fase di inizializzazione. Device Mapper funziona a livello di blocco, quindi quando un file è richiesto, il driver copierà l'interno blocco a cui appartiene il file, ciò porta a molteplici benefici, infatti se una piccola porzione del file viene modificata, soltanto il blocco che ha subito la modifica viene copiato, questo incede in modo positivo sulle performance del container. Questo aspetto è una delle grandi differenze che abbiamo con AUFS, che come abbiamo definito lavora a livello di file, quindi se una piccola porzione del file viene modificata verrà comunque copiato l'interno file.

Capitolo 3

Architettura

In questo capitolo verrà rappresentata l'architettura sviluppata nel contesto di questa tesi in base alle specifiche definite dal Trusted Computing Group ed introducendo una Remote Attestation basata sulla virtualizzazione leggera. Più nel dettaglio, nella sezione 3.1 verrà definito il contesto nel quale andiamo a portare migliorie in base allo stato attuale, nella sezione 3.2 verrà rappresentata l'architettura proposta andando a definire ogni modulo realizzato, in conclusione nella sezione 3.3 verrà descritto e rappresentato il processo di comunicazione tra i vari moduli definendo come si svolge il processo di attestazione.

3.1 Descrizione del problema

Il contesto in cui lo scenario della tesi si svolge è quello del mondo Cloud in particolare all'interno dell'architettura NFV. Con la tesi proposta si vuole tentare di estendere il processo di Remote Attestation all'interno dello scenario NFV. L'ETSI ha definito, tramite un elevato numero di standard, tutti i componenti necessari al mondo NFV per funzionare e per gestire il ciclo di vita della funzioni di rete che possono essere istanziate all'interno di questa architettura. Dal punto di vista della sicurezza informatica, abbiamo analizzato tramite il caso d'uso Security as a Service, l'utilizzo di VNF mirate alla sicurezza, che possono essere in grado di contrastare determinati tipi di attacchi informatici oppure possono essere utilizzate come sonde per il monitoraggio della rete.

Per fornire maggiore sicurezza all'interno dell'architettura NFV, abbiamo bisogno di una metodologia per verificare che le funzioni di rete in esecuzione in un certo istante di tempo, si comportino secondo le specifiche e che quindi non siano state manomesse da terze parti. A tale scopo, l'ETSI ha anche definito come potrebbe avvenire il processo di Remote Attestation delle VNF [24]. Per verificare lo stato di fiducia di una VNF secondo ETSI, occorre avere le seguenti informazioni:

- una firma per verificare la validità di una VNF;
- le misure di integrità relative all'hypervisor;
- le misure di integrità relative a tutte le componenti software in esecuzione all'interno della VNF in esame.

La firma viene eseguita dal produttore della funzione di rete ed è utilizzata per determinare la provenienza della VNF. Nel momento in cui viene fatto l'istanziazione della VNF, la firma deve essere validata tramite un certificato offerto dal creatore della VNF. La firma non deve essere trasferita nell'ambito della procedura di Remote Attestation.

Secondo le specifiche ETSI, il processo di RA deve essere suddiviso in due fasi differenti. La prima per recuperare l'elenco delle misure di tutte le componenti software in esecuzione nella VNF e la seconda relativa a verificare lo stato di integrità dell'hypervisor. Quest'ultima è un'operazione fondamentale, perché le VNF sono basate su l'utilizzo di risorse virtuali che fanno riferimento a

risorse fisiche presenti all'interno dell'host sul quale le VNF sono in esecuzione. L'hypervisor si occupa di eseguire questo disaccoppiamento tra risorse fisiche e virtuali, quindi se tale componente dovesse risultare compromessa, tutte le VNF che fanno riferimento a quello stesso hypervisor sono di conseguenza compromesse. Entrambi gli elenchi delle misure devono essere comunicate a un *Verifier* che si occupa di determinare se una VNF risulti fidata o meno.

ETSI NFV propone due tipi di possibili scenari per recuperare le misure che devono essere comunicate al Verifier per determinare lo stato delle VNF in esecuzione all'interno dell'architettura NFV. Il primo consente di ottenere, tramite una comunicazione tra il Verifier e la virtual machine sulla quale la VNF è in esecuzione, sia l'elenco delle misure relative alle componenti software in esecuzione e sia l'elenco delle misure dell'hypervisor. In termini di scalabilità, questo modello non è adatto allo scenario in cui si hanno un elevato numero di macchine virtuali in esecuzione sullo stesso hypervisor, perché il processo per ottenere tutte le misure potrebbe impiegare molto tempo. È possibile osservare che in questo scenario ogni qual volta che occorre verificare l'integrità di una VNF, verrà verificata anche l'integrità dell'hypervisor. Questo potrebbe causare una perdita elevata di risorse se all'interno dell'architettura NFV sono presenti un gran numero di VNF, ma d'altra parte con questo scenario siamo certi che, in un preciso istante di tempo, verrà verificata l'integrità sia della VNF che dell'hypervisor.

Nel secondo scenario il Verifier stabilisce due canali di comunicazione, uno con la virtual machine per recuperare le misure relative al software in esecuzione all'interno della VNF e il secondo con l'hypervisor per recuperare i suoi dati di attestazione. Questo modello risulta adattarsi molto bene alle distribuzioni che utilizzano un numero elevato di macchine virtuali in esecuzione sullo stesso hypervisor. In un determinato periodo di tempo, il Verifier potrebbe prima verificare lo stato di integrità di tutte le macchine virtuali in esecuzione su un determinato hypervisor e al termine verificare l'integrità del hypervisor. In questo scenario l'attestazione dell'hypervisor non causerebbe un elevato spreco di risorse, perché il suo processo di attestazione verrà eseguito soltanto una volta. Il problema principale di questo scenario è che non si ha un legame molto forte tra i dati di attestazione in esecuzione su una data VNF e quelli del rispettivo hypervisor, perché i due elenchi di misure saranno processati in due fasi differenti. Quest'ultimo aspetto potrebbe risultare non sufficiente per verificare l'integrità dell'architettura NFV.

Tutto questo appena descritto rappresenta il processo di RA delle VNF, pensato da ETSI. Durante il processo di RA le misure relative alle componenti software in esecuzione su una VNF devono essere comunicate a una terza parte fidata all'interno dell'architettura NFV, che precedentemente veniva definita come Verifier. Questa componente, deve essere in un secondo momento, in grado di determinare il livello di "trust" di una VNF in base a queste misure e a una lista di misure di riferimento. Questo per verificare che la VNF non sia stata manomessa durante la sua esecuzione. ETSI ha definito che per verificare l'integrità di una VNF è necessario inserire all'interno dell'architettura NFV più precisamente all'interno del dominio MANO, un modulo chiamato "Trust Manager" [25].

Questa entità permette di verificare l'integrità delle VNF in esecuzione in un dato istante di tempo all'interno dell'infrastruttura NFV. Il Trust Manager dovrebbe implementare tutta la logica per verificare l'attendibilità dell'intera architettura NFV. Tutte le altre entità presenti dovrebbero fidarsi di tale componente. Questo porterebbe ad alleggerire la logica delle altre entità all'interno della infrastruttura NFV, perché il Trust Manager sarebbe l'unico componente in grado di verificare che le VNF si comportino secondo le specifiche. Per tale ragione le altre entità sono certe che le VNF eseguano i compiti per le quali sono state progettate. Questo consente di avere una semplificazione nel determinare lo stato di fiducia di ogni VNF, perché sarebbe possibile comunicare con il Trust Manager per essere certi che tutte le VNF siano integre. L'utilizzo del Trust Manager potrebbe portare anche a dei possibili svantaggi. Infatti, utilizzando questo componente come singola entità in grado di determinare lo stato di fiducia delle VNF, in caso di guasto o di un possibile attacco informatico che causi il suo inutilizzo, non sarà più possibile determinare il livello di fiducia dell'infrastruttura NFV. Per tale motivo risulta fondamentale proteggere il Trust Manager da attacchi.

L'ETSI fornisce solo una definizione dell'utilizzo di questo modulo, ma allo stato attuale non è presente nessun componente che permette di svolgere tale processo. Per quanto riguarda la RA l'ETSI suggerisce una possibile soluzione [24] ma per ora non è stato condotto nessun sviluppo in

merito all'attestazione delle funzioni di rete. È presente, per tale ragione, un gap tra la definizione e l'implementazione di un componente che permette di verificare lo stato di integrità di tale infrastruttura.

Mediante lo sviluppo di questa tesi vogliamo tentare di colmare questa mancanza, implementando un modulo che si occupi di attestare l'integrità delle VNF in esecuzione e sia anche in grado di comunicare con gli altri componenti presenti nell'architettura NFV. Il modulo realizzato prende il nome di Trust Monitor. Tale modulo svolge il processo di attestazione per verificare se le VNF risultino fidate o meno, per fare questo si è seguito il concetto di Remote Attestation definito dal TCG. La differenza rilevante tra il processo di RA definito da ETSI e quello proposto in questa tesi, è rappresentato dal concetto di attestazione leggera. Infatti le VNF istanziate non saranno presenti su un hypervisor che si occupa di effettuare un disaccoppiamento tra le risorse virtuali e fisiche, ma verranno istanziate tramite Docker, quindi direttamente sull'host. Per tale ragione non avverrà la comunicazione con l'hypervisor per attestare la sua integrità, perché tale componente non sarà presente, ma alternativamente verrà attestata anche l'integrità del host fisico sulla quale le VNF sono istanziate.

3.2 Architettura del Trust Monitor

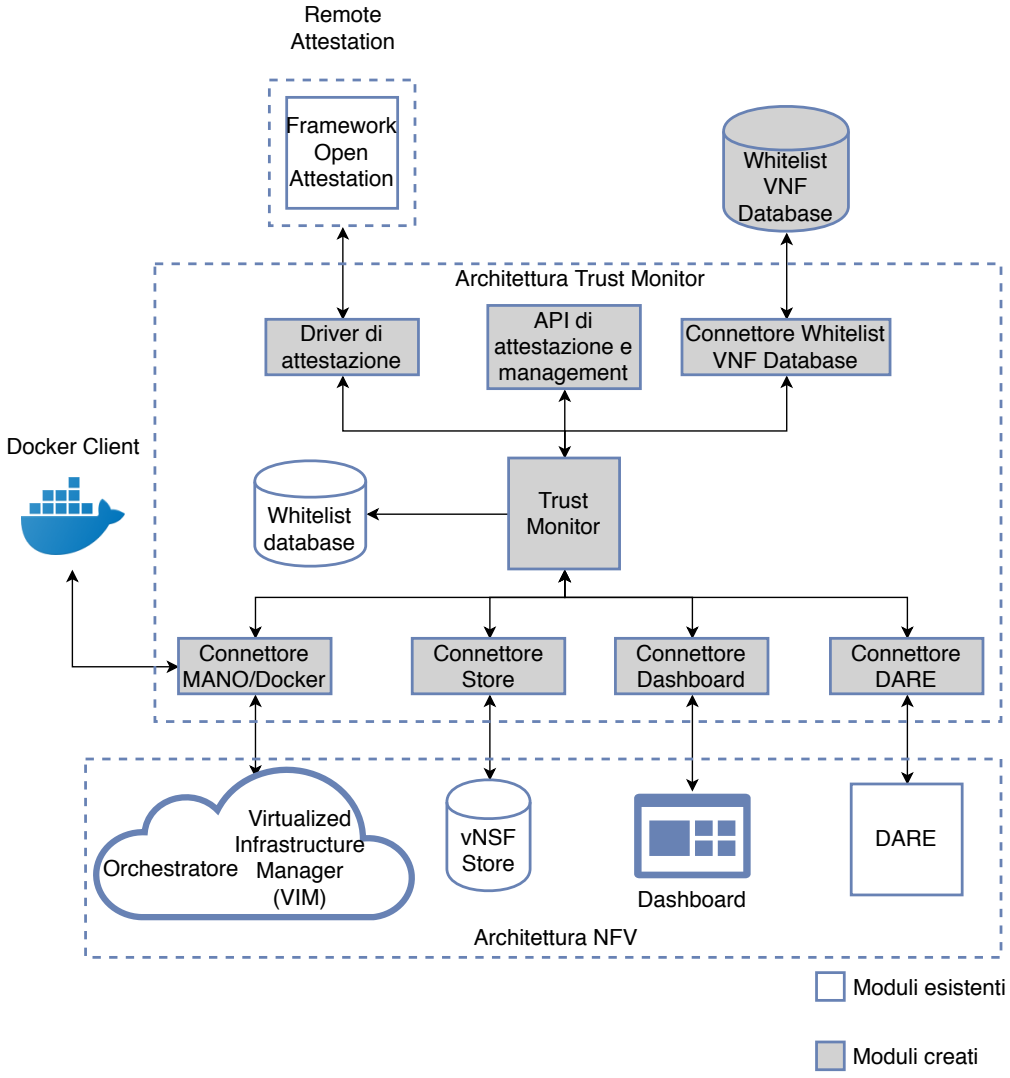

Figura 3.1. Architettura complessiva del Trust Monitor.

In figura 3.1 è rappresentata l'architettura complessiva del Trust Monitor, nella quale sono anche presenti tutti i componenti con cui il modulo sviluppato si interfaccia. I componenti con sfondo grigio sono stati sviluppati nell'ambito di questa tesi, gli altri rappresentano componenti esistenti. In questa sezione verrà analizzato ogni modulo presente nell'architettura andando a descrivere il suo ruolo all'interno dello scenario proposto. Il Trust Monitor permette di andare a realizzare il processo di attestazione all'interno dello scenario NFV, andando ad attestare VNF in esecuzione su host fisici. Nella figura è possibile osservare la presenza di tre insiemi di componenti. Abbiamo il componente che permette di eseguire il processo di Remote Attestation rappresentato dal framework di attestazione, i componenti principali dell'architettura NFV ed infine l'architettura generale del Trust Monitor. Il Trust Monitor per interfacciarsi con i componenti dell'architettura NFV e con il framework di attestazione utilizza dei connettori.

3.2.1 Framework di Attestazione

Il Trust Monitor per verificare l'integrità delle funzioni di rete presenti all'interno di un nodo fisico deve utilizzare un framework di attestazione che si occupi di avviare il processo di RA. La potenzialità del Trust Monitor è che non è vincolato ad utilizzare un particolare framework di attestazione, ma è possibile collegare a tale componente più framework di attestazione. Nel contesto di questa tesi, il componente che permette di svolgere il processo di RA è rappresentato dal Framework di attestazione Open Attestation (OAT) [21].

Il Framework OAT è stato sviluppato da Intel nel 2010 ed il suo scopo è quello di offrire un servizio di RA per la sicurezza nel cloud. OAT è in grado di ottenere l'elenco delle misure del software in esecuzione all'interno di tutte le piattaforme dotate del chip TPM 1.2. Il framework OAT per il suo funzionamento fornisce tutti i componenti suggeriti dal TCG per implementare il servizio di RA, più precisamente implementa:

- HostAgent, rappresenta la componente software in esecuzione all'interno degli host fisici che devono essere attestati.
- PrivacyCA, componente utilizzata per generare certificati associati alle chiavi AIK delle piattaforme da attestare.
- Tabella "Whitelist", che suo interno contiene i valori, considerati validi, dei registri PCR relativi agli host fisici da attestare.
- Appraiser, si tratta del componente che inizia il processo di attestazione contattando una data piattaforma e fornisce indicazioni sull'esito di tale processo.

OAT fornisce un set di API per la comunicazione tra l'HostAgent e l'Appraiser.

Il Framework OAT utilizzato nel contesto di questa tesi è rappresentato da un estensione del Framework OAT nativo, realizzato durante il progetto europeo SECURED [22]. Inizialmente il Framework OAT si occupava di verificare l'integrità esclusivamente di host fisici, ma per estendere il suo funzionamento al mondo virtuale è stato necessario includere durante il processo di attestazione sia le misure relative al host fisico e sia quelle relative ai nodi virtuali. I nodi virtuali attestati da OAT sono basati su container Docker.

Nel contesto di questa tesi, per determinare lo stato di fiducia delle funzioni di rete istanziate all'interno di host fisici è necessario che vengano dispiegate all'interno di container Docker. Docker è un'applicazione di tipo client-server, l'entità server è rappresentata dal demone Docker in esecuzione all'interno di ogni host fisico presente all'interno dell'infrastruttura NFV. Per tale ragione è necessario comunicare con il demone Docker al fine di recuperare informazioni aggiuntive relative alle misure delle VNF presenti all'interno di container Docker. Il modulo che ci permette di effettuare questa comunicazione è rappresentata dalla componente client Docker presente in figura 3.1.

Riassumendo il framework di attestazione è il componente responsabile di svolgere il processo di comunicazione con l'host contenente le VNF, allo scopo di ottenere l'elenco delle misure del software

in esecuzione nelle stesse. Tali misure vengono utilizzate per verificare se le VNF istanziate sull'host fisico, risultino essere affidabili o meno. Per svolgere questo processo ogni componente all'interno dell'architettura NFV deve comportarsi secondo le specifiche definite dal TCG.

3.2.2 Componenti Architettura NFV

Allo scopo di ottenere un processo di attestazione rivolto al cloud NFV è necessario che il Trust Monitor comunichi con i moduli presenti all'interno dell'architettura NFV, questo perché tali componenti permettono al Trust Monitor di recuperare tutte le informazioni necessarie per verificare l'integrità delle VNF in esecuzione.

All'interno dell'architettura NFV l'Orchestratore è il componente incaricato di istanziare le VNF all'interno di una dato host. Il suo compito è quello di mandare in esecuzione all'interno di un host fisico, più precisamente di un dato VIM, uno o più Network Service (NS) i quali sono basati su una o più VNF. Per farlo utilizza un file chiamato descriptor dove al suo interno sono contenute tutte le informazioni che identificano una VNF o un NS. Nel descriptor sono presenti ad esempio informazioni su quale Sistema Operativo è basata una VNF oppure quante risorse virtuali occorre assegnare a una data VNF. Nel momento in cui un NS viene istanziato all'interno di un VIM, vengono anche dispiegate tutte le VNF che formano il servizio di rete. L'Orchestratore al suo interno tiene traccia di tutti i NS che sono in esecuzione in un dato istante di tempo e mantiene anche l'associazione tra il NS e il VIM sul quale è in esecuzione. L'Orchestratore si occupa di gestire il ciclo di vita di un NS. Nel momento in cui un NS viene eliminato il VIM sul quale il NS è in esecuzione si occuperà di liberare le risorse utilizzate dal NS e dalle VNF eseguite al suo interno.

Il Trust Monitor deve comunicare con l'Orchestratore allo scopo di ottenere l'elenco delle VNF istanziate in dato istante di tempo su un dato host, dunque su un dato VIM. Un vincolo fondamentale è che su ogni host deve essere presente un VIM che permetta di mandare in esecuzione all'interno dell'host delle VNF.

Come è possibile osservare nell'immagine oltre all'Orchestratore e al VIM gli altri componenti dell'architettura NFV che comunicano con il Trust Monitor sono, il DARE, la Dashboard e il vNSF Store. Il DARE e la Dashboard ricevono dal Trust Monitor informazioni relative all'esito del processo di attestazione. Il vNSF Store permette al Trust Monitor di recuperare informazioni aggiuntive in merito alle VNF in esecuzione sul host di cui si è richiesta un'analisi di integrità. Tali informazioni sono rappresentate dall'elenco di digest relativi al software custom in esecuzione all'interno della VNF in esame.

3.2.3 Trust Monitor API e database

L'ultimo dominio che andiamo ad analizzare è quello Trust Monitor. Il Trust Monitor è il componente cardine di questa architettura e al suo interno contiene, oltre alle varie funzioni che gli permettono di interfacciarsi con ogni componente, anche tutta la logica per svolgere il processo di verifica di integrità di un host fisico contenente le VNF.

Il Trust Monitor all'interno della sua architettura fa riferimento a un database utilizzato durante la fase di verifica di integrità, che al suo interno contiene un elenco di misure di riferimento utilizzate per verificare il software in esecuzione all'interno delle VNF e all'interno dell'host fisico. Questo database prende il nome di Whitelist Database, le misure presenti al suo interno sono comunicate sotto forma di digest dai produttori del software stesso.

Il Trust Monitor espone a una terza entità delle API utilizzate per richiedere l'avvio della procedura di attestazione e per svolgere operazioni di management. Per il suo corretto funzionamento il Trust Monitor utilizza un database interno allo scopo di memorizzare informazioni relativi ai nodi di rete presenti nell'infrastruttura NFV che necessitano di essere attestati. Il database è anche utilizzato per memorizzare informazioni, più precisamente dei digest, relativi al software custom in esecuzione all'interno dell'host fisico.

3.2.4 Connettori

L'architettura del Trust Monitor per interfacciarsi con il componente che si occupa di svolgere il processo di RA e per interfacciarsi con le entità dell'architettura NFV, necessita di connettori che fungono da tramite tra queste entità.

I connettori presenti nell'architettura del Trust Monitor offrono differenti funzionalità:

- Connettore MANO/Docker: questo connettore permette al Trust Monitor di interfacciarsi con l'Orchestratore presente all'interno dell'architettura NFV e con il client Docker. La comunicazione con l'Orchestratore permette al Trust Monitor di ottenere le informazioni sui VIM nei quali le VNF sono in esecuzione e di recuperare l'elenco delle informazioni relative alle VNF in esecuzione all'interno di ogni host fisico presente nell'infrastruttura NFV. La comunicazione con il client Docker è necessaria per recuperare le informazioni sui container in esecuzione su cui sono eseguite le VNF. Questo aspetto è necessario per considerare durante il processo di attestazione il software in esecuzione all'interno dei container Docker. Nel contesto di questa tesi la comunicazione con Docker è fondamentale perché, per eseguire il processo di RA utilizziamo il Framework OAT esteso che come detto permette di considerare in fase di attestazione il software relativo a determinati container Docker. La comunicazione con Docker ci permette di svolgere successivamente il processo di attestazione delle VNF.

- Connettore Store: permette al Trust Monitor di interfacciarsi con il vNSF Store in modo da ottenere informazioni presenti all'interno del descriptor della VNF in esame. La comunicazione tra il connettore Store e il vNSF Store permette di andare a recuperare dati aggiuntivi, più precisamente digest relativi al software in esecuzione all'interno delle VNF. Questo nell'ottica in cui il tale digest non sia presente nel Whitelist Database.

- Connettore Whitelist VNF Database: viene utilizzato per memorizzare all'interno di un database l'elenco delle misure prelevate dal descriptor che identifica ogni VNF, allo scopo da considerare i digest rappresentanti il software custom in esecuzione all'interno delle VNF durante il processo di verifica di integrità.

- Connettore DARE: viene utilizzato per comunicare al DARE l'esito del processo di attestazione. Sia nel caso in cui una VNF viene indicata come compromessa che nel caso in cui la VNF risulti trusted. Questo tipicamente viene fatto a scopo di monitoraggio della rete.

- Connettore Dashboard: viene utilizzato per comunicare alla Dashboard dell'architettura NFV l'esito del processo di attestazione nel caso in cui la VNF risulti "untrusted". In questo caso l'amministratore di rete può decidere se escludere dalla rete la VNF, in quanto potrebbe essere stata manomessa da terze parti.

- Driver di attestazione: questo componente svolge un ruolo chiave all'interno dell'architettura, infatti consente al Trust Monitor di recuperare l'elenco dei digest di tutte le VNF da attestare e dell'host fisico presente all'interno dell'infrastruttura NFV nella quale le VNF sono in esecuzione. Viene utilizzato per far comunicare il Trust Monitor con il framework di attestazione.

La potenzialità del Trust Monitor è che tramite lo sviluppo di un driver di attestazione è possibile utilizzare qualsiasi framework di attestazione desiderato. Questo perché la sua logica implementativa gli consente di adattarsi a qualsiasi framework di attestazione semplicemente andando a sviluppare il driver di attestazione apposito come è possibile osservare in figura 3.2. All'interno del Trust Monitor possiamo avere anche più framework di attestazione, quindi potenzialmente eseguire processi di attestazione basati su framework differenti allo stesso istante. Nel contesto di questa tesi si è posto attenzione solamente al Framework OAT per cui è stato realizzato un solo driver di attestazione.

La genericità del Trust Monitor permette anche di estendere il processo di RA anche ad altri componenti dell'infrastruttura NFV come ad esempio switch di rete oppure Software Defined Networking (SDN).

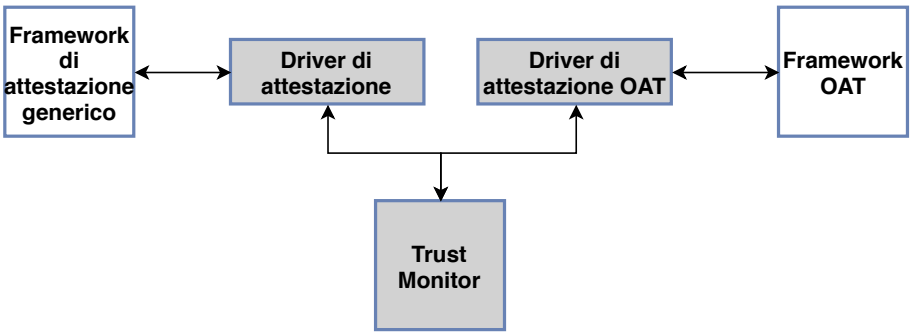

Figura 3.2. Rappresentazione grafica genericità del driver.

3.3 Processo di attestazione del Trust Monitor

Una volta definiti tutti i moduli presenti all'interno dell'architettura del Trust Monitor e aver definito come si interfacciano con i componenti dell'architettura NFV andiamo a descrivere più nel dettaglio come avviene la comunicazione tra tutti questi componenti.

Un terzo componente può comunicare con il Trust Monitor richiedendo tipi di attestazione differenti. È possibile svolgere il processo di attestazione di un determinato host fisico contenente VNF presente all'interno del NFVI, è possibile eseguire il processo di attestazione di tutti gli host fisici presenti nell'infrastruttura NFV oppure scegliere di attestare un set preciso di host fisici nell'infrastruttura NFV specificando informazioni aggiuntive sulle VNF istanziate al loro interno. Tutti questi scenari di attestazione comportano dei passaggi di comunicazione differenti tra i vari moduli presenti all'interno dell'architettura del Trust Monitor.

3.3.1 Requisiti iniziali

Abbiamo detto precedentemente che il Framework di attestazione OAT è rappresentato da un'estensione del framework nativo sviluppato nell'ambito del progetto europeo SECURED. Tale estensione permette di considerare in fase di attestazione di un nodo anche i container Docker lanciati al suo interno, quindi nel nostro caso possiamo attestare le VNF dispiegate all'interno del nodo di rete con la limitazione che vengano eseguite all'interno di container Docker. Per considerare i container Docker nei quali sono presenti le VNF, durante la comunicazione con il Framework OAT occorre specificare l'ID di tali container. Il problema principale è che quando verifichiamo l'integrità di un di un host fisico contenente le VNF non sappiamo quali sono gli ID dei container Docker nei quali le VNF vengono eseguite. Per risolvere questo inconveniente occorre riuscire a comunicare con il demone Docker che esegue e gestisce i container all'interno di un dato host così che, tramite esso, siamo in grado di ottenere l'ID di ogni container in esecuzione all'interno del nodo. Per rendere possibile quanto appena descritto il demone Docker deve essere presente su ogni host fisico presente nell'infrastruttura NFV e deve anche essere contattabile tramite il client Docker. Per eseguire una VNF occorre necessariamente istanziare un container Docker che conterrà la VNF e successivamente recuperare l'ID associato al container creato. Un vincolo fondamentale è che per permettere la creazione di container Docker nei quali dispiegare VNF, su ogni host fisico deve essere presente un VIM che riesca a creare container Docker, così facendo sarà possibile considerare le VNF durante il processo di attestazione.

Ogni VIM deve essere registrato con l'Orchestratore, in modo che successivamente tramite quest'ultimo sarà possibile mandare in esecuzione una determina VNF su un particolare VIM dunque, nel nostro caso, su un dato host. Per svolgere l'intero processo di attestazione è necessario che i nodi di rete, più precisamente gli host fisici presenti nell'infrastruttura NFV siano registrati con il Trust Monitor, in caso contrario non sarà possibile eseguire il processo di attestazione.

Nel momento in cui un nuovo host viene inserito all'interno della rete deve anche essere registrato presso il Trust Monitor andando ad indicare vari parametri tra cui il nome del nodo e il suo indirizzo IP. Il Trust Monitor memorizza i nodi di rete all'interno di un database interno. Tale

database è utilizzato anche per memorizzare particolari digest relativi al software proprietario in esecuzione sul nodo di rete presente all'interno dell'infrastruttura NFV.

3.3.2 Processo comunicazione connettori

Andiamo ora a descrivere come avviene il processo di comunicazione tra i vari moduli presenti all'interno dell'architettura allo scopo di eseguire l'attestazione delle VNF contenute in un dato host. Prima di andare ad eseguire il processo di attestazione occorre recuperare delle informazioni aggiuntive che saranno utilizzate per avviare la comunicazione con il Framework OAT e per verificare che i moduli software in esecuzione all'interno di una VNF siano affidabili.

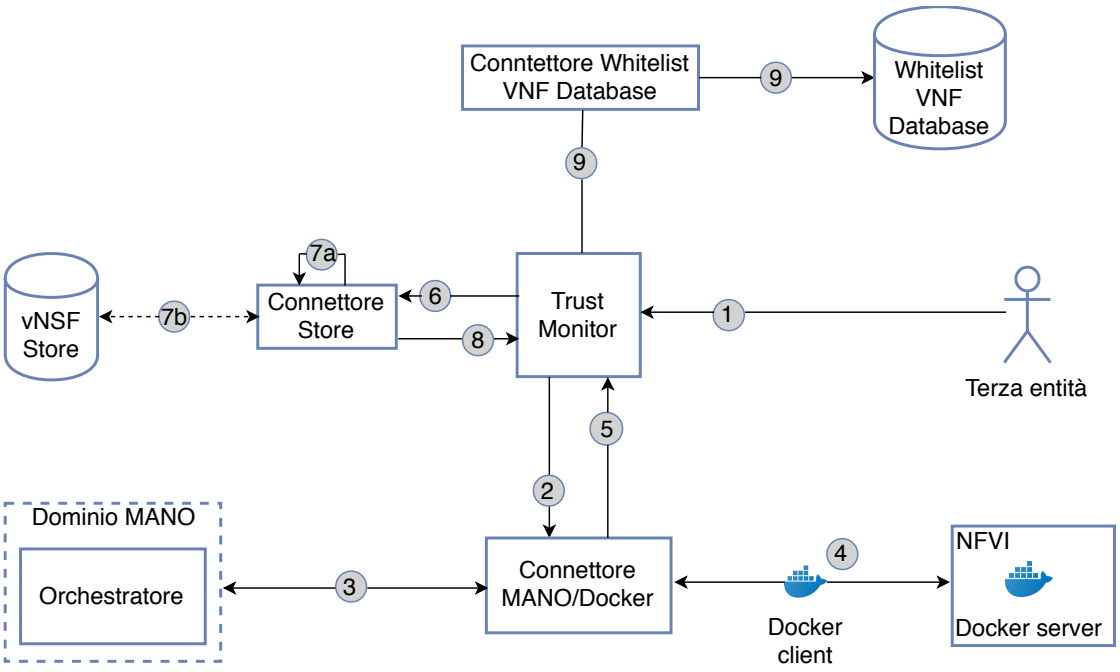

Figura 3.3. Processo di comunicazione tra i vari componenti del Trust Monitor.

Osservando la figura 3.3 è possibile osservare i passaggi necessari per recuperare tutte le informazioni necessarie al Framework OAT per avviare il processo di attestazione verso un dato host fisico presente all'interno dell'infrastruttura NFV.

- Contattare Trust Monitor (freccia 1): è la fase iniziale del processo di attestazione. Una terza entità che è interessata a verificare lo stato di un particolare host presente nell'infrastruttura NFV contatta il Trust Monitor inviandogli il nome del nodo con il quale è stato precedentemente registrato presso il Trust Monitor.

- Contattare connettore MANO/Docker (freccia 2): il Trust Monitor riceve il nome del nodo e avvia il processo di attestazione. Come prima operazione verifica se il nodo è stato precedentemente registrato sul Trust Monitor, in caso negativo invierà un messaggio di errore all'entità richiedente e l'intero processo verrà interrotto. Se questa fase ha esito positivo allora verrà contattato il connettore MANO/Docker che si occuperà di ottenere l'elenco dei nomi delle VNF istanziate sul nodo richiesto e l'elenco degli ID con i quali, i container Docker che contengono le VNF sono stati creati sul host. Al connettore MANO/Docker verrà inviato l'indirizzo IP associato al nodo dato, recuperato mediante una ricerca all'interno del database del Trust Monitor.

- Recuperate informazioni sulle VNF (freccia 3): il connettore MANO/Docker comunica con l'Orchestratore per recuperare l'elenco dei nomi delle VNF dispiegate sul nodo di rete. Per

effettuare questa operazione occorre prima di tutto recuperare l'elenco di tutti i VIM registrati con l'Orchestratore. Dopo di che viene fatto un confronto attraverso l'indirizzo IP del nodo dato e l'IP associato ad ogni VIM, allo scopo di individuare il nome del VIM associato all'host fisico. Questa operazione è importante perché le VNF vengono istanziate tramite NS che vengono mandati in esecuzione su un dato VIM. Dato che su ogni nodo di rete all'interno dell'infrastruttura NFV è presente un VIM, è fondamentale capire il NS in esecuzione all'interno di un dato host. Una volta fatto questo è possibile individuare i nomi delle VNF istanziate sul VIM, quindi di conseguenza sul nodo fornito.

- Contattare Docker (freccia 4): in questa fase il connettore MANO/Docker attraverso l'utilizzo del client Docker comunicherà, mediante l'utilizzo dell'indirizzo IP del nodo da attestare, con il demone Docker presente sul host fisico. Questa fase ha lo scopo di recuperare l'ID dei container Docker contenenti le VNF istanziate.
- Comunicare informazioni ottenute (freccia 5): l'elenco contenente il nome delle VNF associate al nodo di rete dato e l'elenco degli ID corrispondenti ai container Docker vengono comunicati al Trust Monitor.
- Contattare connettore Store (freccia 6): il Trust Monitor trasmette al connettore Store l'elenco contenente i nomi delle VNF allo scopo di ottenere maggiori informazioni sulle stesse.
- Recuperare i digest delle VNF (freccia 7a o 7b): il connettore Store è il componente incaricato di recuperare l'elenco dei digest relativi al software in esecuzione all'interno delle VNF. Tali digest vengono prelevati dal file descriptor che descrive la VNF in esame. Si tratta di un descriptor particolare dove al suo interno sono presenti le misure del software. Il descriptor contenente tali informazioni prende il nome di *security manifest*. Nell'ambito di questa tesi (7a) il recupero di questi digest è gestito internamente, praticamente non avviene la comunicazione con un vero vNSF Store (7b).
- Comunicare elenco digest al Trust Monitor (freccia 8): l'elenco delle misure relative al software in esecuzione all'interno delle VNF viene comunicato al Trust Monitor.
- Memorizzare digest (freccia 9): i digest delle componenti software in esecuzione all'interno delle VNF vengono memorizzati in un database, in modo tale da considerarli durante il processo di verifica di integrità. Se tali digest sono già presenti all'interno del database non verrà aggiunto nessun digest al suo interno.

3.3.3 Processo comunicazione OAT

La comunicazione descritta delle precedente sezione ci permette di riuscire a contattare, avendo tutte le informazioni necessarie, il Framework OAT che si occupa di ottenere l'elenco delle misure sia del nodo fisico che delle VNF eseguite al suo interno. Prima di descrivere come avviene il processo per verificare se un'entità risulti essere "trusted" o meno, occorre prestare attenzione a come avviene la comunicazione tra OAT e l'host fisico.

Come è possibile osservare in figura 3.4 il Framework OAT comunica con l'host fisico presente nell'infrastruttura NFV in modo da ottenere l'elenco dei digest del software in esecuzione al suo interno e nelle VNF. Tali informazioni vengono comunicate sotto forma di un Integrity Report, perché OAT richiede una struttura di questo tipo. Il Framework OAT verifica che le informazioni ricevute provengano dal host contattato, questo viene fatto attraverso la validazione tramite certificato della firma eseguita con la chiave AIK del TPM inviata dal nodo. Se questa fase fornisce esito positivo viene validato anche il contenuto dell'Integrity Report. All'interno del host è necessario l'utilizzo del modulo kernel IMA che si occupa di memorizzare tutti i digest relativi alle componenti in esecuzione all'interno del host fisico analizzato. È necessaria anche la presenza del modulo TPM in modo da essere certi che l'elenco delle misure del software in esecuzione non venga alterato durante il transito nella rete. Il TPM permette anche di calcolare l'aggregato sul PCR10 utilizzato successivamente dal Framework OAT per verificare la validità delle misure. È possibile notare che questo rispecchia in pieno il processo di RA definito dal TCG. All'interno della figura è presente anche Docker, questo perché nel contesto di questa tesi le VNF vengono

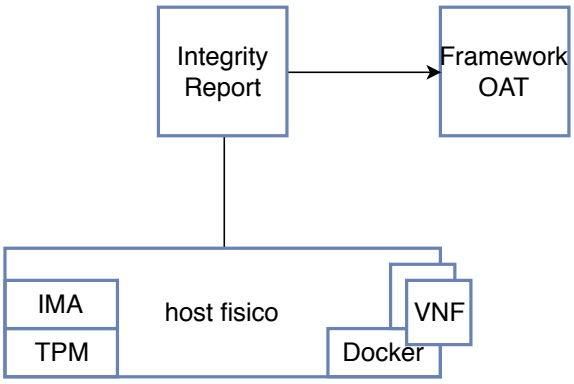

Figura 3.4. Comunicazione tra host fisico contenente VNF e il Framework OAT.

istanziate utilizzando Docker. Docker permette di eseguire una virtualizzazione basata su container nei quali è possibile eseguire software arbitrario, il tutto avviene senza l'utilizzo di uno strato intermedio di virtualizzazione basato su un hypervisor. Si tratta dunque di una "virtualizzazione leggera". Si è utilizzato Docker perché permettere la creazione di container nei quali eseguire le VNF direttamente sull'host fisico. Questo consente al modulo kernel IMA di andare a misurare il software in esecuzione all'interno dei container al pari dei processi eseguiti direttamente sull'host stesso e per tale ragione queste misure possono essere prese in considerazione durante il processo di attestazione.

Riassumendo, ogni host presente all'interno dell'infrastruttura NFV deve essere dotato del modulo TPM e del modulo kernel IMA, quindi ogni nodo deve essere di conseguenza dotato di un Sistema Operativo basato su Linux. In questo modo l'host fisico è in grado di raccogliere le misure di integrità sia del codice in esecuzione che della configurazione, tutto ciò allo scopo di comunicarle a una terza entità in modo sicuro e affidabile. Nel nostro caso tale entità è rappresentata dal Framework OAT. L'Integrity Report contenente tutte le misure deve viene inviato al Trust Monitor che si occuperà di effettuare un confronto per verificare che le misure di integrità corrispondano a quelle presenti all'interno del Whitelist Database di riferimento.

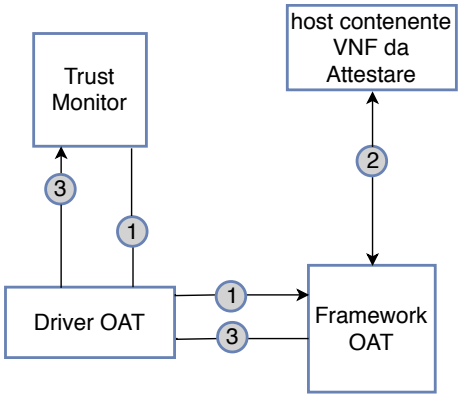

Figura 3.5. Processo di comunicazione tra Framework OAT e Trust Monitor.

Il Trust Monitor comunica con il Framework OAT allo scopo di richiedere l'attestazione di un dato host. In figura 3.5 è rappresentato il processo di comunicazione tra quete due entità e si svolge in questo modo:

- Contattare OAT per eseguire l'attestazione (freccia 1): il Trust Monitor comunica con il driver OAT al fine di avviare il processo di attestazione verso il nodo di rete indicato, fornendo al Framework OAT il nome del nodo da attestare e l'elenco degli ID dei container Docker in esecuzione all'interno del nodo. Queste informazioni sono state ottenuto nel passaggio precedente (figura 3.3)

- Comunicazione tra OAT e l'host fisico da attestare (freccia 2): OAT avvia il processo di attestazione richiedendo al nodo da attestare l'elenco delle sue misure. L'host comunica le misure al Framework di attestazione sotto forma di Integrity Report, a questo punto il Framework OAT valida la risposta.
- Invio Integrity Report al Trust Monitor (freccia 3): l'Integrity Report risultante dal processo di attestazione tra il Framework OAT e il nodo da attestare viene trasmesso al Trust Monitor.

3.3.4 Processo di verifica

Occorre ora andare a considerare come avviene il processo di comunicazione finale in modo da determinare se un host fisico presente all'interno dell'infrastruttura NFV risulti essere fidato o meno. In figura 3.6 è presente quest'ultimo passaggio.

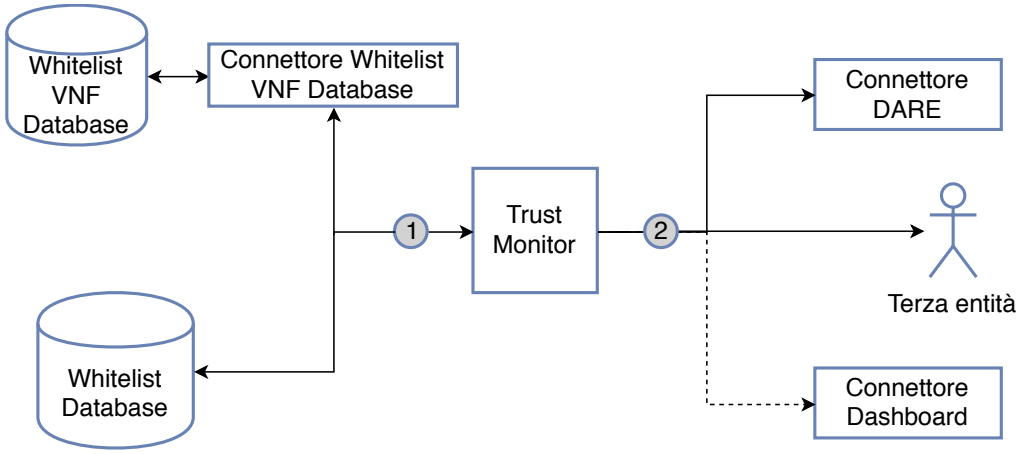

Figura 3.6. Processo di attestazione e verifica integrità.

- Verifica misure con Database (freccia 1): il Trust Monitor estrae, dall'Integrity Report ottenuto attraverso la comunicazione con il framework OAT, l'elenco delle misure relative al software in esecuzione sul nodo e sulle VNF. Ogni misura contenuta nell'elenco delle misure viene confrontata con le misure presenti all'interno di due database. Il primo il Whitelist Database, contiene le misure di riferimento ottenute da repository ufficiali, mentre il secondo, chiamato Whitelist VNF Database, contiene le misure che identificano il software in esecuzione all'interno delle VNF. Quest'ultime sono state memorizzate nella fase precedente dopo la comunicazione con il connettore Store (sezione 3.3.2). La comunicazione tra il Whitelist VNF Database e il Trust Monitor avviene attraverso l'utilizzo di un connettore.
- Comunicazione esito attestazione (freccia 2): il Trust Monitor in questa fase ha tutte le informazioni necessarie per verificare lo stato di integrità del nodo presente all'interno dell'infrastruttura NFV, infatti dispone di tutte le misure inviategli dal Framework OAT e l'elenco delle corrispettive misure di riferimento. Se il confronto tra questi due elenchi ha esito positivo significa che il nodo è "trusted" in caso contrario se abbiamo anche una sola divergenza il nodo è considerato "untrusted". L'esito di questa fase viene comunicato alla terza entità che ha richiesto di eseguire il processo di verifica di integrità ed anche al connettore DARE. Se il nodo è risultato "untrusted" l'esito di attestazione verrà comunicato anche alla Dashboard, questo passaggio avviene attraverso il connettore Dashboard.

Il risultato della procedura di attestazione in caso in cui il processo fornisse un esito negativo, oltre ad indicare lo stato di integrità di un host includerà anche dei dettagli aggiuntivi che indicano il motivo per cui l'entità è stata etichettata come "untrusted".

Lo scenario appena descritto rappresenta il processo di attestazione realizzato in questa tesi per un singolo host fisico presente nell'infrastruttura NFV. Nel caso in cui venga richiesto di attestare

tutti i nodi presenti all'interno dell'infrastruttura NFV occorrerà ripete questa fase per ogni VIM registrato all'Orchestratore, perché ogni VIM potenzialmente potrebbe avere al suo interno delle VNF istanziate e dunque potrebbe rappresentare un host fisico presente in NFVI. È possibile attestare anche un insieme di host registrati con il Trust Monitor, in questo caso la procedura è del tutto analoga a quella descritta precedentemente, l'unica differenza risiede nel non dover comunicare con il demone Docker in esecuzione all'interno dell'host, perché la lista contenente gli ID dei container Docker in esecuzione all'interno del nodo viene trasmessa nel momento in cui viene richiesta l'attestazione.

Capitolo 4

Implementazione

In questo capitolo verranno approfonditi gli aspetti implementativi relativi alla realizzazione del Trust Monitor e dei suoi componenti già discussi nel capitolo precedente. Andremo a descrivere nel dettaglio come contattare il Trust Monitor per avviare il processo di comunicazione al fine di determinare se un'entità risulti essere fidata o meno. Più in particolare all'interno della sezione 4.1 verranno discusse le scelte implementative relative ai componenti realizzati all'interno di questa tesi, nella 4.2 verranno descritte tutte le API offerte dal Trust Monitor, nella sezione 4.3 verranno descritte le API offerte da OAT allo scopo di registrate e attestare un host all'interno dell'infrastruttura NFV e in conclusione nella sezione 4.4 sarà rappresentata l'implementazione dei connettori presenti nell'architettura del Trust Monitor, focalizzandoci sulle API utilizzate per comunicare con tali componenti.

4.1 Scelte implementative

In questa sezione verranno trattate le varie scelte implementative dei moduli software realizzati nel contesto di questa tesi. Ogni modulo presente all'interno dell'architettura del Trust Monitor è un Web Service sviluppato in linguaggio Python. Per la realizzazione di Web Service, il linguaggio Python mette a disposizione diversi tipo di framework che assistono il programmatore nella realizzazione di un servizio web, tra quelli più utilizzati abbiamo Django [27], Flask [28] e Django REST Framework (DRF) [26]. La scelta di quale framework utilizzare è mirata esclusivamente al tipo di modulo che bisogna implementare, questo perché ogni framework offre al programmatore differenti funzionalità.

I connettori presenti all'interno dell'architettura del Trust Monitor sono dei moduli che vengono contattati esclusivamente dal Trust Monitor e hanno lo scopo di comunicare con le altre entità presenti nell'architettura NFV al fine di recuperare particolari informazioni che permettono di perfezionare la procedura di attestazione. Per la realizzazione dei connettori si è deciso di utilizzare Flask, perché offrono pochi servizi e mettono a disposizione del Trust Monitor poche API per svolgere i loro compiti. Si è utilizzato Flask in quanto permette di realizzare Web Service leggeri che mettono a disposizione di una terza entità poche API. La sua grande potenzialità è che è possibile creare un Web Service in maniera molto rapida e veloce.

Il Trust Monitor è il modulo più complesso realizzato nel contesto di questa tesi, perché mette a disposizione vari servizi forniti tramite l'utilizzo di differenti API. Al suo interno è anche presente un database utilizzato per memorizzare informazioni sui nodi registrati con lo stesso, per cui, per la realizzazione della componente Trust Monitor si è utilizzato Django REST Framework, che a differenza di Flask offre più funzionalità e supporto nella realizzazione di un Web Service.

Django REST Framework offre moltissime potenzialità, le più importanti sono elencate di seguito:

- supporta differenti sistemi di autenticazione, tra cui ad esempio BasicAuthentication e TokenAuthentication;

- permette di utilizzare all'interno del Web Service un database gestito facilmente dallo stesso framework, sia database SQL che NoSQL;
- supporta in maniera molto rapida ed efficiente un sistema per la serializzare, che permette di verificare che gli argomenti definiti per ogni API risultino essere validi.

Si è utilizzato DRF e non Django perché Django permette di realizzare applicazioni Web e nel contesto di questa tesi questo aspetto non è stato preso in considerazione. DRF è utilizzato esclusivamente per Web Service che mettono a disposizione di altre entità delle API. Quindi per motivi di prestazione e di performance si è utilizzato DRF.

All'interno dell'architettura del Trust Monitor è presente anche un database utilizzato per immagazzinare le misure relative al software in esecuzione all'interno delle VNF. Per motivi legati alle performance si è deciso di utilizzare per tale database Redis [32], si tratta di un database NoSQL preferito a quelli SQL per la rapidità di inserimento di misure e per la velocità di ottenimento di tutti i digest presenti al suo interno.

Ogni modulo sviluppato all'interno di questa tesi viene rappresentato attraverso l'utilizzo di un Dockerfile, in modo da creare un file di configurazione per Docker Compose che si occupa di istanziare in modo molto rapido ed efficiente i vari componenti e creare facilmente dei collegamenti tra gli stessi. Questo ha lo scopo di facilitare notevolmente il deployment di tutta l'architettura proposta.

4.2 API del Trust Monitor

Tutte le funzionalità offerte dal Trust Monitor vengono fornite tramite API, in modo tale che ogni componente possa comunicare con tale modulo al fine di verificare lo stato di integrità dei nodi contenenti le VNF. In questa sezione verranno descritte le API offerte dal Trust Monitor per Registrare (sezione 4.2.1) e Attestare (sezione 4.2.2) un nodo ed anche le API utilizzate per Management (sezione 4.2.3) e per aggiungere all'interno del Trust Monitor un particolare digest (sezione 4.2.4). Quest'ultima ci permette di aggiungere dei digest custom relativi a applicativi software in esecuzione all'interno del'host fisico presente nell'infrastruttura NFV, al fine di considerare tali digest durante la procedura di verifica di integrità.

Il Trust Monitor utilizza un database di gestione che al suo interno memorizza informazioni relative ai nodi dei quali è possibile richiedere la procedura di attestazione e informazioni relativi a particolari digest che devono essere considerati durante il processo di verifica di integrità. Tale database viene creato al primo avvio del Trust Monitor e viene popolato con due tabelle secondo le specifiche definite nel file `models.py`. All'interno di questo file ci sono due classi, da ognuna di esse viene creata una tabella secondo i campi definiti dalla classe. La prima tabella viene popolata in base ai valori passati all'API di registrazione, mentre la seconda viene popolata in base ai parametri passati all'API di database.

4.2.1 API di Registrazione

Per attestare un host fisico presente all'interno dell'infrastruttura NFV è necessario che tale nodo debba prima essere registrato presso il Trust Monitor. Per memorizzare i dati relativi a un nodo il Trust Monitor utilizza la tabella `Host`.

Per effettuare la registrazione di un host fisico il Trust Monitor fornisce la seguente API: `https://trustmonitor/register_node`, dove `trustmonitor` corrisponde all'indirizzo IP sul quale il modulo è in esecuzione. L'API di registrazione fornisce due metodi HTTP, può essere eseguita come una GET, in questo scenario verrà restituita la lista dei nodi presenti nell'infrastruttura NFV registrati al Trust Monitor, oppure come un POST specificando un elemento di tipo json contenente diversi parametri.

I parametri specificati in fase di registrazione di un nodo possono essere di due tipi, obbligatori oppure opzionali, è possibile indicare:

- `hostName` (obbligatorio): rappresenta il nome dell'host fisico che si intente registrare;
- `address` (obbligatorio): indica l'indirizzo IP del nodo che si sta registrando;
- `pcr0` (obbligatorio): il nodo deve avere necessariamente il TPM e questo valore rappresenta il contenuto del registro PCR 0 presente all'interno del TPM;
- `distribution` (obbligatorio): serve ad indicare il sistema operativo del nodo che si intende registrare (es. CentOS7);
- `analysisType` (opzionale): viene utilizzato per specificare il tipo di analisi che deve essere applicata al nodo durante il processo di attestazione, se non viene specificato verrà applicata la politica di analisi di default;
- `driver` (obbligatorio): viene utilizzato per specificare il framework di attestazione che deve essere utilizzato per attestare tale nodo (nel contesto di questa tesi il driver avrà come valore OAT).

L'attributo relativo al driver è importante perché il Trust Monitor potrebbe potenzialmente essere connesso a più framework di attestazione, quindi tale valore permetterebbe di identificare quale framework utilizzare.

Una volta che la richiesta di registrazione è stata effettuata, il Trust Monitor esegue una validazione dei vari attributi presenti nella richiesta, se non è presente qualche attributo obbligatorio oppure i valori di ingresso non sono del tipo appropriato viene segnalato un errore al richiedente e la richiesta viene annullata. Questa verifica viene fatta mediante l'utilizzo di una classe presente all'interno del file `serializer.py`, che si occupa di verificare che i vari attributi siano definiti correttamente. Per quanto riguarda i parametri definiti opzionali tale classe si occuperà di associare a quel attributo il valore di default.

Se la procedura di verifica ha esito positivo allora la gestione della registrazione di un host viene passato al corrispettivo driver di attestazione, che si occuperà di registrare il nodo di rete presso il corrispettivo framework di attestazione. Se questa fase termina con successo il nodo verrà inserito all'interno del database del Trust Monitor, in caso in cui ci fosse un errore la procedura non andrà a buon fine e verrà mandato un messaggio all'entità richiedente.

4.2.2 API di Attestazione

Una volta che il nodo viene registrato all'interno del database del Trust Monitor è possibile avviare il processo di attestazione. Il Trust Monitor per svolgere il processo di attestazione fornisce tre differenti API ognuna delle quali, al termine della comunicazione con i connettori (così come descritto nel capitolo di architettura), comunica con il driver OAT che si occupa di eseguire la comunicazione con il framework OAT. Le API che permettono di avviare il processo di attestazione sono (dove `trustmonitor` corrisponde all'indirizzo IP sul quale il modulo è in esecuzione.):

- `https://trustmonitor/attest_node`;
- `https://trustmonitor/get_nfvi_pop_attestation_info`;
- `https://trustmonitor/get_nfvi_attestation_info`.

L'API `../attest_node` è identificata dal metodo POST. Nel momento in cui tale API deve essere contattata, nel corpo della richiesta occorre specificare in aggiunta un oggetto json contente la lista dei nomi dei nodi e per ogni nodo, eventualmente, una lista contenente gli ID dei container Docker in esecuzione al suo interno. Tali ID sono necessari nel contesto di questa tesi, perché OAT supporta l'attestazione basata su container Docker. Gli attributi che possono essere forniti a tale API sono:

- `node` (obbligatorio): il suo valore è associato al nome dell'host fisico da attestare. Il nome deve corrispondere a quello con il quale il nodo in esame è stato registrato presso il Trust Monitor;

- `vnfs` (opzionale): serve a specificare la lista dei container Docker in esecuzione all'interno del nodo, se la lista non viene specificata allora verrà attestata solo l'integrità dell'host fisico;
- `node_list` (obbligatorio): è il primo elemento che deve essere inserito all'interno della richiesta e fa riferimento alla lista dei nodi ed eventualmente alla lista delle VNF associate ad ogni nodo da attestare.

Verrà anche qui effettuata una verifica per controllare che tutti gli attributi abbiano un valore corretto. Se questa fase ha esito positivo, verrà effettuato con confronto per verificare che i nodi presenti all'interno della lista trasmetta all'API `../attest_node` siano registrati al Trust Monitor. Al termine di questa fase sarà possibile avviare il processo di attestazione dove, prima di tutto, verranno contattate le API dei connettori per recuperare maggiori informazioni sulle VNF e dopo di che verrà contattato il driver di attestazione associato al nodo da attestare, questo allo scopo di comunicare con il framework di attestazione per eseguire la RA.

L'API `../get_nfvi_pop_attestation_info` offre il metodo GET a cui è obbligatorio trasmettere il nome del nodo del quale si richiede l'attestazione attraverso l'utilizzo dell'attributo `node_id`. Anche in questo caso il nodo deve essere presente all'interno del database del Trust Monitor, in caso contrario non sarà possibile effettuare l'attestazione. Superata questa fase verranno interrogati i connettori per recuperare informazioni aggiuntive sul nodo, dopo di che verrà contattato il framework di attestazione a cui il nodo fa riferimento.

L'API `../get_nfvi_attestation_info` mette a disposizione il metodo GET senza specificare nessun parametro aggiuntivo. Si tratta dell'ultima API che permette di avviare il processo di attestazione. Tale API permette di attestare tutti i nodi in esecuzione all'interno dell'infrastruttura NFV registrati con il Trust Monitor. Per svolgere l'attestazione il Trust Monitor comunicherà con i vari connettori. Come per le API precedenti al termine della comunicazione con i connettori verranno contatto per ogni nodo presente all'interno dell'infrastruttura NFV il framework di attestazione ad esso associato.

Al termine di ogni API di attestazione viene contattato il driver di attestazione associato all'host, di cui si è richiesta l'attestazione, allo scopo di comunicare con il framework di attestazione corrispettivo ed avviare il processo di RA. Per capire quale framework di attestazione contattare viene fatta una ricerca all'interno del database in modo tale da capire quale driver di attestazione è associato ad ogni host da attestare.

4.2.3 API di Management

Il Trust Monitor mette a disposizione l'API `https://trustmonitor/get_status_info` per verificare che l'ambiente di esecuzione è configurato correttamente.

Tale API è utilizzata al fine di determinare se i vari componenti dell'architettura risultino essere connessi e configurati correttamente. L'API `../get_status_info` espone soltanto il metodo HTTP GET e non prende in ingresso nessun argomento. Il suo scopo è quello di interrogare ogni componente collegato con il Trust Monitor, quindi i vari connettori, i framework di attestazione e il Whitelist Database, al fine di verificare se tali componenti rispondano al Trust Monitor. In caso in cui qualcuno di questi componenti non fosse raggiungibile, il processo di attestazione non andrà a buon fine. Per far si che il processo di attestazione venga concluso correttamente è necessario che sia presente almeno un framework di attestazione collegato con il Trust Monitor e che il Whitelist Database sia accessibile, in caso contrario non sarà possibile determinare lo stato di integrità del nodo.

Il Trust Monitor utilizza il driver di attestazione che si occuperà di comunicare con il framework di attestazione per verificare che tale componente sia raggiungibile e sia configurato correttamente. Se ad esempio l'indirizzo IP associato a un dato framework di attestazione non risulta essere configurato allora verrà restituito un messaggio che indica, a una terza entità che ha contattato tale API, che il framework di attestazione non è stato configurato correttamente.

Nel contesto di questa tesi, il Trust Monitor tramite il driver OAT invierà una richiesta di GET alla seguente API: `https://ip_oat:8443/WLMService/resources/oem`, in caso di riposta allora

il Framework OAT risulta essere in esecuzione. Nel caso in cui non è stato specificato l'indirizzo IP del framework di attestazione, quindi nel nostro caso non è stato specificato un indirizzo IP per `ip_oat`, il Trust Monitor invierà in aggiunta un messaggio di errore informando che il driver non è stato configurato correttamente.

Per verificare che i connettori siano in esecuzione correttamente, il Trust Monitor invia una richiesta di GET alle API offerte dai connettori:

- `http://tm_dare_connector:5000/dare_connector`: API offerta dal connettore verso il DARE;
- `http://tm_dashboard_connector:5000/dashboard_connector`: API offerta dal connettore verso la Dashboard;
- `http://tm_manage_osm_connector:5000/manage_osm_connector`: API offerta dal connettore incaricato di comunicare con Docker e OSM;
- `http://tm_store_connector:5000/store_connector`: API offerta dal connettore verso il vNSF Store.

Nel caso in cui il Trust Monitor riceve una risposta da queste API allora i connettori saranno in esecuzione correttamente.

Il Trust Monitor comunica anche con il Whitelist Database per verificare se è in esecuzione correttamente. Questa verifica è molto importante perché in caso non fosse in esecuzione non sarà possibile verificare le misure presenti all'interno dell'Integrity Report di un dato nodo da attestare. Nel contesto di questa tesi il Whitelist Database è basato su Apache Cassandra [29], si tratta di un database NoSQL. I dati presenti al suo interno fanno riferimento ai digest dei pacchetti software. Il database Apache Cassandra viene popolato attraverso i pacchetti software presenti su un mirror [30], cioè un sito web che conserva al suo interno la copia dei pacchetti di varie distribuzioni Linux al fine di renderli più rapidamente disponibili. Per verificare che tale database risulti essere correttamente in esecuzione viene utilizzata pycassa [31], si tratta di un client Python utilizzato per comunicare con Apache Cassandra. Nel caso in cui pycassa riesca a collegarsi con il database allora la comunicazione è andata a buon fine.

4.2.4 API di gestione Database

Il Trust Monitor mette a disposizione l'API `https://trustmonitor/known_digests`, allo scopo di migliorare alcuni aspetti della procedura di attestazione, infatti mediante questa API il Trust Monitor permette di andare ad aggiungere all'interno del proprio database, più precisamente nella tabella `KnownDigest`, i digest non presenti nel Whitelist Database relativi a possibili software custom in esecuzione all'interno degli host presenti nell'infrastruttura NFV. I digest presenti in questa tabella vengono poi considerati in fase di attestazione. Per permettere a una terza entità, come ad esempio l'amministratore di rete, di manipolare i digest all'interno di tale database, il Trust Monitor espone l'API `https://trustmonitor/known_digests` che fornisce tre diversi metodi HTTP:

- GET: tale metodo viene utilizzato per restituire la lista dei digest presenti all'interno della tabella `KnownDigest`;
- POST: viene utilizzato per aggiungere all'interno del database un determinato digest;
- DELETE: viene utilizzato per eliminare un particolare digest presente all'interno del database.

Sia il metodo POST che il metodo DELETE richiedono l'utilizzo di argomenti che devono essere passati tramite un oggetto di tipo json.

Il metodo POST richiede due argomenti:

- `pathFile` (obbligatorio): rappresenta il percorso assoluto relativo al file che si vuole aggiungere all'interno del database del Trust Monitor;
- `digest` (obbligatorio): serve ad indicare il valore del digest associato al nome del file.

Una volta che il Trust Monitor viene contattato alla seguente API, i parametri passati al metodo POST vengono verificati tramite il serializzatore `SerializerKnownDigest` presente all'interno del file `serializer.py`. Se questa operazione ha esito positivo allora viene fatto un controllo all'interno del database per verificare che il valore contenuto all'interno dell'argomento `digest` non sia già incluso all'interno del database. In caso in cui tale digest sia già presente verrà restituito un errore all'entità richiedente, alternativamente il digest verrà inserito all'interno del database.

Il metodo DELETE richiede un solo argomento, `digest`, utilizzato per eliminare dal database il digest specificato. Anche nel caso del DELETE viene effettuata una serializzazione allo scopo di verificare che il valore associato all'attributo digest sia presente e corretto. Se questa fase ha esito positivo viene effettuato una ricerca nel database allo scopo di individuare il digest in questione, se viene trovato allora si procederà alla sua eliminazione.

L'utilizzo dell'API `https://trustmonitor/known_digests` è essenziale perché non tutte le misure relative al software in esecuzione all'interno di un host fisico sono presenti nel database contente le misure di riferimento. Infatti, se ad esempio all'interno di un nodo viene eseguito del software proprietario il digest associato al software non sarà presente nel database e per questo motivo se non ci fosse un altro database nel quale includere tali digest la procedura di verifica di integrità fallirebbe sempre.

4.3 Comunicazione con OAT

In questa sezione verranno discusse l'API utilizzate dal driver di attestazione per comunicare con il framework di attestazione, che nel contesto di questa tesi è rappresentato dal Framework OAT.

Il Trust Monitor comunica con OAT in due fasi distinte, tramite il driver OAT, un modulo Python che si pone da tramite tra queste due entità. La prima fase è necessaria a registrare i nodi che richiedono di essere attestati presso il Framework OAT 4.3.1, mentre la seconda consiste nell'andare a richiedere il processo di attestazione 4.3.2.

4.3.1 API Registrazione del Nodo

Il nodo di rete oltre ad essere registrato con il Trust Monitor deve anche essere registrato con OAT. Il Framework di attestazione OAT registra il nodo all'interno di un database interno di gestione. Se il nodo non viene registrato in questo database il processo di attestazione non potrà svolgersi, una volta registrato sarà possibile avviare l'attestazione dello stesso. Il database utilizzato è di tipo MySQL dunque è organizzato in tabelle. OAT utilizza il database per registrare diverse informazioni che descrivono sia il flusso di registrazione di un nodo che il flusso di attestazione.

All'interno di questo database vengono utilizzate diverse tabelle con scopi differenti. Abbiamo la tabella `HOST` contenente le informazioni sui nodi registrati con il Framework OAT, poi le tabelle relative a `OS` e `OEM` che vengono utilizzate rispettivamente per manipolare le informazioni relative al sistema operativo specificando il nome, la versione e la descrizione e al produttore indicando nome e descrizione. Queste ultime due tabelle vengono associate ai nodi presenti in modo da identificare il loro sistema operativo. Un'altra tabella importante è `MLE` utilizzata per memorizzare il primo modulo del sistema operativo ad essere misurato. È presente anche una tabella che contiene il valore del PCR 0 che identifica il TPM di un dato host fisico presente all'interno dell'infrastruttura NFV. Un'altra tabella utilizzata è quella che identifica i vari tipi di analisi di attestazione che il framework può eseguire verso un generico nodo di rete.

OAT è rappresentato attraverso un sistema di tipo client-server. Il server avvia il processo di attestazione, mentre il client è rappresentato dall'entità che necessita di essere attestata e sta in attesta di ricevere una richiesta di attestazione dal server. Ogni nodo di rete presente all'interno

del cloud NFV deve avere in esecuzione al suo interno un applicativo software cioè il client OAT, che è in attesa di richieste di attestazione provenienti dal server OAT, in caso in cui il nodo non disponga di questo applicativo non sarà possibile effettuare il processo di comunicazione.

Durante la fase di registrazione di un nodo, il driver OAT comunica con il framework di attestazione utilizzando differenti API di quest'ultimo. La combinazione delle API ha il fine di registrare il nodo presso il database di OAT. Le API vengono contattate tramite metodo POST specificando in aggiunta un json contenente i parametri che identificano il nodo di rete. In questa fase ogni API richiede, per il suo corretto funzionamento, l'assegnazione di attributi obbligatori alcuni dei quali però non verranno considerati in fase di attestazione. Per semplificare questa procedura a tali attributi verranno assegnati dei valori simbolici. Questo ha un duplice scopo, non solo quello di non far fallire la procedura di registrazione, ma anche quello di ridurre il più possibile la lista di attributi che l'utente che vuole registrare il nodo deve fornire all'API del Trust Monitor, perché sarebbe complesso l'utilizzo dell'API `register_node` se richiedesse un numero elevato di attributi da definire. Le API contattate sono (dove `url_globale` rappresenta la concatenazione tra l'indirizzo ip del framework OAT e la porta 8443 alla quale è in ascolto, più precisamente `https://ip_oat:8443`):

- `url_globale/WLMService/resources/oem`: utilizzata per registrare all'interno della tabella `OEM` le informazioni relative al produttore del nodo di rete.
- `url_globale/WLMService/resources/os`: utilizzata per registrare all'interno della tabella `OS` il nome del sistema operativo presente sul nodo. Se più nodi utilizzano lo stesso sistema operativo non verrà creata una nuova entry in questa tabella perché già presente.
- `url_globale/WLMService/resources/analysisTypes`: utilizzata per includere all'interno della tabella `analysisType` di OAT il tipo di analisi che si vuole richiedere, occorre dunque specificare il nome dell'analisi richiesta. Potrebbe essere possibile che più nodi richiedano lo stesso tipo di analisi, se cosi fosse non verrà aggiunta una nuova entry in questa tabella perché l'analisi definita sarà già presente.
- `url_globale/WLMService/resources/mles`: utilizzata per fornire a OAT informazioni aggiuntive sul primo device ad essere eseguito all'interno del nodo da registrare. Nel contesto di questa tesi questa informazione è rappresentata dal modulo TPM.
- `url_globale/AttestationService/resources/hosts`: utilizzata per registrare l'host all'interno della tabella `host`. Occorre specificare il nome del nodo, il suo indirizzo IP e informazioni relative al sistema operativo presente.
- `url_globale/WLMService/resources/mles/whitelist/pcr`: utilizzata per registrare il valore dell'attributo `pcr0` associato a un dato nodo. Dunque nella richiesta occorrerà includere il nome del nodo, la sua distribuzione e il valore del pcr0.

Tutte queste informazioni permettono di registrare il nodo di rete presente all'interno dell'infrastruttura NFV, se la procedura di registrazione presso il framework OAT genera un errore allora l'intera operazione di registrazione verrà annullata. Se invece la comunicazione con OAT terminerà con l'aggiunta del nodo presso il database di OAT, il nodo verrà anche registrato presso il Trust Monitor. È possibile notare che tutte le informazioni relative al nodo di rete che necessita di essere registrato, vengono trasmesse utilizzando una sola API offerta dal Trust Monitor. Le informazioni non importanti, come ad esempio la versione del sistema operativo del nodo vengono rappresentate specificando un valore simbolico. Occorre specificare comunque tale informazione perché è obbligatoria, ma può non essere veritiera in quanto viene utilizzata soltanto a scopo informativo.

4.3.2 API di Attestazione

In figura 4.1 è possibile osservare come avviene il processo di comunicazione tra il Trust Monitor e il framework di attestazione. Il Trust Monitor contatta il driver OAT al fine di avviare il processo di comunicazione con OAT. Il driver OAT per comunicare con OAT e richiedere il processo di RA comunica con il framework OAT tramite una richiesta POST all'API:

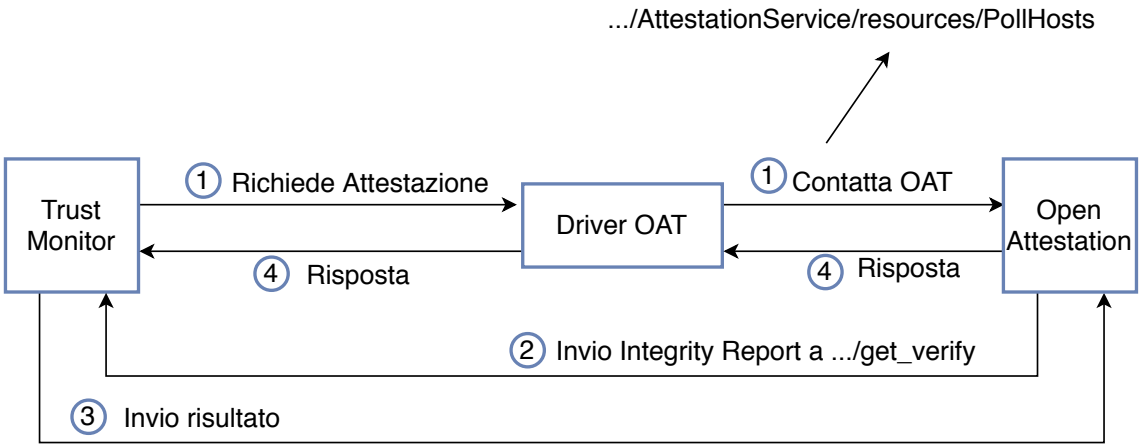

Figura 4.1. Processo di comunicazione tra OAT e il Trust Monitor per il processo di RA.

`https://ip_oat:8443/AttestationService/resources/PollHosts`, dove `ip_oat` corrisponde all'indirizzo IP del server OAT e 8443 rappresenta la porta sul quale il servizio è in esecuzione. L'API offerta da OAT verrà contattata per attestare uno per volta un host fisico presente all'interno dell'infrastruttura NFV. A questa API occorre comunicare:

- `hosts`: rappresenta il nome del nodo che si intende attestare;
- `analysisType`: serve ad indicare il tipo di analisi richiesta per valutare l'integrità del nodo, a questo attributo verrà anche comunicato l'elenco contenente gli ID dei container Docker in esecuzione all'interno del nodo considerato.

Se la comunicazione va a buon fine il server OAT comunicherà con il client OAT presente sul nodo interessato richiedendo il suo Integrity Report, al termine di questa comunicazione OAT invierà l'Integrity Report al Trust Monitor effettuando una richiesta POST all'API `https://trustmonitor/get_verify`. Tale API è stata implementata per permettere la comunicazione dell'Integrity Report tra il server OAT e il Trust Monitor, quindi verrà utilizzata esclusivamente per il framework OAT. I parametri da trasmettere a tale API sono:

- `distribution`: viene utilizzato per indicare il sistema operativo del nodo. Questa informazione viene utilizzata per individuare quali misure del Whitelist Database devono essere prese in considerazione per verificare l'integrità del nodo contenente le VNF. All'interno di tale database le misure presenti fanno riferimento a un sistema operativo, dunque verranno estratte solo quelle relative al sistema operativo definito da tale attributo;
- `report_url`: rappresenta l'url dal quale è possibile ottenere l'Integrity Report del nodo che necessita di essere attestato;
- `report_id`: rappresenta l'identificativo del report, questo perché OAT identifica ogni Integrity Report tramite un ID identificato da un attributo che viene incrementato ogni volta che viene creato un nuovo Integrity Report;
- `analysis`: individua il tipo di analisi richiesta per l'attestazione di quel nodo.

Attraverso le informazioni appena descritte il Trust Monitor è in grado di verificare se il nodo risulti essere integro confrontando le misure presenti nel suo Integrity Report con quelle estratte dal Whitelist Database. L'esito di questa analisi viene trasmessa al framework di OAT come risposta all'API `https://trustmonitor/get_verify`. OAT in base alla risposta ricevuta crea un messaggio contenente l'esito del processo di verifica di integrità che viene trasmesso al driver OAT e successivamente al Trust Monitor.

4.4 Definizione API dei connettori

All'interno all'architettura del Trust Monitor sono presenti altri moduli chiamati connettori, il loro compito è quello di interfacciarsi con gli altri componenti dell'architettura NFV. Sono dunque delle entità che si pongono da intermediari durante il processo di comunicazione del Trust Monitor verso altri moduli dell'architettura NFV. I connettori permettono di recuperare informazioni aggiuntive sulle VNF in esecuzione all'interno di un dato nodo presente nell'infrastruttura NFV. Il Trust Monitor è l'unico componente che comunica con i connettori, la comunicazione tra queste entità avviene mediante l'utilizzo di API. Tra i connettori e il Trust Monitor è presente una relazione unidirezionale, infatti soltanto il Trust Monitor può comunicare con essi, i connettori non contattano mai il Trust Monitor.

I connettori presenti all'interno dell'architettura del Trust Monitor sono:

- `dare_connector`;
- `dashboard_connector`;
- `manage_osm_connector`;
- `store_connector`.

I connettori vengono contattati dal Trust Monitor in due occasioni, durante il processo di attestazione e tramite l'API `https://trustmonitor/get_status_info` utilizzata per verificare che i connettori siano in esecuzione correttamente. Ogni connettore viene contattato dal Trust Monitor in diversi istanti di tempo durante il processo di attestazione. Oltre all'API utilizzata per verificare se il connettore risulti essere in esecuzione correttamente, ogni connettore offre almeno un'altra API.

4.4.1 Connettore DARE

Il `dare_connector` viene utilizzato dal Trust Monitor per trasmettere al DARE l'esito del processo di attestazione di uno o più nodi di rete. In questa tesi non abbiamo una comunicazione tra DARE e `dare_connector` perché, non avevamo a disposizione un vero DARE. Questo connettore allo stato attuale conserva all'interno di un file di log l'esito del processo di attestazione. Il Trust Monitor per comunicare al `dare_connector` il risultato del processo di attestazione utilizza l'API `http://tm_dare_connector:5000/dare_connector/attest_result`, che prevede un metodo POST per trasmettere un json contenente le informazioni complessive sul processo di attestazione.

4.4.2 Connettore Dashboard

Il `dashboard_connector` è utilizzato dal Trust Monitor per trasmettere alla Dashboard l'esito del processo di attestazione nel caso in cui sia "untrusted". Anche in questo caso il messaggio non verrà inviato a una vera Dashboard ma per rendere il tutto più realistico, il messaggio sarà inserito all'interno di una coda RabbitMQ [33]. RabbitMQ è un brocker di messaggi, cioè un'applicazione che memorizza i messaggi in una coda per un'altra applicazione.

Il Trust Monitor comunica con il `dashboard_connector` inviando una richiesta HTTP POST all'API `http://tm_dashboard_connector:5000/dashboard_connector/attestation_failed`. Si tratta di un'API che espone soltanto il metodo POST e richiede l'invio di un oggetto di tipo json contenente le informazione sul processo di attestazione. L'informazione fondamentale che l'oggetto json, contente l'esito del processo di attestazione, deve avere è l'attributo `NFVI` utilizzato per indicare se l'infrastruttura NFV al momento dell'attestazione risulta essere "untrusted". Una volta validato, il messaggio viene immediatamente inserito all'interno di una coda RabbitMQ gestita da un server RabbitMQ, il quale come gli altri connettori viene istanziato all'interno di un container Docker.

4.4.3 Connettore Store

Durante il processo di attestazione il Trust Monitor deve recuperare informazioni sul software in esecuzione all'interno delle VNF, per farlo utilizza lo`store_connector`. Tale connettore recupera dal descriptor delle VNF i valori dei digest relativi alle applicazioni software che sono in esecuzione al loro interno. Tipicamente i descriptor delle VNF sono contenuti all'interno del vNSF Store, nel contesto di questa tesi non avendo a disposizione un vero Store abbiamo simulato questa comunicazione. Allo stato attuale non è presente nessuna implementazione di un descriptor che contenga le misure relative al software in esecuzione all'interno di una VNF, per cui è stato definito un descriptor contenente la lista dei digest delle VNF ed è chiamato `security-manifest`.

```
manifest:vnsf:
    type: OSM
    package: centos_ping_vnfd.tar.gz
    descriptor: centos_ping_vnfd/centos_ping_vnfd.yaml
    properties:
        vendor: some vendor name
        capabilities: ['Virtual OS giovanni:trusty']
    security_info:
        vdu:
          - id: centos_ping_vnfd-VM
            hash_alg: SHA1
            attestation:
                /home/test.sh: abdc0c45af71d5b06912f1d901cd78b5e1a3edc6
```

Figura 4.2. Esempio di security Manifest di una VNF.

Nell'immagine 4.2 è presente un esempio di `security-manifest` per una VNF chiamata `centos_ping_vnfd`. Come è possibile osservare ogni `security-manifest` è legato a una data VNF ed al suo interno contiene informazioni da utilizzare durante il processo di attestazione di una VNF, più precisamente, all'interno di `security_info` sono contenute le informazioni relative al tipo di algoritmo utilizzato per il calcolo del digest e il valore del digest riferito a un particolare applicativo software. Il Trust Monitor contatta lo `store_connector` nel momento in cui ha individuato il nome delle VNF in esecuzione all'interno di un dato nodo di rete, l'interazione Trust Monitor e `store_connector` avviene mediante una richiesta POST alla seguente API `http://tm_store_connector:5000/store_connector/get_vnsfs_digests`. Tale API necessita di un oggetto json con un attributo obbligatorio `list_vnf` al quale occorre associare la lista dei nomi delle VNF. Il connettore internamente verifica se per le VNF indicate è presente un `security-manifest`, se così fosse salva all'interno di una lista le misure associate a ogni VNF. La procedura si conclude con l'invio della lista contenenti i digest degli applicativi software in esecuzione nelle VNF richieste. Per concludere il processo e valutare tali digest durante il processo di attestazione il Trust Monitor memorizza il valore di questi digest all'interno di un database utilizzato per contenere le misure relative alle VNF. Si tratta di un database basato su Redis ed è istanziato all'interno di un container Docker. Il Trust Monitor comunica con il database mediante l'utilizzo di un client Redis offerto da Python.

4.4.4 Connettore MANO/Docker

L'ultimo connettore realizzato nel contesto di questa tesi è chiamato `manage_osm_connector`, svolge un ruolo chiave all'interno dell'architettura del Trust Monitor perché comunica con OSM e con il demone Docker presente all'interno di ogni nodo di rete presente all'interno dell'infrastruttura NFV. Un vincolo fondamentale è che ogni host fisico appartenente a NFVI deve avere in esecuzione al suo interno un VIM. In questa tesi il VIM scelto è VIM Emulator [34], si tratta di un VIM realizzato

dai creatori di OSM e permette comunicando con OSM di istanziare delle VNF tramite l'utilizzo di container Docker. Le API messe a disposizione di `manage_osm_connector` sono:

- `http://tm_manage_osm_connector:5000/manage_osm_connector/osm_list_vim_ip`;
- `http://tm_manage_osm_connector:5000/manage_osm_connector/get_vim_by_ip`;
- `http://tm_manage_osm_connector:5000/manage_osm_connector/osm_vim_docker`;
- `http://tm_manage_osm_connector:5000/manage_osm_connector/get_list_vnf`.

La prima API è accessibile tramite metodo GET, le altre lo sono tramite POST. Il Trust Monitor comunica con il `manage_osm_connector` al fine di ottenere informazioni rilevanti da utilizzare durante il processo di attestazione.

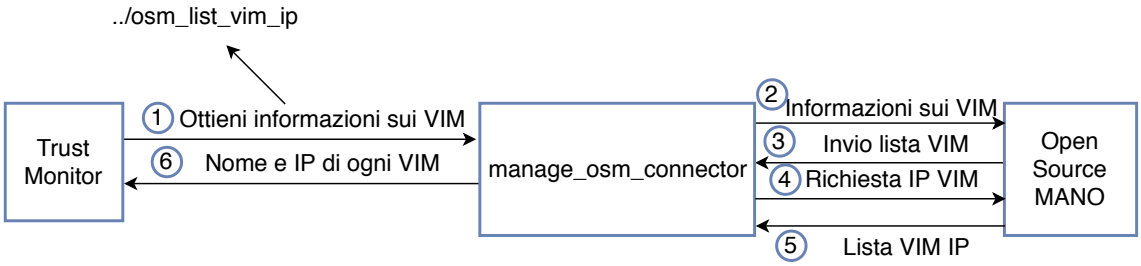

Figura 4.3. Processo di comunicazione per recuperare informazioni sui VIM.

La prima API viene utilizzata quando è richiesto al Trust Monitor di attestare tutte le VNF in esecuzione in un dato istante di tempo. All'interno dell'immagine 4.3 è possibile osservare il processo di comunicazione tra il Trust Monitor e OSM attraverso l'utilizzo dell'API `osm_list_vim_ip` fornita dal connettore, si tratta di un API che espone il metodo HTTP GET al quale non occorre trasmettere nessun argomento. L'API permette di ottenere la lista contenete l'associazione nome VIM e IP di tutti i VIM registrati presso OSM. All'interno di OSM occorre registrare ogni VIM presente nell'infrastruttura NFV, se non vengono registrati non sarà possibile istanziare delle VNF. Per svolgere questo compito `manage_osm_connector` utilizza un applicativo, il client OSM, per comunicare con OSM. In questa fase `manage_osm_connector` comunica con OSM in due occasioni, la prima ha il fine di recuperare la lista di tutti i VIM registrati con l'Orchestratore, la seconda viene utilizzata per recuperare l'indirizzo IP associato ad ogni VIM registrato con OSM, l'associazione nome VIM e IP viene inserita all'interno di una lista che sarà trasmessa al Trust Monitor. In base ai valori presenti all'interno di tale lista il Trust Monitor verificherà se tali IP sono presenti nel suo database, al termine di questa fase per gli indirizzi IP trovati nel database verrà avviato il processo di attestazione.

Alternativamente se al Trust Monitor viene richiesto di attestare un dato nodo presente all'interno dell'infrastruttura NFV, viene utilizzata l'API `get_vim_by_ip` esposta dal `manage_osm_connector`. A questa API occorre comunicare, tramite metodo POST, un oggetto json contenente l'indirizzo IP del nodo di rete che si intende attestare, questo viene fatto mediante l'utilizzo dell'attributo `ip` definito dall'API. Il connettore comunica con OSM al fine di recuperare il nome del VIM associato all'indirizzo IP fornito dal Trust Monitor. La comunicazione tra il connettore e OSM si svolge allo stesso modo dell'API `osm_list_vim_ip` utilizzando sempre il client OSM, ma una volta ottenuta la lista di relazioni contenente il nome VIM e l'indirizzo IP ad esso associato, viene effettuata una ricerca per verificare se esiste un VIM che abbia l'indirizzo IP fornito dal Trust Monitor. Al termine di questa fase il connettore trasmetterà al Trust Monitor il nome del VIM e l'indirizzo IP ad esso associato, in caso in cui non verrà trovato il nome del VIM associato all'indirizzo IP fornito dal Trust Monitor il processo di attestazione si interromperà restituendo un messaggio di errore all'entità richiedente.

Un'altra API offerta dal `manage_osm_connector` e utilizzata dal Trust Monitor è `osm_vim_docker` accessibile mediante metodo POST. Per contattare questa API occorre specificare la lista degli indirizzi IP associati a ogni VIM, quindi ad ogni host da attestare presente all'interno dell'infrastruttura

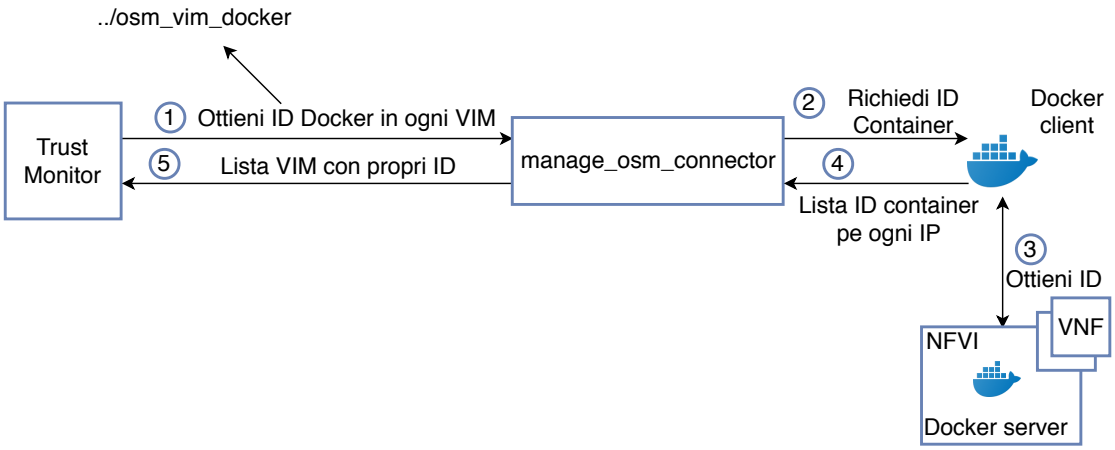

Figura 4.4. Procedura per recuperare lista ID dei container Docker associati ad ogni VIM.

NFV. Nell'immagine 4.4 è possibile osservare l'intero processo di comunicazione definito dall'utilizzo di tale API. Lo scopo di questa API è quella di mettere a disposizione del Trust Monitor la lista degli ID dei container Docker in esecuzione all'interno di ogni nodi di cui si è richiesta l'attestazione. Per farlo il connettore utilizza una libreria Python `docker-py` [35] che funge da client Docker e tramite l'indirizzo IP associato a un dato host è in grado di contattare il demone Docker in esecuzione all'interno del nodo con l'obiettivo di recuperare gli ID dei container in esecuzione al suo interno. Gli ID dei container Docker vengono successivamente utilizzati per contattare OAT ed avviare il processo di attestazione.

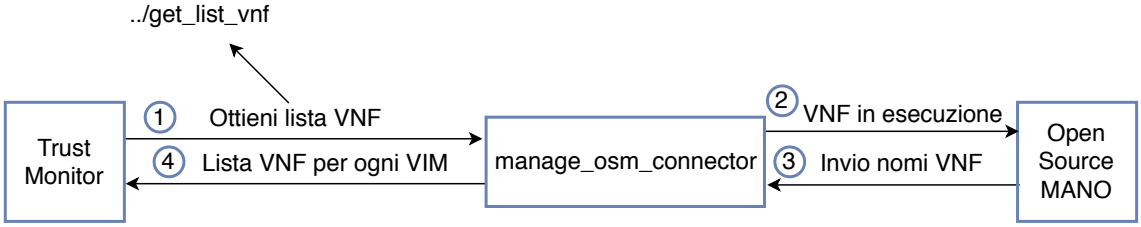

Figura 4.5. Procedura per recuperare nome delle VNF in esecuzione all'interno di ogni VIM.

L'ultima API messa a disposizione dal `manage_osm_connector` è `get_list_vnf` che espone un metodo POST al quale occorre comunicare l'attributo `VIM` che rappresenta una lista contenente l'associazione nome dei VIM e indirizzo IP (sarebbe l'outuput dell'API `get_vim_by_ip`). In figura 4.5 è rappresentata la comunicazione tra il Trust Monitor e OSM, al fine di ottenere la lista di VNF in esecuzione all'interno di ogni VIM. Il connettore per recuperare tali informazioni comunica con OSM mediante l'utilizzo del client OSM. Le VNF vengono istanziate all'interno dell'architettura NFV tramite la definizione di un Network Service (NS), per cui il connettore richiederà prima la lista di tutti i NS in esecuzione all'interno del cloud. Successivamente ogni NS verrà analizzato per verificare che sia in esecuzione all'interno di uno dei VIM passati all'API `get_list_vnf`. Questa procedura viene fatta analizzando il nome del VIM su cui il NS è in esecuzione. Se verrà trovata un associazione tra un NS e il nome di un VIM fornito dalla lista trasmessa dal Trust Monitor, allora saranno estrapolate le VNF istanziate dal NS. Il risultato di questa fase produrrà una lista contenente per ogni VIM l'elenco delle VNF in esecuzione al suo interno. Questa API permette dunque di ottenere la lista dei nomi delle VNF in esecuzione all'interno dei nodi di cui è stata richiesta l'attestazione. Tale lista può successivamente essere comunicata all'API offerta dallo `store_connector`.

Capitolo 5

Risultati

In questo capitolo verranno presentati i risultati ottenuti in base a differenti tipi di test per verificare il funzionamento del Trust Monitor. Più precisamente all'interno della sezione 5.1 verranno fornite informazioni relative ai tipi di piattaforme utilizzate per ospitare l'architettura del Trust Monitor e per ospitare l'host che necessita di essere attestato, nella sezione 5.2 verranno presentati dei test funzionali allo scopo di fornire, all'entità richiedente, il risultato del processo di attestazione, invece nella sezione 5.3 verranno forniti dei dettagli sull'analisi di prestazione del Trust Monitor e del connettore MANO Docker. Si è deciso di analizzare le performance di tale connettore perché all'interno dell'architettura del Trust Monitor risulta essere il connettore più critico. Il connettore MANO Docker svolge molteplici iterazioni con l'Orchestratore dell'architettura NFV e con il demone Docker in esecuzione all'interno dell'host da attestare, per questo motivo è stato necessario analizzare anche le prestazioni di tale connettore. Nell'ultima sezione presente all'interno di questo capitolo verranno forniti test di latenza, test sul consumo della CPU e test sull'utilizzo della RAM.

5.1 Piattaforme di test

In questa sezione verranno descritte le piattaforme fisiche ospitanti l'architettura del Trust Monitor e dell'host contenente le VNF presente all'interno dell'infrastruttura NFV che necessita di essere attestato.

Il Trust Monitor e i connettori utilizzati per comunicare con l'architettura NFV e con il framework OAT sono istanziati tramite l'utilizzo di Docker Compose e sono presenti all'interno di una macchina virtuale. Il Sistema Operativo in esecuzione all'interno della macchina virtuale è Ubuntu 16.04.4 LTS e dispone di 2 GB di RAM, 2 CPU e 18 GB di disco fisso.

L'host fisico è rappresentato da una macchina fisica con Sistema Operativo CentOS7 ed è basata su kernel linux 4.4.19. Al suo interno è presente necessariamente VIM Emulator che permette di istanziare VNF in esecuzione all'interno di container Docker. In aggiunta, il demone Docker presente all'interno dell'host deve essere in ascolto non solo sulla Socker Unix ma anche su uno specifico indirizzo IP, questo al fine di rendere possibile l'accesso al demone Docker anche da remoto permettendo così il recupero degli ID dei container Docker in esecuzione all'interno dell'host. All'interno della macchina fisica deve essere presente e deve essere in esecuzione l'HostAgent OAT utilizzato per comunicare con l'Appraiser OAT allo scopo di eseguire il processo di RA.

Il VIM presente all'interno dell'host da attestare deve essere collegato con l'orchestratoreOpen Source MANO Release 3, questo allo scopo di permettere l'istanziazione di NS contenenti VNF all'interno delle VIM desiderato e dunque di conseguenza all'interno del nodo fisico da attestare.

5.2 Test funzionali

Una volta rappresentata l'architettura sviluppata nel contesto di questa tesi, occorre eseguire dei test per verificare il corretto funzionamento dell'architettura sviluppata. In questa sezione verranno descritti due tipi di test allo scopo di osservare il messaggio di output generato dal Trust Monitor per informare l'entità, che ha richiesto l'attestazione di un host, sull'esito del processo di attestazione verso un nodo all'interno dell'infrastruttura NFV. Verranno considerati due scenari, uno nel caso in cui le VNF istanziate risultino essere "trusted" (sezione 5.2.1), mentre il secondo nel caso in cui una delle VNF sia "untrusted" (sezione 5.2.2).

5.2.1 Test con architettura in stato valido

In questa sezione verrà descritto un primo test utilizzato per testare le funzionalità del Trust Monitor. Per l'esecuzione di questo test è stato necessario creare un NS chiamato `ns_trusted_nsd` che istanzia due VNF `test1_vnfd` e `test2_vnfd`. Entrambe le VNF fanno riferimento all'immagine base `centos:7` che al loro interno non eseguono nessun software custom.

I descriptor utilizzati per descrivere il NS e le VNF devono essere inserite nel catalogo NS e nel catalogo VNF presente all'interno di OSM release 3. Quando i descriptor vengono aggiunti all'interno di OSM è possibile istanziare il NS andando ad indicare il nome con cui il NS debba essere istanziato ed anche il nome del VIM incaricato di ospitare il NS contente le VNF.

Nel momento in cui il NS viene mandato in esecuzione, all'interno del host da attestare vengono creati i container Docker incaricati di ospitare le VNF. Non appena il NS risulti essere istanziato correttamente, cioè all'interno di OSM il NS viene rappresentato con lo stato `Active`, è possibile procedere con la procedura di attestazione. L'entità che richiedere di svolgere l'attestazione di un host fisico presente all'interno dell'infrastruttura NFV contatta il Trust Monitor alla seguente API `https://trustmonitor/get_nfvi_pop_attestation_info/`, dove `trustmonitor` rappresenta l'indirizzo IP associato alla macchina virtuale che ospita il Trust Monitor. Questa API viene contattata tramite metodo HTTP GET e richiede in ingresso il nome del nodo che si vuole attestare, questo viene fatto tramite l'attributo `node_id`. È necessario che il nodo debba essere registrato presso il Trust Monitor, in caso contrario la procedura di attestazione non verrà eseguita e il Trust Monitor invierà un messaggio di errore all'entità richiedente.

Al termine del processo di verifica di integrità, all'entità richiedente verrà inviato un messaggio contenente l'esito del processo di attestazione. In figura 5.1 è possibile osservare l'oggetto Json di output trasmesso dal Trust Monitor. Al suo interno sono presenti dei campi che permettono di descrivere nel dettaglio particolari informazioni del processo di attestazione. I campi presenti sono:

- `NFVI`: utilizzato per fornire informazioni sull'affidabilità dell'infrastruttura NFV in base ai nodi di cui si è richiesta l'attestazione;
- `vtime`: utilizzato per indicare il tempo in cui è stato svolto il processo di attestazione;
- `details`: utilizzato per fornire dettagli aggiuntivi sull'intero processo di attestazione.

Il primo campo che viene mostrato all'entità richiedente è `NFVI`, se il valore ad esso associato è "trusted" allora i nodi attestati risultano essere affidabili, se così non fosse occorre andare ad analizzare nel dettaglio l'oggetto `details` che fornisce altre informazioni utili a descrivere il processo di attestazione, queste sono:

- `node`: tale campo è utilizzato per indicare il nome del nodo di cui è stata richiesta l'attestazione;
- `trust_lvl`: utilizzato per fornire il livello di fiducia del nodo fisico, dunque esclusivamente delle componenti software in esecuzione all'interno dell'host fisico del quale si è richiesta l'attestazione, non considerando il software in esecuzione all'interno dei container Docker e quindi all'interno delle VNF;

```
{
    "NFVI":"trusted",
    "vtime":"2018-07-10T15:25:45+02:00",
    "details": [
        {
            "node":"nfvi-node",
            "trust_lvl":"trusted",
            "analysis_containers": [
                      {
                   "trust_lvl":"trusted",
                   "container":"718c9a2b4eeb"
               },
               {
                   "trust_lvl":"trusted"
                   "container":"fb104a105e7f"
               },
            ],
            "analysis_status":"ANALYSIS_COMPLETED",
            "driver":"OAT",
            "analysis_extra_info": {
                "Digest fake lib": 0,
                "List Digest not found": [],
                "Packages not security": 0,
                "Digest ok": 388,
                "Packages unknown": 0,
                "Packages ok": 110,
                "List Digest Fake Lib": [],
                "Packages security": 0,
                "Digest not found": 0
            }
        }
    ]
}
```


Figura 5.1. Rappresentazione oggetto Json nel caso in cui il risultato sia “trusted”.

- `analysis_containers`: utilizzato per indicare lo stato di fiducia di ogni container che contiene all'interno una VNF. Come è possibile osservare dall'immagine sono presenti le informazioni su due container perché le VNF istanziate tramite il NS erano due. Il campo `analysis_containers` include `container` utilizzato per riportare l'ID del container e `trust_lvl` utilizzato per indicare il livello di fiducia associato al container in esame;
- `analysis_status`: campo utilizzato per indicare se il processo di analisi di integrità si è concluso con successo oppure se si è riscontrato qualche errore durante la comunicazione con l'Appraiser di OAT;
- `driver`: utilizzato per indicare quale framework di attestazione è stato utilizzato per svolgere il processo di RA per il nodo dato;
- `analysis_extra_info`: è una lista json contenente informazioni aggiuntive sulla procedura di attestazione. Tale lista risulta molto utile per capire perché un nodo presente all'interno dell'infrastruttura NFV è stato etichettato come “untrusted”.

Come è possibile osservare dall'immagine in questo scenario il nodo e le VNF in esecuzione al suo interno risultano essere “trusted”.

5.2.2 Test con architettura in stato invalido

In questo scenario andremo ad eseguire un test nel quale una delle VNF in esecuzione risulta essere "untrusted".

Per la realizzazione di questo test è stato necessario creare un NS chiamato `ns_untrusted_nsd` che si occupa di istanziare due VNF `test1_vnfd` e `test_untrusted_vnfd`. La prima basata sull'immagine base `centos:7`, mentre per la seconda viene utilizza un'immagine basata su `centos:7` ma modificata, nella quale viene eseguito un software custom del quale non si dispone del digest che lo identifica, tale immagine prende il nome di `centos:untrusted`.

```
FROM centos:7
LABEL maintainer="Giovanni Trivigno <giovanni.trivigno@studenti.polito.it>"
ADD test.sh /home/test_2.sh
RUN chmod +x /home/test_2.sh
ENTRYPOINT ["./home/test_2.sh"]
```

Figura 5.2. Dockerfile relativo all'immagine `centos:untrusted`.

In figura 5.2 è rappresentato il Dockerfile utilizzato per creare l'immagine `centos:untrusted`. Come è possibile osservare l'immagine base utilizzata per creare tale Dockerfile è `centos:7`, ma al suo interno è stato aggiunto un livello nel quale viene eseguito del software custom. Il file `/home/test_2.sh` non fa altro che stampare tramite `echo` un messaggio. Il file viene posto in modalità di esecuzione quindi nel momento in cui il container Docker verrà eseguito lo script proposto sarà eseguito e dunque verrà misurato da IMA.

Il risultato sarà dunque "untrusted" perché non sarà possibile recuperare il digest relativo al software in esecuzione nella VNF né dal Whitelist Database e né dal `security-manifest` che identifica la VNF in quanto non è stato aggiunto il valore del digest.

I descriptor che identificano il NS e le VNF devono essere aggiunti nel catalogo di OSM al fine di rende possibile l'istanziazione del servizio di rete sul VIM desiderato. Nel momento in cui il NS risulta essere in esecuzione correttamente può essere avviato il processo di attestazione del nodo in esame.

Una terza entità utilizza l'API `https://trustmonitor/get_nfvi_pop_attestation_info/` offerta dal Trust Monitor specificando tramite il metodo HTTP GET l'attributo `node_id` utilizzato per definire il nodo che si intende attestare. Come nel caso precedente verrà attestato il nodo nfvi-node precedentemente registrato presso il Trust Monitor. All'interno della figura 5.3 è possibile osservare il messaggio json risultante dal processo di attestazione del nodo `nfvi-node` presente all'interno dell'infrastruttura NFV e contenente le VNF.

Come è possibile osservare a differenza del caso "trusted" (sezione 5.2.1) all'interno dell'oggetto json sono presenti delle informazioni che ci permettono di comprendere il motivo per cui il nodo in esame è stato etichettato come "untrusted". All'interno di `analysis_containers` è possibile osservare l'ID del container Docker che ha prodotto il risultato di "untrusted".

Per avere maggiori dettagli al fine di comprendere il motivo per cui il nodo è stato definito "untrusted" occorre prestare attenzione ai capi presenti nell'oggetto json `analysis_extra_info`:

- `Digest not found`: questo campo è utilizzato per indicare il numero di digest per i quali non è stato possibile trovare un associazione in base ai digest presenti all'interno del Whitelist Database e del Whitelist VNF Database;
- `List Digest not found`: campo utilizzato per fornire più informazioni sui digest che non sono stati trovati all'interno dei database di riferimento. Tale campo ospita al suo interno una lista dei valori. Ogni oggetto json contenuto al suo interno deve avere il campo `instance` utilizzato per indicare se il digest non presente all'interno dei database è di proprietà dell'host o di uno dei container e il secondo campo serve ad indicare l'associazione tra il nome del

```
{
    "NFVI":"untrusted",
    "vtime":"2018-07-10T15:31:29+02:00",
    "details": [
        {
            "node":"nfvi-node",
            "trust_lvl":"trusted",
            "analysis_containers": [
                {
                    "trust_lvl":"untrusted",
                    "container":"c375e6ed5757"
                },
                {
                    "trust_lvl":"trusted",
                    "container":"357397ec5367"
                },
            ],
            "analysis_status":"ANALYSIS_COMPLETED",
            "driver":"OAT",
            "analysis_extra_info": {
                "Digest fake lib": 0,
                "List Digest not found": [
                    {
                        "instance":"c375e6ed5757",
                        "/home/test_2.sh":
                        "caea4b2b7d892c4222abd34fcff902fd99edf1df"
                    }
                ],
                "Packages not security": 0,
                "Digest ok": 389,
                "Packages unknown": 0,
                "Packages ok": 110,
                "List Digest Fake Lib": [],
                "Packages security": 0,
                "Digest not found": 1
            }
        }
    ]
}
```


Figura 5.3. Rappresentazione oggetto json nel caso in cui il risultato sia “untrusted”.

software per il quale non è stata trovata corrispondenza nel database e il suo corrispettivo digest;

- `Digest fake lib`: utilizzato per indicare il numero di digest che fanno riferimento a librerie condivise (file .so) il cui digest è presente all’interno del Whitelist Database, ma il cui nome non è presente all’interno di tale database. All’interno del Whitelist Database sono presenti anche digest relativi alle librerie, le quali possono essere identificate tramite diversi alias (cioè differenti nomi) con cui una libreria può essere chiamata. Se all’interno di tale database viene trovata solo la corrispondenza con digest di una libreria ma non con il suo nome allora essa viene classificata come fake;

- `Packages not security`: indica il numero di package per i quali esistono aggiornamenti non inerenti la sicurezza;

- `Digest ok`: questo campo è utilizzato per indicare il numero di digest per i quali si è trovata corrispondenza in base ai database di riferimento;
- `Packages unknown`: questo campo è utilizzo per indicare il numero di package per i quali non si conosce la provenienza;
- `Packages ok`: utilizzato per indicare il numero di package per i quali si è trovata corrispondenza all'interno del Whitelist Database;
- `List Digest Fake Lib`: utilizzato per rappresentare una lista di oggetti json dove ognuno dei quali contiene il nome della libreria che è stata classificata come fake;
- `Packages security`: questo campo è utilizzato per indicare il numero di package per i quali esistono aggiornamenti di sicurezza.

Se si vogliono ottenere maggiori informazioni sull'esito del processo di attestazione è possibile consultare il file di log prodotto dal Trust Monitor.

5.3 Test di performance

In questa sezione verranno descritti dei test utilizzati per valutare le performance del Trust Monitor e del connettore MANO Docker. Il connettore MANO Docker presente all'interno dell'architettura dal Trust Monitor risulta essere un componente fondamentale perché viene utilizzato per comunicare con l'Orchestratore e con il demone Docker. La comunicazione con l'Orchestratore è necessaria per recuperare le informazioni sulle VNF in esecuzione su un nodo, mentre la comunicazione con il demone Docker in esecuzione all'interno dell'host da attestare è necessario per ottenere gli ID dei container Docker nei quali sono in esecuzione le VNF. Dal momento che il connettore MANO Docker viene utilizzato per ottenere informazioni necessarie per avviare il processo di attestazione è necessario osservare le sue performance. Più precisamente nella sezione 5.3.1 verrà eseguito un test per osservare il tempo di esecuzione necessario al Trust Monitor per svolgere l'attestazione all'aumentare del numero di VNF istanziate, nella sezione 5.3.2 verrà eseguito un test per osservare il consumo di CPU del Trust Monitor e del connettore MANO Docker, ed infine nella sezione 5.3.3 verrà analizzato il consumo della RAM all'aumentare del numero di attestazioni.

Per ottenere uno scenario più possibile simile ad uno di produzione il Trust Monitor è stato eseguito con il flag `--noreload`, in questo caso se durante la sua esecuzione viene modificato il codice Python del Trust Monitor, questa modifica non avrà effetto sul Trust Monitor presente in memoria. Discorso analogo è stato utilizzato per i connettori sviluppati in Flask, i quali sono stati eseguiti con il flag `use_reloader` impostato a `False`.

5.3.1 Test di latenza

Il primo risultato che verrà proposto è relativo a un test volto ad analizzare nel dettaglio la durata dell'intera procedura di attestazione. All'interno dell'immagine 5.4 è possibile osservare il tempo di attestazione necessario a verificare se un'entità risulti essere fidata o meno. È possibile osservare che il tempo di esecuzione subisce un incremento all'aumentare del numero di container Docker istanziati all'interno dell'host da attestare, dunque nel nostro caso all'aumentare delle VNF istanziate all'interno di container Docker.

Per l'esecuzione di questo test è stato necessario creare quattro NS basati su un numero differente di VNF. Si è deciso di considerare NS basiti su un massimo di 35 VNF istanziate questo perché OSM release 3, utilizzato nell'ambito di questa tesi, non permetteva di istanziare un NS avente più di 35 VNF per motivi prestazionali. Si è tentato di istanziare un NS avente 50 VNF ma questo tentativo ha portato alla saturazione completa della memoria dell'host contenente OSM causando un inutilizzo dell'Orchestratore. Infatti OSM per evitare una possibile saturazione della memoria distrugge il NS che richiede il maggior numero di capacità di elaborazione.

Il grafico presente il figura 5.4 indica i differenti tempi di esecuzione per i quattro NS considerati andando a suddividere il tempo totale di attestazione in base al:

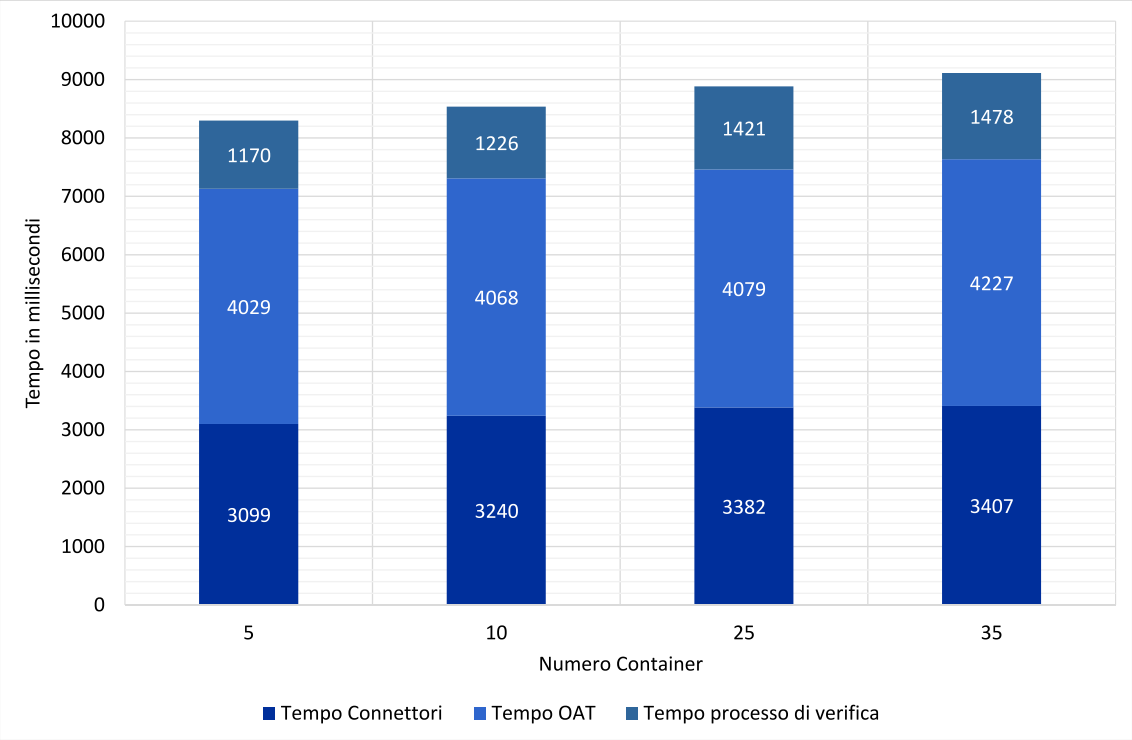

Figura 5.4. Grafico che descrive i tempi di esecuzione all'aumentare dei container.

- Tempo Connettori: rappresenta il tempo utilizzato dai connettori per comunicare con i componenti dell'architettura NFV e con il demone Docker presente all'interno dell'host da attestare al fine di recuperare informazioni relative alle VNF istanziare, all'ID dei container Docker contenente le VNF e al `security-manifest` contenente i digest relativi al software custom in esecuzione all'interno delle VNF considerate.

- Tempo OAT: rappresenta il tempo utilizzato dall'Appraiser OAT per comunicare con l'HostAgest OAT con lo scopo di richiedere l'invio dell'Integrity Report del nodo da attestare che sarà trasmesso al Trust Monitor al fine di avviare il processo di verifica di integrità.

- Tempo processo di verifica: rappresenta il tempo utilizzato per analizzare l'Integrity Report del nodo da attestare al fine di determinare se l'host contenente le VNF risulti essere fidato o meno. Tale processo comunica con il Whitelist Database per recuperare i digest necessari per attestare il nodo in esame.

Per l'esecuzione di questo test è stato necessario effettuare per ogni NS istanziato all'interno di un dato host fisico dieci attestazioni, i tempi presenti all'interno del grafico rappresentano la media delle attestazioni eseguite.

Come è possibile osservare dall'immagine, all'aumentare del numero di VNF da attestare aumenta anche il tempo necessario per svolgere l'interno processo di attestazione. Il tempo che aumenta maggiormente è quello relativo ai connettori, infatti la comunicazione con l'Orchestratore rappresenta un'operazione critica perché all'aumentare del numero di VNF istanziate, OSM fornisce una lista più ampia di nomi di VNF. Il tempo relativo al processo di verifica aumenta in base al numero di container Docker contenenti VNF perché andando a considerare più container durante il processo di attestazione il numero di misure IMA aumenta. Il tempo necessario a svolgere la procedura di RA realizzata dal framework OAT aumenta in maniera minore ma tale aumento dipende dal tempo che intercorre nella comunicazione tra l'Appraiser OAT e l'HostAgent da attestare, in quanto l'HostAgent è in attesa di una richiesta da parte del server OAT che per fare ciò si trova in una fase di polling questo porta a un aumento del tempo di comunicazione tra queste due entità.

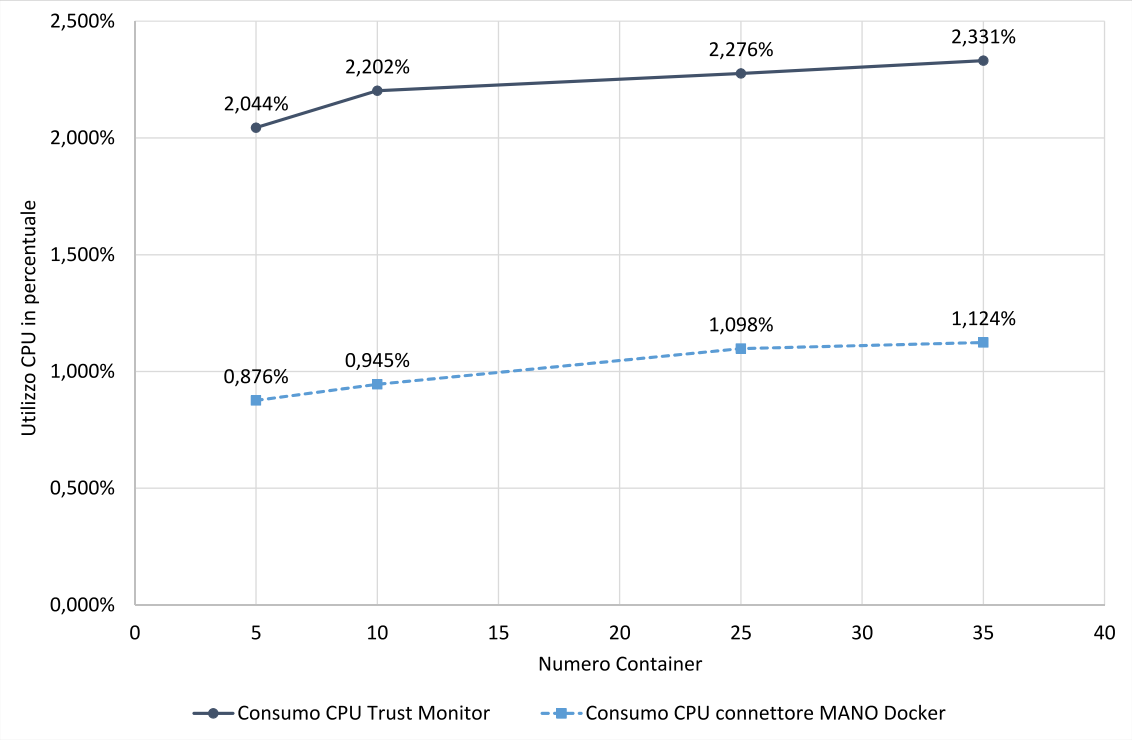

Figura 5.5. Grafico rappresentante l'utilizzo di CPU all'aumentare del numero di container.

5.3.2 Test consumo CPU

In questa sezione verrà presentato un test per valutare il consumo di CPU del Trust Monitor e del connettore MANO Docker in base al numero di VNF da attestare. Come nel caso precedente, per questo test è stato necessario istanziare, in un lasso di tempo differente, quattro NS contenenti un numero differente di VNF.

All'interno del grafico presente in figura 5.5 è possibile osservare il consumo di CPU espresso in percentuale. Analizzando la figura è possibile osservare che il processo di attestazione richiede l'utilizzo di più CPU all'aumentare del numero di container da considerare. Come nel test precedente il valore percentuale di utilizzo di CPU da parte del Trust Monitor e del connettore MANO Docker è rappresentato da una media aritmetica calcolata su dieci attestazioni svolte per ogni NS da attestare.

È possibile osservare che la soluzione proposta richiede un consumo molto ridotto di CPU, per tale ragione per il suo funzionamento l'architettura proposta non richiederebbe un numero elevato di risorse fisiche.

5.3.3 Test consumo RAM

L'ultimo test realizzato è mirato a descrivere il consumo di RAM da parte del Trust Monitor e del connettore MANO Docker. All'interno del grafico presente in figura 5.6 è rappresentato l'utilizzo di RAM espresso in MiB all'aumentare del numero di attestazione.

È possibile osservare che il consumo di RAM non è problematico infatti dal momento in cui viene effettuata la prima attestazione al momento in cui viene effettuata l'ultima attestazione presente all'interno del grafico, l'utilizzo di RAM da parte del Trust Monitor e del connettore MANO Docker non è molto significativo.

Tutti i componenti realizzati nell'ambito di questa tesi sono sviluppati in Python e tale linguaggio non gestisce internamente il processo di garbage collector [36], ma utilizza un garbage collector basato sul concetto di soglia. Questo significa che l'effettiva procedura di eliminazione di un oggetto

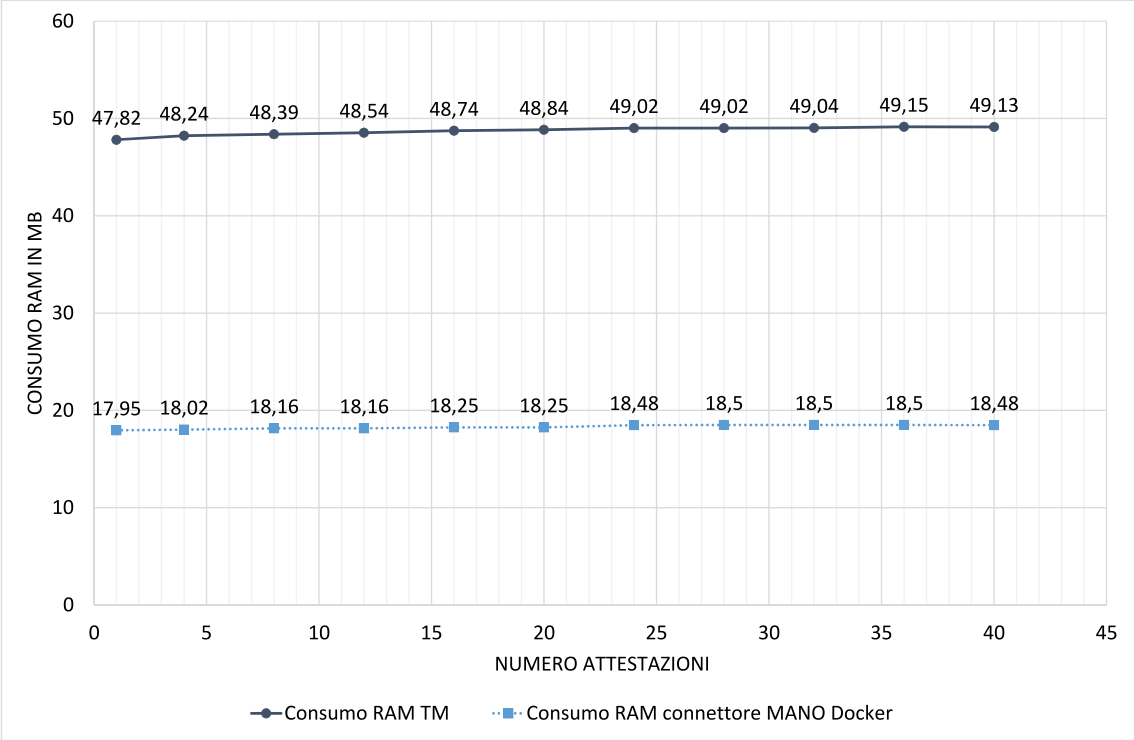

Figura 5.6. Grafico rappresentante il consumo di RAM all'aumentare del numero di attestazioni.

verrà applicata soltanto quanto verranno soddisfatte alcuni vincoli dell'oggetto, come ad esempio il numero di riferimenti che utilizzano tale oggetto posto a 0. Per rilasciare la memoria utilizzata e dunque non causare dei leakage di memoria è stato necessario effettuare una chiamata esplicita al garbage collector utilizzato da Python che in tal caso si occupa di rilasciare la memoria degli oggetti utilizzati. Applicando questa tecnica siamo riusciti ad ottenere un risultato soddisfacente nell'utilizzo di RAM da parte dell'architettura proposta.

All'interno del grafico presente in figura 5.6 è possibile osservare che dal momento in cui viene fatta la prima attestazione al momento in cui viene effettuata l'ultima attestazione si ha un aumento minimo nel consumo di RAM. Questa situazione è motivata dal fatto che Python non restituisce immediatamente la memoria degli oggetti distrutti, ma la memoria viene restituita al Sistema Operativo in un secondo momento. Per tale ragione se viene fatto passare del tempo tra un'attestazione e la successiva avremo una diminuzione nell'utilizzo della RAM. Questo scenario è possibile osservarlo nell'ultimi due valori relativi al consumo RAM del Trust Monitor e del connettore MANO Docker presenti all'interno del grafico.

È possibile osservare che si è prestata particolare attenzione all'analisi delle performance relative al consumo di CPU e all'utilizzo di RAM, questo per permettere di effettuare un deploy dell'architettura proposta anche all'interno di un terminale avente un numero ridotto di capacità di calcolo.

Capitolo 6

Conclusioni

In questo capitolo saranno discusse due tematiche conclusive al fine di analizzare dal punto di vista critico il lavoro di tesi svolto. In particolare all'interno della sezione 6.1 verrà presentato il contesto in cui l'architettura proposta potrebbe portare benefici, mentre all'interno della sezione 6.2 verranno discusse le limitazioni dello sviluppo di questa tesi ed anche dei possibili sviluppi futuri.

6.1 Risultati ottenuti

Per la realizzazione di questa tesi, il punto da cui siamo partiti è stato quello di analizzare come il mondo economico è cambiato nel corso degli anni. Infatti la rivoluzione introdotta dal Cloud Computing ha mutato il modo in cui i servizi IT vengono progettati e impiegati all'interno della rete. Il cambiamento è stato possibile mediante il concetto di virtualizzazione, che ha portato ad un miglioramento in termini di scalabilità e flessibilità del servizio di rete che si va ad utilizzare. Le funzioni di rete presenti all'interno delle reti di comunicazione tradizionali vengono istanziare all'interno di host fisici, tramite il concetto di virtualizzazione tali funzioni sono state sostituite da funzioni di rete virtualizzate che possono essere implementate su un singolo host fisico mediante la condivisione delle risorse fisiche presenti al suo interno.

La tecnologia NFV è stata analizzata nel contesto di questa tesi e permette di istanziare delle funzioni di rete virtualizzate, chiamate VNF. L'utilizzo massivo di servizi IT ha aumentato notevolmente i possibili tipi di attacchi informatici all'interno della rete. Per contrastarli la tecnologia NFV fornisce delle VNF specializzate nel fornire funzioni di sicurezza di rete virtualizzate allo scopo di contrastare o monitorare gli attacchi informatici, tali funzioni sono chiamate vNSF. Le vNSF sono però esposte al mondo quindi possono essere soggette a diversi tipi di attacchi, in particolare un attaccante potrebbe manipolare un nodo per produrre un comportamento diverso da quello atteso. Il focus principale della tesi è stato quello di introdurre all'interno dell'architettura NFV un meccanismo che si occupa di verificare lo stato di integrità di ogni funzione di rete presente all'interno dell'infrastruttura NFV.

Si è iniziato il lavoro analizzando gli standard prodotti da ETSI NFV allo scopo di verificare se era già stato realizzato un sistema che permettesse di eseguire il processo di RA all'interno dell'infrastruttura NFV e che inoltre riuscisse tramite le misure relative al software in esecuzione all'interno del nodo e delle VNF a determinare il livello di fiducia del nodo analizzato. La ricerca ha fornito solo una definizione dell'utilizzo di un modulo che potrebbe svolgere il processo di verifica di integrità all'interno dell'infrastruttura NFV. Per questa ragione era presente un gap tra la definizione e l'implementazione di tale modulo. Mediante lo sviluppo di questa tesi siamo riusciti a colmare questa mancanze implementando un modulo che si occupa di svolgere il processo di attestazione per determinare l'integrità di ogni VNF oppure SDN presente all'interno dell'infrastruttura NFV.

Il modulo realizzato prende il nome di Trust Monitor. Tale modulo è stato reso più generico possibile per riuscire ad utilizzare qualsiasi framework di attestazione per svolgere il processo di

RA. Per fornire una soluzione che si andasse ad integrare all'interno dell'architettura NFV è stato necessario realizzare anche altri componenti, i connettori, che si pongono da intermediari nella comunicazione tra il Trust Monitor e gli elementi che compongono l'architettura NFV.

Con la realizzazione di questa tesi è stato possibile eseguire il processo di verifica di integrità di ogni funzione di rete e di ogni nodo fisico presente all'interno dell'infrastruttura NFV al fine di determinare possibili manomissioni. Il Trust Monitor permette dunque di migliorare l'attuale tecnologia NFV. Il modulo sviluppato in questa tesi porta a miglioramenti in termini di rilevamento di attacchi informatici all'interno del Cloud Computing, portando dunque al rilevamento di manomissioni delle funzioni di rete virtuali.

6.2 Sviluppi futuri

Lo sviluppo dell'architettura proposta ha migliorato l'attuale situazione di verifica di integrità all'interno della piattaforma NFV. In questa sezione verranno discusse le problematiche relative alla limitazione offerta dal framework di attestazione utilizzato all'interno dell'architettura proposta e verranno forniti dei possibili sviluppi futuri che potrebbero migliorare il processo di verifica di integrità delle VNF.

L'architettura proposta in questa tesi utilizza come framework di attestazione una versione di OAT estesa dal gruppo TORSEC realizzata nell'ambito del progetto europeo SECURED. Il Framework OAT è stato esteso per supportare all'interno dell'Integrity Report anche le misure IMA relative ai container Docker istanziati all'interno dell'host fisico da attestare. Nel contesto di questa tesi l'utilizzo del Framework OAT permette di svolgere l'attestazione delle VNF istanziate all'interno di container Docker.

Il Framework OAT è un framework di attestazione deprecato e per il suo funzionamento richiede l'utilizzo del TPM 1.2. Per tale ragione l'utilizzo del Framework OAT potrebbe causare una limitazione nell'esecuzione del processo di RA fornito dal Trust Monitor. Al fine di non rendere vincolante l'utilizzo del Trust Monitor al solo framework OAT, tale componente è stato sviluppato in modo da permettere l'utilizzo di qualsiasi framework di attestazione. Per fare ciò è necessario che venga creato un driver di attestazione specifico che funga da tramite nella comunicazione tra il Trust Monitor e il framework di attestazione scelto, come ad esempio il più recente Open Cloud Integrity Technology (CIT) [37].

La soluzione proposta in questa tesi potrebbe essere migliorata andando a lavorare su due aspetti che allo stato attuale potrebbero essere problematici, questi sono:

- Aggiungere un nuovo framework di attestazione;
- Ottenere l'associazione tra il container Docker ospitante una VNF e il nome o l'ID della VNF stessa, in modo da distinguere su quale container è presente una data VNF.

Il primo punto da discutere è che l'attuale architettura proposta utilizza un framework di attestazione deprecato che fa uso del TPM 1.2. Per questo motivo potrebbe essere interessante utilizzare un framework di attestazione basato sul TPM 2.0, ma occorre anche che il framework scelto venga esteso in modo da permettere di considerare durante la procedura di RA anche le misure relative ai container Docker istanziati all'interno del nodo.

Per quanto riguarda il secondo punto occorre considerare che allo stato attuale non è presente nessuna indicazione che ci permette di distinguere in quale container Docker sono in esecuzione le VNF. Un possibile sviluppo futuro sarebbe quello di riuscire a trovare un associazione tra il container Docker che ospita la VNF e la VNF stessa.

Le VNF, nel contesto di questa tesi, vengono istanziate tramite la comunicazione tra OSM Release 3 e VIM Emulator. OSM fornisce a VIM Emulator il nome delle VNF da istanziare, successivamente le VNF vengono istanziate all'interno di container Docker. Il problema è che non esiste un'associazione tra l'ID utilizzato da OSM per identificare una VNF e l'ID associato a un container Docker che contiene la VNF. Questo non ci permette di distinguere quale sia la VNF

istanziata all'interno di un container Docker. Una possibile soluzione potrebbe essere quella di riuscire ad estendere VIM Emulator al fine di ottenere l'associazione tra l'ID della VNF e l'ID del container Docker in cui è contenuta. Se si riuscisse a fare ciò potrebbe essere esteso anche il Trust Monitor aggiungendo la possibilità di attestare soltanto una data VNF presente in un dato host fisico.

Appendice A

Manuale Utente

Per utilizzare la soluzione realizzata in questa tesi è necessario effettuare una serie di configurazioni per permettere di effettuare correttamente il deployment di tutti gli elementi che compongono l'architettura proposta. In questo capitolo verranno dettate le linee guida per effettuare correttamente il deployment di tutti i componenti utilizzati al fine di replicare correttamente l'ambiente di esecuzione. Più nel dettaglio nella sezione A.1 verrà descritto in che modo devono essere configurati tutti i componenti utilizzati dal Trust Monitor per eseguire la procedura di verifica di integrità, la sezione A.2 ci permette di spiegare come configurare correttamente il nodo fisico da attestare, infine nella sezione A.3 verrà descritto come istanziare correttamente l'architettura realizzata in questa tesi.

A.1 Configurazione architettura

In questa sezione verranno descritte le linee guida per effettuare il deployment di tutta l'infrastruttura utilizzata dal Trust Monitor per concludere con successo il processo di verifica di integrità. I componenti che dovranno essere installati sono:

- Appraiser Open Attestation (OAT);
- Open Source MANO Release 3 (OSM);
- Il Whitelist Database.

A.1.1 Installazione Appraiser OAT

Il Trust Monitor per verificare l'integrità di un nodo contenente delle VNF deve comunicare con il framework di attestazione, che nel contesto di questa tesi è OAT. Al fine di svolgere il processo di RA occorre installare l'Appraiser di OAT, per semplificare la sua installazione viene utilizzato un file ansible che si occupa di installare in automatico il server OAT. L'Appraiser OAT per il suo funzionamento può essere installato su qualsiasi Sistema Operativo, nell'ambito di questa tesi è stato installato all'interno di una macchina con Sistema Operativo CentOS 7.

All'interno della directory `sorgenti_tesi/appraiser_oat` fornita con i sorgenti della tesi è presente il server OAT, i passaggi necessari per installare tale componenti sono:

- Installazione delle dipendenze:

  ```
  sudo yum install ansible
  ```

- Configurazione delle variabili di ambiente all'interno del file `install_centos7.yml` fornito con i sorgenti della tesi all'interno della directory `sorgenti_tesi/oat_appraiser/ansible`:

```
- hosts: localhost

  vars:
    OAT_HOME: /opt/OpenAttestation
    OAT_HOSTNAME: verifier
    OAT_IP: localhost
```

Il valore `OAT_HOSTNAME` rappresenta il nome con cui viene definito l'Appraiser OAT, mentre `OAT_HOME` rappresenta la directory in cui verrà installato il server OAT.

- Installazione Appraiser OAT, per utilizzare questo comando bisogna essere all'interno della directory `ansible`:

```
ansible-playbook -i "localhost," -c local install_centos7.yml
```

Al termine dell'installazione è possibile testare il corretto funzionamento dell'Appraiser OAT provando a collegarsi tramite un browser all'indirizzo `http://nome_appraiser/OAT/alerts.php`, dove `nome_appraiser` fa riferimento al nome con il quale l'Appraiser viene installato all'interno del sistema, a tale indirizzo risponde un'interfaccia web dalla quale è possibile recuperare una serie di informazioni sugli host attestati.

A.1.2 Installazione Open Source MANO

In questa sezione verrà installato l'Orchestratore utilizzato nel contesto di questa tesi per istanziare le VNF all'interno di un VIM. L'Orchestratore utilizzato è Open Source MANO Release 3 (OSM) [19] e per il suo utilizzo deve essere installato all'interno di una macchina con Sistema Operativo Ubuntu16-04 a 64 bit. La macchina deve disporre di 8 CPU, 16 GB di RAM e 100 GB di disco.

I passi necessari per effettuare l'installazione di OSM sono:

- Installare LXD:

```
sudo apt-get update
sudo apt-get install -y lxd
newgrp lxd
```

- Configurare LXD:

```
sudo lxd init
```

- Installare OSM:

```
wget https://osm-download.etsi.org/ftp/osm-3.0-three/install_osm.sh
chmod + x install_osm.sh
./install_osm.sh --source
```

Al termine dell'installazione per verificare che la procedura si è conclusa con successo è possibile aprire un browser e connettersi al seguente indirizzo `https://1.2.3.4:8443`, sostituendo `1.2.3.4` con l'indirizzo IP dell'host, se verrà avviata una pagina di login allora l'installazione è terminata con successo. Per accedere alle funzionalità offerte da OSM è necessario effettuare il login con le credenziali admin admin. È possibile consultare la pagina ufficiale di OSM Release 3 [19] per ottenere maggiori dettagli.

A.1.3 Installazione Whitelist Database

L'ultimo componente che deve essere installato per replicare l'ambiente utilizzato dal Trust Monitor al fine di verificare lo stato di integrità del nodo e delle VNF istanziate al suo interno è il Whitelist Database, si tratta di un database basato su Cassandra. Tale componente è essenziale perché permette al Trust Monitor di confrontare se le misure relative al software in esecuzione all'interno delle VNF risultano corrispondere con quelle presenti all'interno di tale database. Il Whitelist Database è utilizzato per ottenere l'esito del processo di attestazione per verificare se il nodo presente all'interno dell'infrastruttura NFV risulta essere "trusted" o meno.

L'installazione del Whitelist Database richiede per il suo corretto funzionamento dei requisiti minimi di sistema, questi sono:

- 2 CPU;
- 4 GB RAM;
- Il Sistema Operativo CentOS 7.

Per effettuare l'installazione del Whitelist Database è possibile utilizzare Docker che permette di istanziare il database direttamente all'interno di un container Docker.

I sorgenti utilizzati per installare il Whitelist Database sono presenti all'interno della directory `sorgenti_tesi/whitelist_database`. I passaggi necessari per svolgere l'installazione del Whitelist Database all'interno di container Docker sono:

- Installazione dipendenze:

```
sudo yum-config-manager --add-repo
    https://download.docker.com/linux/centos/docker-ce.repo
sudo yum install docker-ce
```

- Installazione di Docker Compose:

```
sudo curl -L https://github.com/docker/compose/releases/download/1.19.0\
/docker-compose-'uname -s'-'uname -m' -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

- (opzionalmente) Modificare il contenuto del file `ra.conf` e `pkgs_download_list.conf` presenti all'interno della directory `conf`. Se si vuole popolare il contenuto del databse con i digest dei pacchetti software relativi a CentOS 7 e per EPEL non bisogna modificare tali file.
- Avviare il container Docker:

```
sudo docker-compose up --build
```

Al termine della procedura di creazione del container il Whitelist Database sarà in esecuzione correttamente.

A.2 Configurazione piattaforma da attestare

In questa sezione verranno discussi passo per passo tutte le operazioni da eseguire nel configurare il nodo fisico da attestare presente all'interno dell'infrastruttura NFV allo scopo che possa essere utilizzato per istanziare delle VNF e che quindi in un secondo momento possa essere verificato dal Trust Monitor.

Nel contesto di questa tesi l'host fisico è dotato del Sistema Operativo CentOS 7.5.1804 e per il suo funzionamento è stato necessario utilizzare il kernel Linux 4.4.19. La scelta di utilizzare un kernel differente da quello distribuito con la versione del Sistema Operativo CentOS è motivata

dall'utilizzo del Framework OAT esteso che permette di svolgere l'attestazione dei container Docker in esecuzione all'interno del nodo. Infatti per permettere il funzionamento del framework OAT è necessario estendere il funzionamento di IMA includendo un nuovo template. Le modifiche applicate ad IMA aggiungono un nuovo template che si occupa di distinguere le misure relative all'host fisico dalle misure relative a un container Docker. Per apportare queste modifiche bisogna utilizzare particolari file sorgenti di IMA non disponibili nel kernel 3.10.0-862.el7 fornito con il Sistema Operativo CentOS 7.5.1804. Un ulteriore vincolo è che per permettere il funzionamento del Framework OAT è necessario che l'host fisico abbia a disposizione il TPM 1.2.

I passi che devono essere eseguiti per configurare correttamente la piattaforma da attestare sono:

- effettuare il download del kernel Linux 4.4.19 e applicare la patch di IMA e infine compilare il kernel Linux modificato, al fine di rendere effettiva l'estensione di IMA per permettere di distinguere le misure appartenenti all'host fisico da quelle prodotte da un container Docker;
- abilitare IMA, con lo scopo di misurare il software in esecuzione all'interno dell'host e dei container Docker;
- installare VIM Emulator per permettere l'istanziazione delle VNF all'interno dell'host;
- configurare Docker;
- installare il client OAT, per permettere la procedura di RA;
- estendere il funzionamento di Docker CLI.

A.2.1 Gestione kernel e applicazione patch IMA

Per utilizzare la versione di IMA modificata è necessario compilare il kernel Linux per rendere effettive le modifiche apportate a tale modulo. In questa sezione verrà eseguito il download del kernel Linux 4.4.19 in modo da ottenere i suoi file sorgenti, successivamente verrà applicata la patch IMA e verrà compilata una versione personalizzata del kernel di Linux.

I passi necessari per effettuare il download dei sorgenti del kernel Linux sono i seguenti:

- Installazione delle dipendenze necessarie:

```
sudo yum groupinstall "Development Tools" -y
sudo yum install hmaccalc zlib-devel binutils-devel openssl-devel
    elfutils-libelf-devel ncurses-devel bc wget -y
```

- Creazione di una directory nella quale inserire i sorgenti del kernel:

```
mkdir myCustomKernel
cd myCustomKernel
```

- Effettuare il download dei sorgenti del kernel Linux 4.4.19 dal sito https://www.kernel.org/:

```
wget https://www.kernel.org/pub/linux/kernel/v4.x/linux-4.4.19.tar.gz
```

- Al termine della procedura di download avremo all'interno di un archivio `tar.gz` i sorgenti del kernel Linux 4.4.19. Occorre estrarre i dati contenuti nell'archivio:

```
tar -xvzf linux-4.4.19.tar.gz
```

All'interno della cartella `linux-4.4.19` sono presenti tutti i sorgenti che compongono il kernel Linux.

Al termine dell'operazione di download è possibile applicare la patch IMA che va a modificare i file:

- `security/integrity/ima/Kconfig`;
- `security/integrity/ima/ima_template.c`;
- `security/integrity/ima/ima_template_lib.c`;
- `security/integrity/ima/ima_template_lib.h`.

La patch IMA è fornita con i sorgenti della tesi all'interno della directory `sorgenti_tesi/patch_ima` e per applicare correttamente la patch occorre:

- Spostare il file `ima_patch.tar.gz` presente all'interno della directory sopracitata, nella directory `myCustomKernel` precedentemente creata. Successivamente bisogna spostarsi nella carella contenente i sorgenti del kernel Linux ottenuti al passo precedente:

  ```
  cd myCustomKernel/linux-4.4.19
  ```

- Applicare la patch IMA:

  ```
  zcat ../ima.patch.gz | patch -p1
  ```

Al termine di questa procedura i sorgenti risulteranno modificati correttamente.

L'ultima operazione da effettuare per utilizzare il kernel Linux 4.4.19 modificato consiste nel compilare e installare il nuovo kernel. Prima di compilare il kernel è possibile selezionare il nuovo template IMA direttamente dal menù di configurazione del kernel. I passaggi necessari per effettuare quest'ultima operazione sono:

- Avviare il menù di gestione del kernel Linux per selezionare il template IMA:

  ```
  make menuconfig
  ```

  Selezionare all'interno del menù mostrato a video la voce `Security options` premendo invio, poi all'interno della sezione `Integrity Measurement Architecture (IMA)` selezionare sempre premendo invio `Default template`. Se `Integrity Measurement Architecture (IMA)` non risulta essere abilitato, abilitare tale modulo. Una volta fatto questo si aprirà una nuova finestra dalla quale è possibile selezionare il template `ima-cont-id` utilizzato nel contesto di questa tesi. Al termine occorre salvare e uscire dal menù.

- Per avviare la compilazione del kernel occorre eseguire il comando:

  ```
  make
  ```

- Al termine il kernel risulterà compilato, per installarlo bisogna eseguire il comando:

  ```
  sudo make modules_install install
  ```

- In conclusione occorre aggiungere il kernel alla lista dei kernel disponibili all'avvio del sistema:

  ```
  sudo /usr/sbin/grub2-mkconfig -o /boot/grub2/grub.cfg
  ```

Al fine di constatare che la procedura si è conclusa correttamente è possibile riavviare la macchina e verificare che il nuovo kernel sia disponibile all'interno della lista dei kernel selezionabili all'avvio.

A.2.2 Configurare IMA

In questa sezione vengono definiti i passi necessari per utilizzare il nuovo template di IMA e definire una politica di misurazione. IMA è un modulo integrato all'interno del kernel Linux e risulta essere attivabile tramite il comando `ima_tcb` passato come parametro all'avvio del kernel stesso. Per aggiungere un parametro al kernel occorre modificare il bootloader grub di Linux, più precisamente occorre aggiungere all'interno del file `/etc/default/grub` una nuova riga in coda al file chiamata `GRUB_CMDLINE_LINUX_DEFAULT`. All'interno di questa ria occorre aggiungere il comando `ima_tcb` e il comando `ima_template=ima-cont-id`, `ima_template` ci permette di definire quale template deve essere utilizzato da IMA, mentre `ima-cont-id` rappresenta il nuovo template IMA utilizzato nel contesto di questa tesi. Per rendere effettive le modifiche applicate al file grub occorre utilizzare il comando:

```
sudo /usr/sbin/grub2-mkconfig -o /boot/grub2/grub.cfg
```

Al riavvio del sistema il modulo IMA dovrebbe essere in grado di misurare correttamente, tramite il nuovo template, tutti i moduli presenti all'interno della macchina secondo le regole di default di IMA, per verificare ciò è possibile osservare il contenuto del file `ascii_runtime_measurements` presente all'interno della directory `/sys/kernel/security/integrity/ima`.

È possibile che IMA non risulti essere attivo di default, per verificare questa condizione occorre constatare se è presente, all'interno della macchina che si sta considerando, la lista delle misure effettuate da IMA sotto il percorso `/sys/kernel/security/integrity/ima`. Se non è presente occorre ricompilare nuovamente il kernel perché non è stato selezionato il modulo IMA dunque non è disponibile all'avvio del sistema.

Il funzionamento di IMA è regolato da un file che ne definisce la politica di misurazione, dunque una una volta abilitato IMA occorre definire tale file. Avviando il modulo IMA con il comando `ima_tcb` verrà avviata la politica di misurazione di default che indica a IMA di misurare tutti gli eseguibili prima di essere lanciati all'interno del sistema, tutti i file mappati in memoria e tutti i file aperti in lettura per l'utente root. All'interno della figura A.1 è possibile osservare la politica di default utilizzata da IMA. Ogni riga presente all'interno di questo file definisce una regola di misurazione applicabile a IMA, le righe che iniziano con la keyword `dont_measure` fanno riferimento a una condizione che non deve essere misurata da IMA, mentre quelle con `measure` rappresentano le condizioni misurate da IMA.

```
dont_measure fsmagic=PROC_SUPER_MAGIC
dont_measure fsmagic=SYSFS_MAGIC
dont_measure fsmagic=DEBUGFS_MAGIC
dont_measure fsmagic=TMPFS_MAGIC
dont_measure fsmagic=SECURITYFS_MAGIC
dont_measure fsmagic=SELINUX_MAGIC
measure func=BPRM_CHECK
measure func=FILE_MMAP mask=MAY_EXEC

< add LSM specific rules here >

measure func=PATH_CHECK mask=MAY_READ uid=0
```

Figura A.1. Politica di default utilizzata da IMA.

IMA permette di misurare diversi tipi di file specificando una funzione (`func`) i cui tre valori principali sono:

- `BPRM_CHECK`: la regola viene applicata a tutti i file eseguibili;
- `FILE_MMAP`: la regola viene applicata a tutti i file mappati i memoria;

- `FILE_CHECK`: la regola viene applicata a tutti i file che non rientrano nelle categorie precedenti.

All'interno della condizione della regola, dopo aver specificato la funzione a cui fa riferimento, è possibile indicare anche una maschera (`mask`) che può assumere tre diversi valori:

- `MAY_READ`: fa riferimento a tutti i file aperti in lettura;
- `MAY_WRITE`: fa riferimento a tutti i file aperti in scrittura;
- `MAY_EXEC`: fa riferimento a tutti i file che possono essere eseguiti nel sistema.

In figura A.2 è presente la politica di misurazione per IMA definita nel contesto di questa tesi, questa politica permette misurare tutti i file in esecuzione mappati in memoria e tutti i file eseguibili in esecuzione.

```
measure func=BPRM_CHECK mask=MAY_EXEC
measure func=FILE_MMAP mask=MAY_EXEC
```

Figura A.2. Politica di misurazione per IMA utilizzata nel contesto di questa tesi.

Nel contesto di questa tesi occorre applicare una politica di misurazione ad IMA, questo viene fatto mediante la creazione di un file contente le regole presenti in figura A.2. I passaggi necessari per fare questo sono:

- Creare una directory chiamata `ima` sotto `/etc`:

  ```
  mkdir /etc/ima
  ```

- Creare il file `ima-policy` contenente la politica di misurazione presente in figura A.2 utilizzata nel contesto di questa tesi.

Al riavvio dalla macchina il modulo IMA è in grado di misurare correttamente i file in base alle regole definite nel file `ima-policy`.

A.2.3 Installazione VIM Emulator

Nel contesto di questa tesi il nodo fisico deve avere al suo interno un VIM che permetta di istanziare delle VNF all'interno di container Docker. Il VIM utilizzato è VIM Emulator [34] che consente di eseguire funzioni di rete reali all'interno di container Docker e il nucleo della piattaforma offerta da VIM Emulator è basata su Containernet [38]. Tale VIM riesce ad interfacciarsi correttamente con Open Source MANO Release 3 che permette di avviare l'istanziazione delle VNF.

Per installare correttamente VIM Emulator all'interno dell'host in questione bisogna effettuare:

- Installazione delle dipendenze:

  ```
  sudo yum install ansible git aptitude
  ```

- Installazione di Containernet:

  ```
  cd
  git clone https://github.com/containernet/containernet.git
  cd ~/containernet/ansible
  sudo ansible-playbook -i "localhost," -c local install_centos.yml
  ```

- Per l'installazione di VIM Emulator, occorre effettuare il download tramite git clone della pull request dal repository https://osm.etsi.org/gitweb/?p=osm/vim-emu.git;a=summary che identifica l'installazione per il Sistema Operativo CentOS, una volta fatto questo si può procedere con l'installazione:

```
cd ~/vim-emu/ansible
sudo ansible-playbook -i "localhost," -c local install.yml
```

Al termine dell'installazione di VIM Emulator, per permettere al Trust Monitor di recuperare la lista degli ID dei container Docker che contengono le VNF in esecuzione all'interno dell'host e a OSM di istanziare le VNF bisogna:

- Avviare il demone Docker affinché sia in ascolto sul socker unix e sul socker tcp, per fare questo occorre prima stoppare il demone Docker e poi riavviarlo:

```
service docker stop
sudo dockerd -H unix:///var/run/docker.sock -H tcp://0.0.0.0:2375
```

- Avviare VIM Emulator in modo che sia in ascolto di richieste di istanziatione di VNF:

```
python vim-emu/examples/osm_default_daemon_topology_2_pop.py
```

Mediante quest'ultimi passi VIM Emulator risulta essere in esecuzione ed è in attesa di richieste di istanziazione da parte di OSM. Affinché OSM sia in grado di contattare VIM Emulator è necessario aggiungere all'interno di OSM il VIM, questo può essere fatto tramite l'interfaccia grafica offerta da OSM oppure tramite il client OSM.

Per far si che il Trust Monitor e OSM riescano a comunicare con VIM Emulator è necessario che vengano aperte delle porte, queste sono: 9005, 9006, 9775, 9776, 8080, 6001, 6002, 6653, 4000, 10243, 10244, 5001, 10697, 10698, 2375.

A.2.4 Configurare Docker

Il prossimo passaggio da effettuare è quello di configurare correttamente Docker. Docker viene installato quando viene eseguita l'installazione di containernet. Nell'ambito di questa tesi Docker deve utilizzare lo storage driver `devicemapper`, per verificare lo storage driver utilizzato da Docker è possibile eseguire il comando `docker info`. Se lo storage driver è `devicemapper` non bisogna configurare ulteriormente Docker alternativamente bisogna modificare lo storage driver. Di default lo storage driver associato a Docker in esecuzione all'interno di CentOS7 è `overlay2`.

Per modificare lo storage driver associato a Docker bisogna:

- Stoppare il demone Docker in esecuzione nell'host:

```
service stop docker
```

- Modificare il file `daemon.json` se è già presente oppure creare tale file sotto il percorso `/etc/docker` e aggiungere al suo interno l'oggetto Json presente in figura A.3.

- Avviare nuovamente il demone Docker:

```
service start docker
```

Al fine di verificare se la procedura si è conclusa con successo è possibile eseguire nuovamente il comando `docker info` e controllare se lo storage driver utilizzato è `devicemapper`.

```
{
  "storage-driver": "devicemapper"
}
```

Figura A.3. Contenuto del file `daemon.json` per la definizione dello storage driver `devicemapper`.

A.2.5 Installazione HostAgent OAT

Il Framework OAT per il suo funzionamento richiede che il nodo che necessita di essere attestato abbia installato al suo interno il software che identifica l'HostAgent OAT. Questa sezione è dedicata alla descrizione dei passi necessari per installare l'HostAgent OAT.

La macchina che necessita di essere attestata deve essere fornita del chip TPM 1.2 che deve essere attivato attraverso il BIOS della macchina. Al fine di concludere questa fase con successo è necessario che la componente Appraiser OAT sia in esecuzione correttamente.

I passi da eseguire per effettuare l'installazione del client OAT sono:

- Installazione dipendenze:

  ```
  yum install bzip2 wget gcc gmp-devel unzip trousers automake autoconf
      libtool pkg-config gettext perl python flex bison gperf
      trousers-devel java-1.7.0-openjdk java-1.7.0-openjdk-devel
  ```

- Inserire all'interno del file `/etc/hosts` l'associazione nome e IP che identificano la località dell'Appraiser OAT:

  ```
  echo 'xxx.xxx.xxx.xxx \t nome_appraiser' > /etc/hosts
  ```

  Dove `xxx.xxx.xxx.xxx` rappresenta l'indirizzo IP dell'Appraiser OAT e `nome_appraiser` rappresenta il nome con cui viene installato l'Appraiser OAT.

- Avviare il servizio di comunicazione con il chip TPM, il chip deve essere attivato tramite il BIOS:

  ```
  systemctl start tcsd
  systemctl enable tcsd
  ```

- Scaricare dall'Appraiser OAT il fle `ClientInstallForLinux.zip` contenente il programma client HostAgent per la costruzione e l'invio dell'Integrity Report contenente le misure di integrità:

  ```
  wget http://verifier/ClientInstallForLinux.zip
  unzip ClientInstallForLinux.zip
  ```

- Modificare il file `OAT.properties` presente all'interno della directory `/OAT` aggiungendo una nuova riga in coda contenente il flag `AddContainerAnalysisSupport` settato uguale a True al fine di supportare l'analisi per i container Docker.

- Definire il nome della macchina:

  ```
  echo 'nome_macchina' > /etc/hostname
  ```

- Registrare l'host presso l'Appraiser:

  ```
  cd ClientInstallForLinux
  sudo sh genera-install.sh
  ```

  Durante questa fase, il client si registra presso il WebService dell'Appraiser con il nome con cui la macchina è chiamata all'interno del file `/etc/hostname`.

- Annotarsi il valore del PCR0 del TPM utilizzato per registrare il nodo presso il Trust Monitor e di conseguenza presso l'Appraiser OAT:

  ```
  cd /sys/deices/pnp0
  find . -iname "pcrs"
  cat 00:0x/pcrs
  ```

  Il valore x deve essere sostituito in base all'output ottenuto dal comando `find`.

- In conclusione far partire il servizio OATClient che resta in attesa di richieste di attestazione da parte dell'Appraiser OAT:

  ```
  service OATClient start
  ```

L'HostAgent fornisce anche il file di log `/var/log/OAT.log` nel quale è possibile osservare se avviene la comunicazione tra l'Appraiser e l'HostAgent.

A.2.6 Estendere Docker CLI

Il Framework OAT esteso per il suo funzionamento richiede che la componente HostAgent abbia installato una versione di Docker CLI estesa, perché viene utilizzato un comando non presente nella versione base. Il comando è stato realizzato nel contesto di un'altra tesi e permette di ottenere in maniera rapida l'associazione tra il numero di device che identifica un container Docker e il valore dell'ID associato al container stesso, tale comando prende il nome di `raInfo`.

Al fine di estendere l'attuale versione di Docker CLI presente all'interno dell'elaboratore occorre partire necessariamente dai sorgenti di Docker CLI ottenibili tramite github e successivamente applicare una patch. La patch prende il nome di `docker-raInfo.patch` ed è fornita con i sorgenti della tesi sotto il percorso `sorgenti_tesi/patch_docker_cli`.

I passaggi necessari per estendere Docker CLI sono:

- Installazione delle dipendenze necessarie:

  ```
  sudo yum install git make
  ```

- Creare una directory nella quale inserire i sorgenti Docker CLI:

  ```
  mkdir docker-cli
  cd docker-cli
  ```

- Effettuare il download dei sorgenti Docker CLI:

  ```
  sudo git clone https://github.com/docker/cli.git
  cd cli
  ```

- Spostare la patch `docker-raInfo.patch` all'interno della directory `docker-cli/cli`.

- Applicare la patch:

  ```
  sudo git apply < docker-raInfo.patch
  ```

- Compilare la versione di Docker CLI:

  ```
  sudo make -f docker.Makefile rabinary
  ```

Al termine di questi passaggi il file binario che rappresenta la nuova versione di Docker CLI è presente sotto la directory `docker-cli/cli/build`.

Per verificare se la procedura si è conclusa con successo è possibile eseguire la nuova versione di Docker CLI per controllare se nell'elenco dei comandi è presente il nuovo comando `raInfo`. Il passaggio conclusivo consiste nel rimpiazzare il file binario rappresentante la precedente versione di Docker CLI con la versione modificata.

A.3 Installazione architettura Trust Monitor

In questa sezione verrà effettuato il deployment di tutta l'architettura realizzata in questa tesi. L'architettura è composta dal Trust Monitor e dei connettori utilizzati per comunicare con le altre entità dell'architettura NFV.

Nel contesto di questa tesi l'architettura del Trust Monitor è eseguita all'interno di una macchina avente come Sistema Operativo Ubuntu16-04. I file sorgenti utilizzati per effettuare il deployment dell'archiettura del Trust Monitor sono presenti nella directory `sorgenti_tesi/ra-trust-monitor` e per effettuare correttamente il deployment dell'architettura partendo dai sorgenti del Trust Monitor bisogna:

- Installare Docker engine:

```
sudo apt-get remove docker docker-engine docker.io
sudo apt-get update
sudo apt-get install apt-transport-https ca-certificates curl
    software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key
    add -
sudo add-apt-repository "deb [arch=amd64]
    https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
sudo apt-get update
sudo apt-get install docker-ce
```

- Installare Docker compose:

```
sudo curl -L https://github.com/docker/compose/releases/download\
    /1.18.0/docker-compose-Linux-x86_64 -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

- Configurare il reverse proxy utilizzato dal Trust Monitor. Per farlo è necessario fornire una chiave e un certificato sotto la directory `reverseProxy/ssl` prima di avviare lo script di Docker Compose. Questo può essere fatto mediante il tool di Ubuntu `make-ssl-cert`. Il valore della chiave ssl e del certificato devono essere inseriti nel file `test.ra.trust.monitor.key` presente nella directory `ssl/private` e nel file `test.ra.trust.monitor.chain` presente nella directory `ssl/private` e `ssl/certs`.

- Configurare l'applicazione del Trust Monitor creando un file chiamato `local_setting.py` da inserire all'interno della directory `trustMonitor/trust_monitor_django`:

```
LOCAL_SETTINGS = True
from settings import *
ALLOWED_HOSTS += ['ip_address_tm']
CASSANDRA_LOCATION = $WHITELIST_DB_IP
CASSANDRA_PORT = '9160'
OAT_LOCATION = $IP_APPRAISER_OAT
```

  Dove `CASSANDRA_LOCATION` rappresenta l'indirizzo IP del Whitelist Database, `OAT_LOCATION` rappresenta l'indirizzo IP associato all'Appraiser OAT e `CASSANDRA_PORT` è la porta alla quale il Whitelist Database è in ascolto.

- Definire indirizzo IP di Open Source MANO:

```
sudo export OSM_IP=<ip OSM>
```

- Configurare l'Appraiser OAT all'interno del file `docker-compose.yml`:

```
tm_django_app:
  image: ra/trust_monitor/tm_django_app
  build: ./trustMonitor
  environment:
    - RUN_DJANGO_APP=1
  depends_on:
    - tm_static_serve
  extra_hosts:
    - "$OAT_VERIFIER_CN":$OAT_VERIFIER_IP"
```

Dove `OAT_VERIFIER_CN` rappresenta il nome con cui l'Appraiser OAT è stato installato e `OAT_VERIFIER_IP` rappresenta l'indirizzo IP associato al server OAT.

- Spostare i file `setting.py` e `start_verify.py` contenuti nella directory `OAThelper` in una posizione scelta dall'utente all'interno della macchina in cui è in esecuzione l'Appraiser OAT. Dopo di che bisogna associare al file `start_verify.py` i permessi di esecuzione e impostare all'interno del file `setting.py` l'URL di base del Trust Monitor utilizzata per effettuare la callback.

- Configurare il percorso remoto relativo al file `start_verify.py` (definito al passaggio precedente) utilizzato da parte dell'Appraiser OAT nel file `driverOATSettings.py` contenuto nella directory `trustMonitor/trust_monitor_driver`:

  ```
  PATH_DRIVER = '/$OAT_TM_DIR/start_verify.py'
  ```

- Scaricare e aggiungere il certificato che identifica l'Appraiser OAT all'interno della directory `trustMonitor/docker/ssl/certs`:

  ```
  openssl s_client -showcerts -connect $OAT_VERIFIER_IP:8443 </dev/null
      2>/dev/null|openssl x509 -outform PEM >
      trustMonitor/docker/ssl/certs/ra-oat-verifier.pem
  ```

  Dove `OAT_VERIFIER_IP` rappresenta l'indirizzo IP associato all'Appraiser OAT. Il comando precedente salverà il certificato X.509 in formato PEM nel file `ra-oat-verifier.pem`.

- Effettuare il deployment dell'architettura tramite Docker Compose:

  ```
  sudo docker-compose up --build
  ```

Al fine di verificare che ogni componente sia in esecuzione correttamente è possibile utilizzare il comando `sudo docker-compose ps` che restituisce l'elenco di tutti i container Docker in esecuzione.

Appendice B

Manuale Programmatore

Nel corso dei precedenti capitoli, abbiamo illustrato l'architettura e l'implementazione proposta per la realizzazione di un sistema che riesca a svolgere l'attestazione di VNF istanziate in host fisici presenti nell'infrastruttura NFV. Nel capitolo precedente abbiamo visto come un utente possa configurare ed utilizzare la soluzione proposta. In questo capitolo, vediamo come estendere il lavoro svolto. Più precisamente all'interno della sezione B.1 verrà discusso come aggiungere una nuova API all'interno del Trust Monitor, nella sezione B.2 verranno dettate le linee guida per creare un nuovo driver di attestazione per effettuare il processo di RA e infine nella sezione B.3 verrà discusso come aggiungere un nuovo connettore all'interno dell'architettura del Trust Monitor.

B.1 Definizione nuova API

In questa sezione verranno definiti i passaggi necessari per inserire una nuova API all'interno del Trust Monitor. Per aggiungere una nuova API occorre:

- creare la classe che si occupa di gestire la nuova API;
- definire il nome tramite il quale l'API risulta essere raggiungibile;
- (opzionale) creare una nuova tabella nel database per memorizzare le informazioni provenienti dall'API;
- (opzionale) creare una classe che si occupa di effettuare una serializzazione dei dati provenienti dall'API.

Quando viene richiesto di aggiungere una nuova API è necessario analizzare se si tratta di un'API che accetta in ingresso dei parametri, oppure un'API che richiede che i dati passati alla stessa vengano memorizzati all'interno di un database. Negli altri casi è possibile creare l'API specificando l'URL a cui l'API risulta raggiungibile e la classe che si occupa di gestire tale API nella quale bisogna specificare i metodi HTTP messi a disposizione dall'API stessa.

B.1.1 Aggiungere classe per gestire API

Al fine di aggiungere una nuova API all'interno dell'applicazione del Trust Monitor è necessario scaricare i sorgenti del Trust Monitor forniti nei sorgenti della tesi e presenti nella directory `sorgenti_tesi/ra-trust-monitor` e successivamente definire all'interno del file `views.py` presente nella directory `trustMonitor/trust_monitor` una nuova classe, chiamata ad esempio `TestAPI`, che si occupa di gestire la nuova API.

All'interno di Django REST framework ogni API definita fa riferimento a una classe nella quale occorre definire tutti i metodi HTTP messi a disposizione dalla stessa e per fare cioè è necessario

che la classe estenda `APIView` una classe offerta dal framework REST che fornisce la possibilità di definire i vari metodi HTTP offerti dall'API che si intende creare. I metodi messi a disposizione da `APIView` sono:

- `get`;
- `post`;
- `delete`;
- `update`.

Ognuno di questi metodi prende in ingresso tre parametri, `self` parametro utilizzato per fare riferimento all'istanza stessa, `request` utilizzato per passare dei dati in ingresso tramite l'API e `format` fa riferimento al tipo di formato associato ai dati passati a tale metodo HTTP (es. Json). Una volta definita la classe è necessario definire l'URL a cui fa riferimento.

B.1.2 Definire URL per contattare API

Le API vengono aggiunte in Django tramite un file che contiene tutte le URL definite per l'applicazione a cui facciamo riferimento. Nel contesto di questa tesi l'API viene aggiunta all'interno del file `urls.py` presente all'interno della directory `ra-trust-monitor/trustMonitor/trust_monitor` fornito con con i sorgenti della tesi. Oltre a specificare il nome dell'API occorre anche specificare il nome della classe definita precedentemente e presente all'interno del file `views.py` che permette di definire tutti i metodi HTTP messi a disposizione dall'API appena definita.

Nella figura B.1 è presente un esempio di API aggiunta a quelle già messe a disposizione dal Trust Monitor. È possibile osservare che è stato necessario specificare a quelle classe presente nel file `views.py` tale API fa riferimento.

```
from django.conf.urls import url
from trust_monitor import views

urlpatterns = [
    ...
url(r'^test_nuova_API/$', views.TestAPI.as_view()),
]
```

Figura B.1. Esempio aggiunta nuova API fornita dal Trust Monitor

In un progetto Django possono essere definite più applicazioni, cioè Web Service differenti ognuno dei quali ha un proprio file `urls.py`. Ogni file `urls.py` contiene tutte le API definite per l'applicazione che si sta analizzato e viene collegato con il progetto complessivo Django tramite il file `urls.py` che lo identifica. Nel contesto di questa tesi quest'ultimo file è presente all'interno della directory `trustMonitor/trust_monitor_django`.

```
from django.conf.urls import include, url

urlpatterns = [
url(r'^', include('trust_monitor.urls'))
]
```

Figura B.2. Contenuto file `urls.py` associato all'interno progetto Django.

Nella figura B.2 è presente il collegamento tra il progetto Django e l'applicativo Django rappresentato in questo caso dall'applicazione del Trust Monitor.

B.1.3 Aggiungere classe in Model

Con i passaggi precedenti abbiamo definito l'API, ma ci sono dei casi in cui i dati processati dall'API oppure i dati passati all'API stessa devono essere memorizzati all'interno del database del Trust Monitor, in questi cosa occorre creare una nuova tabella nella quale memorizzare tali informazioni. Django fornisce un metodo molto semplice per la creazione della tabella all'interno del database al fine di memorizzare le informazioni passata all'API che si sta realizzando, per fare ciò è necessario creare una classe, chiamata ad esempio `TestAPIdb`, all'interno del file `models.py` presente nella directory `trustMonitor/trust_monitor`.

La classe `TestAPIdb` deve estendere la classe `models.Model` messa a disposizione da Django REST framework, si tratta di un classe che permette di gestire il database messo a disposizione da Django. All'interno di questa classe è possibile definire tutti i campi necessari che compongono la tabella da aggiungere all'interno del database. Ad ogni campo viene associato un tipo e in aggiunta è possibile specificare attributi aggiuntivi che definiscono il campo stesso, come ad esempio è possibile specificare il valore di default oppure il numero massimo o minimo di caratteri che il campo in questione può avere. All'interno della classe `TestAPIdb` è possibile in aggiunta indicare una classe interna di metadati chiamata `Meta`. Tramite questa classe è possibile ad esempio specificare il tipo di ordinamento dei vari record contenuti nel database gestito da Django.

```
from django.db import models

class TestAPIdb(models.Model):
    test1 = models.TextField(default='test')
    test2 = models.CharField(max_lenght=20, blank=False)

    class Meta:
        ordering = ('id', )
```

Figura B.3. Esempio di una nuova tabella che verrà aggiunta all'interno del database di Django.

Nella figura B.3 è possibile osservare la definizione della tabella `TestAPIdb` nella quale sono presenti due campi:

- `test1`: si tratta di un campo testuale dove se non viene specificato il valore associato ad esso verrà assegnato il valore rappresentato dall'attributo `default`;
- `test2`: campo di tipo `char` definito da due attributi aggiuntivi, `blank` viene utilizzato per indicare che il campo è obbligatorio mentre il secondo viene utilizzato per impostare la lunghezza massima del campo.

Inoltre è possibile osservare la presenza della classe `Meta` che gestisce i metadati, nell'esempio in figura tale classe aggiunge la colonna `id` all'interno della tabella `TestAPIdb` e ordina i vari record della tabella in base all'`id`, rappresentato da un numero intero autoincrementale.

B.1.4 Serializer

Una volta definita un'API è plausibile che occorre trasmettere alla stessa vari argomenti, perché ad esempio è possibile che determinati valori vengano utilizzati per processare delle operazioni definite dall'API stessa. Allo scopo di verificare che i parametri comunicati all'API risultano essere validi Django REST framework utilizza il concetto di serializzazione servendosi di una classe definita all'interno del file `serializer.py` presente nella directory `trustMonitor/trust_monitor`.

All'interno del file `serializer.py` è necessario creare una classe che si occupa di controllare che i valori passati all'API in questione siano presenti e che siano corretti in base ai campi indicati dalla classe. Questa classe per fornire un esempio durante l'esposizione verrà chiamata `TestSerializer`.

Un esempio di classe che si occupa di effettuare la serializzazione della classe `TestAPIdb` definita in precedenza è presente in figura B.4. È possibile osservare che la classe `TestSerializer` utilizza la classe `serializers.ModelSerializer` offerta da Django REST framework, tale classe permette di collegare il serializzatore a una classe presente all'interno del file `models.py`. L'attributo `model` fa riferimento alla classe presente in `models.py` a cui il serializzatore fa riferimento, mentre mediante l'attributo `fields` il serializzatore verifica che tutti i campi presenti nella classe definita da `model` siano presenti e siano conformi alle specifiche definite dalla classe a cui il serializzatore fa riferimento.

```
from rest_framework import serializers
from trust_monitor.models import TestAPIdb

class TestSerializer(serializers.ModelSerializer):
       class Meta:
              model = TestAPIdb
              fields = ('id',
                              'test1',
                              'test2')
```

Figura B.4. Esempio classe `TestSerializer` che si occupa di verificare se gli attributi trasmessi all'API risultano essere corretti.

Il serializzatore può anche non essere associato a una classe presente in `models.py` in questo caso è necessario specificare soltanto i campi da controllare senza definire la sottoclasse `Meta`. Il serializzatore risulta essere molto utile per verificare la validità dei campi passati a un'API prima di processarli.

B.2 Aggiungere nuovo driver di attestazione

In questa sezione verranno presentate le linee guida per creare un driver di attestazione da essere utilizzato dal Trust Monitor al fine di svolgere l'attestazione di host fisici contenenti delle VNF presenti all'interno dell'infrastruttura NFV oppure switch SDN. Per aggiungere un nuovo driver è necessario servirsi dei file sorgenti del Trust Monitor presenti all'interno dei sorgenti della tesi e contenuti nella directory `sorgenti_tesi/ra-trust-monitor`.

I passi necessari per creare tale componente sono:

- Creazione del file Python contenente il driver di attestazione utilizzato da Trust Monitor e chiamato ad esempio `testDriver.py`. Tale file deve essere inserito all'interno del percorso `trustMonitor/trust_monitor_driver`.
- Inserire all'interno di una classe chiamata ad esempio `TestDriver` del file precedentemente creato tre metodi utilizzati dal Trust Monitor e chiamati rispettivamente `registerNode`, `getStatus` e `pollHost`.
- Creazione di un file chiamato ad esempio `parsingTest.py` da includere all'interno della directory `trustMonitor/trust_monitor/verifier`. Tale file viene utilizzato per effettuare il parsing delle misure IMA contenute all'interno dell'Integrity Report.
- Modificare il file `views.py` andando ad aggiungere le chiamate necessarie per utilizzare il nuovo driver di attestazione.

B.2.1 Contenuto driver di attestazione

All'interno file contenente il driver di attestazione devono essere inclusi all'interno della classe `TestDriver` tre metodi:

- il metodo `registerNode` viene utilizzato per registrare un nuovo nodo presente all'interno dell'infrastruttura NFV presso il framework di attestazione e presso il database del Trust Monitor. In questo modo il nodo può essere attestato da quest'ultimo.

- il metodo `gestStatus` viene utilizzato per verificare se il framework di attestazione utilizzato risulta essere correttamente in esecuzione. Per effettuare questo controllo occorre eseguire una semplice chiamata a un'API offerta dal framework di attestazione. In caso di risposta il framework risulta in esecuzione.

- il metodo `pollHost` viene utilizzato per attestare il nodo passato tramite metodo HTTP POST all'API esposta dal Trust Monitor. In questo metodo viene effettuato il processo di RA verso il nodo scelto comunicando con il framework di attestazione e successivamente occorre effettuare il parsing dell'Integrity Report e chiamare la procedura che si occupa di verificare se il nodo risulti essere "trusted" o meno.

B.2.2 Parsing contenuto Report

Al fine di verificare se il nodo attestato risulta essere fidato o meno occorre ottenere tutte le misure dei digest presenti all'interno del Report inviato dal framework di attestazione. Le misure IMA devono essere parsificate in modo tale da essere gestite come `IMARecord` una classe che inserisce tutti i digest relativi al software in esecuzione all'interno di una lista di digest.

Le varie misure devono contenere informazioni obbligatorie in modo da essere trattate come oggetti della classe `IMARecord`. Un esempio di parging è rappresentato in figura B.5, dove al suo interno sono presenti varie indicazioni che permettono di descrivere che struttura deve avere la riga contenente le informazioni IMA.

I campi presenti in figura B.5 permettono di classificare correttamente la misura IMA analizzata e sono:

- `pcr`: rappresenta il valore del registro PCR del TMP che contiene la misura analizzata;

- `template_digest`, `template_name` e `template_desc`: rappresentano informazioni relative al tipo di template utilizzato da IMA;

- `event_digest`: rappresenta il valore associato alla misura analizzata;

- `event_name`: viene utilizzato per indicare il nome del percorso associato alla misura IMA analizzata;

- `id_docker`: campo utilizzato per indicare se la misura IMA viene associata a un container Docker oppure se la misura è associata all'host fisico;

- `template_data`: rappresenta l'oggetto che contiene le informazioni complessive relative al tipo di algoritmo utilizzato per il calcolo del digest, il valore del digest, il nome del percorso relativo alla componente software associata al digest e l'indicazione che ci permette di distinguere a quale componente, se a un container Docker oppure all'host fisico, la misura analizzata fa riferimento;

- `file_line`: contiene l'elenco di tutti i campi definiti in precedenza.

Nel momento in cui viene definita la stringa complessiva contenente tutte le informazioni associate a una misura IMA occorre effettuare la chiamata alla classe `IMARecord` che includerà la misura IMA analizzata all'interno della lista dei digest in modo da essere processata e analizzata in seguito.

```
pcr = "10"
template_digest = "null"
template_name = "ima-ng"
template_desc = "ima-ng"
event_digest = "data" # the value of measure
event_name = measure['Path'] # the path of measure
id_docker = "host" # the owner of the measure, for example host or container
template_data = ("sha1:" + event_digest + " " + event_name +
" " + id_docker)
# oggetto che include i precedenti valori
file_line = (pcr + " " + template_digest + " " +
template_name + " " + template_data)
# linea complessiva passata alla classe IMARecord
IMARecord(file_line)
```

Figura B.5. Esempio di parsing per includere i digest all'interno della lista di digest.

B.2.3 Procedura verifica integrità

Una volta che le misure vengono parsificate correttamente e che quindi siano presenti all'interno della lista delle misure da analizzare, occorre contattare la procedura di verifica di integrità. Se si vuole utilizzare la procedura di verifica di integrità utilizzata in questa tesi bisogna utilizzare il metodo `verifier` messo a disposizione dalla classe `RaVerifier` presente all'interno del modulo `ra_verifier.py` nella directory `trustMonitor/trust_monitor/verifier`. A tale metodo occorre passare diversi valori:

- l'attributo `distro` rappresenta il nome del sistema operativo presente all'interno della macchina fisica che si intende attestare. Tale campo è utile per capire quali misure presenti all'interno del Whitelist Database occorre considerare per verificare l'integrità del sistema;
- l'attributo `analysis` viene utilizzato per indicare il tipo di analisi da effettuare per verificare l'integrità del nodo presente all'interno dell'infrastruttura NFV;
- l'attributo `infoDigest` rappresenta un oggetto della classe `InformationDigest` presente all'interno della directory `trustMonitor/trust_monitor_driver` che contiene tutte le informazioni dettagliate relative all'esito del processo di verifica di integrità, come ad esempio la lista dei digest non trovati dopo il confronto con i database di riferimento;
- l'attributo `checked_containers` viene utilizzato per indicare se bisogna effettuare anche l'analisi del software in esecuzione all'interno di container Docker in esecuzione all'interno dell'host fisico che si intende analizzare;
- l'attributo `report_id` viene utilizzato per indicare l'id dell'Integrity Report ottenuto tramite il processo di RA tra il nodo analizzato e il framework di attestazione utilizzato;
- l'attributo `known_digest` rappresenta la lista dei digest non presenti all'interno del Whitelist Database ma utilizzati al fine di verificare l'integrità del nodo analizzato;
- l'attributo `port` e `ip` fanno riferimento all'indirizzo IP e alla porta del Whitelist Database, tali campi vengono utilizzati per contattare il database al fine di verificare se le misure contenute all'interno del nodo siano presenti all'interno di tale database.

B.2.4 Modificare file views

Una volta definito il driver di attestazione e il parsing utilizzato da tale driver è necessario modificare le API definite dalle classi presenti all'interno del file `views.py` presente nella directory `trustMonitor/trust_monitor` per utilizzare il nuovo driver di attestazione.

La prima operazione da fare è quella di inserire l'import del file contenente il driver di attestazione all'interno del file `views.py` e successivamente istanziare un oggetto della classe `testDriver` creato in precedenza, in questo modo:

```
from trust_monitor_driver.testDriver import TestDriver

testDriver = TestDriver()
```

Una volta fatto questo bisogna modificare le API del Trust Monitor che permettono di registrate e attestare un nodo e di verificare se il framework di attestazione sia in esecuzione correttamente.

Per esempio è possibile modificare l'API definita dalla classe `RegisterNode` in modo da specificare in una if quale driver di attestazione utilizzare per registrare il nodo in esame. All'interno della figura B.6 è possibile osservare come viene gestito l'inserimento di un nuovo driver di attestazione all'interno del Trust Monitor. Discorso analogo viene fatto per la procedura di attestazione.

```
class RegisterNode(APIView):

    def post(self, request, format=None):
        ...
                if newHost.driver == 'OAT':
                        ...
                elif newHost.driver == 'TestDriver':
                        # registrare il nodo con il framework di attestazione e
                            con il Trust Monitor
```

Figura B.6. Esempio contenuto classe RegisterNode.

B.3 Creazione nuovo connettore

Ogni componente all'interno dell'architettura del Trust Monitor viene istanziato all'interno di un container Docker, per cui è importante che quando viene creato un nuovo connettore venga istanziato tramite l'utilizzo del file Docker Compose utilizzato per eseguire l'intera architettura definita in questa tesi.

Quando un connettore viene creato è necessario aggiungere lo stesso all'interno dell'architettura del Trust Monitor, per gestire questa situazione in maniera molto semplice è possibile aggiungere l'istanziazione di tale connettore all'interno del file Docker Compose che si occupa di istanziare tutta l'architettura del Trust Monitor. I passaggi necessari per creare un nuovo connettore e istanziarlo tramite lo stesso file Docker Compose del Trust Monitor sono:

- creare una directory nel quale aggiungere il connettore;
- creare il connettore secondo la logica appropriata;
- creare un Dockerfile per permettere l'istanziazione del nuovo connettore tramite un container Docker;
- aggiungere il nuovo connettore al file `docker-compose.yml` in modo da permettere l'istanziazione tramite Docker Compose.

Al fine di creare un nuovo connettore è necessario scaricare i sorgenti del Trust Monitor forniti con i sorgenti della tesi e presenti all'interno della directory `sorgenti_tesi/ra-trust-monitor`. Ogni connettore presente all'interno dell'architettura del Trust Monitor è presente all'interno della directory `connectors`. Se si vuole creare un nuovo connettore è necessario creare una nuova directory chiamata ad esempio `connettore_test` all'interno della directory `connectors`.

Nella nuova directory è possibile creare il proprio connettore, chiamato ad esempio `connettore1`, al termine di questa procedura occorre creare un file Dockerfile che permette di istanziare il nuovo connettore all'interno della directory `test_connettore` precedentemente creata.

B.3.1 Dockerfile

L'utilizzo del Dockerfile permette di istanziare correttamente il nuovo connettore all'interno dell'architettura del Trust Monitor, questa scelta porta maggiore scalabilità e portabilità dell'intera architettura proposta.

```
FROM python:2-alpine
LABEL maintainer="nome e indirizzo email del creatore del connettore"
RUN mkdir /logs
WORKDIR /usr/src/app
COPY connettore1.py .
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
EXPOSE 5000
ENTRYPOINT ["python"]
CMD ["connettore1.py"]
```

Figura B.7. Esempio creazione Dockerfile per il connettore `connettore1`.

Nella figura B.7 è possibile osservare un esempio di Dockerfile per il connettore `connettore1`. È possibile osservare che l'immagine base al quale tale connettore fa riferimento è alpine, ma questo non è un vincolo infatti varia in base al tipo di connettore che si realizza. I comandi presenti all'interno della figura sono:

- il primo comando `RUN` permette di creare una directory chiamata `logs` all'interno del container Docker nel quale verrà eseguito il connettore;
- `WORKDIR` viene utilizzato per permette di impostare la directory di lavoro all'interno del container;
- il primo comando `COPY` copia il modulo Python nel quale è stato definito il connettore all'interno del container Docker;
- il secondo comando `COPY` copia il file `requirements.txt` utilizzato per installare i moduli Python utilizzati dal connettore;
- il secondo comando `RUN` viene utilizzato per installare tramite `pip` i moduli Python utilizzati dal connettore;
- il comando `EXPOSE` viene utilizzato per esporre una porta del container Docker;
- i comandi `ENTRYPOINT` e `CMD` vengono utilizzati per eseguire il connettore.

Al termine della creazione del Dockerfile è necessario aggiungere all'interno del file Docker Compose l'istanziazione di tale connettore.

B.3.2 Docker Compose

L'aggiunta del connettore all'interno del file Docker Compose è motivata dal fatto che permette di istanziare con molta facilità tutti i componenti presenti all'interno dell'architettura del Trust Monitor tramite un singolo comando e in aggiunta è anche molto semplice monitorare ogni container Docker istanziato.

Al fine di aggiungere un nuovo connettore all'interno del file Docker Compose è necessario modificare il file `docker-compose.yml` fornito con i sorgenti del Trust Monitor. Nella figura B.8 è possibile osservare un esempio di aggiunta del connettore `test_connettore` definito in precedenza all'interno del file Docker Compose. Il connettore viene definito in `services` e prende il nome di `tm_test_connettore`, l'attributo `image` fa riferimento al nome associato all'immagine con cui viene creato il container Docker che contiene quel connettore, l'attributo `build` viene utilizzato per puntare a quale Dockerfile facciamo riferimento.

Un aspetto interessante è rappresentato dal valore `links` presente in `tm_django_app` mediante tale campo viene specificato un collegamento tra il nuovo connettore e il Trust Monitor, infatti mediante questo link il Trust Monitor può comunicare con il connettore `test_connettore` utilizzando il nome dello `tm_test_connettore` con cui il connettore è stato inserito nel link.

```
version: '3'

services:
  ...
  tm_test_connettore:
    image: ra/connectors/test_conettore
    build: ./connectors/test_connettore

  tm_django_app:
    ...
    links:
      ...
      - tm_test_connettore:tm_test_connettore
```

Figura B.8. Esempio aggiunta connettore `test_connettore` all'interno dell'architettura del Trust Monitor.

Al termine di questa procedura basterà utilizzare il comando `sudo docker-compose up --build` e verranno istanziati tutti i container Docker presenti nel file Docker Compose compreso il nuovo connettore.

Bibliografia

[1] Integrity Measurement Architecture (IMA) project, https://sourceforge.net/p/linux-ima/wiki/Home/#integrity-measurement-architecture-ima

[2] A.Lioy, N.Barresi, T.Su, articolo "Trusted Computing Technology and Proposals for Resolving Cloud Computing Security Problems", "Cloud Computing Security: Foundations and Challanges" a cura di J.Vacca, CRC Press, 2016, ISBN: 9781482260946

[3] Trusted Computing Group (TGC), http://www.trustedcomputinggroup.org/

[4] A.Lioy, G.Ramunno, "Trusted Computing", "Handbook of Information and Communication Security" a cura di P.Stavroulakis, M.Stamp, Springer-Verlag Berlin Heidelberg, 2010, ISBN:978-3-642-04116-7, DOI 10.1007/978-1-84882-684-7

[5] Trusted Computing Group, "TPM Main Specification Level 2 version 1.2 Revision 116, Part 1 - Design Principles", 1 March 2011 https://trustedcomputinggroup.org/wp-content/uploads/TPM-Main-Part-1-Design-Principles_v1.2_rev116_01032011.pdf

[6] Trusted Computing Group, "TPM Main Specification Level 2 version 1.2 Revision 116, Part 3 - Commands", 1 March 2011 http://trustedcomputinggroup.org/wp-content/uploads/TPM-Main-Part-3-Commands_v1.2_rev116_01032011.pdf

[7] E.Brickell, J.Camenisch, L.Chen, "Direct anonymous attestation. In: Proceedings of the 11th ACM Conference on Computer and Communications Security", pp. 132-145, ACM Press, New York, 2004, ISBN:1-58113-961-6, DOI 10.1145/1030083.1030103

[8] TCG Infrastructure Working Group, "Architecture Part II - Integrity Management, specification version 1.0", 17 November 2006, https://trustedcomputinggroup.org/wp-content/uploads/IWG_ArchitecturePartII_v1.0.pdf

[9] TCG Infrastructure Working Group, "Reference Architecture for Interoperability (Part I) specification version 1.0", 16 June 2005, https://trustedcomputinggroup.org/wp-content/uploads/IWG_Architecture_v1_0_r1.pdf

[10] Docker, project https://www.docker.com/

[11] J.Turnbull, "The Docker Book", March 2017, Version: v17.03.0 ISBN: 978-0-9888202-0-3

[12] Repositories on Docker Hub, https://docs.docker.com/docker-hub/repos/

[13] Docker Compose, project https://docs.docker.com/compose/

[14] Storege Driver in Docker, https://docs.docker.com/storage/storagedriver/

[15] ETSI ISG NFV, "Network Functions Virtualisation (NFV) Architectural Framework", http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf

[16] ETSI, "Network Functions Virtualisation, An Introduction, Benefits, Enablers, Challenges & Call for Action", October 2012, http://portal.etsi.org/NFV/NFV_White_Paper.pdf

[17] ETSI GS NFV-MAN 001, "Network Functions Virtualisation (NFV); Management and Orchestration", v1.1.1, December 2014, http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf

[18] ETSI, "Network Functions Virtualisation (NFV); Use Cases", May 2017, http://www.etsi.org/deliver/etsi_gr/NFV/001_099/001/01.02.01_60/gr_nfv001v010201p.pdf

[19] Open Source Mano Release 3 (OSM), project https://osm.etsi.org/wikipub/index.php/OSM_Release_THREE

[20] Gruppo di ricerca TORSEC, http://security.polito.it/

[21] Open Attestation (OAT), project https://github.com/OpenAttestation/OpenAttestation

[22] SECURED, project https://www.secured-fp7.eu/

[23] IEEE, "Containers and Cloud: From LXC to Docker to Kubernets", pp. 81-84, September 2014, IEEE Cloud Computing, DOI 10.1109/MCC.2014.51
[24] ETSI GR NFV-SEC 007, "Network Functions Virtualisation (NFV), Trust; Report on Attestation Technologies and Practices for Secure Deployments", pp. 12-15, October 2017 v1.1.1, http://www.etsi.org/deliver/etsi_gr/NFV-SEC/001_099/007/01.01.01_60/gr_nfv-sec007v010101p.pdf
[25] ETSI GR NFV-SEC 003, "Network Functions Virtualisation (NFV), NFV Security; Security and Trust Guidance", pp. 43, August 2016, v1.2.1, http://www.etsi.org/deliver/etsi_gr/NFV-SEC/001_099/003/01.02.01_60/gr_nfv-sec003v010201p.pdf
[26] Django REST framework, project http://www.django-rest-framework.org/
[27] Django project, https://www.djangoproject.com/
[28] Flask project, http://flask.pocoo.org/
[29] Apache Cassandra db, project http://cassandra.apache.org/
[30] leaseweb, http://mirror.de.leaseweb.net/
[31] pycassa, documentazione, https://pycassa.github.io/pycassa/
[32] Redis db, project, https://redis.io/
[33] RabbitMQ Open Source, project https://www.rabbitmq.com/
[34] VIM Emulator, https://osm.etsi.org/wikipub/index.php/VIM_emulator
[35] Libreria docker-py project, https://github.com/docker/docker-py
[36] A.Golubin, "Garbage collection in Python: things you need to know", 7 October 2017, https://rushter.com/blog/python-garbage-collector/
[37] Open Cloud Integrity Technology (CIT), project https://01.org/opencit
[38] Containernet project, https://containernet.github.io/