



UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

DIPARTIMENTO DI INFORMATICA

CORSO DI LAUREA IN INFORMATICA

TESI DI LAUREA
IN

Analisi dei Pattern delle Chiamate Api per la Classificazione Malware

RELATORI:

Prof.ssa Annalisa Appice
Dr. Giuseppina Andresini

LAUREANDO:

Giacomo Gaudio

ANNO ACCADEMICO 2024 2025

Indice

Sommario	3
1 Introduzione	4
1.1 Formato Windows PE malware	5
1.2 Categorie Malware	6
1.3 Chiamata Api Windows	7
1.4 Contributi	8
2 Stato dell'Arte	9
2.1 Analisi Statica e Dinamica	9
2.2 Machine Learning	11
2.3 Classificazione	12
2.3.1 Algoritmi di Classificazione	13
2.4 Metriche di valutazione	15
2.5 Approcci di Machine Learning per Windows PE Malware Detection	16
Bibliografia	17

Disclaimer

Tutti i marchi, nomi commerciali, prodotti e loghi menzionati in questa tesi sono di proprietà dei rispettivi titolari. L'autore non rivendica alcun diritto di proprietà su tali marchi o nomi commerciali e li utilizza solo a scopo informativo e descrittivo, senza alcun intento di violazione.

Sommario

Andrà inserito il riassunto di tutta la tesi.

Capitolo 1

Introduzione

Con l'avvento dell'era digitale, attività e servizi, in ambito e-commerce, pubblica amministrazione, servizi finanziari e sanitari, vengono sempre di più digitalizzati in Italia. Secondo i dati ISTAT, l'86,2% delle famiglie italiane dispone di un accesso ad internet a testimonianza di un uso sempre più popolare [1]. Tuttavia, l'aumento della connettività comporta inevitabilmente anche una maggiore esposizione alle minacce informatiche. Nel 2024, l'Italia ha registrato il 10% degli attacchi informatici globali; secondo il rapporto Clusit, circa un terzo di tali attacchi è stato veicolato tramite l'utilizzo di malware [2]. Per malware si intende un'istanza di un programma il cui intento sia malevole come carpire informazioni personali e/o arrecare danni ai dispositivi [3]. Per poter identificare i malware esistono due tipi di analisi: statica e dinamica. L'analisi statica prevede l'analisi del codice binario, analizzando ogni ramo di esecuzione possibile alla ricerca di codice malevolo. L'analisi dinamica invece prevede l'esecuzione del programma all'interno di una sandbox, ovvero un ambiente isolato e controllato che impedisce al software di arrecare danni reali al sistema. In questo ambiente è possibile osservare i suoi comportamenti alla ricerca di quelli simili ai malware [4]. Con riferimento alle informazioni veicolate dalle API (Application Programming Interface), per comportamenti intendiamo l'insieme (ordinato) delle chiamate API al Kernel per permettere l'esecuzione dell'applicativo; un API è un modo per interagire con un sistema senza l'ausilio di una interfaccia grafica [5]. Un singolo "comportamento" è effettivamente la singola chiamata API prendendo il nome di "API Call"; mentre la lista (ordinata) prende il nome "API call sequence" [6]. L'obiettivo di questa tesi è classificare un programma in base ai suoi comportamenti. Per raggiungere tale scopo è stato formalizzato e valutato un approccio che apprende un classificatore, utilizzando sequenze di chiamate API come input e algoritmi di machine learning per distinguere le varie categorie.

1.1 Formato Windows PE malware

Per poter essere eseguito in *Microsoft Windows*, un programma deve presentare una logica applicativa interpretabile dal sistema operativo. A tale scopo, il sistema operativo Windows utilizza il formato **PE** (Portable Executable), così denominato in quanto progettato per non essere vincolato a una specifica architettura [7]. Un file in formato **PE** ha nel seguente ordine le seguenti intestazioni:

1. **Stub MS-DOS**: area iniziale del file PE, compatibile con il sottosistema **MS-DOS**. In assenza di istruzioni specifiche da parte dello sviluppatore, contiene un programma di default che visualizza il messaggio: “Impossibile eseguire questo programma in modalità DOS”.
2. **Firma PE**: campo che identifica il file come appartenente al formato **Portable Executable (PE)** e ne consente il corretto riconoscimento da parte del loader di Windows.
3. **Intestazione COFF**: fornisce informazioni fondamentali sul file, come il tipo di macchina su cui è destinato a essere eseguito, il numero di sezioni e la data di compilazione.
4. **Intestazione facoltativa (solo immagine)**: sezione che specifica le informazioni necessarie al caricamento ed esecuzione del programma. È composta da tre sottosezioni principali:
 - **Campi standard intestazione facoltativi** : includono l’indirizzo di entry point del programma, la dimensione del codice e dei dati, e altri parametri di base.
 - **Campi specifici dell’intestazione facoltativa di Windows**: contengono informazioni più dettagliate, come la versione minima del sistema operativo richiesto, le dimensioni massime di heap e stack, e i valori di allineamento della memoria.
 - **Directory dati intestazione facoltative**: riferimenti alle tabelle e alla loro dimensione delle risorse esterne come le ddl.

Una ddl è una libreria che contiene codice e dati utilizzabili da più di un programma contemporaneamente. Ogni programma può utilizzare le ddl del sistema operativo Windows per ottenere memoria, accesso a risorse, far apparire elementi a schermo ed accedere alla rete [8]. I malware utilizzano queste librerie per poter eseguire comportamenti malevoli.

1.2 Categorie Malware

La definizione di *malware* è generica e include tutte le tipologie di software malevolo; per agevolare le attività di rilevamento e classificazione, è utile suddividerli in categorie più specifiche sulla base delle loro finalità e modalità di azione. Nel contesto di questo lavoro di tesi, sono state individuate e considerate le seguenti specializzazioni di software:

- **Unknown:** applicazioni per le quali non è stato possibile determinare con certezza se sono benevoli o meno.
- **Goodware:** software legittimo, privo di finalità malevoli.
- **Malware:** denominazione generica per software malevolo, utilizzata nei casi in cui non sia possibile definire una categoria più specifica.
- **Backdoor:** programma che crea un accesso nascosto al sistema compromesso, permettendo all'attaccante di controllarlo da remoto [9].
- **Trojan:** software che si maschera da applicazione legittima ma che, una volta eseguito, consente operazioni malevole come la modifica o la cancellazione dei dati [10].
- **Virus:** programma in grado di infettare altri file e diffondersi all'interno di un sistema, con lo scopo di danneggiare il corretto funzionamento dell'host [10].
- **Worm:** simili ai virus, ma in grado di diffondersi autonomamente attraverso la rete, senza richiedere l'intervento dell'utente [10].
- **Dropper:** componente malevolo progettato per scaricare o installare altri malware sul sistema target [11].
- **Spyware:** software che raccoglie informazioni sensibili dall'utente o dal sistema infetto e le trasmette a un'entità esterna [10].
- **Adware:** programma che raccoglie informazioni sulle abitudini dell'utente al fine di proporre pubblicità mirata; in alcuni casi può deviare la navigazione verso siti malevoli [10].
- **Packed:** malware compresso (almeno una volta) tramite tecniche di *packing* per eludere i controlli degli antivirus [12].

1.3 Chiamata Api Windows

Le chiamate API (Application Programming Interface) di sistema consentono agli sviluppatori di accedere alle risorse della macchina necessarie per l'esecuzione di un applicativo. Attraverso tali interfacce è possibile interagire con componenti fondamentali come la rete, la memoria, i dispositivi di input/output e l'interfaccia grafica. Le API risultano fortemente dipendenti dal sistema operativo su cui vengono eseguite. In ambiente Windows, le API garantiscono che un'applicazione possa funzionare correttamente su diverse versioni del sistema operativo, sfruttando al tempo stesso le specifiche funzionalità introdotte nelle specifiche versioni [13]. Le API di Windows coprono un ampio spettro di funzionalità, tra cui: interfaccia utente e shell, gestione dell'input e della messaggistica, accesso ai dati e archiviazione, diagnostica, grafica e multimedialità, dispositivi, servizi di sistema, sicurezza e identità, installazione e manutenzione di applicazioni, amministrazione e gestione del sistema, rete e connettività Internet.

Un esempio rappresentato dal malware di tipo **Spyware**, che può invocare ripetutamente la funzione `GetKeyboardState` [14] per recuperare lo stato degli ultimi 256 tasti premuti dall'utente, potenzialmente contenenti credenziali sensibili. Nella documentazione ufficiale, i requisiti della funzione sono riassunti in Tabella 1.1.

Tabella 1.1: Requisiti

Voce	Valore
Client minimo supportato	Windows 2000 Professional (solo app desktop)
Server minimo supportato	Windows 2000 Server (solo app desktop)
Piattaforma di destinazione	Windows
Intestazione	<code>winuser.h</code> (include <code>Windows.h</code>)
Libreria	<code>User32.lib</code>
DLL	<code>User32.dll</code>
Set di API	<code>ext-ms-win-ntuser-rawinput-l1-1-0</code> (Windows 10, versione 10.0.14)

Come si nota, alla voce **ddl** è indicata la libreria `User32.dll`. Questa verrà referenziata nella sezione *Directory dati dell'intestazione facoltativa* dell'eseguibile corrispondente, permettendo al sistema operativo di risolvere la chiamata in fase di esecuzione.

Nel contesto di questo lavoro di tesi, non siamo interessati alla presenza di una singola chiamata API in sé, quanto piuttosto alla frequenza con cui tali chiamate compaiono all'interno delle sequenze. È infatti la distribuzione delle invocazioni a costituire la base per la classificazione del comportamento del software.

1.4 Contributi

La presente tesi è articolata in quattro capitoli principali: *Introduzione*, *Stato dell'Arte*, *Approccio Proposto*, *Validazione Empirica* e *Appendice*.

Nel capitolo di *Introduzione* vengono presentati il contesto di riferimento, le motivazioni, l'obiettivo del lavoro e il contributo originale fornito.

Il capitolo dedicato allo *Stato dell'Arte* offre un inquadramento teorico e metodologico, approfondendo i principali approcci di analisi del software (statica e dinamica), i fondamenti di *machine learning* con particolare attenzione al compito di classificazione, gli algoritmi utilizzati (Random Forest [15] e XGBoost [16]) e le metriche di valutazione. La sezione si conclude con una panoramica dei lavori correlati nel campo della rilevazione di malware in ambiente Windows.

Nel capitolo *Approccio Proposto* viene descritta in dettaglio la soluzione implementata. Si illustra la modellazione mediante diagrammi UML, l'architettura software sviluppata in Python secondo i principi della programmazione a oggetti.

Successivamente, nel capitolo *Validazione Empirica* verranno presentati i dataset adottati, la standardizzazione adottata, le metriche scelte per la valutazione, i risultati sperimentali ottenuti.

Nel penultimo capitolo, *Conclusioni e Sviluppi Futuri*, verranno tratte le conclusioni del lavoro svolto e saranno delineati alcuni possibili ampliamenti e direzioni di ricerca futura.

Infine, l'*Appendice* contiene note tecniche relative all'ambiente di sviluppo e ulteriori dettagli implementativi, utili a comprendere lo sviluppo effettivo.

Capitolo 2

Stato dell'Arte

In questo capitolo vengono presentati i fondamenti concettuali per il riconoscimento dei malware sulla base dei loro comportamenti. Saranno introdotte le due principali tecniche di analisi, statica e dinamica, mettendone in evidenza punti di forza e limitazioni. In seguito, l'attenzione verrà posta sull'analisi dinamica, approccio adottato in questa tesi, illustrandone i principi di machine learning adoperati per la sua realizzazione.

2.1 Analisi Statica e Dinamica

L'analisi statica si concentra sull'esame del codice del programma alla ricerca di specifiche sequenze di istruzioni, denominate *virus signature* (firma del virus) [17]. L'efficacia dei sistemi basati su questa tecnica dipende dalla dimensione e dall'aggiornamento del database delle firme e dall'impossibilità, per il software, di modificare nel tempo la propria struttura. Per aggirare tali sistemi di rilevamento, i creatori di malware hanno sviluppato diverse tecniche di offuscamento, che consentono di generare firme differenti rispetto a quelle originarie, rendendo più complessa la loro individuazione. Tra le principali tecniche ritroviamo:

- **Dead Code Insertion** (inserimento di codice morto): aggiunta di istruzioni prive di funzionalità ai fini malevoli, il cui unico scopo è alterare la firma rilevata [17].
- **Code Transposition** (trasposizione del codice): riordinamento delle istruzioni, in modo che la rappresentazione binaria risulti differente pur mantenendo inalterata la logica di esecuzione [17].
- **Register Reassignment** (riassegnazione dei registri): modifica dei registri utilizzati nelle operazioni interne al programma [17].
- **Instruction Substitution** (sostituzione delle istruzioni): rimpiazzo di blocchi di codice con altri semanticamente equivalenti [17].

Il continuo sviluppo di tecniche di offuscamento e l'aggiornamento dei database di firme ha dato vita a un vero e proprio ciclo vizioso, in cui l'evoluzione dei malware procede di pari passo con quella dei sistemi di rilevamento. Tale approccio, tuttavia, tende ad allontanarsi da ciò che realmente caratterizza un malware: i suoi comportamenti.

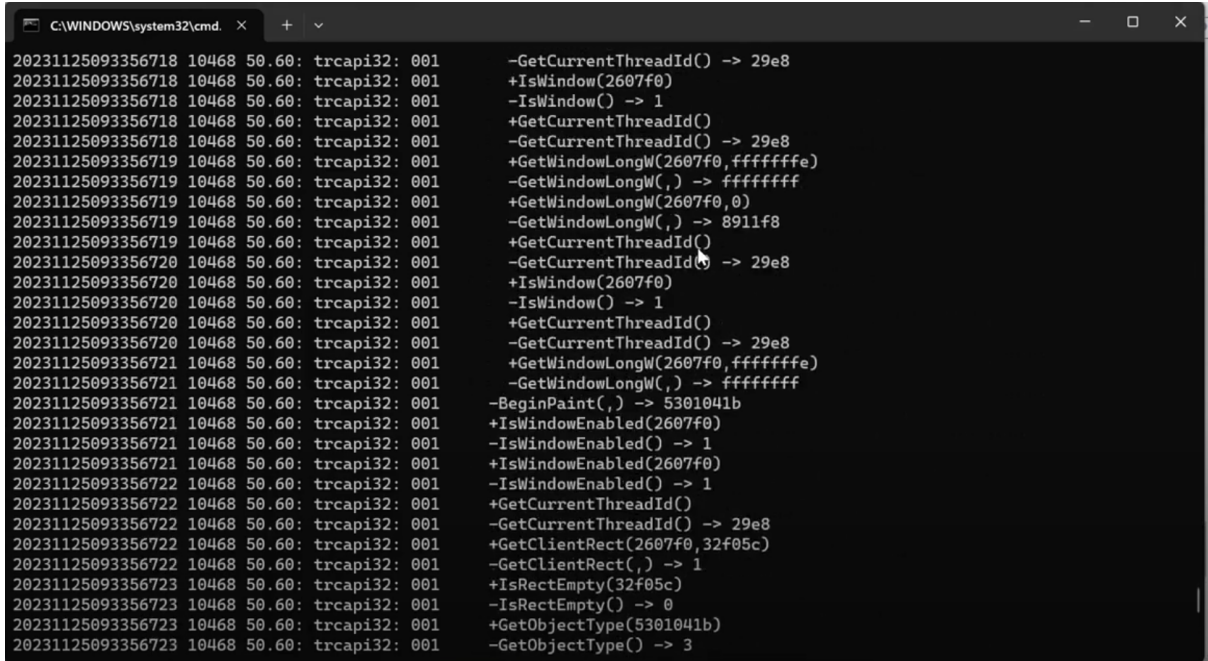
L'analisi dinamica rompe il ciclo generato dai limiti dell'approccio statico, poiché osserva i comportamenti effettivi del software durante l'esecuzione. In questo modo non solo riduce l'impatto delle tecniche di offuscamento sul codice, ma consente anche di evidenziare quei casi limite che l'analisi statica difficilmente riesce a cogliere, risultando complessivamente più efficace [18]. Per poter eseguire un'analisi dinamica, esistono diverse tecniche:

- **Hooking** (Aggancio): intercetta chiamate API per monitorare o modificare il comportamento di un programma. Utilizzato spesso dagli antivirus per controllare processi sospetti. Limite: i malware possono rilevare la presenza di hook tramite checksum o controlli interni [18]. Un software per intercettare le chiamate API di Windows è *Detours* [19].
- **Dynamic Binary Instrumentation** (Strumentazione Dinamica del Binario): analizza e modifica il codice binario mentre il programma è in esecuzione, permettendo di osservare ogni istruzione o comportamento in tempo reale. Più difficile da rilevare rispetto all'hooking [18]. Un software che permette tale realizzazione è *Dynamorio* [20].
- **Virtualization** (Virtualizzazione): esegue un sistema operativo guest su hardware reale (host) isolato, sfruttando meccanismi di separazione. Veloce e relativamente sicuro, ma alcuni malware possono rilevare VM tramite differenze hardware/software [18].
- **Application Level Emulation** (Emulazione a Livello di Applicazione): crea un ambiente finto che emula il sistema operativo e l'architettura di istruzioni per singole applicazioni. Utile per analisi real-time e unpacking¹, ma limitato nella fedeltà: i malware possono rilevare l'emulazione usando istruzioni o API rare [18].
- **Whole System Emulation** (Emulazione dell'Intero Sistema): emula completamente l'hardware di un PC, permettendo l'installazione di un intero sistema operativo guest. Molto utile per osservare comportamenti complessi, unpacking e analisi dettagliata, ma più lento rispetto alla virtualizzazione [18].

Un esempio concreto di implementazione della tecnica di **hooking** è presentato nell'articolo di *Wtrace* [21], dove l'utilizzo della libreria *Detours* permette di intercettare le

¹Per *unpacking* si intende il processo di estrazione del codice originario

chiamate API di un processo in esecuzione. Il risultato di tale analisi può essere osservato nella Figura 2.1, che mostra un output delle API Call registrate di un programma in esecuzione:



```

20231125093356718 10468 50.60: trcapi32: 001 -GetCurrentThreadId() -> 29e8
20231125093356718 10468 50.60: trcapi32: 001 +IsWindow(2607f0)
20231125093356718 10468 50.60: trcapi32: 001 -IsWindow() -> 1
20231125093356718 10468 50.60: trcapi32: 001 +GetCurrentThreadId()
20231125093356718 10468 50.60: trcapi32: 001 -GetCurrentThreadId() -> 29e8
20231125093356719 10468 50.60: trcapi32: 001 +GetWindowLongW(2607f0,fffffffe)
20231125093356719 10468 50.60: trcapi32: 001 -GetWindowLongW(,) -> ffffffff
20231125093356719 10468 50.60: trcapi32: 001 +GetWindowLongW(2607f0,0)
20231125093356719 10468 50.60: trcapi32: 001 -GetWindowLongW(,) -> 8911f8
20231125093356719 10468 50.60: trcapi32: 001 +GetCurrentThreadId()
20231125093356720 10468 50.60: trcapi32: 001 -GetCurrentThreadId() -> 29e8
20231125093356720 10468 50.60: trcapi32: 001 +IsWindow(2607f0)
20231125093356720 10468 50.60: trcapi32: 001 -IsWindow() -> 1
20231125093356720 10468 50.60: trcapi32: 001 +GetCurrentThreadId()
20231125093356720 10468 50.60: trcapi32: 001 -GetCurrentThreadId() -> 29e8
20231125093356721 10468 50.60: trcapi32: 001 +GetWindowLongW(2607f0,fffffffe)
20231125093356721 10468 50.60: trcapi32: 001 -GetWindowLongW(,) -> ffffffff
20231125093356721 10468 50.60: trcapi32: 001 -BeginPaint(,) -> 5301041b
20231125093356721 10468 50.60: trcapi32: 001 +IsWindowEnabled(2607f0)
20231125093356721 10468 50.60: trcapi32: 001 -IsWindowEnabled() -> 1
20231125093356721 10468 50.60: trcapi32: 001 +IsWindowEnabled(2607f0)
20231125093356722 10468 50.60: trcapi32: 001 -IsWindowEnabled() -> 1
20231125093356722 10468 50.60: trcapi32: 001 +GetCurrentThreadId()
20231125093356722 10468 50.60: trcapi32: 001 -GetCurrentThreadId() -> 29e8
20231125093356722 10468 50.60: trcapi32: 001 +GetClientRect(2607f0,32f05c)
20231125093356722 10468 50.60: trcapi32: 001 -GetClientRect(,) -> 1
20231125093356723 10468 50.60: trcapi32: 001 +IsRectEmpty(32f05c)
20231125093356723 10468 50.60: trcapi32: 001 -IsRectEmpty() -> 0
20231125093356723 10468 50.60: trcapi32: 001 +GetObjectType(5301041b)
20231125093356723 10468 50.60: trcapi32: 001 -GetObjectType() -> 3

```

Figura 2.1: API Call catturate

Tuttavia, il semplice elenco delle chiamate API non è sufficiente per mostrare la vera natura di un software. È necessario definire una logica di analisi che trasformi queste sequenze in informazioni discriminanti. In questa tesi tali logiche sono implementate tramite *machine learning*.

2.2 Machine Learning

Il *machine learning* (ML) è una branca dell'intelligenza artificiale (IA) che consente a computer e macchine di apprendere dai dati, imitando i processi cognitivi umani [22]. In termini pratici, il ML consiste nell'**addestramento** di un software, detto **modello**, per svolgere compiti specifici come effettuare **previsioni** o generare contenuti (testo, immagini, audio o video) a partire da dati osservati [23]. Un modello di *machine learning* può essere visto come un costrutto matematico che riceve dati in ingresso (*input*) e produce un risultato (*output*), in modo analogo a una funzione matematica [24]. L'**allenamento** di un modello consiste nel processo di identificazione dei parametri ottimali interni, così che il modello possa compiere previsioni accurate; ciò avviene fornendogli una serie di esempi [25]. Un **esempio** è un'istanza di input descritta da un insieme di variabili organizzate in un **vettore delle caratteristiche** (*feature vector*) [26]. Ogni variabile è detta **caratteristica** (*feature*) e rappresenta un'informazione specifica del dominio del

problema [27]. Le caratteristiche possono essere di tipo **numerico**, quando esprimono valori quantitativi, oppure **categorico**, quando descrivono attributi qualitativi. Una lista di esempi costituisce un *dataset*.

I modelli di *machine learning*, in base al tipo di addestramento e ai dati disponibili, possono essere suddivisi in quattro categorie principali:

- **Apprendimento supervisionato**: il modello impara a fare previsioni a partire da dati etichettati, ossia accompagnati dalle risposte corrette, scoprendo le relazioni tra input e output [23].
- **Apprendimento non supervisionato**: il modello lavora con dati privi di etichette e ha come obiettivo l'identificazione di strutture o pattern nascosti nei dati [23].
- **Apprendimento per rinforzo**: il modello interagisce con un ambiente, ricevendo feedback sotto forma di ricompense o penalità, e sviluppa progressivamente una strategia ottimale per il raggiungimento di un obiettivo [23].
- **Apprendimento generativo**: i modelli non si limitano a classificare o predire, ma generano nuovi contenuti (testo, immagini, audio, ecc.) a partire dagli input forniti dall'utente [23].

In questa tesi ci concentreremo sull'**apprendimento supervisionato**, affrontando in particolare il compito di **classificazione**.

2.3 Classificazione

Il compito di **classificazione** rientra tra quelli affrontabili tramite *apprendimento supervisionato*. In questo paradigma, gli esempi utilizzati durante l'addestramento sono accompagnati dall'output corretto, detto **etichetta** (label) [28]. Nel caso specifico della classificazione, l'etichetta corrisponde a una **classe**, ossia il valore discreto che identifica a quale categoria appartiene l'esempio. Allenando quindi un modello su un dataset etichettato, l'obiettivo della classificazione è predire la classe corretta di un nuovo esempio non etichettato.

Esistono molteplici compiti di classificazione, distinti uno dall'altro dal numero delle classi da distinguere e da quanto sono mutualmente esclusivi [29].

- **Classificazione binaria**: le classi tra cui predire sono due e mutualmente esclusive [29].
- **Classificazione multi classe**: le classi tra cui predire sono più di due e mutualmente esclusive [29].

- **Classificazione a multipla etichetta:** un esempio può appartenere a più classi [29].
- **Classificazione non bilanciata:** la distribuzione delle classi di esempio non è equa [29].

In questo lavoro di tesi verrà affrontata la classificazione multi classe.

2.3.1 Algoritmi di Classificazione

Esistono molteplici algoritmi per la classificazione, in questa tesi verranno affrontati *XGboost* e *Random forest*.

Random forest

Prima di introdurre l'algoritmo di apprendimento *random forest*, è utile esaminare due concetti fondamentali alla sua comprensione: *bagging* e *decision tree*.

L'algoritmo di apprendimento *bagging* appartiene alla categoria dei metodi di **apprendimento d'insieme**, il cui obiettivo è ridurre la varianza² all'interno di un dataset rumoroso [31]. L'apprendimento d'insieme si basa sull'idea che un gruppo di modelli che collaborano possa raggiungere prestazioni superiori rispetto a un singolo modello. Questo principio viene spesso associato al concetto di “saggezza delle folle”, secondo cui un insieme di persone fornisce mediamente decisioni migliori di quelle di un singolo esperto [31]. Nel *bagging*, i singoli modelli dell'insieme vengono addestrati — anche in parallelo — su sottoinsiemi del dataset originario, generati tramite campionamento con reinserimento (bootstrap) [30]. Ciò implica che uno stesso esempio possa comparire più volte all'interno di uno o più sottoinsiemi. Al termine dell'addestramento, la previsione finale del sistema viene ottenuta aggregando le risposte di tutti i modelli. Nel caso della classificazione, la classe predetta corrisponde a quella che ha ricevuto il maggior numero di voti.

L'algoritmo *decision tree* è costituito da una serie di domande organizzate gerarchicamente a forma di albero. Le domande sono generalmente chiamate *condizioni*, *split* o *test*. Ogni nodo interno (non foglia) contiene una condizione, mentre ogni nodo foglia rappresenta una previsione. Per effettuare una previsione, si parte dal nodo radice e si percorre l'albero fino a raggiungere un nodo foglia, scegliendo il ramo da seguire in base all'esito della condizione presente in ciascun nodo. Il percorso che va dalla radice fino alla foglia è chiamato *percorso di inferenza* [32].

L'algoritmo *Random Forest* è un'estensione dell'algoritmo di *bagging*, in cui i modelli interni sono costituiti da *decision tree*. Oltre a essere addestrati su sottoinsiemi

²La varianza si riferisce alla sensibilità del modello rispetto alle fluttuazioni nei dati di addestramento [30].

casuali degli esempi del dataset (*bagging*), ad ogni nodo viene selezionato un sottoinsieme casuale di *feature* candidate per effettuare lo *split*. Questo introduce diversità tra gli alberi, riducendo la correlazione tra di essi e aumentando la robustezza del modello complessivo [33].

XGBoost

Per una corretta comprensione dell'algoritmo di apprendimento *XGBoost*, è utile introdurre preliminarmente l'algoritmo di apprendimento *boosting* e il concetto di *ascesa del gradiente*.

Il *boosting* è una tecnica di apprendimento sequenziale in cui più modelli vengono addestrati uno dopo l'altro, ciascuno dei quali cerca di correggere gli errori commessi dal modello precedente, al fine di ottenere una previsione progressivamente più accurata [34]. Il *boosting* appartiene alla categoria degli apprendimenti d'insieme.

La *discesa del gradiente* [35] è un algoritmo di ottimizzazione che permette di trovare i valori ottimali dei parametri di un modello minimizzando una funzione di costo. L'idea di base è semplice: si parte da valori iniziali arbitrari per i parametri, si calcola la pendenza (il *gradiente*) della funzione di costo in quel punto e si aggiornano i parametri muovendosi nella direzione opposta al gradiente, cioè verso la discesa più ripida. Il processo si ripete finché non si raggiunge un minimo (locale o globale). Un ruolo cruciale è svolto dal *tasso di apprendimento* (*learning rate*), che stabilisce la grandezza dei passi: se troppo grande vi è il rischio di superare il minimo, mentre se troppo piccolo l'addestramento risulta molto lento. Esistono diverse varianti:

- **Batch Gradient Descent:** aggiorna i parametri dopo aver visto tutti i dati di addestramento [35].
- **Stochastic Gradient Descent (SGD):** aggiorna i parametri dopo ogni esempio, introducendo rumore ma anche la possibilità di sfuggire ai minimi locali [35].
- **Mini-batch Gradient Descent:** compromesso tra i due, suddivide i dati in piccoli blocchi [35].

L'obiettivo della discesa del gradiente è ridurre progressivamente l'errore, fino a convergere a un set di parametri che rende il modello il più accurato possibile.

L'algoritmo di apprendimento *XGBoost* si basa sulla tecnica del *boosting*, in cui un modello iniziale, l'albero decisionale, viene progressivamente migliorato tramite l'uso della *discesa del gradiente* [36].

2.4 Metriche di valutazione

Per valutare l'efficacia di un classificatore è necessario quantificare quanto esso sia capace di effettuare previsioni corrette. Uno strumento utile a questo scopo è la **matrice di confusione** [37]. Nella matrice, le righe rappresentano i valori effettivi mentre le colonne indicano quelli predetti; in letteratura è possibile incontrare anche la rappresentazione inversa. La matrice ha tante righe e colonne quante sono le classi presenti nel dataset. L'intersezione tra la riga i e la colonna j fornisce il numero di istanze della classe i che sono state classificate come classe j . In particolare, gli elementi della diagonale rappresentano le istanze correttamente predette. A partire dalla matrice di confusione è possibile calcolare diverse metriche di valutazione, tra cui la **precision**, la **recall** e l'**F1-score** per ciascuna classe. È inoltre possibile derivare le corrispondenti misure aggregate a livello *macro* e *micro*, utili per valutazioni globali sul dataset. Prima di entrare nel dettaglio delle metriche tornano utili i concetti di **true positive**, **true negative**, **false positive** e **false negative** considerando la matrice di confusione M avente n classi.

I **true positive** di una classe k equivalgono al numero di istanze di classe k che sono state predette come classe k :

$$TP_k = M_{k,k}$$

I **false positive** di una classe k rappresentano il numero di istanze appartenenti a classi diverse da k che sono state erroneamente predette come k :

$$FN_k = \sum_{\substack{j=1 \\ j \neq k}}^n M_{k,j}$$

I **false negative** di una classe k rappresentano il numero di istanze appartenenti a k che sono state erroneamente classificate come un'altra classe:

$$FN_k = \sum_{\substack{j=1 \\ j \neq k}}^n M_{k,j}$$

I **true negative** di una classe k rappresentano il numero di istanze appartenenti a classi diverse da k che sono state correttamente classificate come non k :

$$TN_k = \sum_{\substack{i=1 \\ i \neq k}}^n \sum_{\substack{j=1 \\ j \neq k}}^n M_{i,j}$$

La **precision** di una classe k è il rapporto tra i veri positivi e la somma dei veri positivi e dei falsi positivi:

$$\text{Precision}_k = \frac{TP_k}{TP_k + FP_k}$$

La **recall** (o sensibilità) di una classe k è il rapporto tra i veri positivi e la somma dei veri positivi e dei falsi negativi:

$$\text{Recall}_k = \frac{TP_k}{TP_k + FN_k}$$

L'**F1-score** di una classe k è la media armonica tra precision e recall:

$$F1_k = 2 \cdot \frac{\text{Precision}_k \cdot \text{Recall}_k}{\text{Precision}_k + \text{Recall}_k}$$

Le metriche **Precision**, **Recall** e **F1-Score** calcolate per ciascuna classe possono essere aggregate per ottenere valori complessivi sul modello intero. Due approcci comuni di aggregazione sono la media *macro* e la media *micro*:

$$\text{Macro}_{\text{precision}} = \frac{\sum_i^n \text{Precision}_i}{n}$$

$$\text{Macro}_{\text{recall}} = \frac{\sum_i^n \text{Recall}_i}{n}$$

$$\text{Macro}_{F1} = \frac{\sum_i^n F1_i}{n}$$

$$\text{Micro}_{\text{Precision}} = \frac{\sum_i^n TP_i}{\sum_i^n TP_i + \sum_i^n FP_i}$$

$$\text{Micro}_{\text{Recall}} = \frac{\sum_i^n TP_i}{\sum_i^n TP_i + \sum_i^n FN_i}$$

$$TP = \sum_{i=1}^n TP_i,$$

$$FP = \sum_{i=1}^n FP_i,$$

$$FN = \sum_{i=1}^n FN_i,$$

$$\text{Precision} = \frac{TP}{TP + FP},$$

$$\text{Recall} = \frac{TP}{TP + FN},$$

$$\text{Micro-F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

2.5 Approcci di Machine Learning per Windows PE Malware Detection

Bibliografia

- [1] L. Zannella, M. Zanella, and M. Rizzo, “Report Cittadini E ICT,” Istat, Report/Study, 2025. [Online]. Available: https://www.istat.it/wp-content/uploads/2025/04/REPORT_CITTADINI-E-ICT_2024.pdf
- [2] C. , “© Clusit - Rapporto 2025 sulla Cybersecurity in Italia e nel mondo,” Clusit, Tech. Rep., Mar. 2025. [Online]. Available: https://clusit.it/wp-content/uploads/download/Rapporto_Clusit_03-2025_web.pdf
- [3] N. Idika and A. P. Mathur, “A survey of malware detection techniques,” *Purdue University*, vol. 48, no. 2, pp. 32–46, 2007. [Online]. Available: https://profsandhu.com/cs5323_s17/im_2007.pdf
- [4] Y. Ki, E. Kim, and H. K. Kim, “A novel approach to detect malware based on api call sequence analysis,” *International Journal of Distributed Sensor Networks*, vol. 11, no. 6, p. 659101, 2015. [Online]. Available: <https://doi.org/10.1155/2015/659101>
- [5] Microsoft, “Introduction to APIs.” [Online]. Available: <https://learn.microsoft.com/en-us/xandr/industry-reference/intro-to-apis>
- [6] Y. Qiao, Y. Yang, L. Ji, and J. He, “Analyzing malware by abstracting the frequent itemsets in api call sequences,” in *2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, July 2013, pp. 265–270. [Online]. Available: <https://ieeexplore.ieee.org/abstract/document/6680850>
- [7] Microsoft, “Formato pe - win32 apps.” [Online]. Available: <https://learn.microsoft.com/it-it/windows/win32/debug/pe-format>
- [8] —, “Libreria di collegamento dinamico.” [Online]. Available: <https://learn.microsoft.com/it-it/troubleshoot/windows-client/setup-upgrade-and-drivers/dynamic-link-library>
- [9] Kaspersky, “Trojan droppers.” [Online]. Available: <https://encyclopedia.kaspersky.com/glossary/backdoor/>
- [10] Cisco, “What is a malware.” [Online]. Available: <https://www.cisco.com/site/us/en/learn/topics/security/what-is-malware.html>
- [11] Kaspersky, “Trojan droppers.” [Online]. Available: <https://encyclopedia.kaspersky.com/glossary/trojan-droppers/>

- [12] —, “Multipacked.” [Online]. Available: <https://encyclopedia.kaspersky.com/knowledge/multipacked/>
- [13] Microsoft, “Indice di api windows - win32 apps.” [Online]. Available: <https://learn.microsoft.com/it-it/windows/win32/apiindex/windows-api-list>
- [14] —, “Getkeyboardstate - win32 apps.” [Online]. Available: <https://learn.microsoft.com/it-it/windows/win32/api/winuser/nf-winuser-getkeyboardstate>
- [15] D. Heath, S. Kasif, and S. Salzberg, “k-dt: A multi-tree learning method,” in *Proceedings of the Second International Workshop on Multistrategy Learning*, 1993, pp. 138–149.
- [16] T. Chen. (2016) Xgboost: A scalable tree boosting system. Archived on 2016-08-07. [Online]. Available: <https://web.archive.org/web/20160817055008/http://arxiv.org/abs/1603.02754>
- [17] M. Christodorescu and S. Jha, *Static Analysis of Executables to Detect Malicious Patterns* *. USENIX Association, 2003. [Online]. Available: https://pages.cs.wisc.edu/~jha/jha-papers/security/usenix_2003.pdf
- [18] S. Cesare and Y. Xiang, *Software similarity and classification*, 2012th ed. Guildford, England: Springer, 2012.
- [19] “Detours: Detours is a software package for monitoring and instrumenting API calls on windows. it is distributed in source code form.” [Online]. Available: <https://github.com/microsoft/Detours>
- [20] DynamoRIO Project, “Dynamorio: Dynamic binary instrumentation tool,” <https://dynamorio.org/>, 2025, accessed: 2025-09-20.
- [21] wtrance.net, “Using withdll and detours to trace win api calls,” <https://wtrance.net/guides/using-withdll-and-detours-to-trace-winapi/>, Nov. 2023, accessed: 2025-09-20.
- [22] IBM, “Che cos’è il machine learning (ml)?” Mar. 2025, accessed: 2025-09-21. [Online]. Available: <https://www.ibm.com/it-it/think/topics/machine-learning>
- [23] Google, “Che cos’è il machine learning?” accessed: 2025-09-21. [Online]. Available: <https://developers.google.com/machine-learning/intro-to-ml/what-is-ml?hl=it>
- [24] —, “Glossario del machine learning,” accessed: 2025-09-21. [Online]. Available: <https://developers.google.com/machine-learning/glossary?hl=it#model>
- [25] —, “Glossario del machine learning,” accessed: 2025-09-21. [Online]. Available: <https://developers.google.com/machine-learning/glossary?hl=it#training>
- [26] —, “Glossario del machine learning,” accessed: 2025-09-21. [Online]. Available: <https://developers.google.com/machine-learning/glossary?hl=it#example>
- [27] —, “Glossario del machine learning,” accessed: 2025-09-21. [Online]. Available: <https://developers.google.com/machine-learning/glossary?hl=it#feature>

- [28] —, “Glossario del machine learning,” accessed: 2025-09-21. [Online]. Available: <https://developers.google.com/machine-learning/glossary?hl=it#class>
- [29] I. Belcic, “What is classification in machine learning?” Sep. 2025. [Online]. Available: <https://www.ibm.com/think/topics/classification-machine-learning>
- [30] T. Mucci, “Cosa sono l’overfitting e l’underfitting?” Mar. 2025. [Online]. Available: <https://www.ibm.com/it-it/think/topics/overfitting-vs-underfitting>
- [31] “Cos’è il bagging?” Jun. 2025. [Online]. Available: <https://www.ibm.com/it-it/think/topics/bagging>
- [32] “Alberi decisionali (decision trees).” [Online]. Available: <https://developers.google.com/machine-learning/decision-forests/decision-trees?hl=it>
- [33] “Che cos’è la foresta casuale,” Feb. 2025. [Online]. Available: <https://www.ibm.com/it-it/think/topics/random-forest>
- [34] “Che cos’è il boosting,” Nov. 2024. [Online]. Available: <https://www.ibm.com/it-it/topics/boosting>
- [35] “Che cos’è la discesa del gradiente?” Jan. 2025. [Online]. Available: <https://www.ibm.com/it-it/think/topics/gradient-descent>
- [36] E. Kavlakoglu and E. Russi, “Che cos’è xgboost?” Jan. 2025. [Online]. Available: <https://www.ibm.com/it-it/think/topics/xgboost>
- [37] J. Murel and E. Kavlakoglu, “Che cos’è una matrice di confusione?” Feb. 2025. [Online]. Available: <https://www.ibm.com/it-it/think/topics/confusion-matrix>