UNIVERSITÀ
DEGLI STUDI
DI PALERMO

NDS
NETWORKING AND DISTRIBUTED SYSTEMS
RESEARCH GROUP

Progetto e sviluppo di un sistema di riconoscimento di malware per dispositivi mobili tramite analisi ibrida

Tesi di Laurea Magistrale in Ingegneria Informatica

Damiano Cupani

Relatore: Prof. Giuseppe Lo Re
Correlatore: Ing. Marco Morana

UNIVERSITÀ DEGLI STUDI DI PALERMO

DIPARTIMENTO DI INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

PROGETTO E SVILUPPO DI UN SISTEMA DI RICONOSCIMENTO DI MALWARE PER DISPOSITIVI MOBILI TRAMITE ANALISI IBRIDA

TESI DI LAUREA DI
Dott. CUPANI DAMIANO

RELATORE
Ch.mo Prof. GIUSEPPE LO RE

CONTRORELATORE
Ch.mo Prof. SALVATORE GAGLIO

CORRELATORE
Ing. MARCO MORANA

ANNO ACCADEMICO 2019 - 2020

Progetto e sviluppo di un sistema di riconoscimento di malware per dispositivi mobili tramite analisi ibrida

Tesi di laurea di

Dott. Damiano Cupani

Relatore

Ch.mo Prof. Giuseppe Lo Re

Controrelatore

Ch.mo Prof. Salvatore Gaglio

Correlatore

Ing. Marco Morana

Sommario

Al giorno d'oggi, si assiste ad una sempre maggiore diffusione di dispositivi mobili quali gli smartphone, e ad un conseguente incremento delle applicazioni sviluppate per queste piattaforme. In particolare, il sistema operativo Android, grazie alla sua architettura aperta e all'elevato numero di utenti rappresenta un importante mercato per la distribuzione di software. L'obiettivo di questo lavoro di tesi è la progettazione e lo sviluppo di un sistema per il riconoscimento di malware in ambiente Android. Il sistema progettato si basa su un approccio di analisi ibrida che unisce i vantaggi dell'analisi statica, svolta esclusivamente a partire dal codice delle applicazioni, a quelli dell'analisi dinamica, che sfrutta l'osservazione del comportamento delle applicazioni in esecuzione. Il sistema ibrido è stato progettato in due varianti, la prima che fa uso di un solo classificatore e la seconda basata su un sistema di stacking ensemble learning con due livelli. Il sistema proposto sfrutta algoritmi di intelligenza artificiale per determinare se l'applicazione analizzata può essere considerata malevola o legittima. Per valutare le prestazioni del sistema proposto sono stati confrontati i risultati della sola analisi statica e della sola analisi dinamica con quelli delle due varianti del sistema ibrido. I risultati ottenuti hanno dimostrato le potenzialità dell'approccio proposto per il riconoscimento dei malware.

Indice:

1 Introduzione

La diffusione degli smartphone è aumentata drasticamente negli ultimi anni andando di pari passo con l'incremento delle funzionalità fornite da tali dispositivi (He, 2015), tali funzionalità oltre a poter essere utilizzate per migliorare l'esperienza utente vengono sovente sfruttate da attaccanti malevoli per attività illecite. Di tutte le funzionalità la più importante è probabilmente la possibilità di estendere il sistema tramite applicazioni di terze parti chiamate *apps*. Tali apps vengono distribuite tramite dei repository chiamati *store* che rendono estremamente semplice sia la pubblicazione di applicazioni da parte degli sviluppatori che il loro download da parte degli utenti finali. Ovviamente la semplicità nel pubblicare apps che poi possono essere scaricate da un numero considerevole di utenti ha anche suscitato l'interesse di sviluppatori di applicazioni malevole, considerato che, dato l'elevato numero di applicazioni pubblicate, una loro analisi manuale non risulta essere fattibile. Da qui nasce la necessità di sistemi di completamente automatici di rilevazione dei malware mobili che riescano ad ottenere buoni risultati senza richiedere lunghe analisi. Va notato però, che la diffusione di malware negli store ufficiali varia in base al sistema utilizzato. Il mercato dei sistemi operativi mobili è praticamente un duopolio (Steinberg, 2019) di Apple con iOS e Google con Android. Mentre iOS, pur non essendo esente da problemi di sicurezza (Yixiang Zhu, 2017), è la piattaforma preferita dagli sviluppatori legittimi attirati dagli alti margini di guadagno (Perez, 2018). Android invece è la piattaforma mobile preferita dai cracker[1], questo per una serie di motivi per esempio:

- è il sistema mobile più diffuso;
- la sua natura opensource da molta libertà agli sviluppatori che possono modificare il funzionamento di alcune parti del sistema, ad esempio, il *launcher* che gestisce l'interfaccia utente, le app che gestiscono telefonate e SMS e anche la tastiera di sistema[2];
- l'essere opensource garantisce, inoltre, molta libertà agli utenti che possono scaricare applicazioni da store di terze parti semplicemente e senza invalidare la garanzia del dispositivo;

[1] Con cracker si intende un esperto di sicurezza informatica che sfrutta le sue conoscenze per fini malevoli, viene definito semplicemente hacker sebbene questo ultimo termine indica solamente un esperto di cyber security.
[2] Nonostante la potenziale pericolosità nell'utilizzo di una tastiera di terze parti, questa è l'unica, tra le personalizzazioni elencate, che è disponibile pure su iOS.

- la procedura per la pubblicazione e l'aggiornamento delle app non prevede controlli approfonditi;
- il problema della frammentazione, cioè l'elevato numero di varianti di Android create da produttori e operatori che spesso non hanno sufficienti risorse per la manutenzione di un sistema operativo, fa sì che la maggior parte dei dispositivi Android utilizzati non siano aggiornati con le ultime patch di sicurezza.

Per questo si è scelto di concentrarsi su Android per questo lavoro di tesi. Infatti, la crescente diffusione dei dispositivi mobili con sistema operativo Android (Wang, 2016) ha portato ad un aumento del numero e delle tipologie di malware specifici per questa piattaforma.
Secondo i dati dell'International Data Corporation (IDC Corporate USA, 2020) nel 2019 il market share di Android è stato del 86.6% ed è previsto un leggero incremento negli anni successivi.

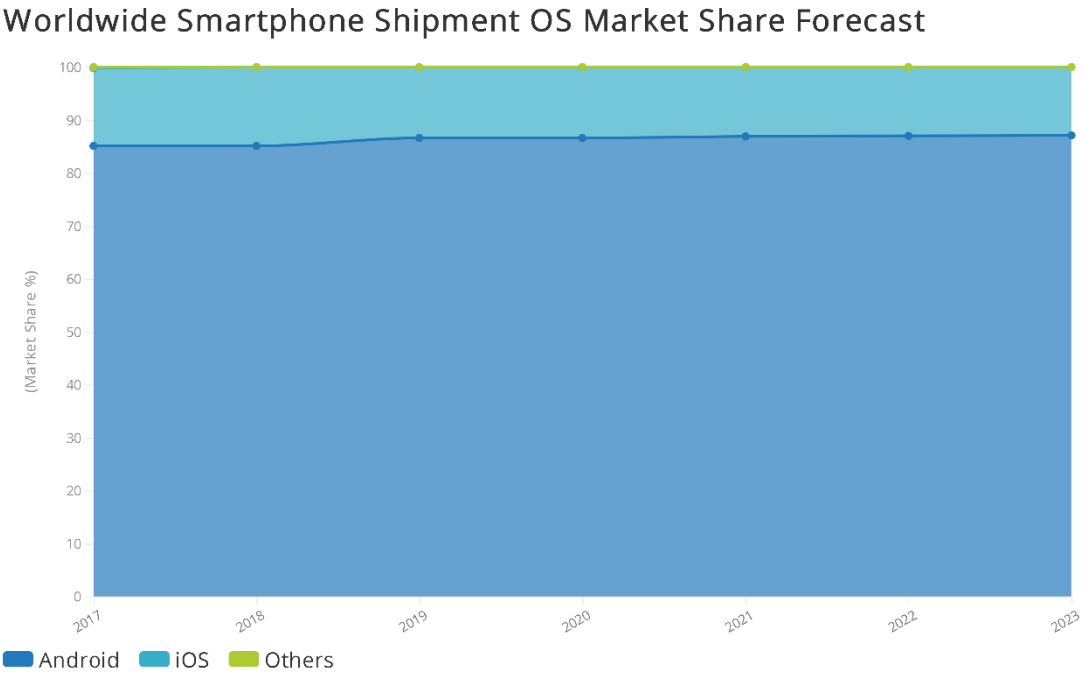

Figura 1 market share Android

Google classifica i malware per il proprio sistema operativo mobile in undici categorie, come mostrato nella seguente tabella (Google, 2019):

Tabella 1 Categorie di malware per Android secondo Google

Click fraud	Trojan	SMS fraud
Spyware	Toll fraud	Backdoor
Hostile downloader	Privilege escalation	Phishing
Others		

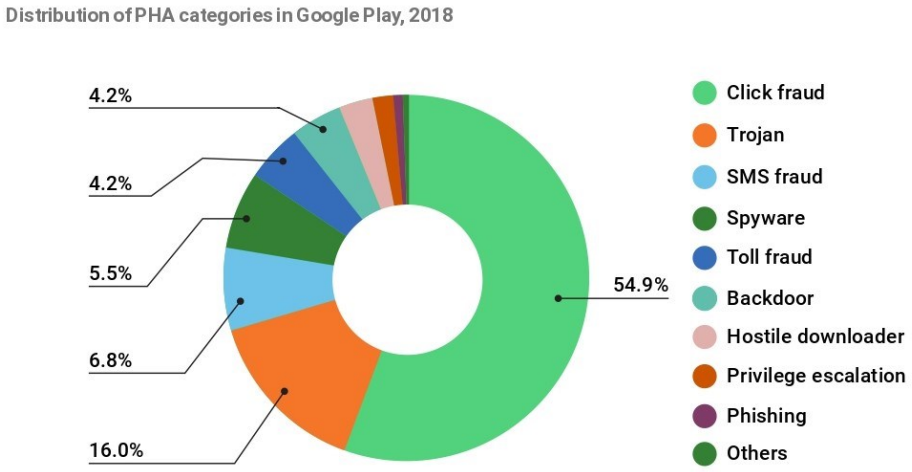

Figura 2 distribuzione delle categorie di malware nel PlayStore

Click fraud: malware che simula click su banner pubblicitari inesistenti al fine ottenere illeciti guadagni truffando le società che gestiscono tali banner. Nonostante tale frode colpisca principalmente le società che gestiscono gli spazi pubblicitari online, si possono verificare anche disagi per l'utente che esegue tali apps. Per esempio, spesso si verificano diminuzioni delle prestazioni dello smartphone e una diminuzione dell'autonomia. Inoltre, l'indirizzo IP del dispositivo può venire contrassegnato come sospetto provocando un rallentamento o l'impossibilità di accedere a determinati servizi.

Trojan: I trojan sono dei software che nascondo i loro comportamenti malevoli promettendo funzionalità utili che possono essere reali o fittizie. Nel primo caso l'utente, utilizzando normalmente il software, difficilmente penserà di essere vittima di un attacco.

SMS fraud: consistono in applicazioni che inviano messaggi di testo a numeri a tariffazione maggiorata controllati dagli attaccanti in modo da rubare il credito della linea di telefonia mobile.

Spyware: si tratta di una tipologia di malware che ottiene illecitamente i dati personali presenti nei dispositivi degli utenti. Tali dati possono in seguito essere monetizzati dagli attaccanti tramite la vendita o l'estorsione.

Toll fraud: malware che implementano una truffa analoga a quella impiegata in SMS fraud utilizzando le chiamate a sovraprezzo invece degli SMS.

Backdoor: le backdoor sono delle vulnerabilità nel sistema che vengono sfruttate dagli attaccanti per prendere il controllo completo o parziale del dispositivo di un utente inconsapevole.

Hostile downloader: si tratta di applicazioni che di per sé non sono malevole ma che scaricano sul dispositivo altre applicazioni che invece lo sono. Non rientrano in questa categoria i browser e i software di file sharing a condizione che scarichino altre app solo su richiesta esplicita dell'utente.

Privilege escalation: questa categoria comprende tutte le applicazioni che consentono di ottenere il livello massimo di permessi (root) nei dispositivi Android. Tale pratica minando alla base il modello della sicurezza Android costituisce sempre un pericolo, ma in alcuni casi questo comportamento è richiesto consapevolmente dall'utente che desidera personalizzare il sistema oltre ciò che Android consente per impostazione predefinita.

Other: in questa categoria rientrano tutti i malware che non fanno parte di nessuna delle categorie elencate precedentemente.

Questi malware sono diffusi sia tramite i canali ufficiali (ad esempio Google Play Store ed Amazon App Store) ma soprattutto attraverso canali alternativi come i marketplace di terze parti (ad esempio Aptoide e Uptodown), l'installazione manuale (consapevole o meno) dei file .apk o la preinstallazione durante la catena di montaggio e distribuzione (Julien Gamba, 2019).

Paradossalmente in alcuni casi gli utenti Android preferiscono continuare ad ud usare una app nonostante siano stati avvertiti dei pericoli legati a tali app, in particolare ciò avviene per le app che consentono di ottenere i privilegi di root per personalizzare il sistema o per quelle che consentono l'accesso gratuito a contenuti protetti da diritti d'autore (conosciute come app *crackate*). Per questo motivo nel 2018 Google ha introdotto la politica di non avvisare ulteriormente (*don't-warn-again*) gli utenti che scelgono deliberatamente di eseguire comunque una app nonostante sia stata segnalata come pericolosa dal sistema anti-malware di Google (Google, 2019, p. 13)

Sebbene gli store di terze parti sono considerati quasi sempre come un indebolimento alla sicurezza di Android, va segnalato un caso interessante, dal punto di vista della sicurezza informatica; F-Droid è uno store per sole app opensource che consente di verificare che le applicazioni scaricate siano state compilate effettivamente dal codice sorgente pubblicato (F-Droid Limited, 2020). Ciò invece non è possibile sullo store ufficiale di Google che, sebbene contenga molte app di cui è possibile visualizzare il sorgente, non da nessuna garanzia rispetto alla corrispondenza tra il codice sorgente e il file apk che viene effettivamente installato sullo smartphone.
Questo, in teoria, potrebbe consentire, tramite una analisi semantica del codice sorgente, di avere l'assoluta certezza di aver scaricato del software benigno; tuttavia questo approccio ha dei problemi pratici di fattibilità: innanzitutto, la gran parte delle applicazioni per Android sono proprietarie e inoltre il processo di verifica semantica della legittimità del codice sorgente è un processo complesso che deve essere effettuato manualmente da un esperto.

La situazione attuale, è che si propongono e si implementano tecniche sempre migliori per il riconoscimento dei malware ma a loro volta gli attaccanti affinano le loro tecniche per cercare di evadere l'analisi.

L'obiettivo di questa tesi è quindi progettare e sviluppare un sistema automatico che sia in grado di riconoscere se una applicazione è sicura oppure no, a tale scopo sono state utilizzate tecniche di analisi ibrida che combinano i vantaggi dell'analisi statica con quelli dell'analisi dinamica.

1.1 Struttura della tesi

L'elaborato di tesi è strutturato come segue: nel capitolo 2 vi è una breve descrizione delle tipologie di analisi dei malware, mostrando le varie differenze che le caratterizzano; nel capitolo 3 vengono mostrate e descritte le tecniche di apprendimento automatico utilizzate per la fase di classificazione; nel capitolo 4 vengono illustrate le scelte progettuali a livello di architettura e configurazione; nel capitolo 5 viene mostrato l'ambiente di sviluppo, illustrando le varie componenti utilizzate nel sistema ibrido e nei sottosistemi statico e dinamico; nel capitolo 6 vengono mostrati i risultati sperimentali ottenuti e, infine, nel capitolo 7 sono presenti alcune conclusioni sul lavoro svolto.

2 Stato dell'Arte

Negli ultimi anni si è assistito ad un incremento della diffusione e dell'utilizzo degli smartphone e di conseguenza si è assistito anche ad un incremento negli attacchi che sfruttano le loro peculiari caratteristiche:

- solitamente sono accesi e connessi in maniera continuativa;
- custodiscono molti dati sensibili come ad esempio contatti, messaggi, e-mail e foto;
- hanno a disposizione un insieme di sensori che possono essere sfruttati dagli attaccanti per carpire dati sensibili[3];
- limitata autonomia energetica, che spesso scoraggia gli utenti dall'utilizzo di software anti-malware avanzati.

Per proteggersi dagli attacchi informatici, la prima linea di difesa è costituita dall'analisi delle minacce (Sartea, 2016), e la letteratura scientifica è ricca di soluzioni proposte per raggiungere tale scopo, tramite sistemi di riconoscimento di malware. Le diverse soluzioni proposte possono essere classificate in base a varie caratteristiche, va tuttavia notato che non esiste una tassonomia rigida, di conseguenza, per ogni caratteristica analizzata è possibile trovare soluzioni che rappresentano un ibrido tra diverse classi descritte:

- **Analisi locale o remota:** l'analisi delle app volta a classificarle in software malevolo o software benigno, può essere effettuata nello stesso dispositivo che si sta cercando di difendere (*locale*) o su un'altra macchina con maggiori risorse computazionali (*remota*). Con primo metodo si privilegia la privacy dell'utente e il risparmio di banda dei dispositivi, poiché non è necessario che il dispositivo mobile invii le applicazioni da analizzare al server remoto. Il vantaggio del secondo metodo invece, è che richiede meno risorse sul dispositivo mobile e di conseguenza comporta un risparmio anche in termini di autonomia. Inoltre, le maggiori risorse disponibili in cloud consentono di effettuare un'analisi più approfondita. Questa distinzione, ovviamente, ha senso solo per sistemi che prevedono di analizzare le app già installate sui dispositivi e non per i sistemi che mirano ad effettuare l'analisi delle app direttamente sugli store (quest'ultimo approccio ricade sempre nella analisi remota).

[3] Ad esempio, sfruttando i sensori presenti negli smartphone è possibile carpire le password digitate in un computer che si trova in prossimità dello smartphone (Mehrnezhad, 2018)

- **Analisi statica o dinamica:** si parla di analisi dinamica quando i software da analizzare vengono effettivamente eseguiti e monitorati su un dispositivo che può essere reale o virtualizzato. In caso contrario, cioè quando si analizza il contenuto degli applicativi senza eseguirli, l'analisi viene definita statica. La suddetta distinzione è il criterio più utilizzato in letteratura quindi di seguito verranno citati alcuni lavori suddividendoli in statici, dinamici, e ibridi statici-dinamici[4].

- **Esecuzione simulata o monitoraggio dell'utilizzo:** nel caso dinamico è possibile considerare un'ulteriore distinzione; infatti l'analisi dinamica prevede solitamente di eseguire i campioni da analizzare all'interno di un ambiente protetto chiamato *sandbox,* ma esiste anche un approccio differente che prevede il monitoraggio dei software mentre vengono eseguiti dall'utente finale.

A prescindere dal tipo di analisi, il modo migliore per seguire l'evoluzione delle minacce è utilizzare delle tecniche di intelligenza artificiale (Pfeffer, 2017). La principale caratteristica di tali tecniche è che possono essere addestrate in modo completamente automatico con campioni malware recenti.

2.1 Analisi Statica

L'analisi statica esamina il contenuto di un software, che nel caso di Android consiste in un file compresso con estensione .apk, senza che l'applicazione stessa venga eseguita al fine di cercare caratteristiche tipiche dei software malevoli.
Esistono diversi tipi di analisi statica che per fini esplicativi possono essere suddivisi in base e avanzati.

2.1.1 Analisi statica di base

Il metodo più semplice consiste nel calcolare l'hash del file da analizzare e confrontarlo con i valori hash dei malware noti, questo metodo è molto efficiente ma può essere aggirato

[4] in tutto il resto di questo lavoro di tesi, salvo diversamente specificato, per analisi ibrida si intenterà l'ibrido tra statica e dinamica come da prassi consolidata in letteratura e nei lavori di ricerca.

facilmente; infatti, basta che cambi un solo bit nel file e per l'effetto valanga il valore di output della funzione (ovvero la firma) cambierà. Per effetto valanga di una funzione hash si intende la proprietà che fa cambiare completamente il suo output al cambiare di un solo bit in input (William, 2016) che in questo caso è il file da analizzare. Per esempio, il solo cambiamento del nome o del valore di una variabile dentro il codice di malware è sufficiente a far variare l'output della funzione hash, pertanto, un approccio basato sull'hashing può rilevare solo copie identiche di un malware noto.
Per questo motivo si è pensato di utilizzare come *signature* per il riconoscimento dei malware non il *digest* della funzione hash bensì dei pattern che corrispondono a porzioni di codice tipicamente presenti all'interno del file malevoli, questo è l'approccio utilizzato per esempio dal software Opensource Yara (YARA, 2020) nondimeno anche quest'approccio è facilmente eludibile tramite semplici tecniche di offuscamento del codice malevolo (Fabrizio Biondi, 2018).

2.1.2 Analisi statica avanzata

Gli approcci più avanzati solitamente sono suddivisi in alcune fasi.
Per iniziare, considerando che tipicamente non si ha accesso al codice sorgente, il codice oggetto viene disassemblato, vale a dire, convertito in un linguaggio di basso livello più adatto all' analisi[5].

Successivamente tale codice viene elaborato ottenendo una rappresentazione più astratta per esempio gli n-grammi della sequenza di chiamate di funzione o il grafo del flusso di controllo. Per n-gramma di una sequenza si intende una qualsiasi sottosequenza di n-elementi della sequenza data, in linguistica solitamente gli elementi sono i caratteri, in questo caso invece gli elementi sono le chiamate a funzione estratte dal codice.
Il grafo del flusso di controllo (CFG) è invece una rappresentazione grafica sotto forma di grafo di tutti i possibili percorsi di esecuzione del codice. In tale grafo i nodi sono i blocchi base cioè blocchi di codici senza salti condizionati e gli archi rappresentano proprio i salti condizionati.

Infine, a partire da tali rappresentazioni astratte viene creato un vettore di caratteristiche utilizzato come input di un algoritmo di classificazione il quale restituirà un output

[5] nel caso di Android solitamente si disassemblano i file *.apk* per ottenere dei file. smali (Marastoni, 2017, p. 2)

benigno/maligno per ogni campione del dataset. Come in molti altri problemi di classificazione, la scelta del vettore delle caratteristiche è cruciale al fine di ottenere dei buoni risultati di classificazione.

Gli autori di (B. Kang, 2016) dopo aver disassemblato le classi dei file apk, utilizzando il software baksmali (smali/baksmali, n.d.), costruiscono i vettori delle frequenze degli n-grammi cercandoli all'interno della sequenza dei soli opcode(senza considerare gli operandi). Tali vettori sono stati utilizzati per la classificazione con Naïve Bayes, SVM, alberi decisionali parziali e foresta random tramite il framework opensource WEKA.
È stato effettuato inoltre un altro esperimento in cui nel vettore le frequenze degli n-grammi veniva indicata la sola informazione binaria sulla sola presenza o assenza del n-gramma invece che il numero di occorrenze. Dal confronto dei risultati della classificazione si evince che questo secondo vettore oltre ad occupare meno spazio ha prodotto sorprendentemente risultati migliori. Il confronto tra gli algoritmi di classificazione ha mostrato invece che i risultati migliori si sono ottenuti con SVM sebbene non molto distanti da quelli ottenuti con la foresta random e gli alberi decisionali parziali.

Gli autori di (Marastoni, 2017) e di (W. Park, 2014) calcolano il grafo del flusso di controllo del campione da analizzare e lo confrontano con quello dei campioni malevoli del dataset cercando delle similarità. Il confronto tra i grafi è stato eseguito tramite la differenza dei loro centroidi.
Quest'approccio oltre a dare un riscontro sulla natura malevola del software fornisce anche, in caso di esito positivo, una classificazione del campione in una delle famiglie di malware conosciute.

In (Leonid Batyuk, 2011) è proposto un sistema che utilizza molteplici analizzatori statici per fornire un riscontro all'utente sulla sicurezza delle app utilizzate consentendo anche di creare una nuova versione del pacchetto apk funzionante (nella maggioranza dei casi) ma con le funzionalità pericolose rimosse. Questo consentirebbe di rafforzare la sicurezza del sistema operativo Android senza la necessità di modificare il sistema stesso.

Per limitare l'uso di risorse computazionali richieste dagli algoritmi di classificazione esistono degli approcci più leggeri che non analizzano interamente il pacchetto apk ma si limitano ad

alcune porzioni di piccole dimensioni ritenute più significative, ad esempio, in (D. Ö. Şahın, 2018) vengono utilizzate solamente le informazioni sui permessi estratte dal file *AndroidManifest.xml* per classificazione. Tra gli approcci utilizzati vi sono i classificatori di tipo *Naive Bayes* e l'algoritmo K-nearest Neighbour (KNN), che raggiunge un'accuratezza superiore al 90%. Gli autori di (De Paola, Favaloro, Gaglio, Lo Re, & Morana, 2018) invece propongono un sistema di analisi statica per il sistema operativo desktop Windows che riceve in input una porzione limitata dei Portable Executable (PE) ed effettua la classificazione tramite una rete neurale profonda.

Essendo l'analisi statica dei malware molto diffusa (la maggior parte dei software anti-malware utilizza proprio tecniche statiche) ed essendo storicamente il primo approccio avanzato usato nella rilevazione di software malevoli, sono spesso adoperate dagli attaccanti tecniche di evasione dell'analisi statica. Per evasione si intende ogni espediente utilizzato per fare in modo che un software malevolo non venga riconosciuto come tale.
Le più diffuse tecniche di evasione sono basate sull'offuscamento del codice ma, il fatto che un software sia oscurato non è indice certo della sua natura malevola, poiché molti software proprietari legittimi impiegano l'offuscamento del codice al fine di tutelare la propria proprietà intellettuale (Arini Balakrishnan, 2012).
Sono stati proposti approcci che tengono in considerazione tale aspetto. Per esempio, il sistema statico basato sul calcolo del flusso di controllo proposto da (Marastoni, 2017) è in grado di riconoscere una particolare tipologia di malware mobili anche nel caso in cui gli attaccanti abbiano offuscato il codice (ad esempio, tramite la sostituzione dei nomi di metodi e variabili) tuttavia fallisce nel caso in cui gli attaccanti abbiano usato tecniche più avanzate di offuscamento del codice basate sull'alterazione del grafo del flusso di controllo.

2.2 Analisi Dinamica

Al contrario dell'analisi statica, quella dinamica si concentra sul comportamento a tempo di esecuzione dell'applicazione.
Come visto in precedenza l'analisi dinamica può consistere in una esecuzione simulata all'interno di una sandbox o nel monitoraggio dell'utilizzo. Considerato che il primo approccio è molto più diffuso (anche a causa delle implicazioni sulla privacy relative al monitoraggio dell'utilizzo da parte dell'utente), d'ora in avanti se non diversamente specificato, per analisi

dinamica si intenterà l'esecuzione simulata.
L'applicazione da analizzare viene quindi eseguita in una sandbox al fine di monitorarne il comportamento.

Una sandbox indica un ambiente virtualizzato isolato dalla macchina in cui fisicamente è eseguita la sandbox stessa. Le sandbox possono essere utilizzate in vari ambiti dell'informatica come lo sviluppo software e la sicurezza informatica. Anche per ciò che concerne la sicurezza informatica le sandbox possono essere impiegate per diversi scopi; ad esempio, si può utilizzare una sandbox al solo scopo di proteggere una sistema da danni dovuti alla esecuzione di software non fidato oppure, come nel caso in esame, oltre alla protezione si desidera anche monitorare l'esecuzione.

Per questo motivo, implementazioni delle sandbox da utilizzare per l'analisi dinamica sono arricchite di funzionalità aggiuntive che consentono di registrare l'attività del sistema e delle applicazioni(*sensori*) ed in alcuni casi di interagire con essi (*attuatori*). Questo tipo di sandbox sono definite *Dynamic Malware Analysis System* (D-MAS).

Il monitoraggio delle attività e la raccolta dati possono includere:

- chiamate di API di sistema;
- traffico di rete;
- accesso o creazione di file;
- caricamento dinamico di codice o librerie;
- accesso alle periferiche hardware;
- modifica delle impostazioni di sistema;
- dump[6] della memoria RAM.

[6] Per dump della memoria RAM si intente un'istantanea che fotografa lo stato della memoria in un determinato momento e che di solito viene utilizzata a scopo di debugging o di analisi del comportamento di un software.

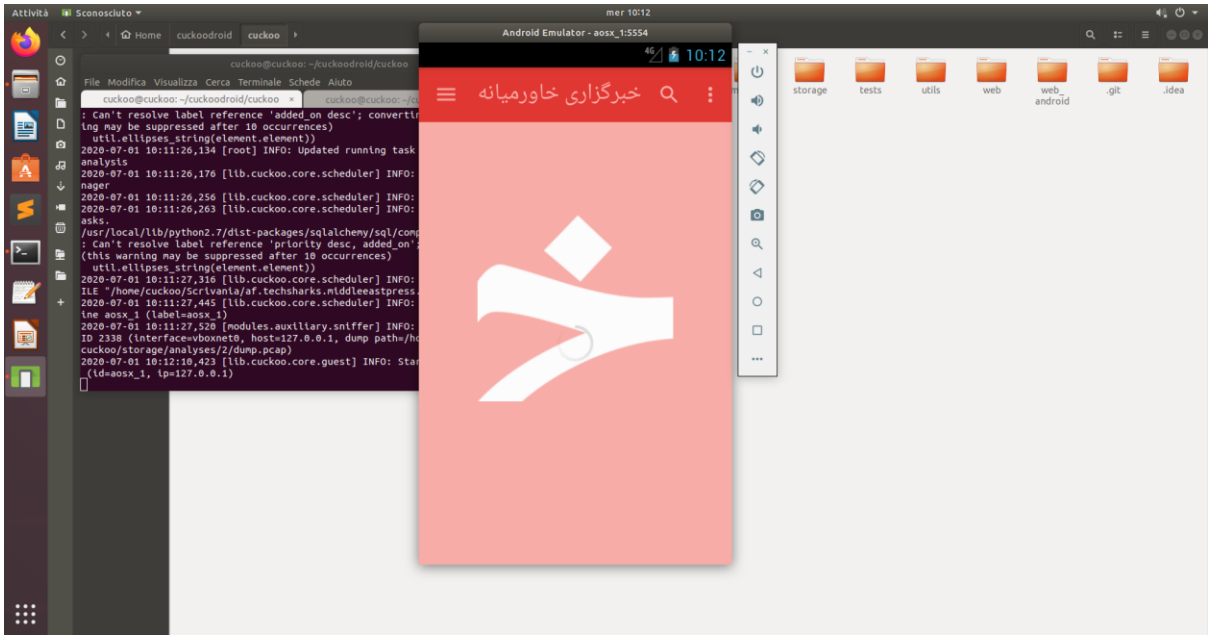

Figura 3 analisi dinamica di un'applicazione Android

Gli autori di (B. Amos, 2013) hanno progettato un sistema di analisi dinamica opensource chiamato STREAM che fa uso di un'infrastruttura di analisi distribuita costituita da un master e da più nodi.

I nodi contengono un'istanza emulata di Android con il relativo software per la gestione dell'analisi. Il master invece si occupa di coordinare le analisi distribuendo i campioni da analizzare ai nodi e raccogliendo da essi i report delle analisi da utilizzare per la classificazione con gli algoritmi naive Bayes, percettrone multistrato, reti Bayesiane, albero decisionale e regressione logistica. I risultati migliori sono stati ottenuti con le reti Bayesiane.

Gli autori di (V. G. Shankar, 2017) hanno implementato un sistema chiamato *AndroTaint* basato sull'analisi *taint* dinamica per tracciare i movimenti dei dati personali degli utenti e rilevare la loro trasmissione non autorizzata. L'analisi *taint* consiste nell'associare alle variabili (o più in generale dati) che si vogliono tracciare delle etichette (chiamate *tag*) che consentono di tracciare tali dati e verificare dove vanno a finire e quali operazioni sono eseguite su di essi. In questo caso, vengono identificati e taggati tutti i dati personali degli utenti (ad esempio codici identificativi univoci, contatti, SMS, coordinate GPS, …), i tag consentono di tracciare tali dati per verificare se raggiungono un'interfaccia di rete e quindi vengono trasmessi.

2.2.1 Analisi dinamica attiva

Al fine di ottenere più informazioni possibili sul software analizzato per determinarne la bontà quasi tutti i sistemi di analisi dinamica utilizzano degli input fittizi il cui obiettivo è indurre il malware a rivelare i suoi comportamenti malevoli. Tuttavia, a differenza di ciò che farebbe un utente umano, tali input sono solitamente indipendenti dalle reazioni del malware, quindi, non è possibile simulare realisticamente un'interazione tra l'utente e l'applicazione.
Per ovviare a questo problema sono stati suggeriti degli approcci basati sulla teoria dei giochi che modellano l'interazione malware-analizzatore come un gioco competitivo in cui l'analizzatore cerca di scoprire il più possibile riguardo al malware mentre il malware cerca di nascondere il suo reale comportamento. Questo approccio è impiegato dagli autori di (Williamson, 2012) e di (Sartea, 2016) che utilizzano l'entropia come funzione di ricompensa per l'analizzatore al fine di ottenere la maggior quantità possibile di informazioni.

2.2.2 Evasione nell'analisi dinamica

Come per l'analisi statica, anche per quella dinamica gli attaccanti hanno sviluppato dei metodi per evitare che i loro malware vengono riconosciuti come tali. Infatti, esistono vari indizi che possono essere sfruttati dagli attaccanti per calcolare la verosimiglianza[7] dell'esecuzione simulata, tra questi:

- **Assenza di connessione ad internet;**
- **Quantità di RAM:** come già scritto in precedenza la RAM ha costo non trascurabile in un sistema di analisi dinamica, per questo motivo, i ricercatori tendono ad utilizzare la minima quantità possibile di RAM per le macchine;
- **Spazio di archiviazione:** valgono considerazioni analoghe al caso precedente;
- **Numero di applicazioni installate:** un numero esiguo di applicazioni installate può essere un indicatore di un ambiente poco realistico;
- **Tipologie di applicazioni installate:** alcuni malware sono programmati per controllare se all'interno della lista di applicazioni installate sono presenti applicazioni tipicamente

[7] In statistica, la funzione di verosimiglianza (o funzione di likelihood) è una funzione di probabilità condizionata, considerata come funzione del suo secondo argomento, mantenendo fissato il primo argomento. Nel linguaggio comune viene indicata anche con il termine probabilità sebbene statisticamente non siano sinonimi.

utilizzate dai ricercatori di sicurezza informatica.

- **Numero contatti:** nell'ambito degli smartphone, considerando che nonostante le continue evoluzioni tali dispositivi nascono per telefonare, una rubrica vuota o con pochi contatti indica quasi sicuramente un ambiente simulato.

L'autore di (Rashid, 2017) ha elencato una serie di metodi utilizzati dai malware per evadere l'analisi in un ambiente Android virtualizzato e le relative contromisure per evitare che ciò avvenga.

2.2.3 Impatto dell'utilizzo della connessione ad internet

Per ciò che riguarda la rete è necessario un approfondimento a parte, infatti, l'assenza di rete può determinare il fallimento dell'analisi anche in assenza di tentativi di evasione da parte dell'attaccante che ha programmato il malware analizzato. Questo è il caso di tutti i malware che richiedono una connessione ad internet per funzionare come ad esempio quelli che ricevono comandi da un server C&C.

Inoltre, la decisione di bloccare o meno le connessioni di rete configura un problema del tipo uovo e gallina (F. Massicotte, 2012):[8]
Dal punto di vista della sicurezza, sebbene l'utilizzo di una sandbox prevenga danni alla macchina su cui è eseguita, se la sandbox stessa è connessa ad internet è possibile che i malware durante l'analisi effettuino connessioni malevole di vario tipo, per esempio possono:

- Contagiare altri dispositivi;
- Partecipare ad attacchi DDOS (Anwar, 2018);
- Truffare gli inserzionisti e le società che gestisce le inserzioni simulando click pubblicitari inesistenti (Crussell, 2014).

[8] Dalla lettura del lavoro citato si evince che il problema dell'uovo e della gallina nell'ambito dei D-MAS non riguarda solo la rete ma è più generale. Tuttavia, a giudizio dell'autore di questa tesi la rete è l'esempio più esplicativo di tale problema.

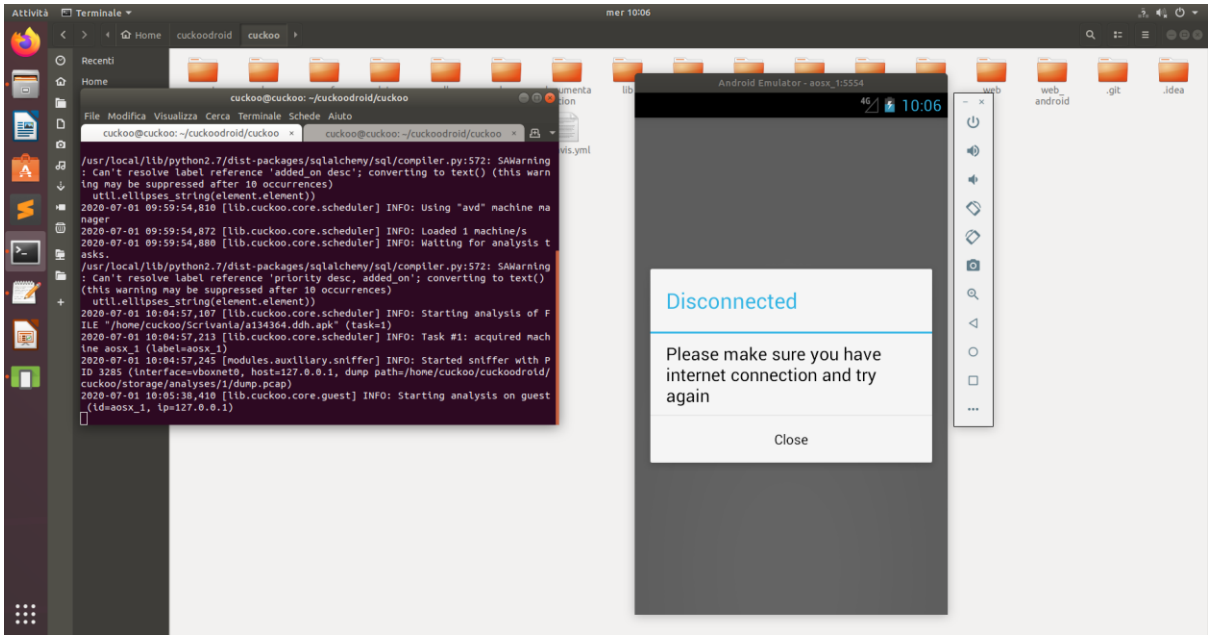

Figura 4 analisi di un'applicazione che richiede l'accesso alla rete per funzionare

Per tutti questi motivi solitamente viene configurato un firewall che blocca tutte le connessioni da e verso la sandbox ad eccezione di quelle necessarie per il funzionamento della sandbox stessa e del sistema di analisi.

Tali misure di sicurezza, tuttavia, limitano la precisione dell'analisi in vari modi, per esempio, un malware potrebbe non essere in grado di scaricare ed eseguire codice malevolo, eseguire comandi ricevuti dall'attaccante, o contagiare altri dispositivi. Tutti questi comportamenti non saranno dunque osservabili.

2.3 Confronto tra analisi statica e dinamica per il riconoscimento di malware

I vantaggi e svantaggi dell'analisi statica e dinamica sono in larga parte complementari, per questo motivo tali vantaggi/svantaggi verranno discussi insieme, suddivisi per categorie.

Prestazioni:

L'analisi statica, non necessitando di eseguire le applicazioni da analizzare, è generalmente molto efficiente anche se ovviamente, la complessità effettiva dipende dal tipo di analisi eseguita. Ad esempio, un'analisi statica avanzata basata sul calcolo del grafo del flusso di

controllo è molto più onerosa di una analisi di base basata sull'hashing.

Al contrario l'analisi dinamica è costosa sotto tutti i punti di vista:

- **Potenza di calcolo:** l'analisi dinamica richiede molta potenza di calcolo in quanto è necessario emulare un dispositivo virtuale all'interno di una sandbox e in alcuni casi è inoltre necessario emulare un'architettura diversa da quella della macchina host.
- **Memoria primaria (RAM):** per lo stesso motivo è necessario allocare un quantitativo di RAM al fine di far funzionare correttamente la sandbox. Inoltre, va considerato che la maggior parte dei software di virtualizzazione (compresi VirtualBox e AVD) allocano l'intera quantità di memoria della macchina guest per intera durata dell'analisi a prescindere dalla memoria effettivamente utilizzata dal sistema operativo della macchina guest[9].
- **Memoria secondaria (spazio di archiviazione sul disco):** oltre allo spazio necessario per memorizzare lo snapshot[10] della macchina virtuale da utilizzare per l'analisi è necessario tenere in considerazione il fatto che i risultati dell'analisi occupano pure parecchio spazio, comunemente dell'ordine dei GB.
- **Tempo:** l'analisi statica richiede molto tempo per il fatto che deve essere simulata un'interazione realistica con l'applicazione come se fosse effettivamente utilizzata da un utente umano. All'aumentare della dimensione del dataset ciò pregiudica gravemente la fattibilità degli esperimenti. Per ridurre il tempo complessivo è possibile parallelizzare l'analisi ma ciò aumenta la complessità del sistema e la richiesta di CPU e RAM di un fattore pari al numero di analisi eseguite in parallelo.

Analogamente al caso dell'analisi statica, anche per quella dinamica l'effettiva richiesta di risorse dipende dalla specifica implementazione. A questo proposito un fattore può essere, per esempio, la versione del sistema operativo usato per l'analisi oppure il numero di applicazioni preinstallate nello snapshot utilizzato per l'analisi.

In conclusione, sebbene l'efficienza dipenda dalla specifica implementazione del sistema di

[9] nell'ambito della virtualizzazione per macchina *Host* (o Host machine o Host VM) si intente una macchina virtuale eseguita su una macchina reale detta *Guest* tramite un software chiamato *Hypervisor*
[10] Nei sistemi di virtualizzazione uno snapshot indica un'istantanea dello stato dell'intera macchina virtuale in un dato instante

analisi adoperato, è un dato di fatto che l'analisi statica richiede meno risorse rispetto a quella dinamica. Questo è particolarmente importante in ambito mobile in quanto l'analisi può essere effettuata anche sul dispositivo stesso a condizione che ovviamente, non vengano utilizzati algoritmi di classificazione onerosi dal punto di vista computazionale.

Accuratezza:

L'analisi dinamica è più accurata di quella statica perché tramite l'esecuzione riesce ad ottenere maggiori informazioni circa i reali comportamenti dei campioni analizzati, per esempio:

- offuscamento del codice di qualsiasi tipo;
- caricamento dinamico di librerie e di codice;
- esecuzione di comandi provenienti da un server C&C;
- comportamenti innescati dall'interazione con altre applicazioni o componenti del sistema (a condizione che tali app e componenti siano presenti nel sistema utilizzato per l'analisi);
- comportamenti innescati dall'interazione con l'utente se questa viene opportunamente simulata.

Tutte queste informazioni non possono essere ricavate, invece, con l'ausilio della sola analisi statica che risulta quindi essere generalmente meno accurata.
Tuttavia bisogna anche notare che le tecniche di analisi statica più avanzate (come quelle utilizzate in (B. Kang, 2016), (Marastoni, 2017) e (W. Park, 2014) e che sono state discusse in precedenza) hanno il vantaggio di analizzare interamente il codice (ad esclusione dell'eventuale codice aggiuntivo caricato a tempo d'esecuzione) cosa che ovviamente non è possibile con l'analisi dinamica poiché in quel caso ci saranno parti del codice che non verranno raggiunte dal flusso di esecuzione.

Sicurezza:

L'analisi statica, non eseguendo i campioni software da analizzare, non comporta nessun rischio[11] per la macchina che effettua l'analisi. Per questo motivo l'analisi statica è, dal punto

[11] In realtà in informatica e in generale nella vita non esiste il rischio zero, per esempio, si potrebbe scrivere un

di vista dell'analizzatore più sicura dell'analisi dinamica.
Va considerato però che con le dovute accortezze, come il sandboxing e la configurazione di un firewall, anche l'analisi dinamica può essere effettuata in sicurezza. Tuttavia, nel caso del firewall questa sicurezza è ottenuta al costo di perdere accuratezza.

2.4 Analisi Ibrida

Come si è visto precedentemente, l'analisi statica e quella dinamica hanno vantaggi e svantaggi che sono in larga parte complementari, perciò, appare naturale provare a combinare i due approcci per ottenere risultati migliori. Questo si ottiene al costo di un incremento nella complessità del sistema.
A seconda degli obiettivi del sistema ibrido esistono numerose implementazioni che combinano in modi diversi i due approcci. Le più comuni si possono suddividere in due categorie: quelle che eseguono sempre entrambi i tipi di analisi e quelle che eseguono prima l'analisi statica e sulla base dei risultati statici decidono se effettuare o meno la più onerosa analisi dinamica. Il primo approccio consente di ottenere maggiore accuratezza a scapito delle prestazioni, in quanto richiede sempre di eseguire entrambe le analisi prima di ottenere i risultati. Al contrario il secondo approccio è un compromesso tra prestazione e accuratezza poiché consente di ridurre significativamente l'impiego medio di risorse per analisi ma al contempo fa ottenere risultati migliori della sola analisi statica o dinamica.

Nella prima categoria rientrano i sistemi che a partire dai i risultati delle due analisi creano un unico vettore di features da utilizzare per la classificazione come proposto dagli autori di SAMADroid (S. Arshad, 2018) che combinano i risultati dell'analisi statica e di quella dinamica in un unico vettore per la classificazione, inoltre, il loro sistema è ibrido su più livelli: infatti oltre a combinare l'analisi statica con quella dinamica, combina anche l'analisi locale sul dispositivo con l'analisi remota.

malware in grado di sfruttare le eventuali vulnerabilità di un particolare sistema di analisi statica in modo simile a quanto riportato in questa Proof of Concept (Peter Ney, 2017).

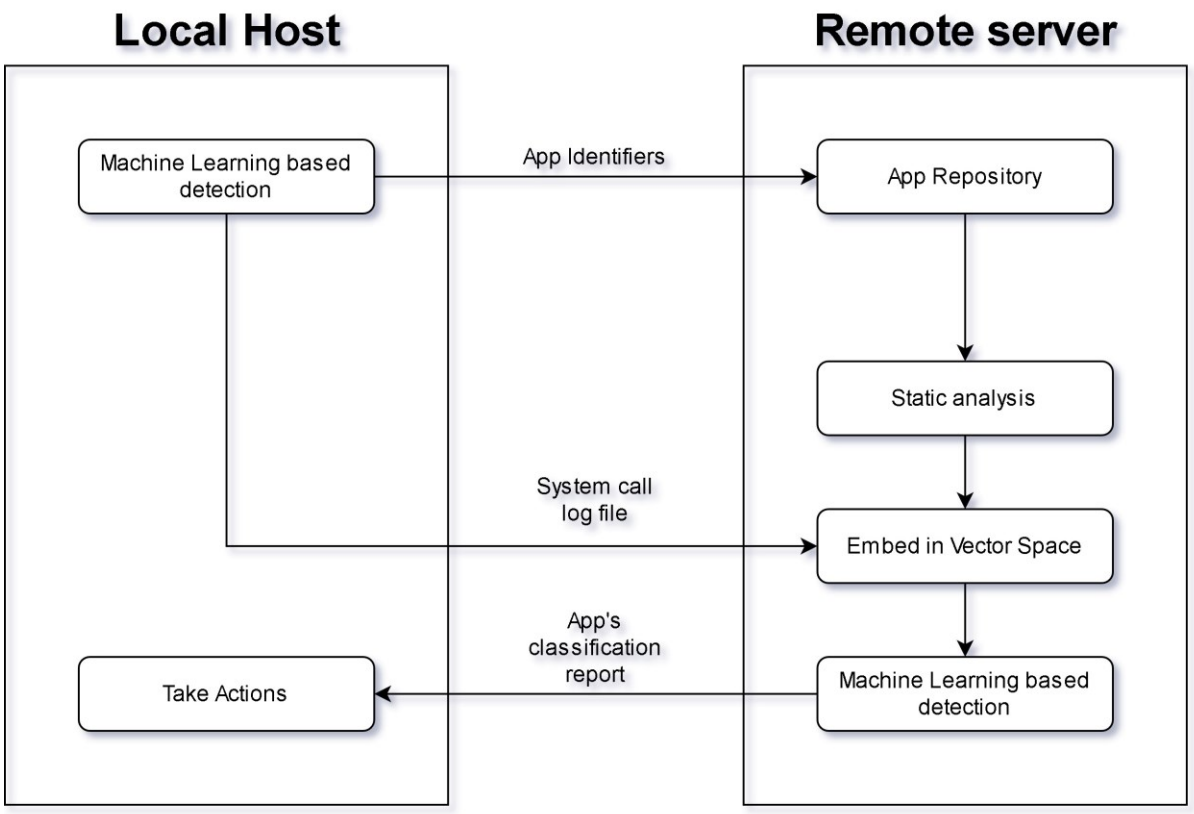

Figura 5 sistema proposto da (S. Arshad, 2018)

L'analisi statica, che non richiede di eseguire le applicazioni, viene effettuata su un server remoto. L'applicazione viene scaricata direttamente dallo store (questo evita al dispositivo mobile di dover inviare il pacchetto apk al server, permettendo di risparmiare banda e autonomia), decompressa, disassemblata e quindi vengono estratte le features statiche. Per le ultime due fasi vengono utilizzati i software Android Asset Packaging Tool e Baksmali.
Per quanto riguarda quella dinamica, invece di simulare l'esecuzione all'interno di una sandbox è stato scelto di monitorare le app mentre vengono utilizzate dagli utenti, questo consente di osservarle in una contesto realistico con interazioni umane autentiche. Le features dinamiche includono la sequenza delle chiamate di sistema, il caricamento dinamico di codice e la decifratura di codice cifrato.

Anche gli autori di (M. Z. Mas'ud, 2013) eseguono sempre le due analisi per ottenere risultati più precisi. Tuttavia, in questo caso non è stato impiegato un sistema di apprendimento automatico ma l'analisi delle features statiche e dinamiche è stata effettuata manualmente. Questo consente ai ricercatori di correlare l'analisi statica con quella dinamica, cioè viene

cercata la corrispondenza tra i comportamenti del malware e il codice responsabile di tali comportamenti. L'analisi dinamica è stata eseguita fisicamente su quattro tablet Android e sono state monitorate le chiamate di sistema e l'attività di rete. Il sistema è stato testato su sei malware appartenenti al dataset Android Malware Genome Project.

Invece un esempio di sistema ibrido che attiva l'analisi ibrida solo al verificarsi di determinate condizioni è (A. De Paola, 2018) che propone un sistema di rilevamento di malware basato sul cloud che esegue indipendentemente le due tipologie di analisi.
L' analisi statica consiste in una rete neurale profonda, capace di ottenere una buona precisione utilizzando solo una piccola parte di dimensione fissa dell'eseguibile, oltre a determinare se il file analizzato è un malware l'analisi statica determina anche il grado di precisione di tale stima. Nel caso in cui tale livello di incertezza sia al di sotto di una soglia allora viene eseguita anche l'analisi dinamica ottenendo una classificazione più accurata. Tale soglia può essere regolata a tempo di esecuzione per effettuare un'analisi più approfondita nei periodi con basso carico.

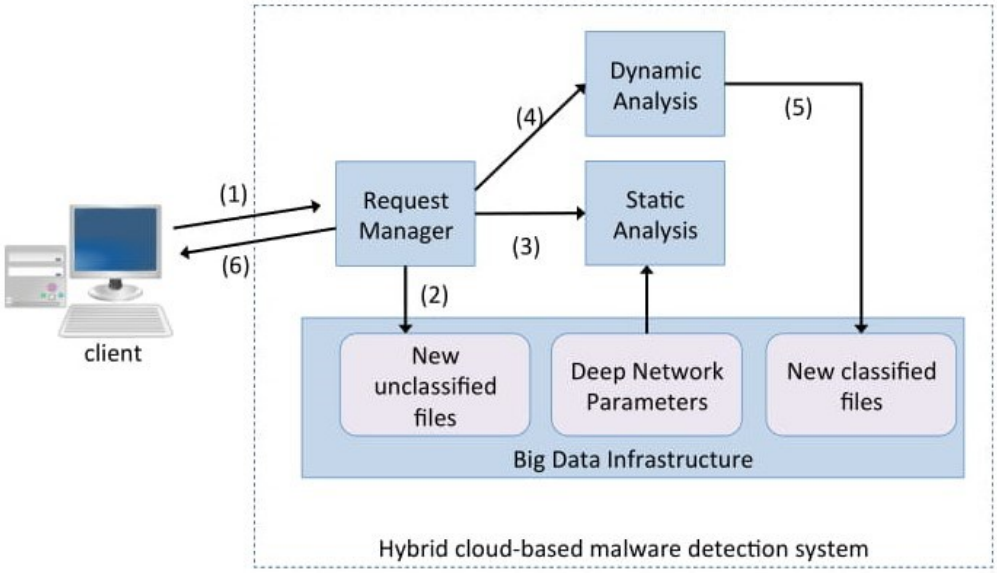

Figura 6 sistema ibrido proposto da (A. De Paola, 2018)

In questo modo oltre ad avere ottenere risultati più accurati rispetto alla sola analisi statica si possono usare i risultati dell'analisi dinamica per addestrare in modo supervisionato il sottosistema statico con software recente senza la necessità di contrassegnare (come benigno o malevolo) manualmente ogni nuovo campione.

Opera in modo simile anche (D. Kim, 2018) in cui l'output dell'analisi può essere di tre tipi:

- Certamente benigno;
- Certamente malevolo;
- Necessaria un ulteriore analisi.

Nei primi due casi il sistema da come responso il risultato della sola analisi statica. Nel terzo caso invece viene effettuata un'ulteriore analisi dinamica avvalendosi del software opensource Cuckoo Sandbox. Sia per l'analisi statica che per quella dinamica è stato usato come algoritmo di classificazione il percettrone multistrato (MLP). Il sistema è stato testato con un dataset di malware estratti dai repository Virus Total e Virus Share.

3 Metodi di classificazione

In questo capitolo, vengono descritti gli algoritmi di apprendimento automatico utilizzati nel sistema di riconoscimento di malware. L' apprendimento automatico (o *machine learning)* è una branca dell'intelligenza artificiale che si occupa dell'implementazione di software in grado di imparare autonomamente ovvero di modificare il proprio funzionamento in risposta all'esperienza acquisita senza la necessità di essere riprogrammati manualmente.

Il processo con cui tali algoritmi apprendono è chiamato addestramento e consiste nella presentazione di un insieme di dati chiamato *dataset*. Il dataset solitamente consiste in un insieme di esempi talvolta chiamati *istanze,* ognuno dei quali è rappresentato da un vettore di caratteristiche (chiamate anche *attributi* o *features*).

Il risultato dell'addestramento consiste in una rappresentazione dello stato di conoscenza acquisito dall'algoritmo chiamata *modello* la cui struttura dipende dal tipo di classificatore addestrato. Il modello ottenuto può essere in seguito utilizzato con dei dati generici per ottenere previsioni riguardo istanze generiche differenti da quelle conosciute (tramite addestramento) dal modello.

Gli algoritmi di apprendimento automatico si possono suddividere in base al tipo di previsione, in tre categorie:

- Classificazione;
- regressione;
- clustering.

Nel caso della classificazione ciò che si desidera ottenere è uno strumento chiamato classificatore che dopo essere stato opportunamente addestrato sia in grado per ogni nuovo input di assegnarlo ad una delle possibili classi. L'insieme delle classi è predeterminato cioè va stabilito prima di iniziare l'addestramento e non può variare successivamente.
Un esempio di classificazione è proprio quello che è stato utilizzato in questo lavoro di tesi, ovvero un classificatore che dato il file di una applicazione determina se appartiene alla classe del software benigno oppure a quella del software malevolo. In questo caso si tratta di un

classificatore binario poiché l'insieme delle classi è costituito da due soli elementi.

La regressione può essere vista come una variante della classificazione in cui invece assegnare ogni istanza ad una classe appartenente ad un insieme discreto e predeterminato si cerca di determinare un valore numerico continuo.

Il clustering, infine, viene utilizzato per suddividere dei dati in delle categorie che, a differenza della classificazione, non sono predeterminate ma vengono scelte automaticamente. Un'applicazione degli algoritmi di clustering è la suddivisone dei malware in famiglie in base alle similitudini comportamentali.

A seconda delle caratteristiche del dataset è possibile individuare tre tipologie di apprendimento:

- Apprendimento supervisionato;
- apprendimento non supervisionato;
- apprendimento semi-supervisionato.

L'apprendimento supervisionato consiste nel fornire in input al modello un dataset in cui ogni campione è contrassegnato con la classe di appartenenza, l'obiettivo dell'addestramento è di estrarre una regola generale che associ i dati alle rispettive classi.

L'apprendimento non supervisionato invece consiste nel cercare una struttura nei dati di input senza che tali dati siano etichettati con la loro classe di appartenenza.

Nel caso dell'apprendimento semi-supervisionato, infine, solo una frazione del dataset è etichettata con la classe di appartenenza. Solitamente la porzione contrassegnata è minoritaria rispetto al dataset complessivo poiché l'etichettatura dei dati è molto costosa e/o lenta. Considerando che l'apprendimento non supervisionato viene utilizzato per apprendere le caratteristiche strutturali del dataset, in genere si procede prima con l'apprendimento non supervisionato e successivamente con quello supervisionato.

Per questo lavoro di tesi si è scelto di considerare i seguenti algoritmi:

- Albero decisionale;
- foresta casuale;
- reti Bayesiane.

La scelta è ricaduta su questi tre algoritmi poiché si tratta di algoritmi spesso utilizzati in letteratura per l'analisi dei software malevoli che inoltre possono essere implementati in modo efficiente senza la necessità di hardware dedicato.

3.1 Albero decisionale

Secondo (Norvig, 2002) gli alberi decisionali sono la più semplice ma al contempo più riuscita forma di apprendimento automatico.

Un albero decisionale è un algoritmo di apprendimento supervisionato che consiste in un albero in cui ogni nodo rappresenta un test su un attributo del vettore di input e ad ogni foglia è assegnato uno dei possibili esiti della classificazione.
Il modello di un albero decisionale, quindi, consiste in un albero con delle regole associate ad ogni nodo interno, tali regole sono semplicemente dei confronti sul valore degli elementi del vettore di ingresso.
Avendo a disposizione un modello è abbastanza semplice trovare l'output della classificazione per una data istanza, infatti, basta partire dalla radice e seguire il percorso indicato dalle regole associate ai nodi per arrivare alla foglia che rappresenta il risultato della classificazione.

Gli alberi decisionali effettuano delle decisioni in un modo molto simile ai processi decisionali umani e di conseguenza i modelli generati risultano essere molto intuitivi. Per questo si dice che gli alberi decisionali operano in una *white box* in contrapposizione con la *black box* in cui operano algoritmi i cui modelli non sono comprensibili dalle persone (per esempio le reti neurali).

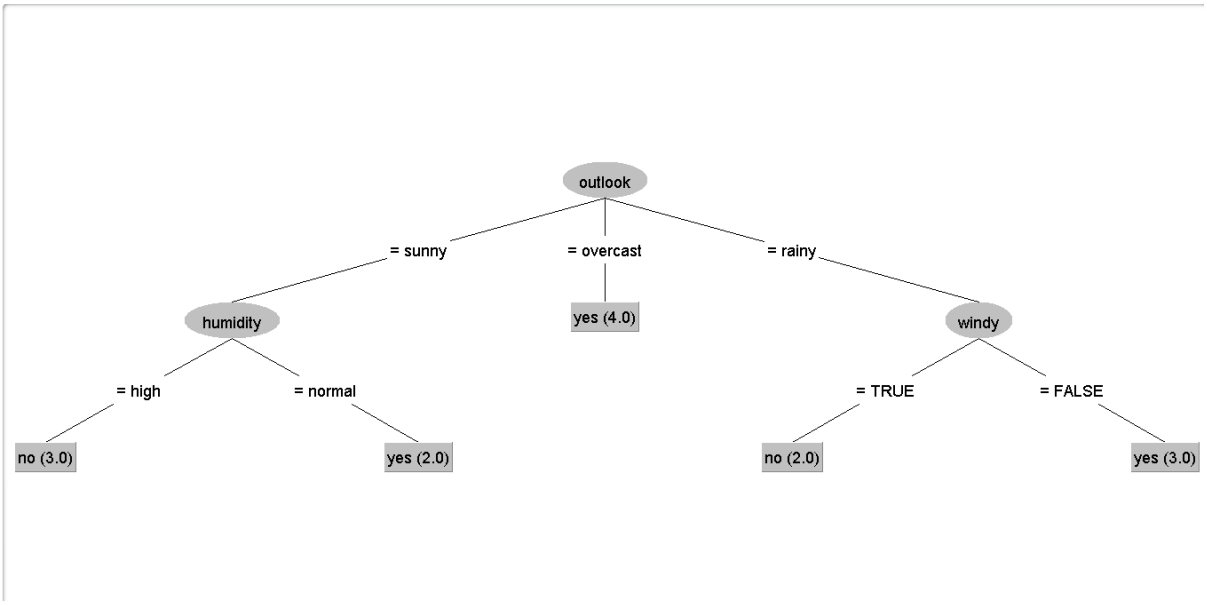

Figura 7: esempio di albero decisionale

L'addestramento supervisionato di un albero decisionale a partire dagli esempi contrassegnati può avvenire tramite diversi algoritmi, i più comuni si basano su ID3(Iterative Dichotomiser 3).
Va comunque notato che una volta generato il modello, il test su nuove istanze è indipendente dallo specifico algoritmo utilizzato per l'addestramento.

3.1.1 Apprendimento tramite ID3

ID3 si basa sul concetto di *information gain* che a sua volta si basa sul concetto di *entropia*, quindi, di seguito verranno entrambi illustrati brevemente.
L'entropia è una misura dell'incertezza di una variabile casuale. Nel caso di evento certo, per esempio, non c'è nessuna incertezza e quindi il valore dell'entropia sarà zero. Diversamente l'entropia indica il numero di bit necessari *in media* per rappresentare un'informazione. Per esempio, l'entropia associata al lancio di una moneta non truccata è 1 poiché ci sono solo due possibili esiti equiprobabili e quindi basta un solo bit per codificare tale informazione.

Conoscendo la distribuzione di probabilità di una variabile aleatoria è possibile calcolare l'entropia con la seguente formula (Shannon, 1948):

$$H(S) = \sum_{x \in X} -p(x) \log_2 p(x) \,,$$

dove:

- S indica la sorgente, in questo caso il dataset di cui si vuole calcolare l'entropia,
- X è l'insieme delle classi in cui è suddiviso S, ad esempio, software malevolo e benigno,
- p(x) è la probabilità che un elemento di S appartenga alla classe x o, più semplicemente il rapporto tra gli elementi della classe x e il numero complessivo di elementi in S.

Da notare che, essendo l'entropia calcolata come un media ponderata dalla probabilità, il valore risultante può anche non risultare intero.
Come detto in precedenza l'entropia è nulla solo se in assenza di qualsiasi incertezza quindi se H(S)=0 allora tutti gli elementi del dataset S apparterranno alla stessa classe.

L'information gain misura l'incremento di informazione ottenuto dopo aver suddiviso il dataset in base ad uno specifico attributo e consiste nella differenza tra l'entropia calcolata prima della suddivisione ed il suo valore dopo la suddivisione sull'attributo selezionato.

$$IG(S, A) = H(S) - \sum_{t \in T} p(t) H(t) \,,$$

dove:

- **H(S)** indica l'entropia prima della suddivisone;
- $\sum_{t \in T} \boldsymbol{p(t) H(t)}$ indica l'entropia dopo aver effettuato la suddivisione sull'attributo A, calcolata come media ponderata dalla probabilità delle entropie dei nodi figli risultanti dalla suddivisone;
- **T** indica l'insieme dei sottodataset ottenuti dalla suddivisione del dataset S secondo l'attributo A, quindi, la cardinalità di T sarà pari al numero di possibili valori (o intervalli di valori nel caso continuo) dell'attributo A.

Per scegliere l'attributo migliore su cui effettuare la suddivisione basta calcolare *l'information gain* su ogni possibile valore e scegliere quello con l'*information gain* più alto. Questo procedimento va ripetuto ricorsivamente finché non si arriva ad avere un insieme di dati tutti

appartenenti alla stessa classe, a questo punto, la classe rappresenterà il valore di classificazione per quella foglia dell'albero.

ID3 costruisce l'albero ricorsivamente in modo *top-down*, partendo dalla radice che rappresenta l'intero training set fino ad arrivare alle foglie, secondo i seguenti passi:

1. Assegna alla radice l'intero training set;
2. calcola l'*information gain* su tutti gli attributi;
3. sceglie l'attributo con il maggiore valore di *information gain*;
4. effettua la suddivisione del dataset secondo l'attributo scelto generando tanti nodi figli quanti sono i possibili valori dell'attributo;
5. per ogni nodo figlio si procede ricorsivamente con i passi 2, 3 e 4 finché non ottiene un sottoinsieme dell'insieme di addestramento perfettamente classificato cioè con tutte le istanze che appartengono alla stessa classe;
6. nel caso in cui, avendo ottenuto un sottoinsieme dell'insieme di addestramento perfettamente classificato, si arresta la ricorsione, viene generato un nodo foglia contrassegnato con la relativa classe di appartenenza.

Di seguito è mostrato lo pseudocodice nel caso di una funzione di classificazione binaria, per esempio, per determinare se un software è maligno o benigno (Mitchell, 1997):

```
Esempi:= istanze di addestramento.
Attributo_obiettivo:= l'attributo il cui valore viene predetto dall'albero. Attributi:= lista di altri attributi.

ID3 (Esempi, Attributo_obiettivo, Attributi)
   Crea un nodo Radice.
   Se tutti gli esempi sono positivi
      restituisci un albero con un unico nodo Radice ed etichetta = +     Se tutti gli esempi sono negativi
      restituisci un albero con un unico nodo Radice ed etichetta = -     Se Attributi è vuoto
      restituisci un albero con un unico nodo Radice ed etichetta = il valore di Attributo_obiettivo più comune tra le istanze di Esempi.
   Altrimenti
      A ← L'elemento di Attributi che riduce maggiormente l'entropia.
      Per ogni valore v di A,
         Aggiungi un nuovo ramo sotto Radice corrispondente al test A = v.
         Esempi(v) ← sottoinsieme di Esempi che hanno valore v per A.
         Se Esempi(v) è vuoto
            Aggiungi una foglia con etichetta = valore di Attributo_obiettivo più comune tra gli esempi.
         Altrimenti
            Aggiungi sotto Radice il sottoalbero ID3 (Esempi(v), Attributo_obiettivo, Attributi – {A}).
   Restituisci Radice.
```

Figura 8 pseudocodice di ID3 nel caso della classificazione binaria

3.2 Random Forest

La *Random Decision Forest*, o più semplicemente *Random Forest* (Ho, 1995), è un algoritmo di classificazione basato sull'utilizzo di un insieme di alberi decisionali. Gli alberi della foresta sono scelti considerando per ogni albero un diverso sottoinsieme di dati scelto casualmente.

L'algoritmo usato in *Random forest* è basato sulla tecnica di *Bootstrap Aggregation* o *bagging* (Breiman, 1994). Quest'ultima consiste nel calcolare modelli di alberi decisionali diversi a partire da un singolo training set ripetutamente suddiviso in diversi sottoinsiemi scelti casualmente. Ognuno di questi sottoinsiemi viene utilizzato per addestrare un albero e quindi gli alberi vengono costruiti indipendentemente l'uno dall'altro producendo risultati generalmente diversi.
Il risultato della classificazione viene determinato tramite la media delle previsioni di ogni albero generato o tramite il voto a maggioranza.
Sebbene tale tecnica risulti generalmente più robusta rispetto ad un singolo albero, può comunque accadere che gli alberi presentino molte somiglianze strutturali compromettendo la bontà delle previsioni.

Random Forest presenta dei miglioramenti rispetto alla tecnica di Bootstrap Aggregation. Queste modifiche riducono la correlazione tra i sottoalberi generati sfruttando non solo la selezione casuale dei dati ma anche la selezione casuale delle caratteristiche (*feature bagging* o *attribute bagging*).
Supponendo che ci siano N campioni ed M caratteristiche nell'insieme di addestramento, l'algoritmo procede come mostrato di seguito:

1. Viene selezionato casualmente un sottoinsieme dell'insieme di N campioni;
2. Viene selezionato casualmente un sottoinsieme dell'insieme di M features;
3. Si costruisce un albero decisionale utilizzando come training set quello ottenuto al punto 1 e considerando ai fini delle suddivisioni solo le caratteristiche scelte al punto 2;
4. Si ripetono i passi precedenti finché non si raggiunge il numero di alberi desiderato;
5. La predizione complessiva della foresta viene calcolata tramite la media delle predizioni proposte da ogni sottoalbero decisionale.

La selezione casuale viene utilizzata poiché è un modo efficiente esplorare l'insieme dei possibili sottoinsiemi di campioni e features che hanno cardinalità rispettivamente di 2^N e 2^M.
Il tipo di campionamento utilizzato per la selezione casuale è *con reinserimento* il che equivale a dire che un elemento può essere selezionato più volte, ciò garantisce l'indipendenza delle selezioni tra tutti gli alberi.
L'utilizzo del *feature bagging* consente alla *Random Forest* di ottenere maggiore variabilità tra gli alberi della foresta rispetto alla *Bootstrap Aggregation*. Questo tipo di approccio permette di gestire anche grandi moli di dati mantenendo una buona accuratezza dei risultati, una minore sensibilità al rumore del dataset e anche una buona efficienza.

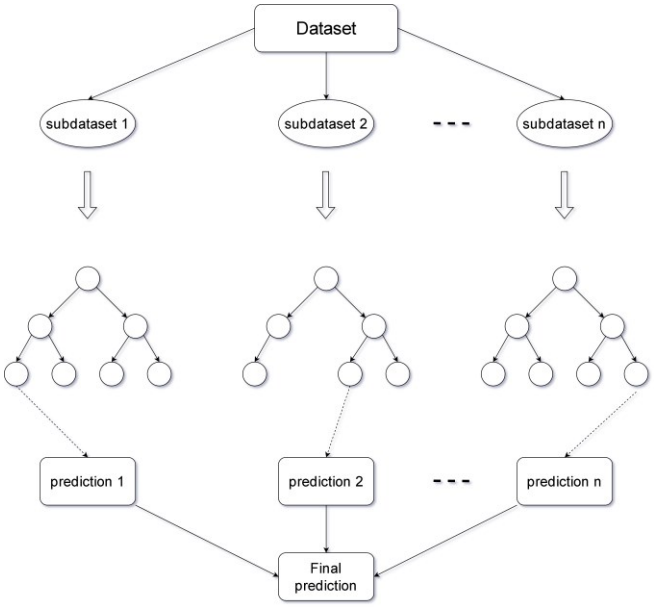

Figura 9 funzionamento del classificatore random forest

3.3 Reti Bayesiane

Le reti Bayesiane sono dei modelli probabilistici che consentono di calcolare in maniera efficiente la probabilità associata ad un evento a partire da un insieme di esempi. Prendono il nome dalla formula utilizzata per calcolare tali probabilità: la regola di Bayes.

Prima di introdurre la regola di Bayes occorre definire che cosa si intende per un evento A subordinato a un altro evento B. Ammettiamo di avere espresso la probabilità del verificarsi dell'evento A e successivamente siamo venuti a conoscenza che si è verificato l'evento B.

Occorre esprimere la valutazione di probabilità su A alla luce della nuova conoscenza acquisita riguardo B. Questa nuova probabilità è detta probabilità di *A dato B* ed è indicata con:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

In particolare, si potrebbe verificare il caso che P(A) = P(A | B), in questo caso gli eventi A e B si dicono *stocasticamente*(o statisticamente) indipendenti. In caso contrario i due eventi si dicono dipendenti.

Un esempio di eventi stocasticamente dipendenti è quello in cui l'evento A indica la vittoria della squadra X sulla squadra Y in un certo incontro sportivo che per semplicità supponiamo non ammetta pareggio, e B il fatto che a un certo punto del gioco, la squadra X è in vantaggio. Naturalmente, la valutazione di probabilità sul fatto che X alla fine vinca alla sarà maggiore

rispetto all'inizio della partita. Come esempio di eventi stocasticamente indipendenti si consideri, invece, il caso in cui A è definito come prima e B indica che durante la partita inizia a piovere; se nessuna delle due squadre è favorita da questo evento, si può supporre che la probabilità P(A) che vinca X resti invariata così come rimane invariata la probabilità che vinca Y.

Possiamo visualizzare visivamente questa formula considerando che il verificarsi di B ha ridotto lo spazio Ω a quella aliquota contenuta in B e quindi ciò che di A può verificarsi è la sua parte che interseca B, mentre la parte rimanente non può più verificarsi. D'altra parte, ora la probabilità non è più quella riferita al vecchio spazio campione Ω, bensì si è ora ristretta a B (che alla luce delle nuove informazioni ottenute costituisce l'insieme di tutti i possibili eventi proprio come lo era Ω prima della consapevolezza del verificarsi di B).

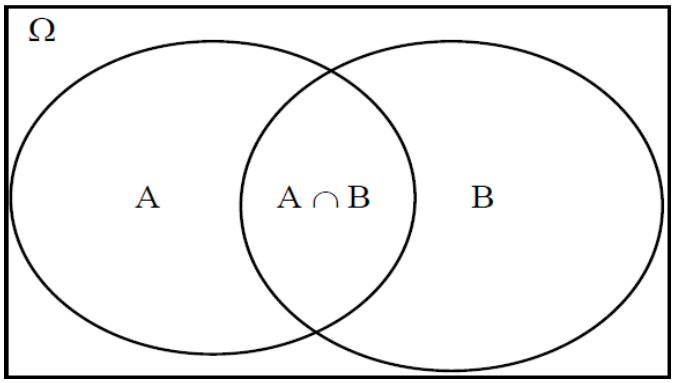

Figura 10 tratta da (Lombardo, 2016)

Nel caso in cui A e B sono statisticamente (o stocasticamente) indipendenti si ha che:

$$P(A \cap B) = P(A)P(B).$$

In questo caso la conoscenza del fatto che un evento si sia verificato o meno non modifica la probabilità del verificarsi dell'altro evento.

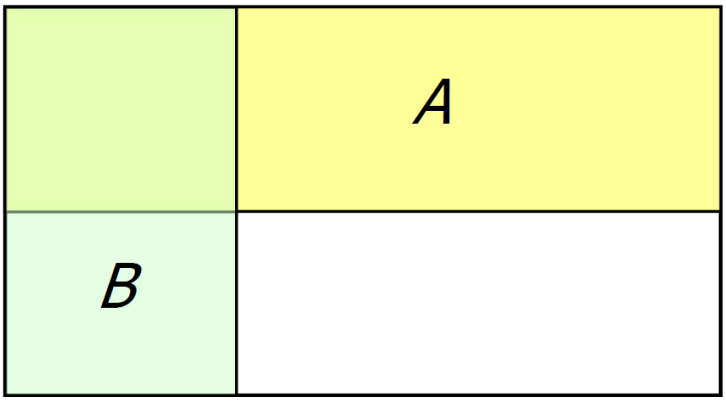

Figura 11 rappresentazione grafica di due eventi statisticamente indipendenti

Nell'esempio mostrato in Figura 11 è chiaro che si tratta di due eventi stocasticamente indipendenti poiché se si riduce la rappresentazione eliminando la parte del diagramma in cui uno dei due eventi si verifica (o non si verifica), la proporzione dell'altro evento non varia. Dal punto di vista numerico se consideriamo l'intero spazio campione Ω, ovvero non siamo nella condizione in cui non abbiamo nessuna informazione riguardo al verificarsi di A o di B abbiamo che:

$$P(A) = \frac{1}{2}\,, \qquad P(B) = \frac{1}{3}$$

Se abbiamo ottenuto l'informazione riguardo al verificarsi di A:

$$P(A|A) = 1, \qquad P(B|A) = \frac{1}{3}$$

Se, al contrario, abbiamo ottenuto l'informazione sul verificarsi di B:

$$P(A|B) = \frac{1}{2}\,, \qquad P(B|B) = 1$$

Al netto dei casi degeneri $P(A|A)$ e $P(B|B)$ riportati solo per completezza la condizione di indipendenza è data da:

$$P(A|B) = P(A) = \frac{1}{2}$$

$$P(B|A) = P(B) = \frac{1}{3}$$

Al contrario in Figura 12 è rappresentato un esempio di eventi statisticamente dipendenti. Si può notare che in questo caso, a differenza di quando accade nel diagramma precedente, se si considera la partizione dello spazio campione che indica il verificarsi di uno dei due eventi, le percentuali relative dei due eventi variano*:* Infatti se consideriamo l'intero spazio campione notiamo che A e B sono equiprobabili entrambi con un valore di probabilità pari a ½ ma se consideriamo la partizione dello spazio campione in cui si verifica A (la metà superiore del rettangolo) si ha che A e B non sono più equiprobabili con la probabilità di A che diventa minore di ½ e quella di B maggiore di ½. Valgono analoghe considerazioni nel caso in cui si sceglie B.

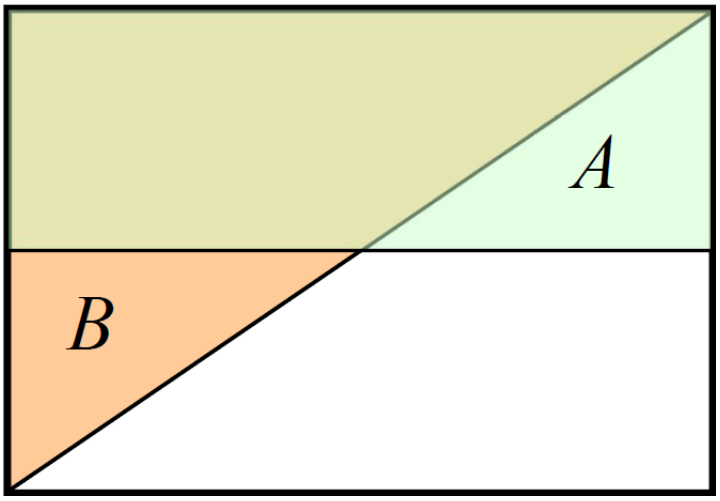

Figura 12 esempio di eventi statisticamente dipendenti

Va inoltre notato che due eventi incompatibili, cioè che non si possono mai verificare simultaneamente, non potranno mai essere indipendenti e di conseguenza saranno sempre statisticamente dipendenti. Questo perché la condizione di incompatibilità tra due eventi implica che il verificarsi di un evento fa sempre variare (annullandola) la probabilità dell'altro evento.

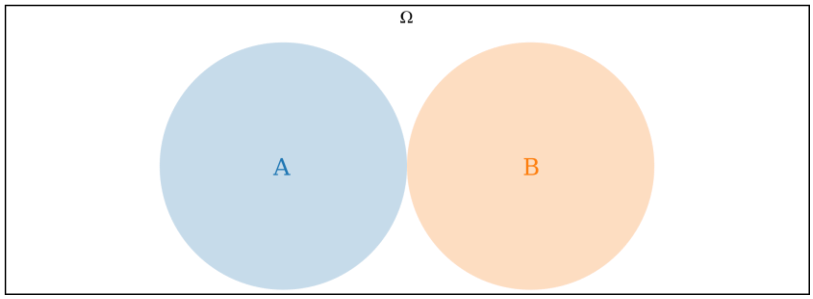

Figura 13 eventi incompatibili e quindi dipendenti

In precedenza, si è vista la formula sulla probabilità condizionata

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Questa regola si può estendere anche al caso in cui un evento dipende da più eventi, ma prima di proseguire può essere opportuno semplificare la notazione con una equivalente ma più semplice; di seguito con la notazione

$$P(A_1 \cap A_2 \cap \dots \cap A_N|B_1, B_2, \dots, B_M) = P(A_1, A_2, \dots, A_N|B_1, B_2, \dots, B_M)$$

verrà indicata la probabilità della congiunzione degli eventi $A_1, A_2, \dots, A_N$
dato il verificarsi degli eventi $B_1, B_2, \dots, B_M$.
Se una variabile statistica V_i è indipendente da $V_{j,}$, possiamo scrivere:

$$P(V_i|V_j, V) = P(V_i|V)$$

E sulla base della definizione di probabilità condizionata abbiamo che:

$$P(V_i|V_j, V)P(V_j|V) = P(V_i, V_j|V)$$

Combinando i due risultati si ottiene:

$$P(V_i, V_j|V) = P(V_i|V)P(V_j|V)$$

Generalizzando al caso di più di variabili:

$$p(V_1, V_2, ..., V_k|V) = \prod_{i=1}^{k} p(V_i|V_{i-1}, ..., V_1, V)$$

Quest'ultima è alla base del funzionamento delle reti Bayesiane che ora verranno descritte.

3.3.1 Funzionamento delle reti Bayesiane

Una rete Bayesiana è un modello probabilistico rappresentato attraverso un grafo orientato aciclico (DAG), cioè un grafo caratterizzato dalla presenza di archi orientati e dall'assenza di cicli diretti, i cui nodi corrispondono a delle variabili casuali.

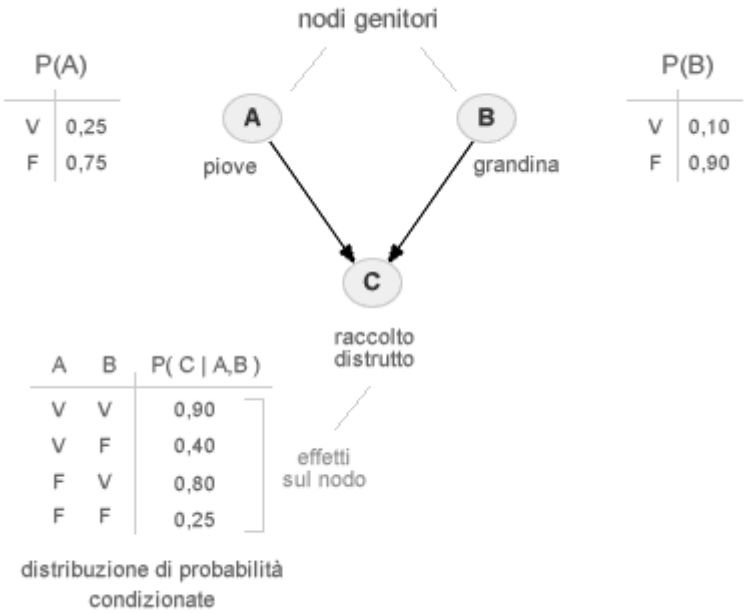

Figura 14 esempio rete Bayesiana con relative tabelle di probabilità condizionata tratto da (Rete bayesiana , 2014)

In una rete bayesiana è necessario che ogni nodo V_i del grafo sia condizionatamente indipendente da qualsiasi sottoinsieme di nodi non discendenti di V_i, dati i genitori di V_i.

Se si considera:

- $A(V_i)$ qualsiasi insieme di nodi del grafo che non sono discendenti di V_i
- $P(V_i)$ i genitori di V_i

La rappresentazione grafica del grafo equivale ad asserire che:

$$P\left(V_i \middle| A(V_i), P(V_i)\right) = P(\, V_i | P(V_i)\,)$$

Dove $P\left(V_i \middle| A(V_i), P(V_i)\right)$ indica la probabilità condizionata di V_i dati $A(V_i)$ e $P(V_i)$.

Quindi l'intera rete bayesiana può essere rappresentata attraverso la distribuzione congiunta di probabilità di un insieme di variabili casuali vista precedentemente:

$$p(V_1, V_2, ..., V_k | V) = \prod_{i=1}^{k} p(V_i | V_{i-1}, ..., V_1, V)$$

3.3.2 Addestramento di una rete Bayesiana

Addestrare una rete bayesiana equivale a trovare una rete che corrisponda al meglio al training set, cioè determinare sia la struttura del grafo aciclico, che le tabelle di probabilità condizionata, associate ad ogni nodo del grafo. Il tipo di addestramento necessario varia a seconda se le informazioni sulla struttura della rete sono disponibili o meno.
Nel primo caso, solitamente la struttura è fornita da un esperto ed è necessario apprendere solo le tabelle di probabilità condizionata.
Nel secondo caso invece l'apprendimento si compone di due fasi; prima si apprende la struttura della rete e dopo si apprendono le tabelle.

3.3.2.1 Apprendimento della struttura

Per quanto riguarda l'apprendimento della struttura della rete, nel caso non sia già nota, è necessaria sia una metrica per la valutazione delle reti che un metodo per effettuare la ricerca nello spazio delle possibili reti. Per la scelta della metrica è possibile considerare le proprietà statistiche dei codici efficienti utilizzate per la compressione dei dati, come ad esempio

Huffman. In questo caso si considera l'insieme di addestramento come un vettore di bit da trasmettere, da codificare con una codifica efficiente.

Dato quindi un training set T e una rete bayesiana B, secondo la teoria dell'informazione (Shannon, 1948) la codifica più efficiente di T distribuito secondo le probabilità di B richiede $L(T, B)$ bit:

dove $L(T, B) = -log(p[T])$
e $p[T]$ che indica la distribuzione di probabilità dei dati di T.

A partire dal training set T è possibile trovare quella rete B_0 che minimizza $L(T, B)$.
Tuttavia, la sola $L(T, B)$ non è una metrica adeguata, visto che il suo uso induce reti sovra-adattate per l'insieme di addestramento. Quindi similmente a quanto avviene in altri algoritmi di apprendimento automatico per limitare il sovra-adattamento si cerca di limitare la dimensione del modello. Nel caso in esame si aggiunge ad $L(T, B)$ la dimensione della rete.

Per quanto riguarda invece la ricerca è possibile adottare un tipo di ricerca non esaustiva chiamata hill-descending per trovare quella che minimizza la metrica scelta. Si parte da una configurazione vuota (in cui tutte le variabili sono indipendenti tra loro) e si applicano piccole modifiche cercando di minimizzare il valore della metrica. Queste piccole modifiche possono consistere, ad esempio, nella rimozione o aggiunzione di archi nel grafo o in un cambio della loro direzione.

3.3.2.2 Apprendimento dei parametri

Il numero e la dimensione delle tabelle da apprendere variano in base al numero di classi e alla struttura della rete. Nel caso più generale se V_i può assumere n possibili valori, avremo bisogno di n-1 tabelle (il meno uno deriva dal fatto che avendo n possibili valori, se V_i non assume uno dei primi n-1 valori allora assumerà sicuramente il valore n-esimo quindi l'aggiunta di un ulteriore tabella non fornirebbe nessuna informazione aggiuntiva) per V_i, di seguito però per semplicità considereremo il caso binario in cui ad ogni nodo è associata una sola tabella di probabilità. Per ogni singola tabella, sempre considerando il caso binario, invece il numero di righe sarà pari a 2^{k_i} con k_i che indica il numero di genitori di V_i che di seguito saranno indicati

con il vettore $\boldsymbol{P}_i$.

Inoltre, ci si può trovare di fronte a due situazioni:

- Dati completi;
- dati incompleti;

Nel caso primo caso l'apprendimento può essere effettuato semplicemente nel seguente modo. Per determinare le tabelle di probabilità di un nodo V_i subordinata a quella di un suo genitore P_j, basta usare la seguente formula:

$$P(V_i = v_i | P_j = p_j) = \frac{N_{ij}}{N_j}$$

Dove N_{ij} indica il numero di istanze nell'insieme di addestramento per cui $V_i = v_i$ e $P_j = p_j$, mentre N_j indica il numero di istanze in cui $P_j = p_j$

Possiamo notare che al fine apprendere i parametri, cioè le tabelle di probabilità condizionata della rete bayesiana, è necessario specificare per ciascun nodo X la distribuzione di probabilità per X subordinata ai genitori di X. Questo pone quindi, nel caso di dati mancanti, il problema della necessità di conoscere alcune porzioni delle tabelle per calcolare le tabelle stesse. Un modo per risolvere questa ricorsione infinita è l'utilizzo del metodo della *massimizzazione del valore atteso* (Dempster, Laird, & Rubin, 1997). Tale metodo si può riassumente nei seguenti passi:

1. Inizializzazione dei parametri della rete con valori pseudocasuali
2. Calcolo dei nuovi parametri utilizzando come i parametri precedenti
3. Verifica della convergenza, se i parametri convergono il metodo termina altrimenti continua ad iterare con il passo 2

È dimostrato che tale algoritmo termina sempre e generalmente la sua convergenza è rapida.

Come nel caso degli alberi decisionali anche le reti bayesiane operano in una white box, infatti, l'apprendimento delle relazioni causali oltre a consentire le predizioni, permette anche di

incrementare il grado di comprensione del dominio del problema. Inoltre, le reti Bayesiane presentano un ragionamento di tipo diagnostico simile a quello degli umani che consente di fare predizioni anche riguardo eventi di cui non si hanno informazioni pregresse.

4 Progettazione del sistema proposto

In questo capitolo verrà descritta la progettazione, da un punto di vista funzionale, del sistema ibrido di rilevazione dei malware proposto e dei suoi sottosistemi ovvero il sottosistema di analisi statica e quello dinamico. Per quanto riguarda il sistema di analisi ibrida sono state progettate due varianti che verranno discusse di seguito.

Il primo metodo di analisi ibrida consiste nell'impiego di un algoritmo di apprendimento automatico utilizzato per classificare un vettore di caratteristiche ibrido unico così ottenuto. Per la fase di addestramento si estrarranno features statiche e dinamiche a partire dai file APK associando a tali vettori l'informazione sulla classe di appartenenza. Le feature statiche sono costruite a partire dalla lista delle chiamate API presenti nel codice mentre quelle dinamiche sono costruite utilizzando i dati raccolti sulle API effettivamente chiamate e sul traffico di rete generato. In seguito, i vettori relativi alle analisi statica e dinamica vengono concatenanti sempre mantenendo l'informazione sulla classe, come mostrato di seguito:

$$vettore\ features\ statiche = [s_1, s_2, \dots, s_{N,}\ class]$$

$$vettore\ features\ dinamiche = [d_1, d_2, \dots, d_{M,}\ \ class]$$

$$vettore\ features\ ibride = [s_1, s_2, \dots, s_N, d_1, d_2, \dots, d_{M,}\ class]$$

Figura 15 creazione del vettore ibrido per l'addestramento e la valutazione

Tali vettori vengono salvati in un dataset che sarà utilizzato per l'addestramento e la valutazione dei risultati.
Successivamente, per la predizione con campioni nuovi, il procedimento è lo stesso ma non si ha a disposizione il valore dell'attributo classe:

$$vettore\ features\ statiche = [s_1, s_2, \dots, s_N]$$

$$vettore\ features\ dinamiche = [d_1, d_2, \dots, d_M\ ]$$

$$vettore\ features\ ibride = [s_1, s_2, \dots, s_N, d_1, d_2, \dots, d_M]$$

Figura 16 creazione del vettore ibrido per la predizione

Tale vettore verrà dato in ingresso al modello generato precedentemente per ottenere l'informazione sulla pericolosità del applicazione analizzata.

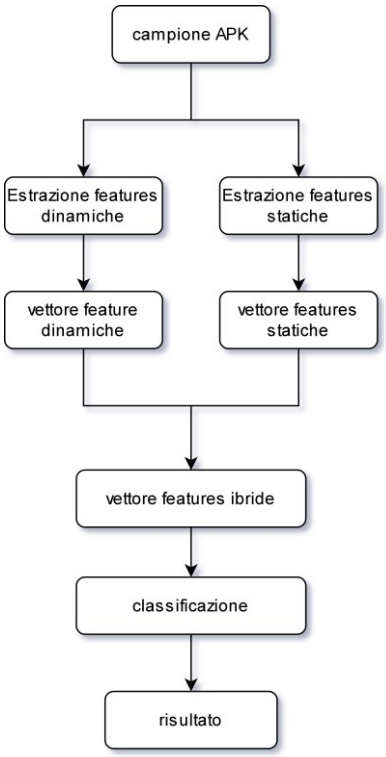

Figura 17 struttura del sistema di analisi ibrida con vettore unico

Il secondo metodo invece classifica separatamente le features statiche e dinamiche per poi utilizzare i risultati di tali classificazioni in un classificatore di tipo stacking ensemble learning. Tale classificatore prevede due livelli: un livello base e un metalivello. Nel livello base sono presenti due classificatori indipendenti, uno a cui vengono date in ingresso le features statiche e l'altro a cui vengono fornite le features dinamiche. Invece nel metalivello è presente il classificatore utilizzato per ottenere il risultato finale al quale vengono forniti in input i dati sulle previsioni dei classificatori base e le relative confidenze che indicano la verosimiglianza delle previsioni.

A differenza del metodo precedente che necessita di una sola classificazione, sebbene con vettori di dimensione maggiore, in questo caso sono richieste tre invocazioni di un algoritmo

di classificazione, due per i classificatori base e uno per il metaclassificatore. Tuttavia, vale la pena evidenziare che il metaclassificatore ha come input un vettore di dimensioni molto ridotte rendendo il suo addestramento e la sua esecuzione molto efficienti. Inoltre, se, come proposto in (A. De Paola, 2018) e discusso nel paragrafo 2.2, si dovesse implementare un sistema che sceglie dinamicamente se effettuare la sola analisi statica o l'analisi ibrida, allora nel caso in cui dopo aver effettuato l'analisi statica sorgesse la necessità di eseguire quella ibrida, i risultati statici già calcolati potrebbero essere riutilizzati per l'ensemble learning. Ciò invece non è possibile con la prima variante di analisi ibrida che, utilizzando un unico classificatore sia per le features statiche che per quelle dinamiche, non è in grado di sfruttare i risultati statici già calcolati. In ogni caso, sebbene il costo dell'analisi ibrida varia a seconda se si utilizza il vettore unico o l'ensemble learning, l'inclusione del meccanismo di analisi dinamica proposto dagli autori di (A. De Paola, 2018) consentirebbe di ridurre il tempo medio di analisi poiché nella maggior parte dei casi si effettua la sola analisi statica.
Di seguito è illustrata la struttura del sistema:

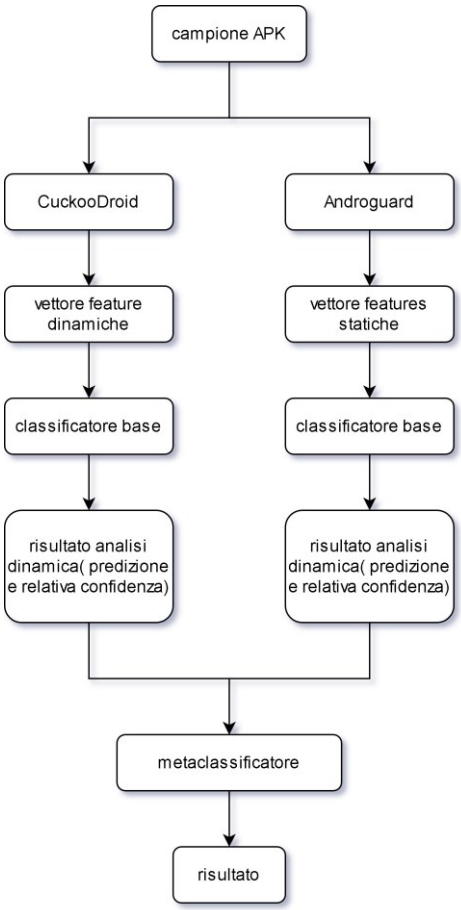

Figura 18 struttura del sistema di analisi ibrida tramite ensemble learning

4.1 Componente statica

Come features statiche è stato scelto di estrarre le chiamate API contenute nel codice delle applicazioni analizzate. Sebbene l'analisi statica sia più semplice ed efficiente di quella dinamica, tale estrazione non è immediata. Per questo motivo ci si è avvalsi della libreria Python Androguard che sarà descritta successivamente nel paragrafo 5.1. Il sistema di estrazione delle features statiche è suddiviso in quattro componenti:

1. estrazione insieme features;
2. selezione features;
3. costruzione vettore delle features
4. creazione dataset delle features statiche.

La componente *estrazione insieme features* estrae tutte le features presenti in ogni elemento del dataset e crea un insieme di tutte le API presenti almeno una volta in uno dei campioni. Viene avviata fornendo in input il dataset delle applicazioni suddiviso in malware e goodware, ed il suo risultato sarà l'insieme di tutte le features presenti in almeno un'istanza del dataset.

La componente di *selezione delle features*, osservato che la cardinalità dell'insieme ottenuto dalla componente *estrazione insieme features* risulta essere troppo elevata (circa 17000 features), prende in input la lista generata al punto precedente ed effettua un filtraggio considerando separatamente le api più frequenti tra i file malevoli e quelli benevoli ed eliminando i duplicati.

La terza componente è costituita da un server REST che riceve in ingresso un file APK e restituisce un vettore di classificazione. Una volta ricevuto il file APK da analizzare, il sistema provvederà a decomprimerlo e ad effettuare un'operazione di reverse engineering in modo da poter estrapolare la lista di API utilizzate, tale lista verrà successivamente confrontata con le caratteristiche precedentemente ottenute con la componente "selezione features" e verrà scritto un vettore binario fine ad indicare lo stato di presenza o meno di una relativa API nel codice disassemblato. Il server REST è stato implementato tramite l'ausilio della libreria Python Flask che consente, tra l'altro, di gestire parallelamente più richieste.

La componente che crea il dataset delle features statiche consiste in un client REST che per

ogni campione effettua una richiesta al server "estrazione features campione" e crea un dataset in formato CSV contenente le informazioni del vettore ritornato dal server e la classe di appartenenza (malevolo o benevolo).

Il dataset ottenuto viene dato in ingresso a WEKA che effettua la classificazione del dataset in modo da ottenere un modello di classificatore e i valori delle metriche di valutazione.

4.2 Componente dinamica

La componente dinamica fa uso di features di tipo comportamentale e di tipo network. Per features comportamentali si intendono le API effettivamente richiamate durante l'esecuzione simulata mentre per features di tipo network si intendono gli host e gli IP contattati e l'informazione sulla loro presenza in una blacklist.

Riguardo le features di rete è necessario precisare che, come evidenziato nel paragrafo 2.2.3, nella configurazione di una sandbox non è possibile per ciò che concerne la rete ottenere simultaneamente la massima precisione e sicurezza, infatti la configurazione un firewall impostato in modo tale da bloccare tutte le connessioni comporta un ambiente meno realistico mentre se si consente una connessione senza restrizioni i malware analizzati possono effettuare alcuni tipi di operazioni malevole nonostante siano eseguiti all'interno di una sandbox. Per questo lavoro di tesi si è privilegiata la sicurezza al costo di una riduzione della precisione, per questo motivo tutte le features di tipo network utilizzate non indicano connessioni ma tentativi di connessione.

Il vettore comportamentale consiste in un vettore binario in cui l'i-esimo elemento è 1 se l'i-esima API è stata richiamata durante l'esecuzione del campione, 0 altrimenti. Nel caso dinamico non essendo il numero delle API (circa 4000) esageratamente alto si è scelto di considerarle tutte senza effettuare una riduzione come invece avvenuto nel sistema statico. Secondo l'autore di questa tesi i motivi della differenza tra la numerosità delle API statiche e quelle dinamiche sono i seguenti:

- Nell'analisi dinamica non viene analizzato l'intero programma ma solo la parte che viene raggiunta dal flusso di esecuzione;
- La mancata connettività ad internet limita il funzionamento di alcuni campioni sia

benevoli che malevoli, che quindi eseguiranno una parte minore del loro codice.

Per ciò che riguarda le features di rete è stato costruito un vettore binario che indica quali destinazioni di rete vengono contattate da un campione, cioè un vettore il cui i-esimo elemento è 1 se il campione ha provato a contattare l'host/IP i-esimo e 0 altrimenti. Inoltre, è stato utilizzato uno script per controllare per ogni indirizzo IP ed host se esso è presente almeno una di un insieme di blacklist. A questo scopo è si è calcolato la percentuale di host/IP blacklistati, tale percentuale è stata convertita in un numero compreso tra 0 e 1 ed è stata utilizzata come feature.

Il sistema di estrazione delle features dinamiche è suddiviso in quattro componenti:

1. Analisi CuckooDroid;
2. estrazione insieme features;
3. verifica appartenenza blacklist
4. creazione dataset delle features dinamiche.

La prima componente consiste nel sistema di analisi dinamica (D-MAS) CuckooDroid e nell'emulatore Android Virtual Device (AVD) così come descritto nel paragrafo 5.2. Tramite uno script Bash vengono mandati in esecuzione, utilizzando l'interfaccia a riga di comando di CuckooDroid, tutti i campioni malevoli e benevoli per ottenere i report in formato JSON contenenti i dati dell'analisi dinamica. Dal punto di vista del tempo impiegato questa è la fase più costosa, infatti, per quanto riguarda l'addestramento iniziale nel caso di questo lavoro di tesi il tempo impiegato è stato di circa due settimane per l'analisi di 8000 file APK.

La componente *estrazione insieme features* consiste in due script Python, uno per le features comportamentali e uno per le features di rete, che utilizzando i report prodotti da CuckooDroid producono gli insiemi di tutte le API chiamate e gli indirizzi contattati almeno una volta da un qualsiasi campione del dataset.

La componente *verifica appartenenza blacklist* verifica per ogni host e indirizzo IP se è presente in almeno una di uno serie di web services di blacklist utilizzati. Tali servizi vengono richiamati tramite una REST API ed indicano se l'indirizzo inviato è stato segnalato come

malevolo. Il prodotto di tale componente sarà un sottoinsieme degli indirizzi contattati in cui sono presenti solamente quelli inseriti in almeno una delle blacklist considerate. Per eseguire tale controllo ci si avvalsi di un apposito script Python opensource (Vallecillo, 2017).

La componente di *creazione del dataset* delle features dinamiche prende in input gli insiemi generati dalla componente *estrazione insieme features*, i sottoinsiemi di indirizzi in blacklist e i file di report suddivisi in quelli relativi ai goodware e quelli relativi ai malware e produce come risultato un file CSV contenente le features dinamiche e l'informazione sulla classe. Il vettore delle features dinamiche può consistere in:

1. Vettore binario contenente l'informazione su quali api sono state chiamate e quali no:

$$[api_1, api_2, \dots, api_N, class] \qquad api_i, class \in \{0,1\}$$

2. Vettore del punto 1 con l'aggiunta del vettore binario con le destinazioni di rete contattate:

$$[api_1, api_2, \dots, api_N, IP_1, IP_2, \dots, IP_M, class] \qquad api_i, IP_j, class \in \{0,1\}$$

3. Vettore del punto 1 con l'aggiunta dell'informazione aggregata sulla percentuale di IP presenti in blacklist:

$$[api_1, api_2, \dots, api_N, bl, class] \quad api_i, class \in \{0,1\} \quad bl = \frac{\#indirizzi\ blacklistati}{\#indirizzi\ totali}$$

Infine, si effettua la classificazione, tramite WEKA, del dataset in modo da ottenere un modello di classificatore e i valori delle metriche di valutazione.

4.3 Sistema ibrido

Di seguito saranno descritti i vettori delle caratteristiche utilizzati nelle due varianti di analisi ibrida introdotte precedentemente:

- vettore ibrido
- ensemble learning

4.3.1 Vettore ibrido

L'approccio del vettore ibrido consiste nel partire da vettori contenenti features statiche e dinamiche e generare un terzo vettore ibrido tramite la loro concatenazione. In quest'approccio l'algoritmo di classificazione viene eseguito una sola volta sul vettore ibrido.
I vettori ibridi utilizzati sono:

1. Vettore statico API con l'aggiunta del vettore equivalente dinamico contente l'informazione binaria sulle API effettivamente richiamate durante l'esecuzione:

$$\left[api_{s,1}, api_{s,2}, \dots, api_{s,N}, api_{d,1}, api_{d,2}, \dots, api_{d,N} class\right] \quad api_{s,i}, api_{d,j}, class \in \{0,1\}$$

2. Vettore 1 con l'aggiunta dell'informazione aggregata sulla percentuale di indirizzi IP in blacklist:

$$\left[api_{s,1}, api_{s,2}, \dots, api_{s,N}, api_{d,1}, api_{d,2}, \dots, api_{d,N}, bl, class\right]$$

$$api_{s,i}, api_{d,j}, class \in \{0,1\} \qquad bl = \frac{\#indirizzi\ blacklistati}{\#indirizzi\ totali}$$

4.3.2 Ensemble learning

L'ensemble learning consiste nel combinare più classificatori in modo da ottenere risultati più accurati rispetto all'utilizzo di un solo classificatore. Per quanto riguarda l'ensemble learning l'architettura impiegata è stata quella stacking (chiamata anche stacking generalization) che consiste in più livelli di classificazione, con il primo livello che ha come input i dati effettivi e

i livelli successivi invece hanno come input le predizioni del livello precedente. I classificatori del primo livello sono chiamati classificatori base mentre quelli dei livelli successivi sono chiamati meta-classificatori poiché classificano avendo in input altri classificatori. Ovviamente i classificatori base devono produrre dei risultati scorrelati affiche il meta-classificatore ne tragga vantaggio, questa variazione può essere introdotta tramite:

1. Variazione delle tipologie di classificatori
2. Variazione dei parametri dei classificatori
3. Variazione degli input dei classificatori
4. Combinazione dei precedenti

In questo caso si è scelta la terza opzione, approfittando del fatto che abbiamo a disposizione i due punti di vista statico e dinamico, utilizzando due livelli. Il primo è costituito da due classificatori uno addestrato con le features statiche e l'altro con quelle dinamiche. Il secondo livello, detto metalivello, invece utilizza come features i risultati della classificazione del livello sottostante.

4.4 Classificazione

Gli algoritmi di classificazione utilizzati in questo lavoro di tesi, per ciò che riguarda il vettore unico, sono:

- Albero decisionale
- Random Forest
- Reti Bayesiane

Invece per ciò che riguarda l'ensemble learning è stato impiegato Random Forest per il livello base mentre per il metalivello sono stati impiegati gli alberi decisionali e le reti Bayesiane.
Per la classificazione è stato utilizzato il framework WEKA (The University of Waikato, 2020) che sarà descritto nel paragrafo 5.3 e le relative API per il linguaggio di programmazione JAVA. Per quanto riguarda l'ensemble learning di tipo stacking generalization WEKA dispone di classificatore apposito che tuttavia non era adatto agli scopi di questo lavoro di tesi poiché consentiva di utilizzare features differenti nei classificatori base. Per questo motivo è stata creata una variante del classificatore stacking estendendo la classe Java di quello già esistente in WEKA (*weka.classifiers.meta.Stacking*).

5 Ambiente di sviluppo

Di seguito verranno descritti i software e framework utilizzati per le componenti di analisi statica e analisi dinamica descritte nei paragrafi 4.1 e 4.2 e per la classificazione. Per ciò che riguarda la gestione dell'analisi ibrida, invece, è stata implementata senza basarsi su sistemi preesistenti. L'addestramento e la valutazione dei risultati sono stati implementati in Java estendendo una classe di WEKA, mentre per ciò che riguarda la successiva predizione su istanze nuove è stato scritto un programma Python che necessita comunque accedere alle API Java di WEKA tramite un server REST appositamente configurato.

5.1 Analisi statica

L'estrazione delle chiamate ad API presenti nel codice delle applicazioni si compone di una fase di preprocessing del file apk e nell'estrazione vera e propria. Nella fase di preprocessing il file APK compresso viene spacchettato ottenendo i file DEX, ODEX e le altre risorse utilizzate dall'applicazione, ad esempio, file xml o file multimediali. Per i fini dell'estrazione delle API i file di interesse sono esclusivamente quelli DEX e ODEX che contengono le classi compilate rispettivamente in versione non ottimizzata e ottimizzata. A partire da tali file è possibile risalire ad *un* codice sorgente Java dell'applicazione. Tale sorgente non coincide esattamente con quello effettivamente scritto dai programmatori dell'applicazione ma è comunque sufficiente per consentire l'estrazione della lista di API.
Per tali elaborazioni è stato utilizzato il tool Androguard, che a sua volta richiama DAD(Dad is A Decompiler) per la decompilazione dei file DEX/ODEX. Androguard è utilizzabile sia tramite interfaccia a linea di comando che come libreria Python. Per gli scopi di questo progetto si è scelto di utilizzare la libreria Python in modo da ottenere una maggiore flessibilità nell'utilizzo.

5.2 Analisi dinamica

Di seguito verranno presentati i principali software utilizzati per l'analisi dinamica. Verrà descritto, in particolare CuckooDroid, un Dynamic Malware Analisys System (D-MAS) per il sistema operativo mobile Android, e AVD, software necessario per creare l'ambiente virtuale

utilizzato da CuckooDroid per eseguire i malware in sicurezza.

5.2.1 CuckooDroid

CuckooDroid è un'estensione per il sistema operativo mobile Android di *Cuckoo Sandbox*, un software opensource utilizzato per effettuare l'analisi automatica di file sospetti su sistemi desktop Windows. È possibile eseguire qualsiasi file sospetto all'interno di una sandbox e ottenere un report dettagliato che descrive il comportamento dell'eseguibile.
Grazie sua natura opensource e alla sua strutturazione in moduli, tutti scritti in linguaggio Python, è possibile, inoltre, personalizzare qualsiasi aspetto dell'ambiente di analisi, l'elaborazione dei risultati dell'analisi e la fase di reporting.
Ad esempio, è possibile scegliere il tipo di sandbox da utilizzare tra tre configurazioni:

1. La macchina guest consiste in una macchina virtuale Linux all'interno della quale viene emulato un dispositivo virtuale Android tramite il sistema di virtualizzazione ufficiale per Android AVD (Android Virtual Device)

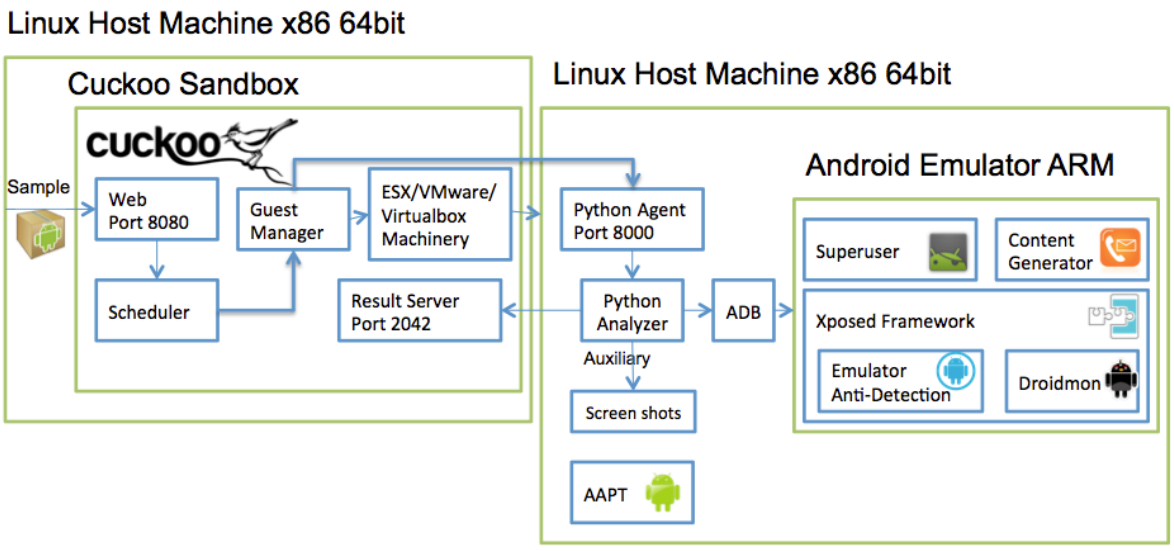

Figura 19 configurazione di CuckooDroid con AVD eseguito dentro una macchina virtuale Linux, tratta dalla documentazione ufficiale di CuckooDroid (Guest Machine Architecture, 2015)

2. La macchina guest consiste in un'istanza di AVD senza la necessità di un ulteriore livello di virtualizzazione;

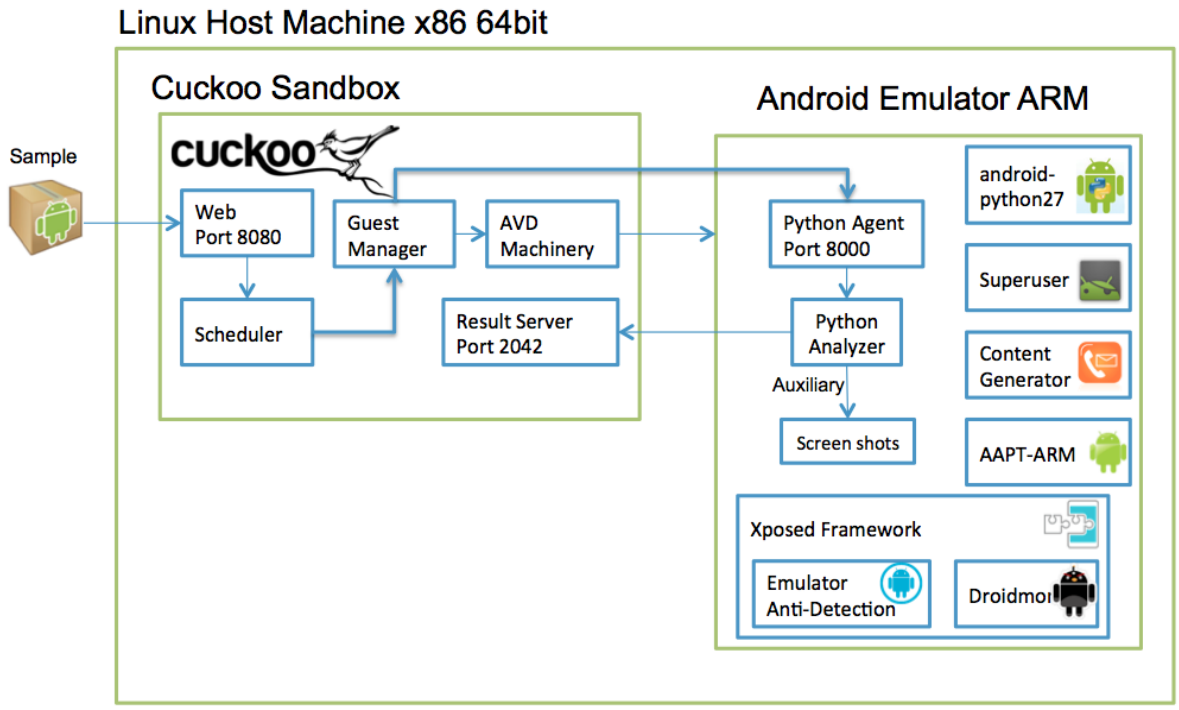

Figura 20 rappresentazione grafica della configurazione scelta, tratta dalla documentazione ufficiale di CuckooDroid (Guest Machine Architecture, 2015)

3. La macchina guest consiste in una macchina virtuale con un sistema operativo Android, utilizzando un software per la generazione di macchine virtuali, per esempio VirtualBox.

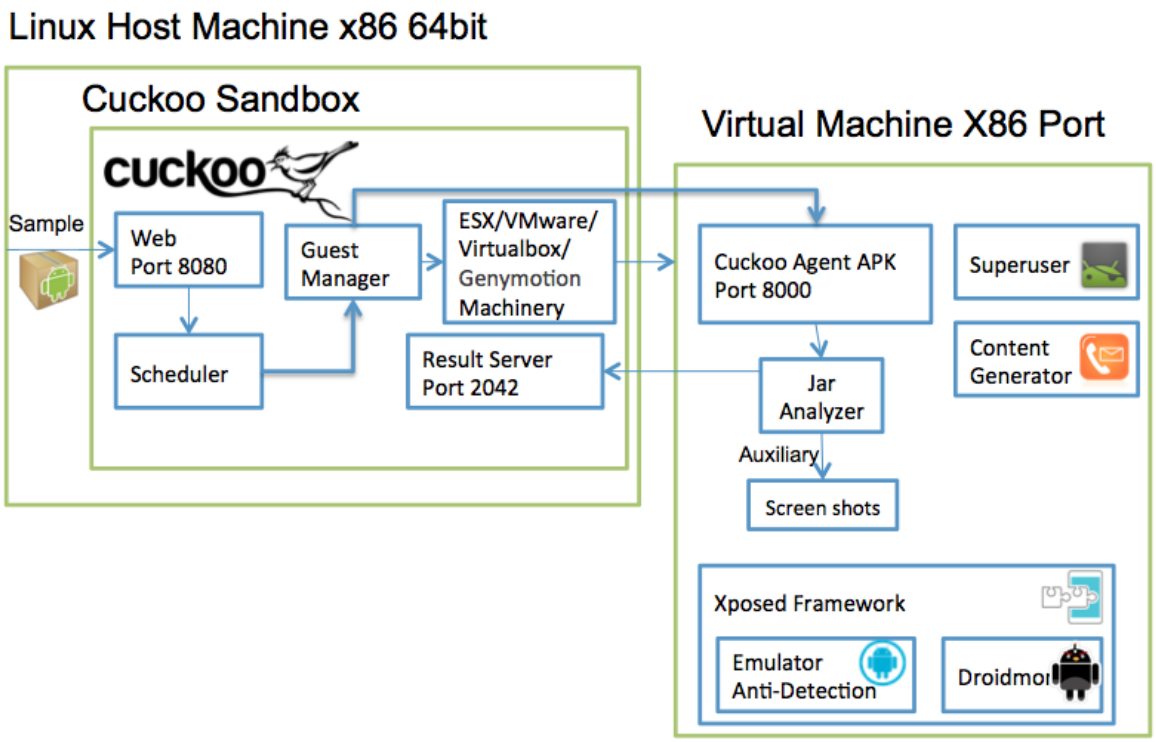

Figura 21 configurazione di CuckooDroid che non fa uso di AVD, tratta dalla documentazione ufficiale di CuckooDroid (Guest Machine Architecture, 2015)

Per gli scopi di questo lavoro di tesi si è scelta la configurazione che utilizza solamente AVD perché è la più semplice ed evita l'ulteriore costo computazionale dovuto ad una doppia virtualizzazione.

CuckooDroid è costituito da una componente host, ovvero il software centrale che gestisce l'esecuzione dei campioni e la raccolta delle informazioni per l'analisi, e da una componente guest, all'interno della quale vengono eseguiti i campioni software da analizzare. Ogni volta che la componente host avvia una nuova analisi, viene avviata un'istanza della macchina guest e viene caricato il campione da analizzare, il quale avvia tutte le componenti necessarie per l'analisi.
È possibile inoltrare il campione da analizzare sia tramite una interfaccia web-based sia da riga di comando in modo da automatizzare il processo di analisi.

5.2.1.1 Macchina Host

Quando CuckooDroid viene avviato, vengono inizializzati i suoi moduli abilitati e le configurazioni necessarie per effettuare l'analisi e viene avviato il ResultServer che si occupa di recuperare i risultati delle analisi e di eseguirne la post-elaborazione (per esempio la conversione da JSON a HTML).
Ogni volta che viene inviato un nuovo task a CuckooDroid, esso identifica automaticamente il tipo di file (a meno che non venga specificato manualmente) e avvia l'analizzatore all'interno dell'ambiente virtuale Android con apposito modulo per quel tipo di file.

5.2.1.2 Macchina Guest

La macchina virtualizzata in cui avviene l'analisi è costituita da:

- l'agent, un server che ascolta su una porta nella macchina guest in attesa della ricezione del app da analizzare;
- il sistema di monitoraggio, utilizzato per registrare il comportamento dell'eseguibile durante l'esecuzione simulata.

Quando viene avviata una nuova analisi nella macchina guest, l'host invia il campione da analizzare assieme ai parametri dell'analisi. L'analisi viene interrotta quando il campione termina la sua esecuzione o viene raggiunto un timeout (uno dei parametri che si possono specificare per l'analisi). A questo punto l'host raccoglie i risultati e li invia al ResultServer che genererà il report JSON e se specificato nella configurazione anche quello HTML. In questo caso il report HTML conterà le stesse informazioni di quello JSON ma a causa della struttura dei file html la loro dimensione sarà di molto superiore rispetto ai corrispondenti file JSON. Per questo motivo, dovendo analizzare un elevato numero di file APK, si è scelto di disattivare la generazione dei report in formato HTML nella configurazione di CuckooDroid.

5.2.1.3 Reports

I risultati dell'analisi di CuckooDroid vengono salvati principalmente in un file JSON, chiamato *report.json* e in parte anche nel file log di del framework droidmon. Questo file contiene informazioni sia relative all'analisi dinamica sia all'analisi statica effettuata sul

malware nell'ambiente "sandbox".
La tipica struttura di un file di report è la seguente:

- info: questa sezione contiene i metadati relative all'analisi come la data e l'ora di inizio e fine, la versione di cuckooDroid e il tipo di VM usati per l'analisi;
- procmemory: questa sezione è non vuota solo se nella configurazione di CuckooDroid è abilitata il dump della memoria RAM in questo caso tale sezione contiene proprio le informazioni sul contenuto della memoria relativa all'applicazione analizzata registrate tramite il tool *Volatility;*
- **network:** contiene i dati registrati riguardo al traffico di rete in entrata e in uscita dalla sandbox;
- virustotal: contiene i risultati dell'analisi statica effettuata tramite il web service VirusTotal, tale servizio consente di analizzare file di qualsiasi tramite più di settanta diversi antimalware riportando sia i risultati di ogni singola analisi che l'informazione aggregata sul numero di risultati positivi. Nel caso in cui CuckooDroid non riesca a raggiungere il web service, ad esempio, perché bloccato da un firewall, tale sezione sarà vuota;
- **droidmon:** questa sezione è l'equivalente CuckooDroid della sezione behaviour di Cuckoo(analisi su Windows);
- signatures: indica se l'applicazione presenta dei comportamenti sospetti come richiedere i permessi di amministratore o accedere alla lista di applicazioni installate;
- static: in Cuckoo contiene ulteriori dati inerenti all'analisi statica ma in CuckooDroid per impostazione predefinita questa sezione è vuota;
- apkinfo: contiene le informazioni relative ai metadati presenti all'interno del file APK analizzato, ad esempio, la lista dei permessi richiesti dall'applicazione. L'estrazione dei metadati è eseguita con l'ausilio della libreria apkinfo;
- behaviour: in Cuckoo contiene le informazioni relative alla parte comportamentale dell'applicazione analizzata, qui risulta essere vuota, poiché le informazioni riguardanti questo aspetto vengono fornite nella sezione *"droidmon"* e nel file *droidmon.log*, facendo uso dell'omonimo framework di monitoraggio;
- target: questa sezione riporta i metadati relativi file analizzato tra cui nome, path, dimensione e il digest di vari algoritmi di hashing;
- debug: qui sono registrati gli eventi relativi al normale funzionamento di CuckooDroid

o eventuali errori;

- strings: contiene la lista delle stringhe stampabili presenti nel codice che può essere utilizzata per effettuare un analisi statica tramite strumenti di pattern matching come YARA;
- dropped: lista degli eventuali file scaricati dal applicazione durante la sua esecuzione, tali file vengono salvati da CuckooDroid in un'apposita cartella.

Le informazioni considerate per le estrazioni delle caratteristiche sono quelle relative al *comportamento* del malware e quelle relative all'aspetto *network*.
La sezione droidmon contiene tutte le informazioni del file analizzato dal punto di vista comportamentale. Essa contiene al suo interno varie sottosezioni dove vengono raccolte diverse informazioni come, ad esempio, le api richiamate dal file durante la sua esecuzione, i "data leak" ovvero i trasferimenti di dati non autorizzati, le "reflected API" ovvero le API call che non vengono rilevate come api e che cercano di nascondere aspetti del comportamento del file analizzato. Le chiamate API sono particolarmente importanti poiché, secondo il modello di sicurezza Android, sono l'unica via per un'applicazione per interagire con il sistema.
La sezione "network", invece, contiene le informazioni riguardo al traffico di rete. Essa contiene al suo interno diverse sottosezioni, relative ai diversi tipi di richieste effettuate dal malware durante la sua esecuzione, ad esempio, informazioni riguardo ai protocolli di rete UDP, TCP, HTTP, SMTP, ICMP, DNS, e le porte utilizzate dai vari protocolli di comunicazione.

5.2.2 Android Virtual Device

AVD è un emulatore per Android che fa parte dell'ambiente di sviluppo ufficiale per Android chiamato Android Studio. Si tratta di un emulatore completo che non si limita a consentire l'esecuzione delle app ma anche dell'intero sistema operativo. A tale scopo è possibile personalizzare sia le caratteristiche hardware del sistema utilizzato che la versione del sistema operativo da eseguire.

Al fine di consentire il corretto funzionamento di CuckooDroid, come scritto in precedenza, è necessario che nel sistema virtuale sia installato un agente scritto in Python. Questo agente ha il compito di dover monitorare il comportamento di altre applicazioni in esecuzione, operazione non consentita dal modello di sicurezza adottato da Android ed in generale da tutti i sistemi

mobili. Per questo motivo il sistema Android da utilizzare con CuckooDroid è stato modificato in modo da abilitare i permessi di root[12] e garantire all'agente l'accesso completo al sistema. Inoltre, è stato necessario modificare il codice Python di CuckooDroid che avvia l'analisi in modo da renderlo compatibile con le recenti versioni di AVD.
Per effettuare le analisi e rendere la sandbox riutilizzabile, CuckooDroid invece che creare uno snapshot del sistema virtualizzato come avviene in Cuckoo, crea una copia temporanea della macchina virtuale da utilizzare per l'analisi. Questo si traduce semplicemente nel creare una copia della cartella che contiene i dati della VM. Non appena il sistema termina di analizzare file la copia temporanea viene eliminata, ripristinando lo stato del sistema e preparando la macchina virtuale per l'analisi successiva.

5.3 Classificazione

Per la classificazione si è scelto di utilizzare il framework WEKA. Weka (Waikato Environment for Knowledge Analysis) è un software open source per l'apprendimento automatico scritto in Java e sviluppato nell'Università di Waikato in Nuova Zelanda, che raccoglie numerosi algoritmi di apprendimento automatico e contiene strumenti per la preparazione, la classificazione, la regressione, il clustering e il mining dei dati. La scelta è ricaduta su tale framework perché si tratta di un software semplice da utilizzare che comprende molti algoritmi di classificazione ed è estensibile con classificatori scritti appositamente. Weka accetta come dataset di input una tabella in cui le istanze corrispondono alle righe e gli attributi selezionati alle colonne. Il formato utilizzato in Weka per la lettura dei dataset è il formato ARFF (Attribute Relationship File Format), simile al formato CSV (Comma Separated Value), per il quale esiste un convertitore bidirezionale incluso in WEKA.
Inoltre, è possibile aggiungere altri classificatori sia creandoli da zero che basandosi su classificatori esistenti estendendo la relativa classe Java. Il vantaggio di scrivere un classificatore WEKA anche se non si estende un classificatore già esistente consiste nella possibilità di utilizzare i metodi di valutazione dei risultati di WEKA, in modo da evitare che un'implementazione non corretta del sistema di valutazione possa dare dei risultati fuorvianti.

[12] I permessi di root sono il massimo livello di autorizzazione che un applicazione può ricevere su Android ma per motivi di sicurezza tale possibilità è disabilitata per impostazione predefinita su tutti gli smartphone Android. Sebbene in ambito Linux spesso si utilizzano root e amministratore come sinonimi, su Android i privilegi di amministratore sono di livello inferiore rispetto a quelli di root e per tale motivo non sono disabilitati.

WEKA può essere utilizzato tramite:

- Interfaccia grafica(GUI): la GUI è il modo più semplice per utilizzare WEKA, per esempio, si possono caricare file .arff oppure file .csv che verranno convertiti automaticamente, filtrare i dati, effettuare la classificazione, visualizzare i risultati e salvare il file contenente il modello del classificatore addestrato;
- Interfaccia a riga di comando *SimpleCLI*: tale interfaccia consente di eseguire WEKA anche su sistemi sprovvisti di interfaccia grafica e in parte di automatizzare il processo di classificazione. Tuttavia, le funzionalità fornite da *SimpleCLI* sono molto ridotte addirittura minori di quelle offerte dalla GUI;
- API Java: Weka consente di richiamare delle API all'interno di programmi scritti in java in modo da avere maggiore flessibilità e poter automatizzare tutti le elaborazioni supportate.

WEKA mette anche a disposizione diverse metodologie per la valutazione dei risultati, come la tecnica dello Split Percentuale, in cui viene scelta, appunto, una percentuale per scindere il dataset in una frazione per il Training Set e la restante parte per Test Set, oppure è possibile indicare manualmente un apposito dataset come Training Set e un altro come Test Set.
Inoltre, è possibile utilizzare la tecnica k-cross-validation. La k-cross-suddivide il dataset in k parti di uguale dimensione dove, ad ogni passo, la k-esima parte del dataset viene utilizzata come insieme di validazione, mentre la restante parte verrà utilizzata come insieme di addestramento. In questo modo si esclude iterativamente un gruppo alla volta e lo si cerca di predire con i gruppi non esclusi evitando, quindi, sovrastime nella valutazione dovute all'overfitting.

6 Valutazione sperimentale

6.1 Dataset utilizzato

Per effettuare l'addestramento dei classificatori e valutare i risultati della classificazione è stato utilizzato un dataset di applicazioni Android annotate con l'indicazione esplicita sulla natura malevola o benigna di ogni applicazione. In questo progetto sono state utilizzati 8000 applicazioni, di cui 4000 files .apk malevoli e 4000 files .apk benigni.
Le applicazioni malevole presenti nel dataset provengono principalmente da una collezione di malware presenti in una repository di *VirusShare* (virusshare, s.d.) e, in parte, sono state estratte dai dataset Genome Project (Android Malware Genome Project , s.d.), Drebin (Drebin, s.d.) , Pwnzen Infotech Inc (kuafuDet, s.d.), e Contagio Mobile (Contagio Mobile, s.d.). Invece i file .apk benigni, considerando che lo store ufficiale di Android non consente di scaricare i file apk direttamente, sono stati estratti dal dataset Androzoo (Androzoo, s.d.), che include file .apk di vari marketplace, tra cui Google Play Store, App China, FDroid, ecc.

Al fine di valutare l'effettiva bontà della classificazione su dati diversi da quelli utilizzati per costruire il modello, evitando sovrastime dovute all'overfitting è stata utilizzata la tecnica di convalida incrociata o cross-validation. In particolare, è stata utilizzata la tecnica k-cross-validation, con k uguale a 10. Questa tecnica divide i dati casualmente i dati di addestramento in k insiemi per ognuno dei k insiemi effettua la generazione del modello utilizzando i restanti k-1 insiemi e lo valuta sull'insieme selezionato. In questo modo vengono effettuate k valutazioni tutte con dati differenti da quelli utilizzati per l'addestramento ed infine per ottenere la valutazione finale si calcola la media delle k valutazioni. Sebbene la k-fold validation è un utile strumento al fine di ottenere valutazioni accurate, essa non è adatta per generare il modello da utilizzare successivamente per valutare istanze non contrassegnate con la relativa classe. Questo deriva dal fatto che nei k modelli generati non viene utilizzato l'intero dataset per il loro addestramento riducendo potenzialmente la loro efficacia, inoltre, per ottenere una singola predizione sarebbe necessario eseguire k classificatori ed implementare un sistema di votazione. Per questi motivi, anche quando si impiega la k-fold validation, viene generato un modello addestrato sul totale dei campioni contrassegnati disponibili, tuttavia, tale modello non è da utilizzarsi per la valutazione dei risultati ma solamente per invocare successivamente il classificatore su nuove istanze non contrassegnate.

6.2 Metriche valutate

In questo paragrafo, verranno descritte le metriche utilizzate per la valutazione dei risultati della classificazione, le quali sono state estratte dai report di classificazione di Weka.
Nei report vengono indicati il numero di campioni classificati correttamente e il numero di campioni classificati in modo scorretto sia per la classe goodware che per la classe malware. Tali valori costituiscono la *matrice di confusione.*
La matrice di confusione è, nel caso più generale di classificazione con N classi, una matrice quadrata di ordine N, in cui il generico elemento nella riga i e colonna j indica il numero di campioni la cui classe effettiva è l'i-esima mentre quella predetta è la j-esima. Di conseguenza le predizioni corrette saranno solamente quelle nella diagonale principale della matrice. Nel caso binario in cui sono presenti solamente due classi, la matrice di confusione è una matrice quadrata di ordine due i cui elementi sono chiamati:

True positives(TP):
numero di istanze predette come malware che sono realmente malware.

False positives(FP):
numero di istanze predette malware che in realtà sono goodware.

True negatives(TN):
numero di esempi predetti goodware che sono veramente goodware.

False negatives(FN):
numero di esempi predetti goodware che in realtà sono malware.

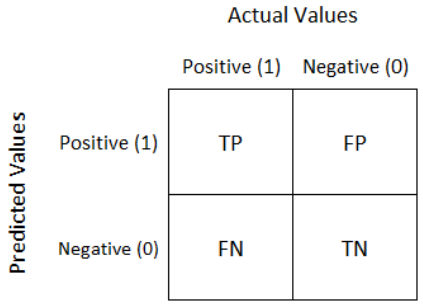

Figura 22 matrice di confusione nel caso di classificazione binaria

A partire da questi valori, vengono calcolate la seguenti metriche prese da (Hossin, 2015) : TP Rate**,** FP Rate**,** precisione, accuratezza e F-Measure.
Ad eccezione di FP rate, tutte le metriche considerate indicano risultati migliori all'aumentare del loro valore.

TP Rate (True Positive Rate): rappresenta il tasso di malware identificati correttamente come tali, è chiamata anche *recall*.

$$\textbf{TP Rate} = \frac{TP}{TP+FN}$$

FP Rate (False Positive Rate): indica la frazione di goodware correttamente indentificate come benevole.

$$\textbf{FP Rate} = \frac{TN}{TN+FP}$$

Precisione: indica l'aliquota di campioni predetti positivi(malware) che sono effettivamente positivi.

$$\textbf{Precision} = \frac{TP}{TP+FP}$$

Accuratezza : si calcola come il rapporto tra il numero di predizioni corrette diviso il numero totale di predizioni, a prescindere dal fatto che la predizione sia stata malware o goodware.

$$Accuracy = \frac{TP+TN}{TP+FP+TN+FN}$$

F-Measure: metrica ottenuta combinando precisione e recall tramite la media armonica, tale metrica è considerata particolarmente efficace per la valutazione di classificatori binari (Joshi, 2002).

$$\textbf{F} - \textbf{Measure} = 2 * \frac{Precisione * Recall}{Precisione + Recall}$$

6.3 Risultati

Essendo lo scopo di questo lavoro di tesi la progettazione di un sistema di analisi dinamica di malware, costituito da una componente statica e una dinamica, di seguito verranno confrontati i risultati ottenuti dalla sola analisi statica e dinamica con quelli delle due proposte di analisi ibrida.

6.3.1 Analisi statica

Per l'analisi statica, le features prese in considerazione sono le chiamate ad API presenti nel codice. Di seguito sono mostrati i risultati suddivisi per metodo di classificazione utilizzato.
Il numero delle API estratte dal codice è risultato essere troppo elevato per essere elaborato quindi è stato effettuato un filtraggio che ha tenuto conto delle api più richiamate sia dai benevoli che malevoli. I risultati ottenuti sono stati i seguenti:

Tabella 2 risultati dell'analisi statica

Classificatore	Accuracy	TP Rate	FP Rate	Precision	F-Measure
Albero decisionale	0.826	0.826	0.176	0.826	0.826
Reti bayesiane	0.658	0.658	0.332	0.67	0.658
Random Forest	0.858	0.858	0.139	0.86	0.858

6.3.2 Analisi dinamica

Nel caso dell'analisi dinamica sono state utilizzate features di tipo comportamentale e di tipo network. Dal punto di vista comportamentale sono state utilizzate le API effettivamente chiamate durante l'esecuzione dei campioni, invece per la componente network sono state analizzati gli host contattati tramite i protocolli di rete TCP, UDP, HTTP, e le richieste di risoluzione degli hostname.
A differenza dell'analisi statica si è notato un più ridotto di features probabilmente dovuto al fatto che sono considerate solo le API effettivamente chiamate rispetto a tutte le API presenti nel codice. Per questo motivo si è scelto di utilizzare tutte le features senza prefiltraggio. Inoltre, in sostituzione del vettore degli indirizzi IP e host contattati è stata impiegata una feature aggregata calcolata tramite il rapporto tra il numero di destinazioni presenti in una

almeno una blacklist diviso il numero complessivo di destinazioni contattate.

Quindi i vettori utilizzati per l'analisi dinamica sono:

1. Vettore binario contenente l'informazione su quali api sono state chiamate e quali no;
2. Vettore del punto 1 con l'aggiunta del vettore binario con le destinazioni di rete contattate;
3. Vettore del punto 1 con l'aggiunta dell'informazione aggregata sulla percentuale di IP presenti in blacklist.

Altri vettori sono stati valutati e successivamente scartati, per esempio, si è visto che le informazioni di rete da sole non fornivano risultati significativamente superiori rispetto ad un classificatore che ritorna un valore costante e quindi tali vettori sono stati scartati. Inoltre, si è valutato di aggiungere al secondo vettore l'informazione sulla percentuale di IP blacklistati ma ciò non ha portato ad un miglioramento dei risultati.

Di seguito sono indicati i risultati ottenuti con il vettore delle sole API chiamate:

Tabella 3 risultati analisi dinamica con vettore API

Classificatore	Accuracy	TP Rate	FP Rate	Precision	F-Measure
Albero decisionale	0.813	0.813	0.19	0.814	0.813
Reti bayesiane	0.741	0.741	0.246	0.769	0.741
Random Forest	0.831	0.831	0.172	0.831	0.831

La seguente tabella, invece, indica i risultati ottenuti aggiungendo al vettore usato precedentemente l'informazione binaria sugli IP e host contattati:

Tabella 4 risultati analisi con dinamica con vettore API e destinazioni di rete contattate

Classificatore	Accuracy	TP Rate	FP Rate	Precision	F-Measure
Albero decisionale	0.813	0.813	0.19	0.814	0.813
Reti bayesiane	0.741	0.741	0.246	0.769	0.741
Random Forest	0.831	0.831	0.171	0.831	0.831

Infine, sono elencati i risultati della classificazione utilizzando invece del vettore di rete binario il rapporto tra indirizzi in blacklist diviso il totale:

Tabella 5 risultati analisi dinamica con vettore API e percentuale blacklist

Classificatore	Accuracy	TP Rate	FP Rate	Precision	F-Measure
Albero decisionale	0.813	0.813	0.19	0.814	0.813
Reti bayesiane	0.741	0.741	0.246	0.769	0.741
Random Forest	0.831	0.831	0.172	0.831	0.831

Si può notare che l'aggiunta di informazioni sul traffico di rete non ha comportato un miglioramento apprezzabile delle prestazioni dell'analisi ibrida, ciò è quasi sicuramente dovuto alle limitazioni della connettività discusse nel paragrafo 4.2.

6.3.3 Analisi ibrida

Per l'analisi ibrida sono state valutate due tecniche; la prima consiste nel creare un vettore unico contenente sia le features statiche che quelle dinamiche, la seconda consiste invece nell'utilizzo di tecniche di stacking ensemble learning in cui si effettua una classificazione su due livelli:

- Nel primo livello le features statiche e dinamiche vengono classificate indipendentemente
- Nel secondo livello vengono utilizzati i risultati del primo livello, comprendenti sia la predizione che la stima sulla relativa confidenza, per effettuare una seconda classificazione

6.3.3.1 Classificazione con vettore unico

I vettori considerati per la classificazione sono:

1. Vettore statico API + vettore equivalente dinamico contente l'informazione binaria sulle API effettivamente richiamate durante l'esecuzione;
2. Vettore 1 con l'aggiunta dell'informazione aggregata sulla percentuale di indirizzi IP in blacklist.

Per quanto riguarda il primo vettore i risultati suddivisi per metodo di classificazione sono stati i seguenti:

Tabella 6 risultati analisi ibrida utilizzando come features le API estratte dall'analisi statica e dall'analisi dinamica

Classificatore	Accuracy	TP Rate	FP Rate	Precision	F-Measure
Albero decisionale	0.836	0.836	0.167	0.837	0.836
Reti bayesiane	0.785	0.785	0.23	0.799	0.785
Random Forest	0.864	0.864	0.141	0.865	0.864

Invece con il secondo vettore si sono ottenuti i seguenti risultati:

Tabella 7 risultati analisi ibrida utilizzando come features le API estratte dall'analisi statica e dall'analisi dinamica, e la percentuale di indirizzi IP in blacklist

Classificatore	Accuracy	TP Rate	FP Rate	Precision	F-Measure
Albero decisionale	0.836	0.836	0.167	0.837	0.836
Reti bayesiane	0.785	0.785	0.23	0.799	0.785
Random Forest	0.867	0.867	0.138	0.868	0.867

6.3.3.2 Ensemble learning

Per quanto riguarda l'ensemble learning l'architettura impiegata è stata quella di tipo stacking. In questo caso si è scelto di utilizzare due livelli, il primo costituito da due random forest una addestrata con le features statiche e l'altra con quelle dinamiche. La scelta del classificatore di primo livello è ricaduta sulla random forest poiché si tratta del classificatore che ha fatto ottenere i risultati migliori in tutti i casi visti precedentemente. Per il secondo livello invece sono stati considerati i seguenti classificatori:

- Albero decisionale
- Reti bayesiane

Si è scelto di non utilizzare la Random Forest come meta-classificatore (classificatore di secondo livello) poiché l'ingresso del secondo livello, contenendo un esclusivamente le predizioni del livello inferiore, risulta avere un numero esiguo di features mentre il classificatore Random Forest, che esso stesso un classificatore di tipo ensemble learning, è indicato principalmente per dataset con un numero elevato di caratteristiche.
Le features utilizzate per la componente dinamica consistono nel vettore binario delle API

richiamate. È stata testata anche l'aggiunta al vettore di informazioni riguardo alla traffico di rete ma non si sono ottenuti miglioramenti, quindi, tale vettore è stato scartato.
I risultati ottenuti sono riepilogati nella seguente tabella:

Tabella 8 risultati analisi ibrida con ensemble learning

Metaclassificatore	Accuracy	TP Rate	FP Rate	Precision	F-Measure
Albero decisionale	0.866	0.866	0.14	0.868	0.866
Reti Bayesiane	0.865	0.865	0.141	0.868	0.865

Dall'analisi dei risultati si è notato che il classificatore che ha performato meglio in ogni caso è stato Random Forest applicato ad un vettore di features unitario che tiene conto sia delle API che degli indirizzi IP invocati durante l'esecuzione del codice, tale risultato non è inaspettato poiché le foreste casuali nascono proprio per gestire vettori di grandi dimensioni come quelli qui utilizzati. Inoltre, si è notato che, sebbene la classificazione dinamica abbia a disposizione maggiori informazioni sul comportamento dei campioni software, i risultati ottenuti non hanno superato di molto quelli dell'analisi statica. Le motivazione di una differenza così esigua potrebbe risiedere nelle limitazioni introdotte dal firewall che hanno limitato non solo la parte network ma anche quella comportamentale e nella versione, non recente, del sistema operativo Android utilizzato in CuckooDroid.

7 Conclusioni

La sicurezza informatica, al giorno d'oggi, ha assunto una importanza fondamentale, e può avere ripercussioni anche nella vita quotidiana degli utenti, a causa della pervasività dei dispostivi smart a disposizione degli utenti. Ciò è vero in particolar modo per i dispositivi che vengono utilizzati perlopiù in maniera spontanea e senza la dovuta attenzione, come ad esempio gli smartphone, in cui vengono conservate informazioni personali e sensibili come foto, conversazioni e dati sulla geolocalizzazione. Inoltre, possono anche essere usati per accedere a servizi come posta certificata e online banking sia direttamente dagli stessi smartphone che autorizzando tramite autenticazione a due fattori l'accesso che avviene con un altro dispositivo.

La principale minaccia per gli smartphone è costituita dalle app malevole e la maggiore protezione contro questa minaccia è costituita sui sistemi di riconoscimento dei malware. I sistemi esistenti in commercio ed in letteratura utilizzano informazioni che possono essere ricondotte alla natura maligna o benevola della app, tramite l'analisi statica, dinamica o ibrida. In questo lavoro è stato presentato un sistema ibrido di riconoscimento malware per dispositivi mobili Android che sfrutta sia l'analisi statica che quella dinamica e che utilizza tecniche di apprendimento automatico per addestrare dei classificatori in grado di individuare la natura malevola o benevola delle applicazione analizzate. Il lavoro di tesi svolto ha dunque riguardato oltre alla progettazione e sviluppo del sistema ibrido, anche progettazione e sviluppo dei sottosistemi statico e dinamico.

La scelta degli algoritmi di machine learning per quanto riguarda l'analisi statica è ricaduta sui seguenti classificatori: Albero decisionale, Random forest e Reti Bayesiane. In generale i risultati ottenuti sono stati molto soddisfacenti ed i migliori risultati sono stati ottenuti con il classificatore Random Forest.

Per l'analisi dinamica sono stati impiegati gli stessi classificatori impiegati in quella statica. Inoltre, si sono confrontate le prestazioni al variare della composizione del vettore delle caratteristiche le quali rappresentano in maniera sintetica il comportamento degli eseguibili analizzati. Anche in questo caso i migliori risultati si sono ottenuti con Random Forest. Tuttavia, i risultati sperimentali non hanno mostrato un incremento della bontà delle predizioni

rispetto all'analisi statica, anzi si è verificato un leggero calo delle prestazioni. Questo risultato inaspettato potrebbe essere dovuto a vari fattori, tra cui la versione non recente di Android utilizzata per l'analisi e l'assenza di connettività ad internet durante l'esecuzione simulata. Quest'ultimo è un problema cruciale e tuttora aperto dato che nella stragrande maggioranza dei casi le applicazioni mobili richiedono una connessione di rete per funzionare ed in molti casi la logica di funzionamento è in buona parte implementata lato server. Inoltre, non si tratta neanche di un fattore discriminante in quanto ciò vale sia per le applicazioni malevole che per quelle benevole. Sebbene siano presenti in letteratura (Williamson, 2012) (S. Arshad, 2018) delle possibili soluzioni a tale problema ognuna di esse presenta svantaggi e dalla ricerca bibliografica svolta durante questo lavoro di tesi non risulta che nessuna di esse sia stata ancora utilizzata in un sistema pronto all'uso.

Per ciò che concerne l'analisi ibrida sono state proposte due architetture per combinare i due sottosistemi di analisi e si sono confrontati i risultati ottenuti: la prima combina le features estratte in un unico vettore da utilizzare per la classificazione, la seconda invece implementa un sistema di stacking ensemble learning in cui le features statiche e dinamiche sono classificate separatamente e i loro risultati sono dati in ingresso ad un metaclassificatore che darà la predizione sulla natura malevola o benigna del file.
Nel primo caso si ha un sistema più semplice e leggermente più efficiente. Nondimeno, con il secondo sistema, effettuando separatamente le classificazioni, si ha una maggiore flessibilità nel decidere quali classificatori utilizzare e se eseguire solamente uno dei due sottosistemi di analisi. I due esperimenti hanno prodotto risultati molto simili ma a giudizio dell'autore di questa tesi l'approccio con il metaclassificatore è da preferire poiché e più flessibile e si può prestare più facilmente all'implementazione di un sistema di analisi ibrida in cui l'analisi dinamica venga eseguita solo al verificarsi di determinate condizioni. Tra i classificatori utilizzati i migliori risultati sono stati ottenuti nel primo caso con Random forest e nel secondo caso utilizzando Random Forest come classificatore per i due sottosistemi e l'albero decisionale come metaclassificatore.

Analizzando i risultati ottenuti, dei possibili sviluppi futuri che potrebbero portare a dei miglioramenti potrebbero essere:

- L'analisi dinamica potrebbe avvenire su sandbox che eseguano versioni più recenti del

sistema operativo Android in modo da seguire al meglio la costante evoluzione dei malware. Inoltre, si potrebbero valutare delle modifiche per mitigare i problemi dovuti all'assenza di connessione a internet durante l'analisi. Una possibile soluzione potrebbe essere quella di monitorare l'effettiva esecuzione delle applicazioni nei dispositivi degli utenti finali ma nel fare ciò bisognerebbe considerare le implicazioni per la privacy degli utenti ed inoltre tale analisi potrebbe essere effettuata solo a posteriori. Un'altra possibile soluzione è creare una simulazione di rete Internet da utilizzare per l'analisi ma questa risulterebbe utile solo nel caso di connessioni non cifrate.

- Per quanto riguarda i classificatori si potrebbero valutare i risultati dell'utilizzo delle reti neurali profonde che non sono state prese in considerazione per via dei lunghi tempi di addestramento in questo lavoro di tesi ma sono utilizzate in altri lavori, ad esempio, in (D. Kim, 2018). Per ottenere dei risultati in tempi ragionevoli ci si potrebbe avvalere di acceleratori hardware per le reti neurali.
- Nel sottosistema di analisi statica si è reso necessario, al fine di per ridurre la dimensione dei vettori, un filtraggio delle features. Tale filtraggio, nato dalla necessità di ridurre i tempi di addestramento e basato sulla frequenza delle API non ha affrontato il problema della correlazione tra i dati e della presenza di eventuali dati rumorosi. Si potrebbe pensare di utilizzare tecniche più avanzate di filtraggio al fine di ridurre la correlazione dei vettori di features e migliorare le prestazioni dei classificatori. Tale filtraggio si potrebbe quindi estendere anche al sottosistema di analisi dinamica.

Bibliografia

(s.d.). Tratto da virusshare: https://virusshare.com/

(s.d.). Tratto da Android Malware Genome Project : http://www.malgenomeproject.org/

(s.d.). Tratto da Drebin: https://www.sec.cs.tu-bs.de/~danarp/drebin/

(s.d.). Tratto da kuafuDet: https://nsec.sjtu.edu.cn/kuafuDet/kuafuDet.html

(s.d.). Tratto da Androzoo: https://androzoo.uni.lu/

A. De Paola, S. G. (2018). A hybrid system for malware detection on big data. *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS).*

Anwar, S. Z. (2018). Android botnets: A serious threat to android devices. *Pertanika Journal of Science & Technology 26.1*.

Arini Balakrishnan, C. S. (2012). Code Obfuscation Literature Survey. Tratto da http://pages.cs.wisc.edu/~arinib/writeup.pdf

B. Amos, H. T. (2013). Applying machine learning classifiers to dynamic Android malware detection at scale. *2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC).* Sardinia.

B. Kang, S. Y. (2016). N-opcode analysis for android malware classification and categorization. *2016 International Conference On Cyber Security And Protection Of Digital Services (Cyber Security).* London.

Breiman, L. (1994). *Bagging Predictors.* Bagging PredictorsByLeo Breiman*Technical Report No. 421September 1994*PartiaDepartment of StatisticsUniversity of CaliforniaBerkeley, California 94720.

Chalk., S. J. (1997). probability. *IUPAC. Compendium of Chemical Terminology, 2nd ed. (the "Gold Book").* Tratto da http://goldbook.iupac.org/terms/view/P04855

Contagio Mobile. (s.d.). Tratto da http://contagiomobile.deependresearch.org/index.html

Crussell, J. R. (2014). "Madfraud: Investigating ad fraud in android applications.". *Proceedings of the 12th annual international conference on Mobile systems, applications, and services.*

D. Kim, D. M.-K. (2018). A Hybrid Static Tool to Increase the Usability and Scalability of Dynamic Detection of Malware. *2018 13th International Conference on Malicious and Unwanted Software (MALWARE).* Nantucket, MA, USA.

D. Ö. Şahın, O. E. (2018). New Results on Permission Based Static Analysis for Android Malware. *6th International Symposium on Digital Forensic and Security (ISDFS).* Antalya.

De Paola, A., Favaloro, S., Gaglio, S., Lo Re, G., & Morana, M. (2018). Malware Detection through Low-level Features and Stacked Denoising Autoencoders. *ITASEC.* Milan.

Dempster, M. L., Laird, N. M., & Rubin, D. B. (1997). Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological), Vol. 39, No. 1.*

Dmitrievsky, M. (2018, 07 06). *Random Decision Forest in Reinforcement learning*. Tratto da mql5.com: https://www.mql5.com/en/forum/263400

F. Massicotte, M. C. (2012). A Testing Model for Dynamic Malware Analysis Systems. *IEEE Fifth International Conference on Software Testing, Verification and Validation*, (p. 826-833). Montreal, QC.

Fabrizio Biondi, T. G.-W. (2018). Tutorial: anOverview of Malware Detection and Evasion Techniques. *ISoLA 2018 - 8th International SymposiumOn Leveraging Applications of Formal Methods, Verification and Validation,.* Limassol,.

F-Droid Limited. (2020). *Verification Server*. Tratto da f-droid.org: https://f-droid.org/en/docs/Verification_Server/

Finetti, D. (1970). *Teoria delle probabilità,.* G.Einaudi.

Google. (s.d.). Tratto da Play Store: https://play.google.com/store

Google. (2019). *Android Security 2018 Year in Review.* Tratto il giorno 04 05, 2020 da https://source.android.com/security/reports/Google_Android_Security_2018_Report_Final.pdf

Google. (2020). Tratto da https://developer.android.com/: https://developer.android.com/reference/android/Manifest.permission.html#SYSTEM_ALERT_WINDOW

Guest Machine Architecture. (2015). Tratto da https://cuckoo-droid.readthedocs.io/en/latest/installation/guest_android_avd/architecture/

He, D. &. (2015). Mobile application security: Malware threats and defenses. *Wireless Communications, IEEE.*

Ho, T. K. (1995). Random decision forests. *Proceedings of 3rd International Conference on Document Analysis and Recognition.* Montreal, Quebec, Canada.

Hossin, M. a. (2015). A review on evaluation metrics for data classification evaluations. *International Journal of Data Mining & Knowledge Management Process*.

IDC Corporate USA. (2020, 01). *Smartphone Market Share*. Tratto da idc.com: https://www.idc.com/promo/smartphone-market-share/os

Joshi, M. V. (2002). On evaluating performance of classifiers for rare classes. *2002 IEEE International Conference on Data Mining.* Maebashi City, Japan.

Julien Gamba, M. R.-R. (2019). An Analysis of Pre-installed Android Software. *41st IEEE Symposium on Security and Privacy, 18-20 May 2020,.* San Fransisco, CA, USA.

Kolmogorov, A. K. (1956). *Theory of probability.* CHELSEA PUBLISHING COMPANY NEW YOURK.

Kopelo Letou, D. D. (2013). *Host-based Intrusion Detection and Prevention System (HIDPS)* .

L. C. G. Rogers, W. D. (2000). *Diffusions, Markov Processes, and Martingales.* Cambridge University Press.

Leonid Batyuk, M. H.-D. (2011). Using Static Analysis for Automatic Assessment and Mitigation of Unwanted and Malicious Activities Within Android Applications, In Proceedings of the International Conference on Malicious and Unwanted Software. Tratto il giorno 04 06, 2020 da http://lilicoding.github.io/SA3Repo/papers/2011_batyuk2011using.pdf

Lombardo, A. (2016). *Probabilità e statistica per ingegneri.*

M. Z. Mas'ud, S. S. (2013). Profiling mobile malware behaviour through hybrid malware analysis approach. *2013 9th International Conference on Information Assurance and Security (IAS).*

Marastoni, N. &. (2017). GroupDroid: Automatically Grouping Mobile Malware by Extracting Code Similarities. *SSPREW-7 , December 4–5, 2017.* San Juan, PR, USA.

Mehrnezhad, M. T. (2018). Stealing PINs via mobile sensors: actual risk versus user perception. *Int. J. Inf. Secur*. Tratto da https://doi.org/10.1007/s10207-017-0369-x

Mitchell, T. M. (1997). *Machine Learning.* McGraw-Hill Science.

Norvig, P. R. (2002). *Artificial Intelligence. A modern approach.* Prentice Hall.

Perez, S. (2018, 07 16). *Apple's App Store revenue nearly double that of Google Play in first half of 2018*. Tratto da techcrunch.com: https://techcrunch.com/2018/07/16/apples-app-store-revenue-nearly-double-that-of-google-play-in-first-half-of-2018/?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=AQAAAJToyw6USEJOwzmXR3cYfDDz7j94d-

XCp9FDBMEsWuAwKCm8_fVj-zKhit
Peter Ney, K. K. (2017). Computer Security, Privacy, and DNA Sequencing:Compromising Computers with Synthesized DNA, Privacy Leaks, and More. *2017 USENIX Security Symposium.*
Pfeffer, A. R. (2017). Artificial intelligence based malware analysis.
Rashid, W. (2017). *Automatic Android Malware Analysis.* Kiel University of Applied Sciences,, Computer Science and Electrical Engineering. Tratto il giorno 03 13, 2020 da https://github.com/waqarrashid33/internship_report/blob/master/main.pdf
Rete bayesiana . (2014, 11 15). Tratto da Okpedia: https://www.okpedia.it/rete_bayesiana
S. Arshad, M. A. (2018). SAMADroid: A Novel 3-Level Hybrid Malware Detection Model for Android Operating System.
Sartea, R. &. (2016). Active Android malware analysis: an approach based on stochastic games.
Shannon, c. E. (1948). A Mathematica lTheory of Communication. *The Bell SystemTechnical Journal*.
smali/baksmali. (s.d.). Tratto il giorno 2020 da https://github.com/JesusFreke/smali
Steinberg, M. (2019). *The Platform Economy. How Japan Transformed the Consumer Internet.* University of Minnesota Press.
The University of Waikato. (2020). *WEKA The workbench for machine learning*. Tratto il giorno 2020 da WEKA: https://www.cs.waikato.ac.nz/ml/weka/
V. G. Shankar, G. S. (2017). AndroTaint: An efficient android malware detection framework using dynamic taint analysis. *2017 ISEA Asia Security and Privacy (ISEASP).*
Vallecillo, J. (2017). Tratto da https://github.com/jgamblin/isthisipbad/blob/master/isthisipbad.py
W. Park, K. L. (2014). Analyzing and detecting method of Android malware via disassembling and visualization. *International Conference on Information and Communication Technology Convergence (ICTC).* Busan.
Wang, H.-T. W.-C. (2016). The diffusion process, competitive relationship,and equilibrium analysis of smartphone operating systems. *Technology Analysis*, p. 8.
William, S. (2016). *Cryptography and network security : principles and practice (Seventh ed.).*
Williamson, S. &. (2012). Active Malware Analysis using Stochastic Games. *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AA-MAS 2012).*
YARA. (2020, 04 06). Tratto da https://virustotal.github.io/yara/
Yixiang Zhu, K. Z. (2017). Review of iOS Malware Analysis. *IEEE Second International Conference on Data Science in Cyberspace (DSC) DSC Data Science in Cyberspace.*