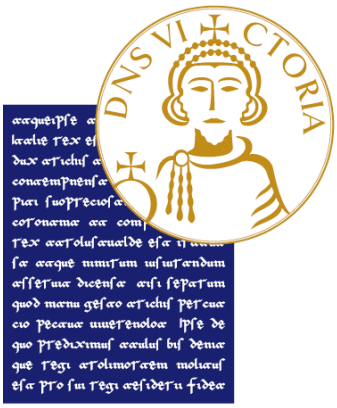
UNIVERSITÀ DEGLI STUDI DEL SANNIO



Dipartimento di Ingegneria

Corso di laurea triennale in Ingegneria Informatica

Tesi di Laurea

in

Sicurezza Informatica

STUDIO DI ATTACCHI DI POISONING A CLASSIFICATORI PER MALWARE DETECTION BASATI SU MACHINE LEARNING

Relatore

Prof. Corrado Aaron Visaggio

Candidato

Roberto Garzone 863000793

Correlatrice

Dott.ssa Sonia Laudanna

Anno Accademico 2020/2021

Indice

Capitolo 1

Introduzione

Gli smartphone, con il passare del tempo, sono diventati parte integrante della quotidianità delle persone, utilizzati come il mezzo più comune di trasmissione di informazioni sensibili, quali: foto, contatti o email. Un tale utilizzo ha traslato l'interesse dei criminali informatici al mondo della telefonia, portandoli alla ricerca di un sistema efficace di ottenimento di informazioni sensibili. Difatti, nell'agosto del 2010, Kaspersky [1] ha registrato il primo SMS contenente un Trojan, a causa del quale uno smartphone ha inviato, senza il consenso dell'utente, un SMS a numeri con tariffa maggiorata. Nel report generato da McAfee's Global Threat Intelligence [2], risalente al 2018, è stata dichiarata la presenza di 65.000 applicazioni false all'interno dell'app store di Google, 55.000 delle quali caricate nel giugno dello stesso anno. A queste si aggiunge un significativo incremento della presenza di Trojan e Backdoor. Un tale incremento ha generato, in Google, la necessità di un innalzamento dei livelli di sicurezza, utili all'identificazione e, conseguentemente, alla rimozione di applicazioni sospette. Gli interventi sulla sicurezza

hanno portato, il 3 febbraio 2019, all'eliminazione di 29 applicazioni maligne, le quali attività includevano la pubblicazione in modalità full-screen di pubblicità con contenuti pornografici, ad ogni sblocco del telefono, e la ridirezione dell'utente verso una pagina per attuare attacchi di phishing. Nel 2015, l'Università di Cambridge, analizzando oltre 20.000 smartphone di vari vendor, ha riportato che l'87,7% dei dispositivi Android è suscettibile di almeno una vulnerabilità critica. L'attenzione della sperimentazione si è focalizzata sul sistema Android, in quanto StatCounter [3] ha statisticamente definito che oltre il 75% degli smartphone in commercio usano tale sistema operativo. In questo scenario, una tecnica di intelligenza artificiale che ha recentemente preso piede è nota come machine learning [4] [5]. Si tratta di una tecnica che consente al calcolatore di apprendere dall'esperienza e, quindi, decidere sulla base della stessa, non necessitando di istruzioni esplicite. Mediante il machine learning è quindi possibile individuare potenziali malware ancora sconosciuti [6] [7]. Tale tecnica si distingue in due categorie: supervisionata o non supervisionata. La fondamentale differenza tra le due categorie è il tipo di apprendimento: nel primo caso l'apprendimento viene effettuato con il supporto di una serie di esempi, che costituiscono il dataset, che consentono al calcolatore di effettuare una distinzione degli elementi inviatigli in input. Nel caso non supervisionato, sarà lo stesso calcolatore a dover identificare le caratteristiche dei singoli dati in input e suddividerli sulla base di esse [7] [8]-[10]. Il machine learning viene ampiamente utilizzato nell'individuazione e la classificazione dei malware attraverso tre diverse tipologie di analisi: statica, dinamica ed ibrida [11]. L'analisi statica si ottiene senza l'esecuzione dell'applicazione, quindi senza alcun effetto sul dispositivo. Essa si basa

sullo studio di specifici elementi del codice sorgente, come: permessi, chiamate API o intent (category o action) [12] e risulta fortemente vulnerabile agli attacchi basati sull'offuscamento [13]-[15]. L'analisi dinamica, invece, consiste nell'esecuzione di un'applicazione all'interno di una sandbox e nella registrazione di tutti gli eventi di log. Grazie a questa analisi è possibile registrare tutte le chiamate a sistema effettuate dall'applicazione [16] [17] e monitorare il comportamento del malware stesso. Ultima tipologia di analisi è l'analisi ibrida, che riprende metodi e pregi dell'analisi statica e dinamica. L'analisi ibrida può essere effettuata sia in locale che su un server remoto [18]. Sebbene, le soluzioni di machine learning siano adottate con successo in molteplici contesti, l'applicazione di queste tecniche al dominio della cyber security è complessa e ancora immatura. Tra le molte questioni aperte che interessano i sistemi di sicurezza basati sul machine learning, in questo lavoro viene esaminata la robustezza di tali modelli rispetto ad attacchi avversariali che mirano a influenzare le capacità di rilevamento e predizione.

1.1 Obiettivi

L'obiettivo della sperimentazione è quello di valutare la robustezza dei detector basati su machine learning rispetto ad attacchi di tipo poisoning. I classificatori utilizzati per la valutazione si basano su algoritmi di: Logistic Regression, Random Forest e Support Vector Machine. I contributi di questo studio includono:

- Classificazione di apk di tipo benigno e di tipo malware e valutazione dell'accuratezza dei classificatori;

- Generazione di poisoning attack basati su distanza euclidea;

- Generazione di poisoning attack basati su distanza di Hamming;

- Generazione di poisoning attack basati su K-Means Clustering;

- Identificazione della più piccola perturbazione che può essere aggiunta ad un campione per renderlo avversariale;

- Valutazione dell'efficacia dei modelli considerati nei vari scenari di attacco.

1.2 Motivazione e contesto

Android risulta essere il sistema operativo mobile più popolare. La crescente popolarità di Android ha portato ad un aumento dello sviluppo di applicazioni di terze parti e, di conseguenza, l'autenticità di queste app è sempre in discussione. Le app di terze parti sono quelle app create da un fornitore diverso dal produttore del dispositivo. Esse hanno lo scopo di soddisfare le intenzioni del venditore, non dell'utente. Le applicazioni di terze parti risultano vulnerabili a diversi tipi di attacchi. Queste app, anche se presenti su app store ufficiali, possono essere dannose. Diversi scenari di attacco risultano essere eseguiti su applicazioni di terze parti. Gli scenari di attacco, ad esempio, includono il repackaging dell'app Android che si realizza quando l'attacker aggiunge funzionalità dannose e le ridistribuisce agli utenti che credono di utilizzare un'app legittima o l'app originale. L'atto di 'riconfezionare' l'applicazione viene utilizzato anche quando si tenta di decodificare l'applicazione stessa. Il repackaging è una pratica ampiamente utilizzata per distribuire soprattutto Trojan [21]. L'adversarial machine learning è un campo che si trova all'intersezione tra machine learning e sicurezza informatica. Questa tecnica tenta di 'ingannare' i modelli di apprendimento automatico tramite input dannosi [22] - [24]. Diversi lavori hanno dimostrato che alcuni esempi avversariali generati per un modello possono anche essere classificati erroneamente da un altro modello. Questa proprietà, che non si basa sul modello per cui è stata creata, è denominata trasferibilità. Essa può essere sfruttata per eseguire attacchi di tipo black-box [25]. Esistono diversi metodi per creare esempi avversariali come Fast Gradient Sign Method (FG-

SM), Jacobian-Based Saliency Map Approach (JSMA) [26],[27]. Il Poisoning attack è una delle più importanti tecniche di creazione di attacchi avversariali. Tale attacco viene effettuato nelle fasi di apprendimento di un qualsiasi algoritmo di intelligenza artificiale, introducendo una serie di training set, accuratamente progettati, utili alla compromissione dell'intero processo di apprendimento e alla creazione di una backdoor [28]. L'avvelenamento può essere di tre tipologie:

1. avvelenamento del Dataset: uno degli attacchi più comuni, consistente nell'introduzione, nel training set, di dati errati oppure nella modifica del comportamento dell'algoritmo;

2. avvelenamento dell'algoritmo: in questo caso viene sfruttato uno specifico algoritmo per comprenderne il modello di apprendimento. Esistono diverse modalità di avvelenamento di un algoritmo, tra cui:

   - avvelenare l'apprendimento del trasferimento: gli attaccanti istruiscono un algoritmo avvelenato e lo diffondono ad altri algoritmi di machine learning
   - iniezione e manipolazione dei dati: gli attaccanti iniettano nel dataset una serie di dati errati
   - corruzione della logica: l'attaccante modifica il metodo di apprendimento dell'algoritmo

3. avvelenamento del modello: è la forma di avvelenamento più semplice. Il modello viene sostituito con uno avvelenato, il quale resta all'interno del calcolatore facendo da backdoor per l'attaccante.

Come esempio, è possibile menzionare uno dei più celebri esperimenti sul poisoning attack che vede un gruppo di ricercatori apportare delle perturbazioni alle immagini di un panda, caricate in un algoritmo di apprendimento per l'identificazione delle specie, portando lo stesso algoritmo ad identificare come panda l'immagine di un gibbone [29] [44].
In questo lavoro vengono discussi tre scenari di poisoning attack:

1. Poisoning Attack basato sulla distanza euclidea;
2. Poisoning Attack basato sulla distanza di Hamming;
3. Poisoning Attack basato sul K-Means Clustering.

Tutti gli scenari di attacco vengono eseguiti su *feature* statiche di un'applicazione, come i permessi. L'efficacia dell'attacco creato viene esaminata utilizzando diversi modelli di classificatori di machine learning.

1.3 Organizzazione del lavoro

La tesi si compone di 4 capitoli più la bibliografia.

Nel primo capitolo viene data una visione generale delle motivazioni che hanno portato alla nascita di questa sperimentazione e degli obiettivi che si vogliono perseguire. Viene ripreso il concetto di machine learning, le sue applicazioni e la sua implementazione. Allo stesso modo viene analizzato il concetto di Poisoning Attack e le forme che questo può assumere.

La seconda parte della tesi prevede l'analisi dei lavori, presenti in letteratura, attinenti agli obiettivi della sperimentazione. Lavori che vengono poi utilizzati come punto di partenza per la costruzione del modello della sperimentazione stessa.

Il terzo capitolo prevede una descrizione pratica dell'implementazione degli algoritmi di machine learning e l'analisi dei risultati. Per quanto riguarda la descrizione degli scenari di attacco, è necessario specificare che quando si parla di passi, si intendono le righe di codice del relativo pseudo-codice. I passi mancanti sono passi che indicano la chiusura di cicli iterativi e costrutti condizionali.

Nel quarto capitolo saranno presenti le considerazioni finali, poste sulla base dei dati ottenuti dalle analisi effettuate.

Ultimo capitolo è la bibliografia, contenente le fonti delle informazioni inserite all'interno della tesi.

Capitolo 2

Stato dell'arte

Esistono in letteratura diversi lavori che riguardano sia la generazione degli attacchi avversariali che degli attacchi di tipo poisoning in particolare.
Xuanzhe Liu ha condotto uno studio sul pattern comportamentale degli utenti Android [30]. Esso si focalizza sull'analisi del diverso utilizzo delle applicazioni, sulla base della popolarità, della gestione delle attività, dell'uso della rete e della scelta del dispositivo. La conoscenza di tali pattern aiuta la comunità di ricerca nella scoperta di nuovi trend da poter seguire per lo sviluppo di nuove applicazioni e la loro distribuzione.
Lo studio di Sen Chen, invece, illustra lo sviluppo di un approccio basato sull'advesarial machine learning, diviso in due fasi. Tale approccio è noto come KuafuDet [31]. La prima fase consiste nell'estrapolazione delle *feature*; che verranno poi analizzate, nella seconda fase, con un'analisi remota. Gli algoritmi di adversarial machine learning vengono applicati per l'offuscamento di applicazioni maligne.
Il lavoro di Sen Chen si pone come filtro di identificazione dei falsi negativi,

proponendo un sistema capace di incrementare l'accuratezza della detection del 15%.

Un altro studio di Sen Chen prende il nome di StormDroid [33] un set costituito da una serie di *feature* estratte per eseguire una classificazione delle stesse. Vengono, inotre, estratte una serie di chiamate API, mediante metriche basate su alberi. É dimostrato come StormDroid raggiunge un'accuratezza di detection pari al 94%.

Weilin Xu ha proposto un caso di studio basato sulla classificazione di malware PDF e sui metodi di classificazione automatica delle evasioni [34]. In tale studio le varianti malware vengono generate mediante tecniche di progammazione genetica, dette mutazioni. Le valutazioni vengono effettuate mediante due particolari classificatori: PDFrate e Hidost. Al fine di individuare i malware all'interno del dataset, vengono applicati tool messi a disposizione da Oracle. Dopo aver individuato i sample malevoli, questi vengono opportunamente manipolati, con particolari tecniche di mutazione.

Infine, Fei Zhang [35] ha dedotto che le *feature* possono essere selezionate sulla base delle capacità di generalizzazione e della sicurezza riguardo gli attacchi di evasione. A tal proposito è necessaria l'implemetazione di un algoritmo di selezione delle stesse, basato sul wrapper. Viene poi generato un esempio di attacco di evasione, utile alla valutazione della robustezza del gruppo di *feature* selezionate. L'esperimento ha previsto l'uso di uno spam filtering e di un sistema di individuazione di malware su PDF.

Capitolo 3

Disegno dell'esperimento

3.1 Domande di ricerca

Questo lavoro ha l'obiettivo principale, come già anticipato, di valutare la robustezza, in termini di accuratezza, di un modello di classificazione in caso di attacco di tipo poisoning e di determinare quale e quanto piccola deve essere la perturbazione per rendere un campione di tipo avversariale.
A fronte della sperimentazione sono state poste due domande di ricerca:

RQ1: Qual è la più piccola perturbazione che può essere aggiunta ad un campione per renderlo avversariale?

RQ2: Quanto risultano robusti i modelli di machine learning considerati sottoposti ad attacchi di tipo poisoning?

Per rispondere alla prima domanda di ricerca è necesserio eseguire diversi test durante i quali la perturbazione subisce delle variazioni arbitrarie ed incrementali. Lo scopo è quello di definire la minima perturbazione necessaria

a contribuire all'alterazione del malware e la sua successiva evasione.
Invece, per rispondere alla seconda domanda di ricerca è necessario sottoporre gli algoritmi di machine learning, precedentemente implementati, ad un attacco di tipo poisoning. I malware alterati vengono raccolti in un file che poi viene usato come test set per la predizione. Sulla base delle predizioni ottenute mediante i tre classificatori: Logistic Regression, SVM e Random Forest, sarà possibile definire quale, tra i tre algoritmi testati, si dimostrerà il più robusto, ovvero quale algoritmo sarà maggiormente capace di riconoscere i malware nonostante l'alterazione. Tale robustezza viene, quindi, analizzata mediante il detection rate degli algoritmi, con cui è possibile comprendere, per ogni algoritmo e per ogni classificatore, in quale scenario si ottiene il minor numero di sample di tipo malware evasi.

3.2 Descrizione del dataset

Il dataset utilizzato include 2416 sample, scaricati dal repository AndroZoo [19], un dataset da oltre un milione di applicazioni Android raccolte da vari store. Di questi sample distinguiamo due tipologie: benigni e malware. Differenziati, nel dataset, dall'etichetta "CLASSE", i sample benigni vengono indicati con "0", mentre i malware vengono indicati con "1".

Dai sample ottenuti vengono esaminati i permessi delle singole applicazioni, estratti mediante un processo di reverse-engineering utilizzando uno specifico script scritto in Python: ExtractorAIO.py [20].

Tale script esegue le operazioni di estrapolazione e collezione dei permessi dei file .apk. L'estrapolazione avviene mediante il modulo jadx [38]; un decompilatore Java-Based, la cui funzione consiste nell'estrapolazione dei permessi di applicazioni per OS Android. La collezione dei permessi, correlati al nome dell'applicazione, avviene mediante comandi di shell Unix, utili alla creazione e modifica del file .csv, contenitore delle informazioni ottenute.

In fig. 3.1 viene rappresentato il processo di estrapolazione dei permessi.

Dal processo di estrapolazione dei permessi viene realizzato un file .csv come quello mostrato in fig. 3.2.

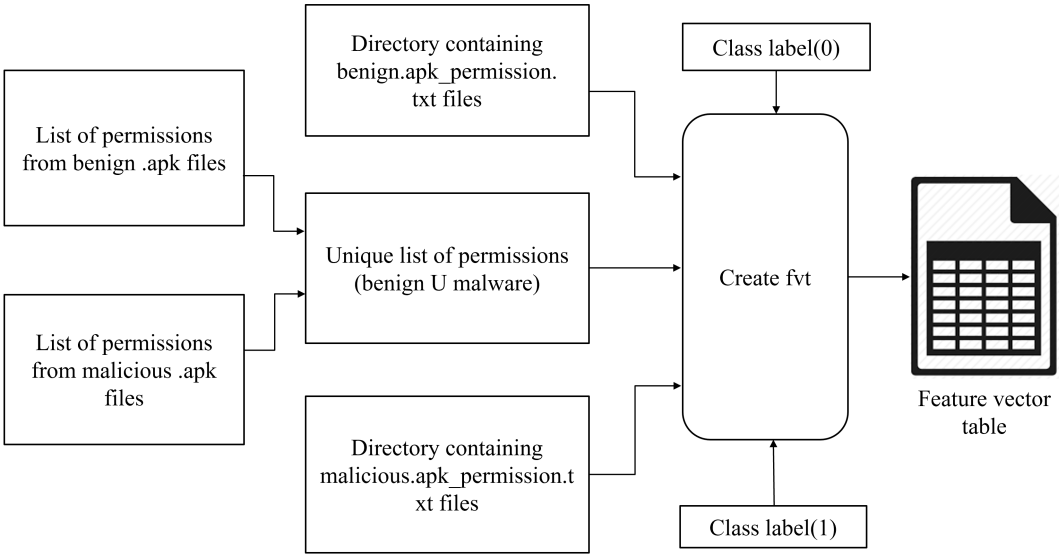


Figura 3.1: Creazione della tabella dei vettori delle feature

	A	ASH	ASI	ASJ	ASK	ASL	ASM	ASN	ASO
1	NAME	biz.app4m	com.googl	com.noma	com.indiai	com.subsp	br.com.tec	android.ha	CLASS
853	067157FB	0	0	0	0	0	0	0	0
854	06716825	0	0	0	0	0	0	0	0
855	06716981	0	0	0	0	0	0	0	0
856	06716D9A	0	0	0	0	0	0	0	0
857	06716EB0	0	0	0	0	0	0	0	0
858	067170F8	0	0	0	0	0	0	0	0
859	067190EF	0	0	0	0	0	0	0	0
860	0671941C	0	0	0	0	0	0	0	0
861	0671946F	0	0	0	0	0	0	0	0
862	06719C33	0	0	0	0	0	0	0	0
863	06719E0C	0	0	0	0	0	0	0	0
864	0671A331	0	0	0	0	0	0	0	0
865	0671A716	0	0	0	0	0	0	0	0
866	0671A99A	0	0	0	0	0	0	0	0
867	0000003B	0	0	0	0	0	0	0	1
868	0000014A	0	0	0	0	0	0	0	1
869	000002B6	0	0	0	0	0	0	0	1
870	000003D3	0	0	0	0	0	0	0	1
871	00000439	0	0	0	0	0	0	0	1
872	0000049D	0	0	0	0	0	0	0	1
873	0000090C	0	0	0	0	0	0	0	1
874	00000989	0	0	0	0	0	0	0	1
875	00000F8E	0	0	0	0	0	0	0	1
876	00001091	0	0	0	0	0	0	0	1
877	000010F3	0	0	0	0	0	0	0	1
878	00001112	0	0	0	0	0	0	0	1
879	0000120C	0	0	0	0	0	0	0	1
880	0000143E	0	0	0	0	0	0	0	1
881	000014F7	0	0	0	0	0	0	0	1
882	00001802	0	0	0	0	0	0	0	1
883	00001991	0	0	0	0	0	0	0	1

Figura 3.2: Dataset ottenuto

3.3 Metodologia di analisi

Il poisoning attack è un processo di iniezione, il quale produce delle perturbazioni che vanno ad incrementare il tasso d'errore dell'algoritmo di machine learning.

Affinchè sia possibile effettuare una valutazione della robustezza degli algoritmi implementati, è necessaria la presenza di un classificatore, detto H, e di un dataset per la classificazione, che fungerà da training set e da test set opportunamente selezionato.

Il dataset si presenta come un array multidimensionale a d dimensioni del tipo:

$$[D = (X_i, y_i)_{i=1}^{n}] \tag{3.1}$$

dove

$$X_i \in [1, 0]^d \tag{3.2}$$

Il classificatore H, preso in input il dataset, esegue la classificazione e produce un risultato y del tipo: H(X) = y.

Obiettivo del poisoning attack è quello di aggiungere piccole perturbazioni alle *feature* di:

$$X \wedge H(X + \mu) = H(X^*) \mid H(X^*) = y^{'} \rightarrow y^{'} \neq y \tag{3.3}$$

Nello scenario, proposto in questo lavoro, verranno eseguiti tre diversi attacchi di poisoning:

1. Poisoning attack basato sulla distanza euclidea tra il malware e l'intero

set di sample benigni;

2. Poisoning attack basato sulla distanza di Hamming tra il malware e l'intero set di sample benigni;

3. Poisoning attack basato sul K-Means Clustering.

Nella fase di sperimentazione viene analizzato, in forma generale, uno scenario del tipo mostrato in fig. 3.3:

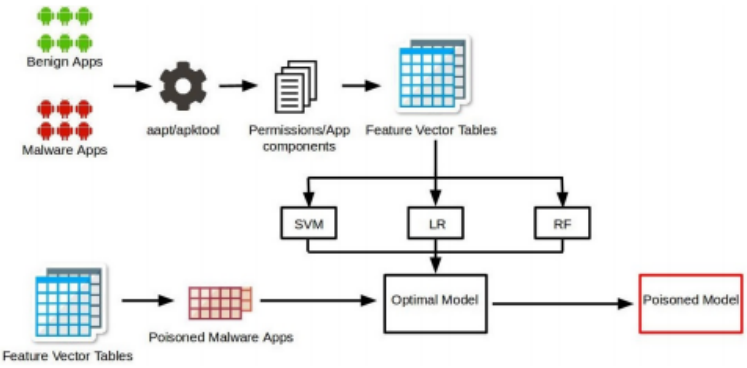


Figura 3.3: Architettura: Trovare un modello ottimale ed eseguire un attacco di tipo poisoning

L'attaco verrà eseguito su un dataset contenente 1120 .apk maligni, forniti da AndroZoo [19] e 1296 .apk benigni, forniti dal Canadian Institute for Cybersecurity [36].

Per ogni file .apk ottenuto, verranno estratti i permessi. Successivamente si andrà a verificare l'efficacia dei classificatori.

Il dataset usato per la classificazione conterrà entrambe le tipologie di apk ed i relativi permessi.

Questi ultimi saranno di natura binaria, ma dovranno essere convertiti in interi nel momento della sperimentazione.

Dal dataset verranno estratti i sample maligni, che fungeranno da test set e saranno sottoposti al poisoning attack, nel momento in cui verranno soddisfatte precise condizioni.

Il malware alterato sarà, infine, mandato in input al modello ottimale per la predizione, ovvero la verifica dell'efficacia dell'algoritmo. Quest'ultimo sarà tanto più efficace quanti meno saranno i sample evasi.

La valutazione si baserà sull'analisi dell'accuratezza, tasso di evasione, precisione e recall, dei singoli classificatori, calcolati prima e dopo l'attacco.

Per l'esecuzione della classificazione e della predizione è stato adoperato il software Weka. Esso è un software realizzato appositamente per tali operazioni. Prendendo in input file .arff, riesce ad estrapolare le *feature* di dataset e test set per poi eseguire le operazioni di classificazione degli elementi del dataset e di predizione degli elementi presenti nel test set, sulla base del classificatore indicato. Il risultato può essere salvato in diverse estensioni (come .csv, .html, ecc...). Per comodità di analisi è stata indicata l'estensione .csv.

3.3.1 Scenario di attacco basato sulla distanza euclidea

Per questo attacco verrà usato il 10% dell'intero test set, ottenuto mediante il calcolo della distanza euclidea tra il malware e i benigni.

Per ogni apk benigno verrà scelto un permesso in maniera del tutto randomica. L'alterazione verrà effettuata esclusivamente nel caso in cui il permesso sarà presente nell'apk benigno.

Lo scenario vedrà il vettore maligno del tipo:

$$M_n = (m_{n1}, m_{n2}, m_{n3}, ..., m_{nm}) \tag{3.4}$$

mentre il vettore benigno sarà del tipo:

$$B_n = (b_{n1}, b_{n2}, b_{n3}, ..., b_{nm}) \tag{3.5}$$

Ottenuti i due vettori, verrà calcolata la distanza euclidea, calcolabile con la seguente formula:

$$\sqrt{(m_{n1} - b_{n1})^2 + (m_{n2} - b_{n2})^2 + (m_{n3} - b_{n3})^2 + ... + (m_{nm} - b_{nm})^2} \tag{3.6}$$

dove m_{ni} rappresenta l'assenza o la presenza dell'i-esimo permesso nel malware M_n e b_{ni} rapprensenta la presenza o l'assenza dell'i-esimo permesso nel file benigno B_n

Una visione generale dell'implemetazione viene fornita dallo pseudo-codice mostrato in fig. 3.4:

Algorithm 1 Poisoning Euclidean

Input: $Test\_setT, classifierH, benigndatasetB, perturbation\delta$
Output: poisoned sample x_i'

```
1: evasion_count ← 0
2: repeat
3:   count ← 0
4:   repeat
5:     r ← 0, avg ← 0
6:     repeat
7:       sample ← x_i
8:       select random feature r_num
9:       if B[r_num] = 1 and T[r_num] = 0 then
10:        count ← count + 1
11:        sample[r_num] ← 1
12:        r ← r + 1
13:        dist ← 0, sum_dist ← 0
14:        repeat
15:          dist ← distance.euclidean(sample : j[0 :
             δ])
16:          edist.append(dist)
17:          sum_dist ← sum_dist + dist
18:        until j ≤ |B|
19:      end if
20:      avg ← sum_dist/|B|
21:      edist.sort()
22:      flag ← 1
23:      repeat
24:        if avg < edist[l] then
25:          flag ← 0
26:        end if
27:      until l ≤ |B| * δ
28:      if flag = 0 then
29:        p.append(H.predict_class(sample))
30:      end if
31:      if p = 0 then
32:        evasion_count ← evasion_count + 1
33:      end if
34:    until r ≤ δ
35:  until i ≤ |B|
36: until k ≤ |T|
37: goto 28
```

Figura 3.4: Pseudo-codice basato su distanza euclidea

In questo codice vengono forniti specifici input:

- Test Set (T)
- classificatore (H)
- benign dataset (B)
- perturbazione (δ)

Lo scenario viene rappresentato in fig. 3.5:

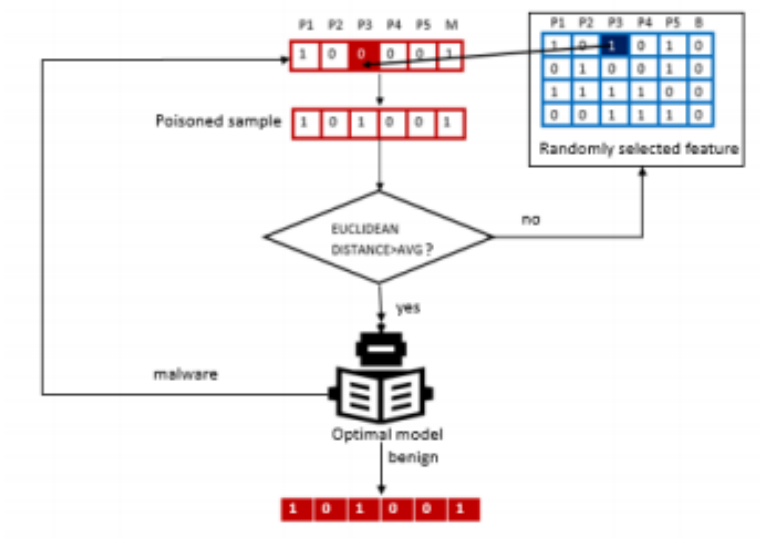


Figura 3.5: Poisoning attack basato sulla distanza euclidea

L'algoritmo può essere così riassunto: prima cosa, è necessario inizializzare la variabile *evasion_ count* (**passo 1**) a 0, questa variabile andrà a contare il numero di malware evasi. Successivamente viene avviato un ciclo iterativo (**passo 2**) che, ad ogni iterazione, andrà ad azzerare la variabile *count* (**passo 3**): un contatore delle alterazioni subite da uno stesso malware. Il ciclo al passo 2, limitato dall'incremento di una variabile k, termina quando k assumerà un valore pari alla dimensione del test set, precedentemente

riempito.

Dopo l'azzeramento di *count* si apre un nuovo ciclo (**passo 4**), limitato dall'incremento della variabile i, che termina quando i assumerà un valore pari alla dimensione del set di benigni, precedentemente riempito. Al **passo 5** si ha l'azzeramento delle variabili r e *avg*, rispettivamente: la perturbazione introdotta e la media delle distanze distanze euclidee tra il malware assunto e i benigni presenti nel dataset.

Si apre un nuovo ciclo iterativo (**passo 6**), che termina quando la perturbazione raggiunge un limite arbitrariamente imposto.

In questo ciclo vediamo il caricamento dell'i-esimo malware nella variabile *sample* (**passo 7**) e l'esecuzione dell'alterazione (**righi 9 - 11**).

Viene selezionato un valore casuale, caricato nella variabile *r_num* (**passo 8**), che va ad indicare la posizione della *feature* sia nel malware che nel benigno. Questo perché affinché sia possibile l'alterazione è necessario il rispetto di una condizione: la *feature* deve essere presente nel benigno (1 nel dataset) ed assente nel malware (0 nel dataset) (**passo 9**).

Se la condizione viene rispettata si passa all'alterazione.

Viene incrementato di 1 *count* (**passo 10**), il permesso viene forzato ad 1 in *sample* (**passo 11**) e la perturbazione viene incrementata di 1 (**passo 12**).

Terminata l'alterazione, ma sempre nel costrutto condizionale, vengono azzerate le variabili *dist*, che conterrà la distanza euclidea successivamente calcolata, e *sum_dist*, che calcolerà la somma delle distanze (**passo 13**).

Si apre poi un nuovo ciclo iterativo (**passo 14**) che termina nel momento in cui la variaibile j eguaglia la lunghezza del set di benigni.

In questo ciclo viene calcolata la distanza euclidea tra il malware ed il j-

esimo benigno (**passo 15**), la distanza calcolata viene salvata in una lista *edist* (**passo 16**) e sommata alla variabile *sum_dist* (**passo 17**).

Terminato il ciclo, termina anche il costrutto condizionale.

Successivamente viene calcolata la media delle distanze (**passo 20**), caricata in *avg*.

La lista delle distanze viene ordinata con un ordine ascendente (**passo 21**).

Viene inizializzato un *flag* ad 1 (**passo 22**). Tale flag, se pari a 0, consente l'esecuzione della predizione sul malware assunto.

Si apre un nuovo ciclo iterativo (**passo 23**), che termina quando l raggiunge un valore pari alla lunghezza del set di benigni moltiplicato per la perturbazione limite data in input all'algoritmo, nel quale viene verificata che la media delle distanze sia minore del 70% delle distanze contenute in *edist* (**passo 24**). Se la condizione è verificata *flag* viene posto a 0 (**passo 25**).

Al termine del ciclo si pone una condizione sul valore di *flag* (**passo 28**): se flag è 0 viene eseguita la predizione, altrimenti si passa al malware successivo (**passo 29**).

Il risultato della predizione viene caricato nella variabile p.

Successivamente viene posta una condizione su p (**passo 30**): se p è uguale a 0 la variabile *evasion_count* viene incrementata di 1 (**passo 32**), altrimenti si passa al malware successivo. Questo perché se p è uguale a 0, vuol dire che il malware è stato predetto come benigno.

3.3.2 Scenario di attacco basato su distanza di Hamming

Nell'implementazione del poisoning attack basato sulla distanza di Hamming, una volta dato in input il dataset, viene scelto un insieme casuale di malware da essere alterati.

Una volta terminata la scelta, verrà calcolata la distanza di Hamming di ogni malware con tutti i file benigni.

Avendo valori binari, è possibile calcolare la distanza di Hamming mediane lo XOR trai due vettori;

$$hd = 1011011001 \oplus 0100110011 = 111111010 \tag{3.7}$$

$$d(1011011001, 0100110011) = 7 \tag{3.8}$$

I file benigni verranno ordinati, in ordine ascendente, sulla base del valore della distanza. A questo punto solo lo 0.5% dei benigni verrà preso in considerazione per la verifica delle condizioni di alterazione.

Come nell'implementazione precedente, il malware verrà alterano solo se la *feature*, casualmente estratta, sarà presente solo nel file benigno.

Soddisfando tale condizione, il malware verrà alterato, quindi, la *feature* gli verrà aggiunta.

L'algoritmo può essere pensato come rapprensentato in fig. 3.6.

Saranno necessari alcuni input affinchè l'attacco possa essere efficace:

- Dataset (D)
- Test_set, casualmente scelto

- Numero di file .apk benigni (β)
- Limite di perturbazione (δ)

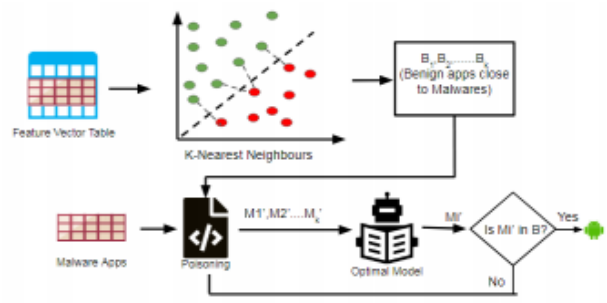


Figura 3.6: Poisoning attack basato sulla distanza di Hamming

L'algoritmo può essere così riassunto: si parte con l'inizializzazione del contatore di iterazioni i a 0 (**passo 1**), con cui si va a definire il termine dell'intero algoritmo.

Successivamente viene avviato un ciclo iterativo (**passo 2**), il quale termina quando la variabile i raggiunge un valore pari alla lunghezza del test set, precedentemente calcolato.

Avviato il ciclo, viene assegnato l'i-esimo malware alla variabile x (**passo 3**) e viene inizializzato a 0 il contatore di iterazioni j (**passo 4**).

Viene quindi avviato un secondo ciclo iterativo (**passo 5**) che termina quando j assume un valore pari al numero di apk presenti nel dataset.

All'interno di questo ciclo viene caricato nella variabile b il j-esimo apk presente nel dataset (**passo 6**) e viene verificato che sia un apk benigno (**passo 7**), sulla base della classe dell'apk.

Se la condizione non viene soddisfatta, si passa all'apk successivo; altrimenti, si procede con il calcolo della distanza di Hamming tra malware e benigno

assunti (**passo 8**). Distanza che viene caricata nella variabile *h*.

Se la distanza non è nulla (**passo 9**), viene aggiunta ad un array bidimensionale (**passo 10**), che prevede l'array benigno nella prima colonna e la relativa distanza nella seconda.

Terminato il ciclo, l'array bidimensionale viene ordinato, in ordine crescente, (**passo 14**) e nella lista *l* viene caricato il beningo con distanza minore, o più benigni se dovessero avere pari distanza (**passo 15**).

Infine, viene azzerata la variabile *j*, che farà da contatore di iterazioni per il ciclo successivo (**passo 16**).

Tale ciclo termina quando *j* assume un valore pari alla lunghezza di *l*.

In questo ciclo viene inizializzato un contatore di alterazioni *c* (**passo 18**), viene caricato nella variabile *b* il j-esimo benigno presente in *l* (**passo 19**) e viene eseguito lo xor, il cui risultato viene caricato nella variabile *a*, tra il benigno assunto ed il malware attuale (**passo 20**).

A questo punto viene casualmente scelto un bit dal risultato dello xor pari ad 1 (**passo 21**). La ricerca del bit pari ad 1 continua fin tanto che: il bit casualmente preso è pari a 0 oppure non sono stati verficati tutti i bit di *a*.

La posizione del bit preso rappresenta la posizione della *feature* sottoposta alla condizione per l'alterazione.

Individuato il bit, viene posta la condizione per l'alterazione (**passo 22**).

Tale condizione consente l'alterazione nel momento in cui la *feature* è presente solo nel benigno altrimenti, si passa al malware successivo.

Se la condizione viene soddisfatta si passa alla fase di alterazione della *feature*: viene forzata ad 1 la *feature* nel malware (**passo 23**).

Successivamente viene incrementato il contatore di alterazioni (**passo 24**).

Terminata l'alterazione viene predetta la classe del malware alterato (**passo 26**). Se il risultato della predizione, caricato nella variabile p, è pari a 0 (**passo 27**) il malware viene aggiunto alla lista di malware evasi e si può passare al malware successivo (**passi 28-29**); altrimenti viene eseguita una nuova alterazione (**passo 31**).

L'alterazione, su uno stesso malware, viene eseguita fin tanto che l'algoritmo di classificazione preveda il malware come benigno oppure il contatore di alterazioni raggiunge il limite di perturbazioni (**passo 30**), arbitrariamente definito.

Una visione completa dell'algoritmo può essere vista in fig. 3.7

Algorithm 2 Poisoning Hamming Distance

Input: $DatasetD, Test\_set, H, \beta, \delta$
Output: Evaded samples

$i \leftarrow 0$ {iteration counter}
repeat
  $x \leftarrow Test\_set[i]$
  $j \leftarrow 0$
  repeat
    $b \leftarrow D[j, 1 : m]$
    if $b[m] == 0$ **then**
      $h \leftarrow hamming\_distance(x, b)$
      if $h \neq 0$ **then**
        $A[j][2] \leftarrow h$ {A is a 2 dimensional array where, 1st column has benign samples 2^{nd} column has the distance to x}
      end if
    end if
  until $j \leq |D|$
  sort A in ascending order of hamming distance
  $l \leftarrow A[i : \beta]$ {l is the 2-dimensional array of benign samples with the shortest distance to malware x}
  $j \leftarrow 0$
  repeat
    $c \leftarrow 0$
    $b \leftarrow l[j]$
    $a = b$XORx
    select a random number γ s.t $a[\gamma] = 1$
    if $b[\gamma] = 1$ and $x[\gamma] = 0$ **then**
      $x[\gamma] \leftarrow 1$
      $c \leftarrow c + 1$
    end if
    $p \leftarrow H\_predict(x)$ {testing classifier with evaded sample}
    if $p \leftarrow 0$ **then**
      $i \leftarrow i + 1$
      goto 2
    else if $c \leq \delta$ **then**
      goto 21
    end if
  until $j \leq |l|$
until $i \leq |Test\_set|$

Figura 3.7: Pseudo-codice basato sulla distanza di Hamming

Scenario di attacco basato su K-Means Clustering

Quest'algoritmo va a suddividere i dati in input, sulla base delle loro caratteristiche, in k cluster [37].

I cluster vengono generati, inizialmente, assumendo k vettori benigni come centroidi, che verranno poi iterativamente ridefiniti. Tale passaggio viene effettuato mediante il calcolo della distanza euclidea tra i centroidi ed i restanti vettori benigni.

Ogni vettore viene poi assegnato al gruppo avente il centroide meno distante. La ricerca di un nuovo centroide si basa sul calcolo della media degli elementi presenti nel cluster.

L'immagine in fig. 3.8 pone una visione del funzionamento di tale algoritmo:

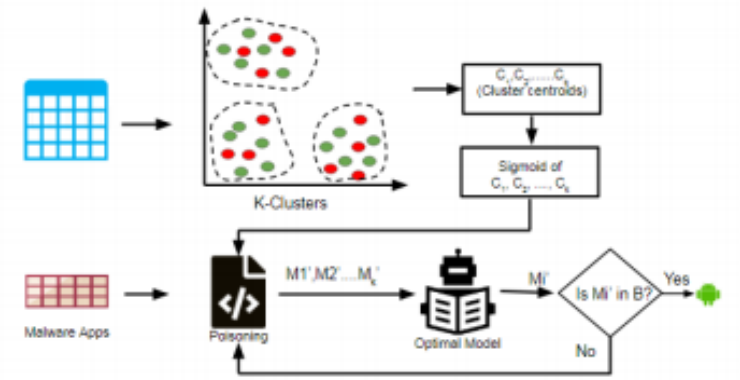


Figura 3.8: Poisoning attack basato su K-Means Clustering

Come per i precedenti, l'algoritmo necessita di specifici input:

- Dataset (D)
- Test set, casualmente scelto

- modello ottimale (H)
- numero di cluster (δ)
- threshold per la funzione sigmoide (τ)

L'algoritmo può essere così riassunto: primo elemento è la definizione di una funzione in grado di individuare i corretti centroidi dei singoli cluster e di suddividere gli elementi del set di benigni in oppotuni cluster (**passi 1 - 6**). La funzione andrà a restituire la lista *cluser_ means*, caricata poi in *centers*, contenente i centroidi corretti.

Viene poi inizializzato un contatore di iterazioni i (**passo 7**), che limiterà l'escuzione dell'intero algoritmo.

Successivamente viene avviato un ciclo iterativo (**passo 8**), che termina nel momento in cui i assume un valore pari al valore limite della perturbazione, arbitrariamente imposto.

Tale ciclo serve per convertire il centroide in un valore binario.

Per prima cosa viene caricato in c l'i-esimo centroide (**passo 9**) e viene inizializzato a 0 il contatore di iterazioni j (**passo 10**). Viene quindi avviato un ciclo iterativo (**passo 11**) che termina quando j assume un valore pari alla lunghezza del centroide.

All'interno del ciclo viene calcolato un valore, caricato in s mediante la funzione sigmoide (**passo 12**). Se il valore calcolato è maggiore del vaore τ di riferimento (**passo 13**) la j-esima posizione del centroide viene forzata ad 1 (**passo 14**); altrimenti viene posto a 0 (**passo 16**).

Una volta che l'operazione è stata ripetuta su ogni posizione del controide, si passa a quello successivo (**passo 18**).

Terminato anche il ciclo iniziale, il contatore di iterazioni i viene azzerato (**passo 20**) e viene aperto un nuovo ciclo. Quest'ultimo termina nel momento in cui i assume un valore pari alla dimensione del test set.

Viene caricato in x l'i-esimo malware (**passo 23**) e viene inizializzato a 0 il contatore di iterazioni j (**passo 25**).

Viene quindi avviato un secondo ciclo che termina quando j assume un valore pari alla dimensione della lista di centroidi (**passo 26**).

Nella variaible c viene caricato il j-esimo centroide (**passo 27**) e, successivamente, viene eseguito uno xor tra il centroide ed il malware considerati (**passo 28**), il cui risultato viene caricato nella variabile a.

A questo punto viene casualmente scelto un bit, dal risultato dello xor, pari ad 1 (**passo 29**). La ricerca del bit pari ad 1 continua fin tanto che: il bit casualmente preso è pari a 0 oppure non sono stati verficati tutti i bit di a.

La posizione del bit preso rappresenta la posizione del bit di centroide e malware sottoposto alla condizione per l'alterazione.

Individuato il bit, viene posta la condizione per l'alterazione (**passo 30**). Tale condizione consente l'alterazione nel momento in cui solo il bit del centroide, in quella posizione, vale 1; altrimenti, si passa al malware successivo.

Se la condizione viene soddisfatta si passa alla fase di alterazione della *feature*: viene forzata ad 1 la *feature* nel malware (**passo 31**).

Viene incrementato il contatore di alterazioni (**passo 32**) e viene eseguita la predizione sul malware alterato (**passo 33**), il cui risultato viene caricato in p.

Se la predizione restituisce il bit 0 (**passo 34**) il malware viene aggiunto alla lista *evaded_ samples*, contenente i malware evasi, e l'algoritmo passa al

malware successivo (**passi 35 - 36**); altrimenti si cerca una nuova posizione (**passo 38**). La ricerca della nuova posizione ha due condizioni di blocco: quando la predizione restituisce bit 0 e quando il contatore di alterazioni raggiunge il limite di perturbazione (**passo 37**).

Raggiunta la condizione di blocco è necessario passare al malware successivo (**passo 41**).

Una più chiara visione dell'implementazione è fornita dallo pseudo-codice in fig. 3.9

Algorithm 3 Poisoning K-Means

Input: *Dataset : D, T est set, classifier : H, # clusters : δ, Threshold : τ*

Output: Evaded Samples

```
1: Function K-Means clustering (Dataset D)
2: initially choose p data points from D as centroids
3: (re)assign each vector in D to the cluster to which it is
   more close to based on the mean value of the object in
   the cluster
4: update the cluster means
5: centres ← cluster_means
6: EndFunction
7: i ← 0
8: repeat
9:     c ← centers[i]
10:    j ← 0
11:    repeat
12:        s ← f[c[j]]
13:        if s > τ then
14:            c[j] = 1
15:        else
16:            c[j] = 0
17:        end if
18:        j ← j + 1
19:    until j ≤ |c|
20:        i ← i + 1
21: until i ≤ δ
22: i ← 0
23: repeat
24:    x ← T[i]
25:    j ← 0
26:    repeat
27:        c ← centers[j]
28:        a ← c XORx
29:        count ← 0
30:        select a random number y s.t a[y]=1
31:        if c[y] = 1andx[y] = 0 then
32:            x[y] ← 1
33:            count ← count + 1
34:            p ← H_predict(x)
35:            if p == 0 then
36:                j ← j + 1
37:                goto 26
38:            else if c < δ then
39:                goto 29
40:            end if
41:        end if
42:        j ← j + 1
43:    until j ≤ |centers|
44: until i ≤ |T|
```

Figura 3.9: Pseudo-codice basato su K-Means Clustering

3.4 Classificatori

In questo paragrafo si andrà a spiegare le caratteristiche dei classificatori utilizzati per la sperimentazione e le metriche adoperate per la valutazione delle loro prestazioni.
Come detto in precedenza, la sperimentazione basa i suoi risultati sul confronto tra tre diversi classificatori: Logistic Regression (LR), Support Vector Machine (SVM), Random Forest (RF).
Per comprendere al meglio i tool utilizzati per la sperimentazione è opportuno analizzarne le caratterstiche.

3.4.1 Logistic Regression

Tale classificatore si avvale di un algoritmo di classificazione basato su dati storici. Ciò vuol dire che maggiore è il numero di dati analizzati, maggiore è l'accuratezza delle sue previsioni.

La Regressione Logistica è uno strumento di classficazione appartenente alla famiglia dei classificatori supervisionati.

Mediante metodi statistici, genera un risultato rapprensentante la probabilità (P) che un determinato valore in input appartenga ad una certa classe. Dove P è un numero compreso tra 0 ed 1.

La Regressione Logistica applicata nella sperimentazione è detta binomiale, in quanto il risultato della predizione può assumere i due valori logici 1 (malware) e 0 (benigno).

Come tutte le analisi di regressione, la Regressione Logistica è un'analisi predittiva, ovvero: misura la relazione tra la variabile dipendente (ciò che si vuole prevedere) e le variabili indipendenti (le caratteristiche note).

La probabilità di appartenenza viene calcolata mediante una ***funzione logistica***.

La predizione sarà, infine, un'analisi del risultato della funzione: se la funzione restituisce una probabilità avente valore al più pari a 0,4 verrà definita la classe 0 come classe di appartenenza; altrimenti la classe di appartenenza sarà la classe 1.

La fig. 3.10 mostra i passaggi utili all'ottenimento della predizione:

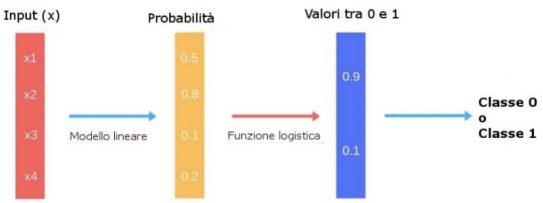


Figura 3.10: Processo predizione Logistic Regression

La Regressione Logistica, in matematica, viene rappresentata mediante la seguente equazione:

$$y = \frac{e^{(b_0+b_1*x)}}{1 + e^{(b_0+b_1*x)}} \tag{3.9}$$

Dove:

- x è il valore in input
- b_0 e b_1 sono i coefficienti dei valori in input
- y è l'output della predizione

La stima dei coefficienti viene effettuata mediante il medoto di **massima verosimiglianza**, che consente la ricerca di valori capaci di minimizzare l'errore delle probabilità viste dal modello.
Ancora, la regressione logistica può essere espressa in termini probabilistici, definendo che:

$$P(X) = P(Y = 1|X) \tag{3.10}$$

Come noto, la probabilità dell'input è il risultato dell'equazione della regressione logistica. Di conseguenza si può affermare che:

$$P(X) = \frac{e^{(b_0+b_1*x)}}{1 + e^{(b_0+b_1*x)}} \tag{3.11}$$

Quindi, eliminando l'esponenziale, si ottiene che:

$$ln(\frac{p(x)}{1 - p(x)}) = b_0 + b_1 * x \tag{3.12}$$

Dove:

$$ln(\frac{p(x)}{1-p(x)}) \tag{3.13}$$

è detto **funzione logit**, mentre

$$E(\frac{p(x)}{1-p(x)}) \tag{3.14}$$

in statistica, è detto **odds**

La funzione logit, rappresentante l'inverso della funzione logistica, consente di associare la probabilità di una predizione ad un'intervallo di numeri reali. Ovvero, è il logaritmo della probabilità che l'output appartenga ad una delle due classi possibili.

L'odds rappresenta il rapporto tra la probabilità di un evento e la probabilità che l'evento non accada [39].

3.4.2 Support Vector Machine

Il classificatore SVM è un'algoritmo di classificazione automatico supervisionato, utile sia per problemi di classificazione che di regressione, le cui prestazioni si massimizzano nei problemi di classificazione binari.

L'idea che si pone alla base dell'SVM è quella di individuare un iperpiano capace di dividere il dataset in due classi.

Quindi, si può dire che alla base dell'SVM vi sono tre concetti chiave:

iperpiano una linea, o un piano, capace di dividere l'intero dataset, come mostrato in fig. 3.11

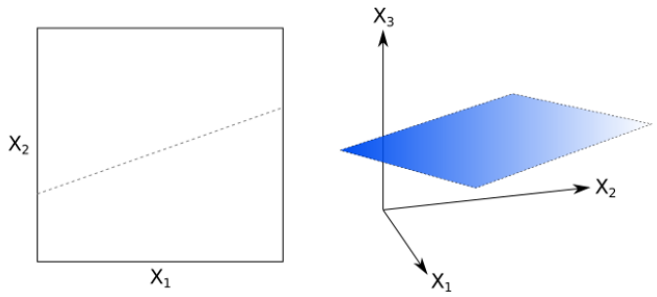


Figura 3.11: Iperpiano

vettori di supporto elemenmti critici del dataset, sono i dati più vicini all'iperpiano, come mostrato in fig. 3.12. Critci perché capaci di alterare la posizione dell'iperpiano stesso nel momento in cui vengono modificati o rimossi

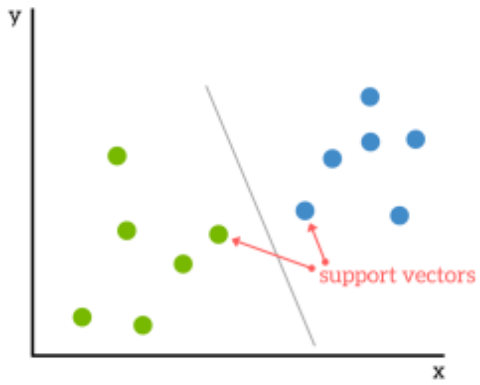


Figura 3.12: Vettori di supporto

margine la distanza trai vettori di supporto, di classi differenti, più vicini all'iperpiano. Il centro del margine si incrocia con l'iperpiano, come mostrato in fig. 3.13.

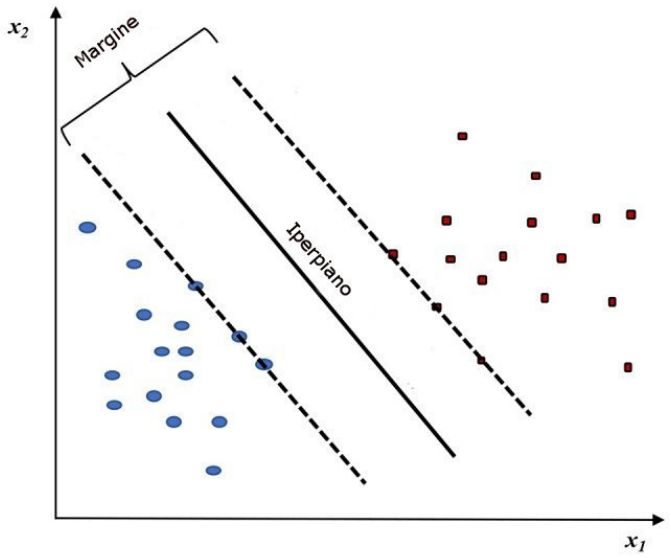


Figura 3.13: Margine

Obbiettivo dell'SVM è quello di identificare l'iperpiano che meglio riesce a dividere i dati presenti nel dataset. Per tale scopo sono necessari due passaggi:

1. ricerca di un **iperpiano linearmente separabile** capace di separare i dati appartenenti a classi diverse. Se vi sono più iperpiani possibili viene scelto quello che garantisce il maggior numero di vettori di supporto.

2. nel caso l'iperpiano non viene trovato, viene applicata una **mappatura non lineare**, con la quale il training set viene posto su una dimensione superiore (passando da due a tre dimensioni, nel caso binario). In tal modo viene garantita la divisione.

L'iperpiano lienarmente separabile risulta essere un concetto importante per l'SVM, che, di conseguenza, necessita di un chiarimeto.

Iperpriano lienarmente separabile

É un tipo di iperpiano capace di semplificare la divisione delle due classi. Si basa sull'individuazione di più iperpiani, teoricamente infiniti, capaci di suddividere il dataset, come mostrato in fig. 3.14.

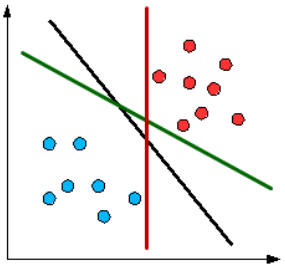


Figura 3.14: Ricerca iperpiano ottimale

Terminata la ricerca si passa all'individuazione dell'iperpiano ottimale, ovvero di quello avente maggiore distanza dai dati del dataset, come mostrato in fig. 3.15.

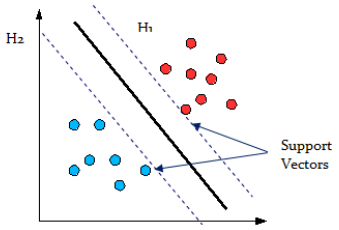


Figura 3.15: Iperpiano ottimale

Nel caso vi sia l'aggiunta di nuovi dati, questi verranno immediatamenti inseriti nel gruppo di appartenenza.

Modello lineare dell'SVM

In matematica è possibile definire le operazioni eseguite dall'SVM mediante un modello lineare.
In tale modello vediamo l'iperpiano come la somma tra un prodotto vettoriale ed uno coefficiente

$$\overrightarrow{w}\ \overrightarrow{x} + w_0 = 0 \tag{3.15}$$

Dove w rapprensenta il vettore di peso, x il vettore delle caratteristiche e w_0 il bias.
Tale equazione può essere scritta in maniera più dettagliata come

$$w_0 + w_1x_1 + w_2x_2 + ... + w_nx_n = 0 \tag{3.16}$$

Quindi è possibile affermare che su n dimensioni, l'iperpiano è combinazione lineare di tutte le dimensioni eguagliate a zero. In un caso binario, i punti al di sopra dell'iperpiano devono rispettare la condizione:

$$w_0 + w_1x_1 + w_2x_2 > 0 \tag{3.17}$$

mentre i punti al di sotto dell'iperpiano devono rispettare la condizione:

$$w_0 + w_1x_1 + w_2x_2 < 0 \tag{3.18}$$

Includendo anche i limiti di margine si ottengono due nuove condizioni:

$$w_0 + w_1 x_1 + w_2 x_2 \geq 0 \rightarrow y = 1 \tag{3.19}$$

$$w_0 + w_1 x_1 + w_2 x_2 \leq 0 \rightarrow y = -1 \tag{3.20}$$

Dove l'etichetta di classe (y) può assumere esclusivamente i valori -1 e 1.

Le due condizioni consentono l'individuazione dei confini di margine.

I vettori di supporto sono quei dati che cadono su tali confini, fig. 3.16.

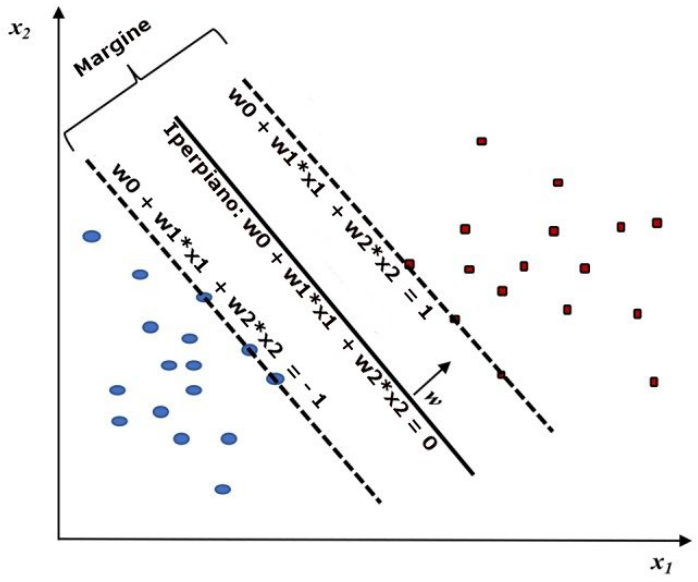


Figura 3.16: Coefficienti di margine

C'è da dire che se w rappresenta il vettore dei pesi ||w|| rappresenta la lunghezza del vettore, la cui massima dimensione è:

$$\frac{1}{|w|} + \frac{1}{|w|} = \frac{2}{|w|} \tag{3.21}$$

Da tale calcolo deriva che minimizzando w si ottiene l'iperpiano ottimale [40].

3.4.3 Random Forest

Come gli altri classificatori, il Random Forest è un algoritmo di classificazione supervisionato.
Si basa sulla combinazione di più alberi decisionali che, singolarmente, mostrano una poca accuratezza nella predizione, accuratezza che viene incrementata nel momento in cui le predizioni vengono combinate tra loro.
Il risultato del classificatore non è altro, quindi, che la media del risultato numerico ottenuto dai vari alberi decisionali su cui si basa, per un problema di regressione, o la classe restituita dal maggior numero di alberi decisionali per un problema di classificazione.
Il modello di classificazione si basa su due concetti fondamentali, da cui deriva il nome random:

- campionamento casuale di punti dati di allenamento durante la costruzione di alberi
- sottoinsiemi casuali di funzionalità considerate durante la divisione dei nodi

Campionamento casuale di punti dati di allenamento durante la costruzione di alberi

Durante la fase di training, ogni albero decisionale si allena su un campione casuale di dati presi dal dataset. É facile che un albero utilizzi più volte uno stesso campione.

L'idea alla base di questo addestramento è che utilizzando campioni causali si ha una varianza più elevata rispetto all'uso di campioni scelti in serie. Si ottiene inoltre, nel complesso, una varianza inferiore di un caso ordinato, senza incrementare la distorsione dei risultati.
Dalla media dei singoli alberi decisionali si otterrà poi il risultato finale, come mostrato in fig. 3.17.

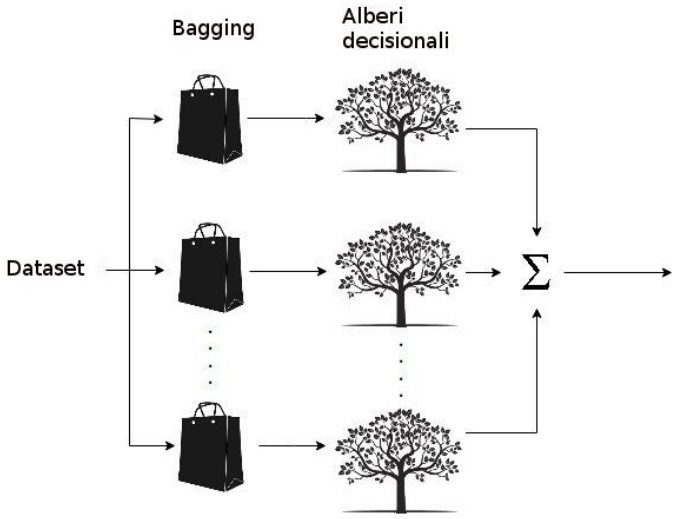


Figura 3.17: Processo predizione Random Forest

Sottoinsiemi causali di funzionalità considerate durante la divisione dei nodi

Altro concetto fondametale, alla base del Random Forest, è che, in ogni albero decisionale, ciascun nodo debba essere diviso sulla base di caratteristiche casualmente assunte.
In genere viene preso in considerazione un numero di caratteristiche pari alla

radice quadrata delle caratteristiche totali. Ad esempio, su un totale di 16 caratteristiche ne verranno prese in considerazione solo 4, casualmente assunte.

Si può quindi affermare che il Random Forest combina un numero significativo di alberi decisionali, allenati con insiemi di osservazioni che differiscono di poco l'uno dall'altro, suddividendo i nodi dei singoli alberi sulla base di un numero limitato di caratteristiche [41].

3.5 Metriche di classificazione

Fino ad ora si è discusso dei classificatori, dei loro processi di funzionamento e delle loro caratteristiche, ma come viene valutato l'output della classificazione e della predizione?

A tal proposito siamo soliti applicare diverse metriche di prestazioni, tra le quali vi sono alcune fondamentali:

Precision il rapporto tra il numero di previsioni corrette di una classe ed il numero di volte che tale classe viene prevista. Calcolato come

$$precision = \frac{vero\_positivo}{vero\_positivo + falso\_positivo} \tag{3.22}$$

Per un esempio di precisione si può pensare ad un antifurto. Esso risulta essere molto preciso nel momento in cui suona solo se recepisce un ladro. Occorre però porre attenzione sul fatto che la precisione non diminuisce se a volte non rileva il ladro, l'importante e che non rilevi altre forme viventi. Questa precisazione è dovuta al fatto che tale metrica si basa esclusivamente sui veri o falsi positivi, ovvero solo quando l'algoritmo prevede l'evento [42], come mostrato in fig. 3.18.

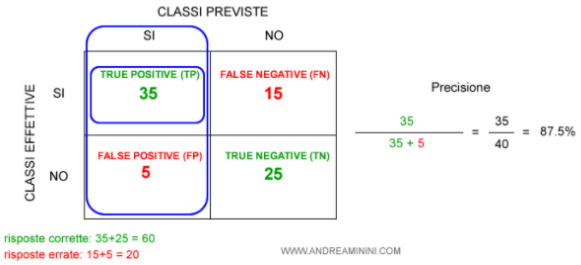


Figura 3.18: Precision

Recall misura la sensibilità del modello, mediante il rapporto tra le previsioni corrette, su una classe, ed il numero totale di volte in cui l'evento si verifica.

$$recall = \frac{vero\_positivo}{vero\_positivo + falso\_negativo} \tag{3.23}$$

Tornando all'esempio dell'antifurto, il valore di recall varia sulla base del numero di volte in cui l'algoritmo è stato capace di individuare l'ingresso del ladro in casa [42], come mostrato in fig. 3.19.

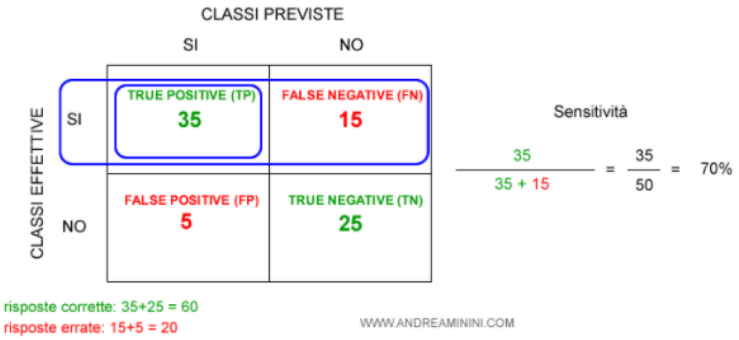


Figura 3.19: Recall

Accuracy rappresenta la percentuale delle istanze correttamente classificate. Problema di questa metrica è che può essere fuorviante nel momento in cui i dati sottoposti a classficazione risultano non essere bilanciati [43].

Detection Rate indica la capacità di individuazione del malware, in questo scenario, da parte dell'algoritmo di machine learning. Presenta la

seguente formula:

$$detection\_rate = \frac{n\_test\_sample - n\_sample\_evasi}{n\_test\_sample} \tag{3.24}$$

Le prime tre metriche vengono calcolate sia prima che dopo l'attacco; il detection rate, invece, viene calcolato esclusivamente dopo.

3.6 Analisi dei risultati

I seguenti risultati sono stati ottenuti mediante una duplice esecuzione degli algoritmi.

La prima esecuzione ha consentito la produzione del test set, contenente i malware che l'algoritmo è riuscito ad alterare.

La seconda esecuzione è stata utile al fine di individuare i malware evasi, sulla base delle predizioni ottenute. In particolare questa esecuzione è stata ripetuta per ogni classificatore.

Mediante le analisi effettuate è stato possibile osservare i valori di Precision, Accuracy e Recall dei tre classificatori adoperati: Logistic Regression (LR), Random Forest (RF) e Support Vector Machine (SVM). Le analisi sono state effettuate mediante il software Weka. Dall'analisi eseguita, preso l'intero dataset come trainig set (2416 sample), sono stati ottenuti i risultati mostrati nella tabella 3.1.

Classificator	Precision	Recall	Accuracy (%)
LR	0,785	0,776	77,649
SVM	0,764	0,733	73,649
RF	0,881	0,875	87,459

Tabella 3.1: Risultati analisi prima dell'attacco

Invece, per la costruzione dei modelli relativi agli algoritmi basati sul K-Means Clustering e sulla distanza di Hamming è stato adoperato l'intero dataset, prima addestrati e successivamente ri-valutati sul test set; mentre per l'algoritmo basato sulla distanza euclidea la predizione è stata ottenuta con uno split del dataset in: 90% training set ed il restante 10% test set. Una volta eseguita la predizione, è stata possibile la valutazione dei classificatori

in termini di accuracy e detection rate.

Nello specifico, nel caso di un attacco di poisoning di tipo K-Means Clustering, con un test set comprendente 1120 malware, si ottengono i risultati riportati nella tabella 3.2.

Classificator	Evaded Sample	Detection Rate
LR	820	0,27
SVM	802	0,28
RF	957	0,14

Tabella 3.2: Risultati poisoning di tipo K-Means Clustering

Nel caso, invece, di un attacco di poisoning di tipo Hamming distance, con un test set comprendente 567 malware, ha restituito i risultati riporati nella tabella 3.3.

Classificator	Evaded Sample	Detection Rate
LR	321	0,43
SVM	328	0,42
RF	489	0,14

Tabella 3.3: Risultati poisoning di tipo Hamming distance

Nel caso, infine, di un attacco di poisoning basato su distanza euclidea con un test set comprendente 242 malware alterati, ha restituito i risultati riportati nella tabella 3.4.

Clasificator	Evaded Sample	Detection Rate
LR	139	0,43
SVM	163	0,33
RF	149	0,38

Tabella 3.4: Risultati poisoning di tipo Euclidean distance

Capitolo 4

Conclusioni

Dalle analisi effettuate si evincono sostanziali differenze tra l'analisi prima dell'attacco e quella dopo l'attacco. Da subito è possibile individuare come l'analisi dopo l'attacco ha prodotto differenti risultati per ogni algoritmo di machine learning.

Poichè gli output delle metriche e del conteggio delle evasioni differsiscono per ogni algoritmo e per ogni classificatore, è bene argomentarle separatamente. Alla fine dell'esposizione delle differenze sarà possibile definire quale algoritmo si è dimostrato più robusto rispetto l'attacco subito e con quale classificatore.

Dalle analisi è inoltre possibile definire l'algoritmo di poisoning più efficace. In particolare un algoritmo di poisoning si definisce più efficace se presenta il maggior numero di malware evasi; mentre, un algoritmo di machine learning, sottoposto ad un attacco di tipo poisoning, si definisce più robusto se presenta un detection rate più alto.

Dal punto di vista della robustezza degli algoritmi, rispetto l'attacco effet-

tuato, è necessario avere una visione completa, basata sul detection rate.
Per quanto riguarda l'algoritmo basato sul K-Means Clustering si nota un detection rate pari a: 0,27 con il classificatore LR, 0,28 con il SVM e 0,14 con il RF. Si può quindi notare che il modello costruito con il classificatore di tipo RF risulta il meno robusto.
Per quanto riguarda l'algoritmo basato sulla distanza di Hamming si nota un detection rate pari a: 0,43 con il LR, 0,42 con il SVM e 0,14 con il RF. In tal caso l'attacco presenta maggiore efficacia nel caso del modello costruito con il classificatore RF.
Per quanto riguarda, infine, l'algoritmo basato sulla distanza euclidea si nota un detection rate pari a: 0,43 con il LR, 0,33 con il SVM e 0,38 con il RF. In tal caso l'algoritmo di machine learning meno robusto risulta quello costruito con il classificatore SVM.
Sulla base di tutti i risultati ottenuti è possibile definire quale algoritmo si dimostra maggiormente robusto rispetto ad un attacco di tipo poisoning.
Nonostante le differenze tra gli output degli algoritmi per ogni classificatore siano sostanzialmente diverse, è facile notare che gli attacchi che hanno dimostrato una maggiore efficacia sono stati quelli basati sulla distanza di Hamming e sul K-Means Clustering, nell'analisi mediante il classificatore RF, con una parcentuale di malware evasi pari all'86%; mentre gli algoritmi basati sulla distanza euclidea e sulla distanza di Hamming, entrambi con il classificatore LR, si sono dimostrati gli algoritmi più robusti, in quanto aventi il detection rate più elevato.

Bibliografia

[1] www.kaspersky.com/blog/insecure-android-devices/10296

[2] www.mcafee.com/enterprise/en-in/threat-center/global-threatintelligence-technology.html

[3] https://gs.statcounter.com/os-market-share/mobile/worldwide

[4] Hossien Sayadi, Nisarg Patel, Sai Manoj P D, Avesta Sasan, Setareh Rafatirad and Houman Homayoun, "Ensemble Learning for Effective Run-Time Hardware-Based Malware Detection: A Comprehensive Analysis", ACM/ESDA/IEEE Design Automation Conference (DAC), 2018

[5] Monica Kumaran and Wenjia, "LiLightweight malware detection based on machine learning algorithms and the android manifest file", Published in: I (URTC), 2016

[6] Ambra Demontis, Marco Melis, Battista Biggio, Davide Maiorca, Daniel Arp, Konrad Rieck, Igino Corona, Giorgio Giacinto and Fabio Roli, "Yes, Machine Learning Can Be More Secure! A Case Study on Android Malware Detection", arXiv:1704.08996v1 [cs.CR], 2017

[7] Justin Sahs and Latifur Khan, “A Machine Learning Approach to Android Malware Detection”, European Intelligence and Security Informatics Conference, 2012

[8] Eli (Omid) David, Nathan S. and Netanyahu, ”DeepSign: Deep Learning for Automatic Malware Signature Generation and Classification”,International Joint Conference on Neural Networks (IJCNN), pages 1–8, Killarney, Ireland, 2015

[9] Yao Wang, Wan-dong Cai and Peng-cheng Wei,”A deep learning approach for detecting malicious JavaScript code”, SECURITY AND COMMUNICATION NETWORKS, Security Comm. Networks; 9:1520–1534, 2016

[10] Alec Radford, Luke Metz and Soumith Chintala, ”Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks”, ICLR 2016

[11] Parvez Faruki, Ammar Bharmal, Vijay Laxmi, Vijay Ganmoor, Manoj Singh Gaur and Mauro Conti, ”Android Security: A Survey of Issues, Malware Penetration, and Defenses”, IEEE Communication Surveys & Tutorials, vol. 17, NO. 2, 2015

[12] Sen Chen, Minhui Xue, Lingling Fan, Shuang Hao, Lihua Xu, Haojin Zhu, and Bo Li, ”Automated Poisoning Attacks and Defenses in Malware Detection Systems: An Adversarial Machine Learning Approach”, Elsevier Computers & Security, Volume 73, 2018

[13] V. Rastogi, Y. Chen, and X. Jiang, "Droidchameleon: Evaluating android anti-malware against transformation attacks", ACM SIGSAC Symposium on Information, Computer and Communications Security, ASIA CCS '13, , 329-34, 2013

[14] Guillermo Suarez-Tangil1, Santanu Kumar Dash1, Mansour Ahmadi2, Johannes Kinder1, Giorgio Giacinto2 and Lorenzo Cavallaro1, "DroidSieve: Fast and Accurate Classification of Obfuscated Android Malware", Seventh ACM on Conference on Data and Application Security and Privacy,pages 309-320, 2017

[15] Gerardo Canfora, Andrea Di Sorbo, Francesco Mercaldo and Corrado Aaron Visaggio, "Obfuscation techniques against signature based detection: a case study", Mobile Systems Technologies Workshop (MST), 2016

[16] Andrea Saracino, Daniele Sgandurra, Gianluca Dini and Fabio Martinelli, "MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention", IEEE Transactions on Dependable and Secure Computing, 2016

[17] Xi Xiao, Shaofeng Zhang, Francesco Mercaldo, Guangwu HuEmail and Arun Kumar Sangaiah, "Android malware detection based on system call sequences and LSTM", Multimedia Tools and Applications, vol. 78, n. 4, p. 3979–3999,Springer, 2019

[18] Saba Arshad, Munam A. Shah, Abdul Wahid, Amjad Mehmood, Houbing Song and Hongnian Yu, "SAMADroid: A Novel 3-Level Hybrid

Malware Detection Model for Android Operating System", IEEE. Translations, vol. 6, 2018

[19] https://androzoo.uni.lu/api_doc

[20] https://github.com/Saket-Upadhyay/Android-Permission-Extraction-and-Dataset-Creation-with-Python

[21] https://ieeexplore.ieee.org/abstract/document/8782574

[22] https://dl.acm.org/doi/abs/10.1145/2046684.2046692

[23] https://www.morganclaypool.com/doi/abs/10.2200/S00861ED1V01-Y201806AIM039

[24] https://arxiv.org/abs/1611.01236

[25] http://proceedings.mlr.press/v97/guo19a.html

[26] https://ieeexplore.ieee.org/abstract/document/1004360/authorsauthors

[27] https://www.mdpi.com/2504-4990/2/4/30

[28] https://pralab.diee.unica.it/en/PoisoningAttacks

[29] https://analyticsindiamag.com/what-is-poisoning-attack-why-it-deserves-immediate-attention/

[30] Xuanzhe Liu, Huoran Li, Xuan Lu, Tao Xie, Qiaozhu Mei, Feng Feng, and Hong Mei, "Mining Behavioral Patterns from Millions of Android Users", IEEE Transactions on Software Engineering, 2017

[31] Sen Chen, Minhui Xue, Lingling Fan, Shuang Hao, Lihua Xu, Haojin Zhu, and Bo Li, "Automated Poisoning Attacks and Defenses in Malware Detection Systems: An Adversarial Machine Learning Approach", Elsevier Computers Security, Volume 73, 2018

[32] Sen Chen, Minhui Xue, Zhushou Tang, Lihua Xu and Haojin Zhu, "StormDroid: A Streaminglized Machine Learning-Based System for Detecting Android Malware", ASIA CCS'16, Xi'an, China, 2016

[33] Sen Chen, Minhui Xue, Zhushou Tang, Lihua Xu and Haojin Zhu, "StormDroid: A Streaminglized Machine Learning-Based System for Detecting Android Malware", ASIA CCS'16, Xi'an, China, 2016

[34] Weilin Xu, Yanjun Qi, and David Evans, "Automatically Evading Classifiers A Case Study on PDF Malware Classifiers", NDSS '16, 21-24 February , San Diego, CA, USA, 2016

[35] Fei Zhang, Patrick P. K. Chan, Battista Biggio, Daniel S. Yeung, and Fabio Roli, "Adversarial Feature Selection Against Evasion Attacks", IEEE Transactions on Cybernetics, VOL. 46, NO. 3, 2016

[36] https://www.unb.ca/cic/datasets/andmal2020.html

[37] Kiri Wagstaff, Claire Cardie, Seth Rogers and Stefan Schroedl, "Constrained K-means Clustering with Background Knowledge"

[38] https://github.com/skylot/jadx

[39] https://www.lorenzogovoni.com/regressione-logistica/

[40] https://www.lorenzogovoni.com/support-vector-machine/

[41] https://www.lorenzogovoni.com/random-forest/

[42] https://www.andreaminini.com/ai/machine-learning/la-differenza-tra-precision-e-recall

[43] https://docs.microsoft.com/it-it/azure/machine-learning/classic/evaluate-model-performance

[44] https://openai.com/blog/adversarial-example-research/