Università Politecnica delle Marche

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in Ingegneria Informatica e dell'Automazione

---

Tesi di Laurea

Rilevamento di malware da traffico DNS reale mediante classificazione basata su feature

Malware detection from real DNS traffic using feature-based classification

Relatore
Prof. Luca Spalazzi

Candidato
Luca Mannini

Correlatori
Prof. Alessandro Cucchiarelli

Prof. Christian Morbidoni

Anno Accademico 2019/2020

Indice

Elenco delle figure

Abstract

Recentemente le botnet sono diventate uno dei maggiori pericoli per gli utenti di Internet. L'evoluzione del software, negli anni, ha portato i cyber-criminali a creare reti di bot sempre più complesse, al fine di evitare il rilevamento del malware e le tecniche di reverse-engineering. Tra i metodi per evitare il rilevamento, uno dei più utilizzati è basato sulla generazione di domini DNS utilizzati per il collegamento tra bot e botmaster. D'altro canto le tecniche per rilevare codice malevolo sono migliorate all'aumentare della complessità dei malware stessi, motivo per il quale si è iniziato ad implementare antivirus e firewall basati su tecniceh euristiche e/o sul machine learning. In questa tesi si utilizzerà una rete neurale, in particolare un Multi-Layer-Perceptron pre-addestrato che lavora su caratteristiche (feature) estratte dai nomi di dominio al fine di classificare i DNS in benevoli e malevoli. La si testerà su un dataset nuovo, creato da traffico reale anonimizzato e si andrà poi a confrontare i risultati con quelli di una rete chiamata "Long Short Term Multiclass Imbalance" (LSTM-MI), addestrata sullo stesso dataset.

Capitolo 1

Introduzione

1.1 Obiettivi e struttura della tesi

Questa tesi sarà una trattazione sperimentale che avrà l'obiettivo di descrivere le performance delle attuali tecniche di machine learning allo stato dell'arte allo scopo di poter discernere il traffico generato da botnet basate su DGA dal normale traffico utente. In particolare si cercherà di quantificare l'accuratezza di un algoritmo che prenderà in input delle stringhe rappresentanti i Domain Name Server (DNS) di alcuni siti web e che manderà in output la famiglia a cui si suppone appartenga il DNS in questione.

La trattazione inizierà con una descrizione generale sulla scena del cybercrimine internazionale, passando a discutere dei malware, delle botnet e in particolare degli algoritmi malevoli su cui è focalizzata la classificazione, dopodiché si parlerà delle tecniche di *machine learning* utilizzate per la classificazione. Verranno descritti: il materiale iniziale, i metodi di ottenimento del dataset di test, i classificatori utilizzati, il tipo di esperimento e il flusso di lavoro.
Sarà mostrato il codice e spiegato in linea di massima per dare un'idea delle parti che lo compongono, motivando le scelte implementative e le problematiche affrontate durante la stesura del codice.
Verranno descritti i risultati sperimentali, confrontati con quelli di altri classificatori e infine si cercherà di dare un giudizio sul risultato ottenuto paragonandolo alle aspettative, aggiungendo indicazioni su possibili miglioramenti futuri.

Capitolo 2

Contesto applicativo

L'aumento di utilizzo dei computer nell'era moderna ha fatto sì che si creasse un'enorme rete interconnessa capace di scambiare informazione tra i più remoti angoli della Terra. L'ampliamento delle possibilità di comunicazione ha anche dato modo a malintenzionati di sfruttare le capacità della rete, in particolare utilizzando software malevolo e strumenti quali botnet, ransomware e in generale malware per ottenere il controllo dei personal computer degli ignari utenti di internet. La cronaca Internazionale è piena di notizie riguardo attachi hacker a infrastrutture e aziende tra cui possiamo citare la famosissima vicenda di Marcus Hutchins e Wannacry[1], l'attacco alle banche del malware Carbanak[2] e l'attacco a Playstation Network da parte della Lizard Squad[3]. Questi attacchi informatici hanno mobilitato la comunità internazionale della cybersecurity che negli anni ha sempre cercato di trovare soluzioni affinché minacce simili possano essere prevenute. Attacchi simili a quelli di Wannacry si sono infatti verificati molteplici volte a seguire, seppur abbiano avuto in genere impatti minori (purtroppo ci sono state eccezioni[4] a questa regola) proprio grazie alla maggior consapevolezza riguardo le tematiche legate alla sicurezza informatica. La frode online è passata da tempo dall'essere un semplice hobby a un mezzo per i criminali informatici per guadagnarsi da vivere. I truffatori considerano Internet il loro campo di gioco. Internet infatti ha molti siti vulnerabili e una grande quantità di dati non protetti. Sebbene esistano dati "protetti", i luoghi in cui sono archiviati possono comunque essere violati. Alcuni cybercriminali hanno condiviso la loro esperienza nell'hacking tramite articoli online. Oramai scrivere trojan, exploit, generare traffico malevolo è così semplice a tal punto che esistono veri e propri forum[5] nei quale si

[1] https://www.youtube.com/watch?v=vveLaA-z3-o

[2] https://www.youtube.com/watch?v=XXSn5lwRF5o

[3] https://www.forbes.com/sites/insertcoin/2015/07/09/lizard-squad-hacker-who-shut-down-psn-xbox-live-and-an-airplane-will-face-no-jail-time

[4] https://fortune.com/2020/09/18/ransomware-police-investigating-hospital-cyber-attack-death

[5] https://www.recordedfuture.com/russian-chinese-hacking-communities/

spiega passo passo come imparare. Ci sono decenni di storie su questi forum[6] che nel tempo si sono guadagnati la "fama" per aver permesso a numerosi cybercriminali di interagire e cooperare, molti dei quali arrestati a seguire di vari raid di FBI, Europol e altre agenzie governative e di intelligence. Se volessimo citarne alcuni si inizierebbe dal forum "Runet", di cui ormai si sono perse le tracce, considerato il primo e vero forum di cybercrimine internazionale, passando poi per Maza.la[7], Dark0de[8], Damagelab, Antichat.ru[9], Zloy[10] e soprattutto Exploit.in [11], fino ai più recenti come Xss.is [12] che si fa vanto di essere nato dalle ceneri del vecchio Damagelab, o Maza-forum.ru[13] che sfrutta il nome del vecchio forum Maza.la per farsi pubblicità "ingannevole".

Sotto queste enorme community, che come si può ben notare hanno per lo più origini russe, c'è una vera e propria fiamma continua alimentata dal lucro e dall'inganno, che permette ai rappresentanti del cybercrimine di continuare a proliferare. Purtroppo c'è anche chi riesce a farla franca, come testimonia il giornalista investigativo Brian Krebs nel suo blog[14].

Lo studio degli strumenti utilizzati dagli attaccanti, in gergo denominati "threat actors", è di fondamentale importanza per poter prevenire le minacce sia dal punto di vista dell'individuo, ovvero l'utente web che naviga e lavora tramite internet, sia dal punto di vista delle comunità che utilizzano la rete.

2.1 Struttura e funzionamento delle Botnet

Una botnet è una rete controllata da un botmaster e composta da dispositivi infettati da malware specializzato, detti bot, zombie o slaves. I dispositivi connessi ad Internet al cui interno sussistono vulnerabilità nella loro infrastruttura di sicurezza informatica possono talvolta diventare parte della botnet, e, se l'agente infettante è un trojan, il botmaster può controllare il sistema tramite accesso remoto. I computer così infettati, possono scagliare attacchi denominati Distributed Denial of Service contro altri sistemi e/o compiere altre operazioni illecite, in alcuni casi persino su commissione di organizzazioni criminali.

[6] https://www.digitalshadows.com/blog-and-research/forums-are-forever-part-1-cybercrime-never-dies/
[7] https://threatpost.com/report-major-russian-hacker-forumhacked-022811/74972/
[8] https://krebsonsecurity.com/2015/07/the-darkode-cybercrime-forum-up-close/
[9] https://forum.antichat.ru
[10] https://forum.zloy.bz
[11] https://exploit.in/
[12] https://xss.is
[13] https://maza-forum.ru
[14] https://krebsonsecurity.com/2019/07/whos-behind-the-gandcrab-ransomware/

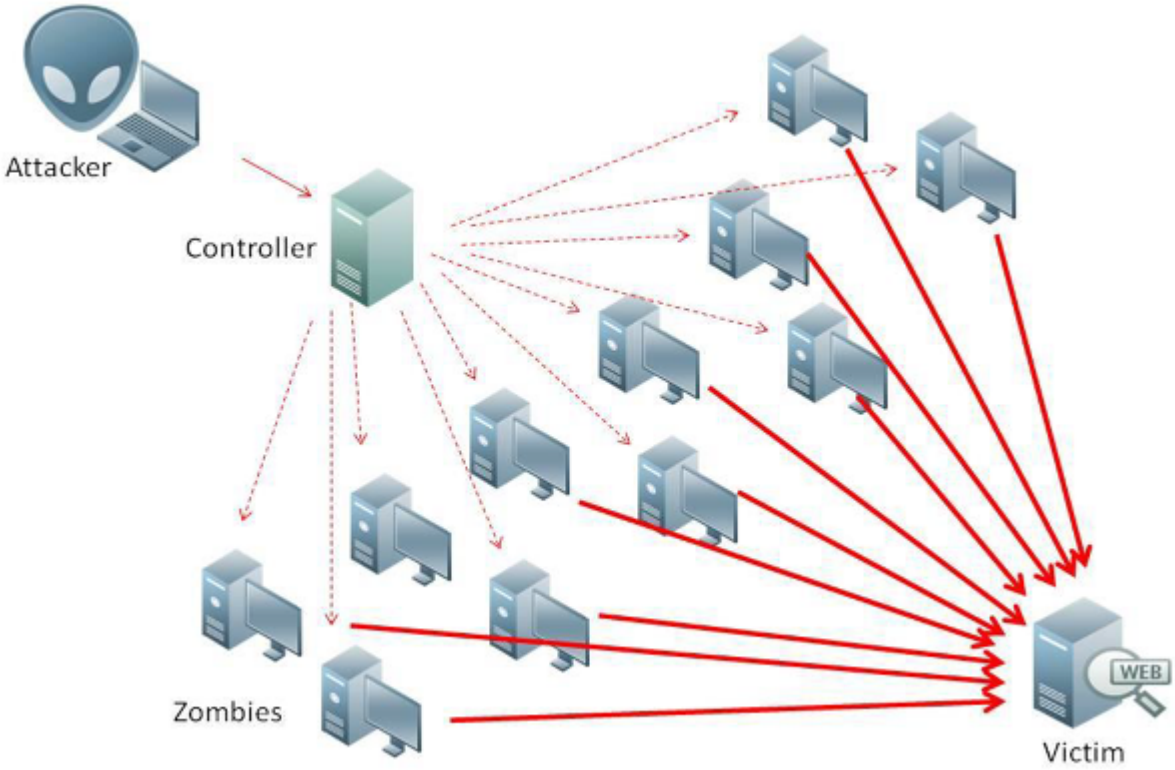

Figura 2.1: Attacco DDOS

I botmaster progettano le loro botnet in modo che diventino il più possibile modulari, robuste e invisibili. Inoltre, i controller delle botnet sono in grado di propagare facilmente le modifiche all'intera botnet o, se lo si desidera, a singoli bot. L'approccio più tradizionale per eliminare una botnet consiste nell'identificare l'indirizzo IP del server C&C e provare a rimuoverlo, interrompendo la capacità del botmaster di inviare nuovi comandi alla botnet. Sebbene questo approccio produca buoni risultati per le botnet più vecchie, le botnet più recenti e avanzate hanno iniziato a creare ridondanza a livello di C&C. In un certo senso, non siamo più di fronte a un serpente ma piuttosto a un'idra: ogni volta che i ricercatori o le autorità sono in grado di identificare e sequestrare un server C&C, ne vengono creati altri per sostituirlo. Questo comportamento è un motivo per cui le botnet sono diventate così difficili da rilevare e smantellare. Anche se ogni botnet ha le sue caratteristiche particolari implementate in base all'esigenza del botmaster, in genere sono tutte molto simili tra di loro. La comprensione di quali caratteristiche siano condivise tra le diverse famiglie di botnet, compreso il ciclo di vita delle botnet e i meccanismi difensivi, è di fondamentale importanza per comprendere il fenomeno delle botnet.

2.1.1 Ciclo di vita

Il ciclo di vita di un bot è mostrato nella **Figura 2.2**. Il ciclo di vita può essere suddiviso in un totale di 11 step, a loro volta raggruppabili in diverse fasi. Nella prima fase è fondamentale decidere i requisiti e scrivere le specifiche della botnet: il botmaster deve sapere chi è l'obiettivo. Cosa si otterrà una volta che il bot inizierà a infettare?

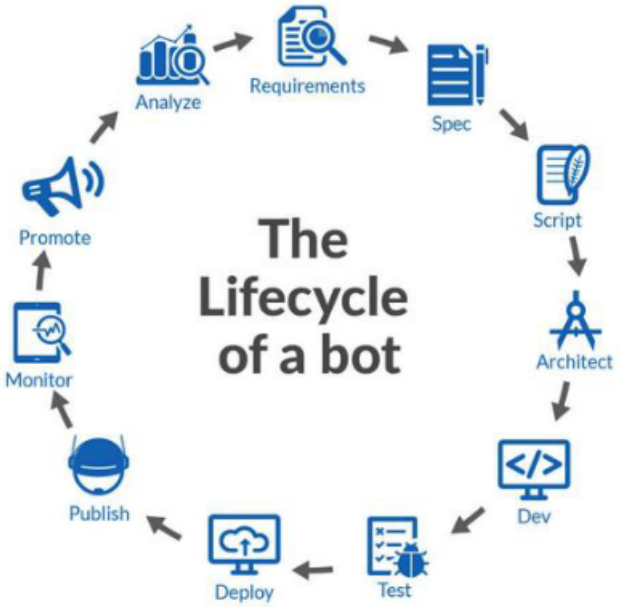

Figura 2.2: Ciclo di vita della Botnet

È necessario considerare anche ciò che sarà utile per il futuro e quindi scrivere le specifiche della botnet a fronte di futuri cambiamenti ed implementazioni della botnet. La seconda fase, composta dai 3 step successivi, è quella nella quale si andrà a scrivere il codice della botnet nella sua interezza, compreso di:

- Script per l'interazione del bot con gli utenti
- Front-end e pannello di controllo (C&C)
- Back-end per la gestione dei dati raccolti
- Test del software in ognuna delle sue componenti

Una volta terminato il test, il bot deve essere distribuito in un ambiente stabile, cercando di simulare una distribuzione reale del malware. Dopo il processo di test e distribuzione, il file utilizzato per l'infezione dei dispositivi deve ottenere l'approvazione dagli app store nel caso in cui siano necessari per poter diffondere la botnet: un esempio di ciò è la diffusione di malware per cellulare, che devono essere approvati per poter essere poi scaricati dal Play Store o dall'App Store. Dopo l'approvazione deve essere monitorata particolarmente l'interazione dell'agente infettante con l'utente per fare in modo che non si verifichino errori che potrebbero far scattare la curiosità degli antivirus e/o degli utenti stessi. Le prestazioni del bot devono essere analizzate non appena viene utilizzato e dopo un'adeguata analisi, si può passare alla fase di riscrittura delle funzionalità, nella quale la botnet viene ulteriormente migliorata. Come si può intuire, non è facile scrivere una botnet senza errori e affinché il software finale faccia il suo lavoro senza problemi è necessario eseguire molti test.

2.1.2 Topologia

Il controller della botnet deve garantire che l'infrastruttura C&C sia sufficientemente robusta da controllare una grande quantità di bot e da resistere a tentativi di sinkholing o shutdown. È necessario valutare diversi parametri per poter decidere, dal punto di vista dell'attaccante, quale botnet è meglio utilizzare dipendentemente dal modello di business e da fattori che per lo più decideranno la scalabilità e le funzioni della stessa. Le caratteristiche scelte relative alla topologia della botnet influenzeranno in modo diverso la complessità della botnet, la latenza del messaggio, la sopravvivenza una volta infettato il sistema e il rilevamento. Dipendentemente dalla topologia infatti, la botnet potrà essere più o meno difficile da smantellare dalle autorità.
C'è da dire che la topologia influenza pesantemente la velocità di comunicazione all'interno della rete di bot, il che può essere un punto cruciale in alcuni tipi di operazione. In generale le botnet più difficili da abbattere sono le quelle basate su protocolli p2p,

TOR o, in generale, le botnet decentralizzate. Il motivo è molto semplice: trovare il C&C è molto più difficile. Se volessimo definire alcuni topologie classiche potremmo elencarne 4:

- **A stella**:
  questa tipologia ha un solo C&C centralizzato (Centralized Command & Control) per comunicare con ogni singolo bot. È proprio il C&C centralizzato che emette tutte le istruzioni ad ogni bot. Una volta che viene violato un nuovo computer, esso contatterà automaticamente il centro di controllo della botnet di cui diventerà membro, attendendo poi le istruzioni future. La configurazione della topologia a stella è mostrata nella **Figura** 2.3.

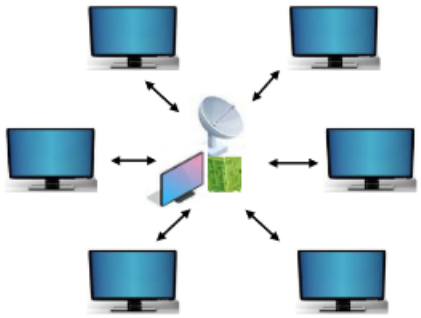

Figura 2.3: Topologia a stella

- **Multi-server**
  Questa topologia è un'estensione della topologia precedente . Qui vengono impiegati più server per inviare le istruzioni dei C&C ai vari bot. Questi server comunicano tra loro per la gestione della botnet e per questo la costruzione di questa configurazione è molto complessa ma, una volta implementata, può essere utilizzata sia da topologie a stella che multi-server. La configurazione multi-server è mostrata nella **Figura** 2.4.

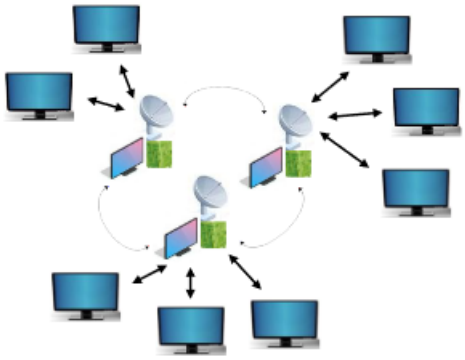

Figura 2.4: Multi-server

- **Gerarchica**
  Il singolo bot non conoscerà la posizione del pannello di controllo , il che induce i ricercatori a indovinare la dimensione della botnet: in questo caso le botnet enormi sono divise in sotto-botnet. La **Figura 2.5** mostra la configurazione gerarchica.

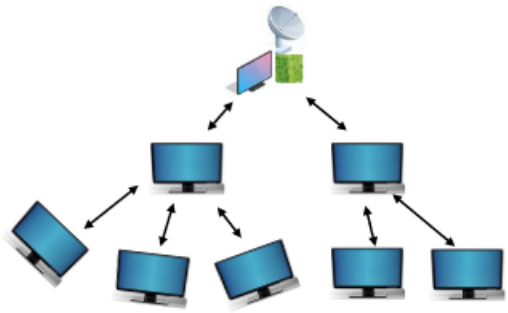

Figura 2.5: A gerarchia

- **Decentralizzata**
  Qui non ci sarà alcuna configurazione centralizzata di C&C ed è, come scritto in precedenza, il tipo di rete il cui centro di comando è più difficile da trovare. Il C&C inserirà i comandi nella rete di bot tramite alcuni bot, i quali a loro volta li distribuiranno automaticamente a tutti gli altri. Per comunicare ai bot, la struttura decentralizzata fa uso di numerosi canali di comunicazione tra i bot. La **Figura 2.6** mostra la configurazione decentralizzata.

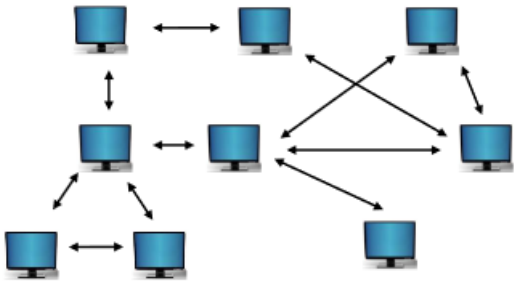

Figura 2.6: Decentralizzata

2.2 Tecniche di evasione dal rilevamento

Le botnet operano di nascosto per eludere il rilevamento e aumentare la loro probabilità e durata di sopravvivenza. Possiamo studiare le strategie di evasione adottate da una botnet dal punto di vista del bot, del botmaster, del server C&C e della comunicazione tra le due parti di cui la botnet è composta, ma per ragioni di lunghezza verrà trattato solamente l'offuscamento del loader dagli antivirus (punto di vista del bot) e l'offuscamento del server di controllo.

- **Offuscamento del file binario**
  Per evitare di essere rilevati dalle applicazioni di sicurezza basate su host, vengono impiegate diverse tecniche di evasione per nascondere l'agente infettante (che da ora chiameremo "loader" o file binario). Gli approcci di rilevamento basati su pattern sono sconfitti dall'uso del polimorfismo. La proprietà di polimorfismo si riferisce alla capacità del file binario di mutare e riscrivere parte del proprio stesso codice. Uno dei modi per implementare ciò è utilizzare la crittografia. Lo stesso effetto può essere ottenuto anche comprimendo il loader: la compressione aiuta a offuscare il codice malevolo. Alcuni packer sono in grado di produrre nuovi file binari ogni volta che l'eseguibile maligno originale viene compresso. Sebbene il polimorfismo del codice riesca a nascondere il loader dagli antivirus basati sulla firma del loader, può comunque essere rilevato da approcci di rilevamento basati sulla memoria. Quando viene eseguito, il loader deve essere decrittografato o decompresso per ottenere lo stesso codice. Questo problema è risolto dal cosiddetto codice metamorfico.

- **Anti-Analysis**
  I ricercatori analizzano il comportamento delle botnet eseguendo i loader su macchine virtuali o sandbox. Un altro metodo per l'analisi bot-net consiste nell'utilizzare honeypot che emulano il software noto e le vulnerabilità di rete che

possono essere infettate dalle botnet. Gli honeypot sono progettati per essere autonomi e prevenire la diffusione di botnet oltre l'honeypot. Per eludere tale analisi, alcuni loader eseguono controlli per determinare l'ambiente in cui vengono eseguiti. Se il binario rileva una macchina virtuale o sandbox, può rifiutarsi di eseguire o modificare la sua funzionalità per eludere l'analisi. Dopo un'ondata iniziale di botnet compatibili con VM, come Conficker, Rbot, SDbot/Reptile, Mechbot, SpyBot e AgoBot, la tendenza per tale tecnica di evasione sta diminuendo principalmente perché i programmi legittimi raramente eseguono test per l'ambiente di esecuzione e quindi gli antivirus considerano inutilmente sospetto il file eseguito.

- **Security Suppression**
  Dopo aver infettato con successo una macchina, una botnet può procedere e disabilitare il software di sicurezza esistente sulla macchina vittima. Se l'host è stato già infettato da altri malware, anche questi vengono eliminati. Ad esempio, Conficker disabilita diversi servizi Windows relativi alla sicurezza e chiavi di registro al momento dell'installazione. Include una blacklist dei nomi di dominio che utilizza per bloccare l'accesso a determinati siti web relativi alla sicurezza e una blacklist dei processi per terminare i processi che possono aiutare nel suo rilevamento.

- **Rootkit Technology**
  Un rootkit è un programma che si installa in modo persistente e non rilevabile sulla macchina infetta sovvertendo il normale comportamento del sistema operativo. Le botnet possono installare rootkit su macchine compromesse per ottenere un accesso privilegiato ad esse. Ciò consente loro di svolgere attività dannose aggirando i tipici meccanismi di autenticazione e autorizzazione. Di conseguenza, il software antivirus tradizionale non riesce a rilevare le intrusioni.

La botnet differisce da altri malware nella capacità del botmaster di controllare e coordinare in remoto tutte le macchine infette tramite i server C&C. Ciò significa essenzialmente che il "cervello" della botnet risiede nel server C&C e proprio per questo il server di controllo può trasformarsi nel suo tallone d'Achille. Pertanto, le botnet investono risorse considerevoli per nascondere il server C&C. Al fine di vanificare gli sforzi di rilevamento, i criminali informatici hanno quindi ideato alcune ulteriori tecniche evasive che si basano sulla rapida modifica dei record DNS. Tali tecniche consentono agli aggressori di modificare rapidamente le informazioni DNS delle loro macchine in modo tale da cambiare costantemente l'indirizzo IP e il nome di dominio a cui si connettono. Esempi di tali tecniche sono chiamate Fast-Flux (FF) e "Domain Generation Algorithm" (DGA). In questa tesi ci concentreremo sul secondo metodo, motivo per il quale è dedicata una sezione a parte.

Il **Fast Flux** è una tecnica utilizzata nelle botnet basata sul DNS per nascondere il phishing e i siti di malware dietro una rete di host compromessi che agiscono da proxy e che cambiano in continuazione. Il tipo più semplice di fast flux, conosciuto come "single-flux", è caratterizzato da molti nodi che, all'interno della rete, registrano e de-registrano il proprio indirizzo come parte della lista degli indirizzi DNS di tipo A per un singolo dominio. Questo sistema unisce il "round robin DNS" con valori molto bassi di TTL (Time to Live), per creare una lista di indirizzi per un certo dominio che è in continuo cambiamento. Questa lista può comprendere centinaia di migliaia di indirizzi. Un tipo più sofisticato di fast flux, conosciuto come "double-flux", è caratterizzata da nodi nella rete che registrano e de-registrano il proprio indirizzo come parte della lista dei record DNS per una certa zona. Questo fornisce uno strato addizionale di ridondanza e di sopravvivenza all'interno della rete di malware. Durante un attacco malware, il record DNS punterà ad un sistema compromesso che agirà da proxy. Il metodo può anche mascherare i sistemi dell'attaccante, che sfrutteranno la rete attraverso una serie di proxy e renderanno più arduo identificare la rete dell'attaccante. Il record normalmente punterà ad un indirizzo IP dove i bot vanno per registrarsi, per ricevere istruzioni o per attivare degli attacchi. Siccome gli IP passano attraverso un proxy, è possibile contraffare l'origine di queste istruzioni, aumentando la possibilità di superare le liste di controllo degli accessi che sono state messe nella rete.

2.3 DGA

Gli algoritmi di generazione del dominio (DGA) sono algoritmi implementati in alcuni tipi di malware che vengono utilizzati per generare periodicamente un gran numero di nomi di dominio che possono essere utilizzati come punti di incontro con i loro server di comando e controllo. L'elevato numero di potenziali punti di incontro rende difficile per le forze dell'ordine chiudere efficacemente le botnet, poiché i computer infetti tenteranno ogni giorno di contattare alcuni di questi nomi di dominio per ricevere aggiornamenti o comandi. L'uso della crittografia a chiave pubblica nel codice del malware rende impossibile per le forze dell'ordine e altri attori imitare i comandi dei controller del malware poiché alcuni worm rifiuteranno automaticamente qualsiasi aggiornamento non firmato dai controller del malware. Ad esempio, un computer infetto potrebbe creare migliaia di nomi di dominio come: www. <caratteri a caso> .com e tentare di contattare una parte di questi allo scopo di ricevere un aggiornamento o comandi. Incorporando il DGA invece di un elenco di domini generati in precedenza (dai server di comando e controllo) nel binario non offuscato del malware si protegge da un dump di stringhe che potrebbe essere inserito preventivamente in una blacklist di rete per tentare di limitare la comunicazione in uscita da infetti ospita all'interno di un'azienda. La tecnica è stata resa popolare dalla famiglia di worm Conficker.a e .b che inizialmente generavano 250 nomi di dominio al giorno. A partire da Conficker.C, il malware genererebbe 50.000 nomi di dominio ogni giorno di cui tenterebbe di contattare 500, dando a una macchina infetta una possibilità dell'1% di essere aggiornata ogni giorno se i controller di malware registrassero un solo dominio al giorno. Per impedire ai computer infetti di aggiornare il proprio malware, le forze dell'ordine avrebbero dovuto pre-registrare 50.000 nuovi nomi di dominio ogni giorno. Dal punto di vista del proprietario della botnet, deve registrare solo uno o pochi domini tra i diversi domini che ogni bot interroga ogni giorno.

Le DGA possono essere classificate in base al seed che usano per generare i nomi di dominio:

- **Tempo dipendente**: in questo caso l'algoritmo utilizza come seed un tempo (orario o data) per generare i domini.

- **Tempo indipendente**: in questo caso l'algoritmo utilizza come seed un valore costante per generare i domini

- **Deterministico**: il DGA in questo caso fa uso di parametri di tipo deterministico. Questi valori sono noti dall'algoritmo e assumono valori fissi; DGA con questa caratteristica generano una sequenza di nomi di dominio molto semplice

da prevedere, in quanto si farà sempre uso degli stessi valori per la generazione della sequenza.

- **Non deterministico**: in questo caso l'algoritmo genera una sequenza di nomi di dominio casuali e difficili da prevedere.

Possiamo quindi distinguere 4 classi di DGA:

- **TID-DGA** (tempo indipendente e deterministico): in questo caso l'algoritmo genererà lo stesso nome di dominio ogni volta che viene mandato in esecuzione.
- **TDD-DGA**(tempo dipendente e deterministico): in questo caso il seme del DGA varia con il tempo, i dominigenerati con questo seme risultano di facile predizione grazie alla proprietà deterministica del seme;
- **TDN-DGA** (tempo dipendente e non deterministico): in questo caso la predizione dei nomi di dominio non può essere eseguita in quanto il seme non può essere anticipato.
- **TIN-DGA**(tempo indipendente e non deterministico): predizione impossibile ed è difficile da individuare

Oltre ad essere classificati in base al tipo di seme utilizzato, gli algoritmi DGA posso essere divisi anche in base al loro schema di generazione:

- **Arithmetic-based DGA**: questo tipo di algoritmi calcolano una serie di valori ASCII che possono essere utilizzati direttamente per il nome di dominio oppure per creare un vocabolario che verrà poi utilizzato per creare i nomi di dominio.
- **Hash-based DGA**: in questo caso l'algoritmo si baserà sulla rappresentazione esadecimale di un hash per generare i domini.
- **Wordlist-based DGA**: l'algoritmo concatena sequenze di parole provenienti da una o più liste di parole, generando nomi di dominio meno casuali e quindi più camuffabili.
- **Permutation-based DGA**: si derivano tutti i possibili DGA attraverso la permutazione di un nome di dominio iniziale.

Ecco un esempio in linguaggio Python di algoritmo di generazione casuale:

```
1 import numpy as np
2
3 def generate_domain(year: int, month: int, day: int) -> str:
4     """Generate a domain name for the given date."""
```

```
    domain = ""

    for i in range(16):
        year = ((year ^ 8 * year) >> 11) ^ ((year & 0xFFFFFFF0)<<17)
        month = ((month ^ 4 * month) >> 25)^ 16 *(month & 0xFFFFFFF8)
        day = ((day ^ (day << 13)) >> 19) ^ ((day & 0xFFFFFFFE)<<12)
        domain += chr(((year ^ month ^ day) % 25) + 97)

    return domain + ".com"
```

Alcuni esempi di domini generati da un DGA:

```
dsmicmyhicyjixqh.co
lqcouvubglcsvauxci.com
liwcpvgbghyoueyh.to
gpjlwhjtuigo.la
tyykxdvbpbgbtinsjhb.so
yfckcikuotbiobat.bz
sudbjibcqtvlyajsarqe.ki
vdwvdgbqwxtbayw.to
inompasfyu.to
gpuoiuhkeobxcsjxe.im
bxevdbdw.kz
gxfowiabjsxoqf.ru
crqencenoehyrudh.nf
tmbeunyvciwwvtiu.org
etbcvjw.sh
ynlbbvexsfkh.us
whcnlxilcsxqaamwse.ir
wnslsgkrimsshofmcli.tj
bresloyxepgcf.net
dfikhlxyfypgmtgooxx.mx
gdyxedmwrqblepfgqw.co
cvmedawbn.tv
einqfdygcw.cx
qnxqhgtyogclb.ac
gsuhwnalhf.sh
aeosygdqrxgdwlt.in
oixnlqcprc.cx
iihiqfeprivko.us
rrcivwwxvtkbuebhobqau.im
```

Capitolo 3

Machine Learning

Come è stato preannunciato nei capitoli precedenti, in questa tesi verrà utilizzato il Machine Learning per riuscire a classificare i domini DGA in benevoli o malevoli. In particolare i modelli che andremo ad utilizzare saranno 2: il Multi-Layer-Perceptron e la rete LSTM.MI. La seconda non verrà trattata nel dettaglio dato che è lavoro di ricerca descritto in altri paper [7].

Il Machine Learning è un metodo di analisi dei dati che automatizza la costruzione di modelli analitici. È una branca dell'intelligenza artificiale e si basa sull'idea che i sistemi possono imparare dai dati, identificare modelli autonomamente e prendere decisioni con un intervento umano ridotto al minimo. Il Machine Learning utilizza algoritmi che imparano dai dati in modo iterativo permettendo, per esempio, ai computer di individuare informazioni anche sconosciute senza che venga loro segnalato esplicitamente dove cercarle. L'aspetto più importante dell'apprendimento automatico è la ripetitività, perché i modelli più sono alimentati con dati, più sono in grado di adattarsi a loro in maniera autonoma. Infatti, i computer imparano da elaborazioni precedenti per produrre e prendere decisioni autonome che siano affidabili e replicabili. Il rinnovato interesse verso il Machine Learning in questi ultimi anni è stato incentivato dalla continua crescita dei dati a disposizione e dai potenti processi di elaborazione disponibili a basso costo. Ciò consente di realizzare automaticamente modelli complessi per l'analisi dei dati ed elaborare velocemente risultati accurati anche su larga scala. La costruzione di tali modelli, consente per esempio alle aziende di identificare nuove opportunità di profitto o di evitare rischi non preventivati.

I metodi di Machine Learning più utilizzati sono l'apprendimento supervisionato e l'apprendimento non supervisionato. Nell'apprendimento supervisionato, gli algoritmi vengono addestrati utilizzando dati già classificati, cioè vengono forniti dei dati di input di cui già si conoscono gli output. Nel lavoro qui illustrato, per esempio,

si sono forniti i nomi di dominio già classificati con “0” (benevolo) o “1” (malevolo). L’algoritmo di apprendimento, impara a sua volta ad abbinare gli input agli output corrispondenti, comparando i risultati per trovarne gli errori e modificare il modello che sta costruendo. Attraverso metodologie come classificazione, regressione, previsione e gradient boosting, l’apprendimento supervisionato utilizza modelli per prevedere il valore della classe da assegnare ai dati non ancora classificati. Concentriamoci sulla classificazione. Tale metodologia è una sottocategoria dell’apprendimento supervisionato il cui obiettivo è quello di predire le etichette corrispondenti ai dati di nuove istanze basandosi sulle osservazioni passate. Per etichetta, o label, si fa riferimento a un valore assegnato ad ogni dato per indicare all’algoritmo la classe in cui esso rientra. Seppur la classificazione più semplice sia quella binaria, in cui i dati possono essere classificati con un’etichetta che può assumere solo due valori, le etichette possono anche essere più di due a patto che l’algoritmo abbia avuto modo di impararle nella fase di addestramento. Gli step fondamentali di un algoritmo di classificazione sono due:

- **Fase di training**: viene passato un dataset di addestramento (training set) che permette all’algoritmo di studiare i dati e le loro features al fine di capirne i pattern e creare un modello di previsione.

- **Fase di Test**: Il training set, è formato da dati di cui si conoscono già le classi di appartenenza, cioè di dati contenenti anche delle relative label. È la fase più onerosa dal punto di vista computazionale. Una volta terminata la fase di training e creato quindi un modello di previsione, viene fornito un ulteriore dataset da testare (test set), anche questo munito di label per valutare la bontà del modello creato in precedenza. A tale scopo, l’algoritmo cerca di classificare il test set assegnando delle label ai dati e successivamente, confronta le label predette con quelle reali per valutare l’accuratezza della previsione attraverso degli indicatori, che citeremo più avanti.

Generalmente, sia il training set che il test set vengono presi da uno stesso dataset di partenza, il quale viene suddiviso in due parti stabilendo delle percentuali di grandezza (per esempio 80% training e 20% test)

Apprendimento semi-supervisionato

L’apprendimento semi-supervisionato, ha le stesse applicazioni dell’apprendimento precedente con la sola differenza che per la fase di training utilizza sia dati classificati che non: solitamente un ridotto volume di dati classificati e un volume più ampio di dati non classificati. Questo apprendimento è utile se la classificazione ha

un costo troppo elevato per permettere un processo di apprendimento completamente supervisionato.

Apprendimento non supervisionato

L'apprendimento non supervisionato utilizza per la fase di training dati non classificati, non fornendo quindi al sistema la "risposta giusta". È l'algoritmo che deve scoprire cosa gli viene passato, esplorando i dati per individuare una qualche struttura interna. Questa tecnica di apprendimento, funziona bene con i dati transazionali. Ad esempio, per individuare consumatori con caratteristiche simili a cui rivolgere campagne di marketing specifiche oppure per scoprire le caratteristiche principali che differenziano le diverse tipologie di consumatori.

Apprendimento con rinforzo

L'apprendimento per rinforzo, è un tipo di apprendimento molto usato in robotica e videogiochi. Con l'apprendimento per rinforzo, l'algoritmo ricerca le azioni che generano le ricompense maggiori, passando per esperimenti ed errori. Questo tipo di apprendimento è basato su un processo a tre componenti: l'agente (chi impara o prende decisioni), l'ambiente (tutto ciò con cui l'agente interagisce) e le azioni (cosa può fare l'agente). L'obiettivo dell'agente è quello di scegliere le azioni che massimizzano la ricompensa prevista per un determinato lasso temporale. Scegliendo le azioni giuste, l'agente raggiungerà l'obiettivo più velocemente.

3.1 Reti Neurali

Una rete neurale è un modello computazionale usato nei problemi di apprendimento automatico, basato sul funzionamento semplificato di una rete neurale biologica. Possiamo vedere il modello come un gruppo di nodi, neuroni artificiali, interconnessi tra loro. Si presenta come un sistema adattivo in grado di modificare la sua struttura (nodi e interconnessioni) basandosi sia su dati esterni, che su informazioni interne che si correlano e attraversano la rete neurale durante la fase di apprendimento. Ricevono segnali esterni su uno strato di nodi, detti nodi di ingresso, e ognuno di questi sono a loro volta collegati ad altri nodi interni della rete, che tipicamente sono organizzati a più strati. Ogni nodo elabora i segnali ricevuti e li trasmette ai successivi fornendo via via elaborazioni sempre più dettagliate fino ad arrivare allo strato di uscita dove vengono raccolti i risultati. I parametri della rete vengono determinati, nel caso di

apprendimento supervisionato, sulla base dei dati del training set. Lo scopo dell'addestramento è quello di costruire un modello di processo che genera i dati e non di interpolare i dati di training. Una volta addestrata, la rete è capace di fornire una risposta corretta a nuovi dati di ingresso non presentati nella fase di addestramento.

Lo strumento di valutazione per valutare la bontà di un algoritmo di classificazione è la matrice di confusione. Quest'ultima, è una tabella che rappresenta la distribuzione sia delle classi assegnate inizialmente che di quelle predette dal classificatore, evidenziando sulla diagonale secondaria la ripartizione degli errori mentre su quella principale le unità statistiche classificate, come mostrato in **figura 3.1**. Tipicamente le righe rappresentano le etichette iniziali mentre le colonne riportano le etichette predette. Ogni cella contiene quindi il numero di predizioni effettuate dal classificatore della relativa combinazione riga/colonna. In un'applicazione di classificazione binaria, come nel nostro caso, se per esempio classifichiamo il dataset iniziale convenzionalmente con i valori "P" per i domini benevoli e il valore "N" per i domini malevoli, gli esiti predetti dal classificatore possono essere ricondotti a quattro casi possibili:

- **Vero Positivo (TP, True Positive)**: il classificatore assegna il valore "P" a un dato appartenente alla classe "P".
- **Falso Positivo (FP, False Positive)**: il classificatore assegna il valore "P" a un dato appartenente alla classe "N".
- **Vero Negativo (TN, True Negative)**: il classificatore assegna il valore "N" a un dato appartenente alla classe "N".
- **Falso Negativo (FN, False Negative)**:il classificatore assegna il valore "N" a un dato appartenente alla classe "P".

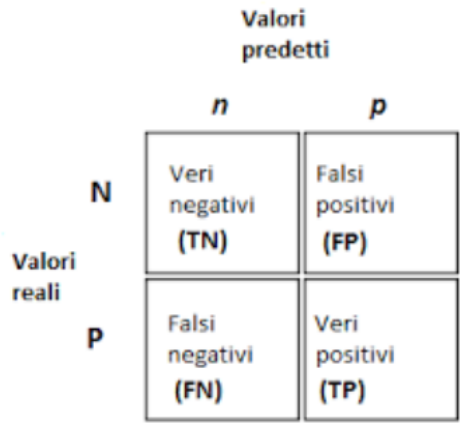

Figura 3.1: Matrice di confusione nei test di classificazione binaria.

Dalla matrice di confusione, possono essere calcolati a loro volta diversi indici di valutazione come:

- **Accuracy**: valuta l'accuratezza del modello. Assume il valore 1 nel caso di accuratezza massima, mentre assume il valore zero quando l'accuratezza è minima.

$$Accuracy = \frac{TP + TN}{TN + FP + FN + TP} \tag{3.1}$$

- **Precision**: la precisione corrisponde alla capacità di un classificatore di non etichettare un campione malevolo come benevolo. Di conseguenza, più questo indice è grande e minore è il numero di falsi positivi.

$$Precision_m = \frac{TP}{TP + FP} \tag{3.2}$$

- **Recall**: la recall (o True Positive Rate) è la capacità del classificatore di trovare tutti i campioni positivi, di conseguenza, più questo indice è grande e minore è il numero di falsi negativi.

$$Recall = \frac{TP}{TP + FN} \tag{3.3}$$

- $\mathbf{F1}_{score}$: Il $\mathbf{F1}_{score}$ rappresenta la media armonica della precision e della recall, dove un punteggio $F1_{score}$ raggiunge il suo valore migliore a 1 e il punteggio peggiore a 0. La formula per il F1score è:

$$F1_{score} = 2 * \frac{(precision * recall)}{(precision + recall)} \tag{3.4}$$

3.2 Multi-Layer-Perceptron

Un percettrone multistrato (MLP) è una classe di reti neurali artificiali (ANN) feedforward. Il termine MLP è usato in modo ambiguo, a volte vagamente per indicare qualsiasi ANN feedforward, a volte per riferirsi strettamente a reti composte da più strati di percettroni (con attivazione di soglia). I percettroni multistrato sono talvolta denominati colloquialmente reti neurali "vaniglia", soprattutto quando hanno un singolo strato nascosto.

Come possiamo vedere dalla **figura 3.2**, un MLP è costituito da almeno tre livelli di nodi: un livello di input, un livello nascosto e un livello di output. Ad eccezione dei nodi di input, ogni nodo è un neurone che utilizza una funzione di attivazione non lineare. MLP utilizza una tecnica di apprendimento supervisionato chiamata back-propagation per la creazione. I suoi molteplici strati e l'attivazione non lineare distinguono MLP da un percettrone lineare. Può distinguere dati che non sono separabili linearmente.

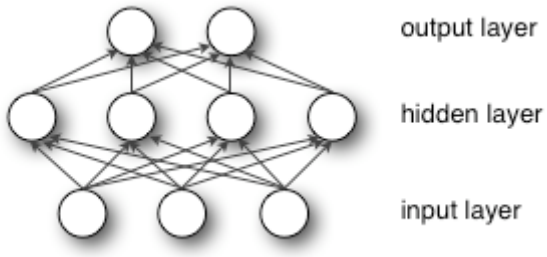

Figura 3.2: Modello di Percettrone Multi-Strato

3.2.1 Funzione di attivazione

Se un percettrone multistrato ha una funzione di attivazione lineare in tutti i neuroni, cioè una funzione lineare che mappa gli input ponderati all'output di ciascun neurone, l'algebra lineare mostra che qualsiasi numero di strati può essere ridotto a un input a due strati con modello di output. Nelle MLP alcuni neuroni utilizzano una funzione di attivazione non lineare che è stata sviluppata per simulare la frequenza dei potenziali d'azione, o attivazione, dei neuroni biologici.

Le due funzioni di attivazione più conosciute sono entrambe sigmoidi e sono descritte da:

$$y(v_i) = tanh(v_i) \tag{3.5}$$

e

$$y(v_i) = (1 + e^{-v_i})^{-1} \tag{3.6}$$

Nei recenti sviluppi del deep learning l'unità lineare raddrizzatore (ReLU) è più frequentemente utilizzata come uno dei possibili modi per superare i problemi numerici legati alla funzione sigmoide.

La prima è una tangente iperbolica che va da -1 a 1, mentre l'altra è la funzione logistica, che è simile nella forma ma varia da 0 a 1. Qui y_i è l'output del i-esimo nodo (neurone) e v_i è la somma ponderata delle connessioni di input. Sono state proposte

funzioni di attivazione alternative, comprese le funzioni raddrizzatore e softplus. Le funzioni di attivazione più specializzate includono funzioni di base radiale (utilizzate nelle reti di base radiale, un'altra classe di modelli di rete neurale supervisionati).

3.2.2 Strati

L'MLP è costituito da tre o più livelli (un livello di input e uno di output con uno o più livelli nascosti) di nodi ad attivazione non lineare. Poiché gli MLP sono completamente connessi, ogni nodo in un livello si collega con un certo peso w_{ij} a ogni nodo nel livello successivo.

3.2.3 Apprendimento

L'apprendimento avviene nel percettrone modificando i pesi della connessione dopo l'elaborazione di ogni dato, in base alla quantità di errore nell'output rispetto al risultato atteso. Questo è un esempio di apprendimento supervisionato e viene effettuato tramite back-propagation, una generalizzazione dell'algoritmo dei minimi quadrati medi nel percettrone lineare.

Possiamo rappresentare il grado di errore in un nodo di output j nel punto dati n-esimo (esempio di addestramento) da $e_j(n) = d_j(n) - y_j(n)$, dove d è il valore target e y è il valore prodotto dal percettrone. I pesi dei nodi possono quindi essere regolati in base a correzioni che riducono al minimo l'errore nell'intero output, dato da:

$$\varepsilon(n) = \frac{1}{2}\sum_j e_j^2(n) \tag{3.7}$$

Usando la discesa del gradiente, il cambiamento in ogni peso è

$$\Delta w_{ji}(n) = -\eta \frac{\partial \varepsilon(n)}{\partial v_j(n)} y_i(n) \tag{3.8}$$

dove y_i è l'output del neurone precedente e η è il tasso di apprendimento, che è selezionato per garantire che i pesi convergano rapidamente in una risposta, senza oscillazioni.

La derivata da calcolare dipende dal campo locale indotto v_j, che a sua volta non varia. È facile dimostrare che per un nodo di output questa derivata può essere semplificata

$$-\frac{\partial \varepsilon(n)}{\partial v_j(n)} = e_j(n)\phi'(v_j(n)) \tag{3.9}$$

dove $\phi'(v_j(n))$ è la derivata della funzione di attivazione sopra descritta, che a sua volta non varia. L'analisi è più difficile in un nodo nascosto per la modifica dei pesi, ma si può dimostrare che la derivata rilevante è

$$-\frac{\partial \varepsilon(n)}{\partial v_j(n)} = \phi'(v_j(n)) \sum_k -\frac{\partial \varepsilon(n)}{\partial v_k(n)} w_{kj}(n) \tag{3.10}$$

Questo dipende dalla variazione dei pesi dei k-esimi nodi, che rappresentano lo strato di output. Quindi, per modificare i pesi dello strato nascosto, i pesi dello strato di output cambiano in base alla derivata della funzione di attivazione, e quindi questo algoritmo rappresenta una propagazione all'indietro della funzione di attivazione.

Capitolo 4

Materiale, metodi e workflow

L'esperimento, soggetto di studio di questo testo, è stato basato sul confrontare i due modelli citati in precedenza: il primo è un Multi-Layer-Perceptron, il secondo una rete LSTM.MI, entrambi addestrati sullo stesso dataset vecchio di un anno. Come già detto ci concentreremo sulla descrizione e sul test del MLP per poi confrontare i risultati con quelli della rete LSTM.MI.

Il classificatore basato su MLP è diviso in 7 moduli i quali hanno 2 compiti principali: i primi sei moduli devono calcolare le features con delle tecniche che vedremo nella prossima sezione, l'ultimo modulo, il settimo, ha invece il compito di testare il dataset utilizzando le features dei primi sei, per poi mostrare le varie statistiche sul risultato ottenuto dal test.

4.1 Struttura del classificatore basato su MLP

Il MLP è diviso in 7 moduli i quali hanno 2 compiti principali: i primi sei moduli hanno il dovere di calcolare le features con delle tecniche che vedremo nella prossima sezione, l'ultimo modulo, il settimo, aveva invece il compito di testare il dataset utilizzando le features dei primi sei, per poi mostrare le varie statistiche sul risultato ottenuto dal test. Nella figura 4.1 è mostrato il workflow del classificatore diviso nei 7 moduli, a loro volta divisi nei vari script importati dai moduli. Sono presenti anche i vari input ed output di ogni modulo.

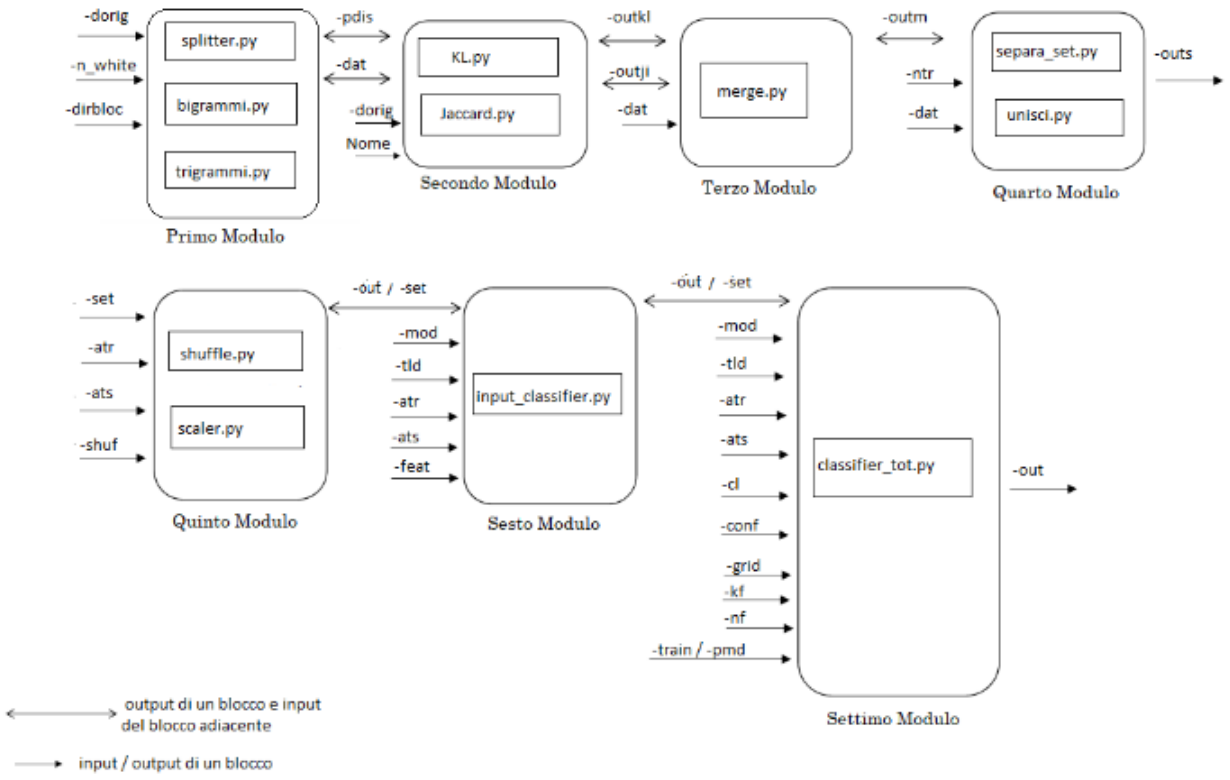

Figura 4.1: Workflow classificatore e divisione in moduli.

È importante sottolineare che tali moduli possono anche essere utilizzati singolarmente in quanto non è presente nessun tipo di correlazione tra di loro.

Primo modulo

In questo modulo viene eseguita l'operazione di split o separazione del Domain Name (DN), che viene implementata attraverso la funzione splitter. In particolare il modulo prende i Domain Name e li divide in SLD e TLD utilizzando il punto come carattere di divisione tra i due. Completata la divisione inizia il calcolo dei bigrammi e dei trigrammi di ogni singolo dominio. Nei capitoli successivi questi passaggi verranno spiegati meglio.

Secondo modulo

Il secondo modulo calcola le features tramite la Divergenza di Kullback-Leibler che è una misura della perdita di entropia nell'informazione e tramite l'Indice Jaccard, che invece è un indice statistico che misura la similarità di due insiemi campionati. Per una spiegazione più dettagliata si rimanda al **capitolo 6**.

Terzo modulo

Questo modulo esegue l'unione in unico file dei due file prodotti in output dalla KL e dalla Jaccard per ogni blocco del dataset; dopo aver caricato i due file che gli sono stati passati relativi alla KL e alla Jaccard, estrae i due header dei due file e crea un header unico contenente: un indice per tutte le features della KL e della Jaccard, il SLD, il TLD, la famiglia e la label; tale header sarà inserito in testa all'interno del file prodotto in output. Successivamente procede per ogni DN del blocco, all'unione di tutte le features che saranno inserite all'interno del file di output, verificando opportunamente che i domini di entrambi i file siano uguali, aggiungendo anche in coda ad ogni DN la famiglia di appartenenza e la label d'identificazione. In output produrrà un unico file chiamato: nomeblocco_features.txt

Quarto modulo

Normalmente il classificatore necessita di due dataset: uno di training e uno di dataset, motivo per il quale questo modulo effettua la divisione del blocco di output del terzo modulo in due parti. In questo caso però il classificatore era già addestrato, motivo per il quale è stato necessario fornire allo script un particolare parametro di input per far sì che il dataset non venga diviso.

Quinto modulo

Questo modulo mischia i DN all'interno di ogni set, attraverso la funzione shuffle. L'header di intestazione del file viene preventivamente escluso dallo shuffle evitando così che possa creare problemi negli script successivi, in quanto esso contiene esclusivamente caratteri e non numeri; In output si ottiene un unico file contenente tutti i Domain Name del Set, il cui nome sarà composto dal nome iniziale del Set con l'aggiunta in coda del termine "_shuf", (ad es. set_A_train_shuf.txt), situato all'interno della cartella definita attraverso il parametro path. Viene quindi effettuata la normalizzazione delle features attraverso l'algoritmo Min-Max Scaler che si basa sulla seguente formula:

$$X_{new} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{4.1}$$

La formula ritorna un valore compreso tra 0 e 1.

Sesto Modulo

In questo script vengono preparati i dati che saranno utilizzati per la classificazione dei domini: si carica dal file definito attraverso path_file_features le features da estrarre, che saranno inserite all'interno di un array. Se viene definita attraverso il parametro classification_task una classificazione multi-classe si invoca una funzione per entrambi i file A-train e A-test, che assegna ad ogni DN un dga_family_code, ossia un indice che identifica a quale famiglia i DN appartengano; tale indice viene ricavato da un dizionario Python, contenente all'interno un indice che identifica ogni famiglia del dataset; in output da questa funzione ottengo un file,contenuto all'interno della cartella definita da path_set, il cui nome sarà composto dal nome definito dall'argomento file con in aggiunta _dga_code. Tali file saranno poi eliminati, in quanto non più utili.

Settimo modulo

Il settimo è il modulo che effettua la classificazione e che produce il report finale. Il classificatore Multi-Layer-Perceptron viene utilizzato importando la libreria scikit-learn, che viene utilizzata anche precedentemente per la normalizzazione e per scrivere il report delle statistiche.

Capitolo 5

Creazione dataset

La maggior difficoltà dell'esperimento è stata riuscire a creare il dataset di test: per farlo è stato necessario scrivere uno script che scaricasse i file da `https://osint.bambenekconsulting.com/feeds/dga-feed.txt`[1]. Lo stesso processo è stato applicato ai domini del Majestic Million, che fortunatamente permette di scaricare il file csv direttamente dal sito: `https://it.majestic.com/reports/majestic-million`.
Una volta scaricato i file contenenti tutti i domini, è stato necessario processare i dati in modo da togliere tutto il superfluo e infine unire il tutto in un unico file csv. A differenza dei dati Majestic Million, i dati Bambenek vengono aggiornati giornalmente, il che significa che il download del file txt deve essere effettuato ogni giorno, motivo per il quale abbiamo aggiunto lo script al cronjob di un server dell'università per permettere il download quotidiano.
Sotto è presente il listato dello script "prototipo" aggiunto al cronjob.

```
'''
Written by Luca Mannini
'''
import urllib3
from datetime import date
import os
from os import path

fileName = "dnsBamb/" + date.today().strftime("%m-%d-%y") + ".txt"
list = []
if path.exists(fileName):
    print("Testo gia' scaricato in: " + os.getcwd())

else:
    http = urllib3.PoolManager()
```

[1]Purtroppo il sito ora richiede una licenza.

```
    response = http.request('GET', "http://osint.bambenekconsulting.
   com/feeds/dga-feed.txt")
    print("response OK")
    data = response.data.decode('utf-8')
    print("decode OK")
    f = open(fileName, "x")
    for line in data.splitlines():
        splitted=line.split(",")
        list.append(splitted[0])
    print ("split OK")
    print("Download del txt completato.")
    for e in list:
        f.write("\n" + e)

    f.close()
```

Una volta scaricati i domini Bambenek, è stato essenziale a fini dell'esperimento trovare l'intersezione tra l'insieme dei domini Bambenek e l'insieme dei domini DNS presenti nei log di traffico della rete GARR, questo per far sì che siano presenti solamente DN di traffico reale. Il codice sotto è il prototipo dello script che effettua la ricerca dell'intersezione tra i due insiemi citati.

```
# E' NECESSARIO CHE NEL FILE DEL BAMBENEK NON CI SIANO SPAZI VUOTI O
   ELEMENTI ERRATI (TIPO I CANCELLETTO)

'''
Written by Luca Mannini
'''
import urllib3, glob, os, sys
import json
from datetime import date
from os import path


def intersection(listaBambenek, listaGarr):
    setBambenek=set(listaBambenek)
    setGarr=set(listaGarr)
    return setBambenek.intersection(setGarr)

def writeInFile(dictInters):
    os.chdir("../dnsIntersection")
    fileName = date.today().strftime("%m-%d-%y") + "_intersection.txt
   "
    with open(fileName, 'a') as file:
        for o in dictInters:
            file.write(o + "\n")
```

```
23          file.truncate()
24
25
26
27  def getGarrFiles(dirGarr):
28      arrayFiles = []
29      os.chdir(dirGarr)
30      for file in glob.glob("*.log"):
31          arrayFiles.append(file)
32
33      os.chdir("..")
34      return arrayFiles
35
36  def getBambenekFiles(dirBamb):
37      arrayFiles=[]
38      os.chdir(dirBamb)
39      for file in glob.glob("*.txt"):
40          arrayFiles.append(file)
41      os.chdir("..")
42      return arrayFiles
43
44
45  def findIntersection():
46      dirGarr="dnsGarr"
47      dirBamb="dnsBamb"
48      garrFiles=getGarrFiles(dirGarr)
49      bambenekFile=getBambenekFiles(dirBamb)
50      listaGarr = []
51      listaBambenek = []
52      rigaGarr = []
53      num_el_File=0
54      counterFile=0
55      num_el_Garr=0
56      intersezione = set()
57
58      os.chdir(dirBamb)
59      for bambFile in bambenekFile:
60          with open(bambFile, "r") as f1:
61              for line in f1:
62                  line = line.strip() #elimina \n alla fine della riga
63                  if(line!="" or  not line.startswith("#")):
64                       listaBambenek.append(line)
65      print("\nLunghezza lista da Bambenek: " + str(len(listaBambenek))
        )
66      os.chdir("..")
67      os.chdir(dirGarr)
```

```
68
69
70      for file in garrFiles:
71
72          if(counterFile % 12 == 0):
73              print("listaGarr flushata")
74              tempSet=intersection(listaBambenek, listaGarr)
75              intersezione=intersezione.union(tempSet)
76              listaGarr=[]
77          num_el_File=0
78          with open(file, "r") as f2:
79              counterFile+=1
80              for line in f2:
81                  num_el_File+=1
82                  num_el_Garr+=1
83                  line = line.strip()
84                  if(not line.startswith("#")):
85                      rigaGarr=line.split(";")
86                      if rigaGarr[5].endswith("."):
87                          rigaGarr[5]=rigaGarr[5][:-1]
88                      listaGarr.append(rigaGarr[5])
89              print("Numero di elementi nel file " + file + ": " + str(
    num_el_File))
90
91      print("Numero totale di elementi dal Garr: " + str(num_el_Garr))
92      print("Lunghezza intersezione: " + str(len(intersezione)))
93
94
95      answer=input("\nVuoi salvare l'intersezione? [Y/N]: ")
96      while(answer!="Y" and answer!="N"):
97          answer=input("\n Vuoi salvare l'intersezione? [Y/N]: ")
98      if(answer=="Y"):
99          writeInFile(intersezione)
100
101
102
103 def main():
104     print("\nRicerca dell'intersezione dei domini...")
105     findIntersection()
106
107 if __name__ == "__main__":
108     main()
```

Le famiglie di domini DGA accettate dal classificatore sono le seguenti:

- murofet
- pykspa
- padcrypt
- ramnit
- ranbyus
- simda
- ramdo
- suppobox
- gozi
- qadars
- symmi
- tinba
- rovnix
- fobber
- corebot
- matsnu
- vawtrak
- necurs
- pushdo
- cryptolocker
- dircrypt
- emotet
- kraken
- nymaim

- conficker

Come ultimo ritocco, è stato importante rimodellare il dataset e si è deciso di optare per la costruzione dello stesso componendolo di 30000 domini Majestic Million (whitelist) e circa 15000 Bambenek (blacklist), di cui solo un centinaio appartenenti alla famiglia "Vawtrak" e i restanti tutti appartenti alla famiglia "Necurs". Lo script finale comprendente tutti i moduli sopra citati sarà listato in Appendice. Il risultato finale è il dataset sotto rappresentato. (Alexa indica i domini Majestic Million, quindi i benevoli)

```
legit,famiglia,dominio
legit,alexa,preventionweb.net
legit,alexa,marantz.co.uk
legit,alexa,mini.it
legit,alexa,massimorebecchi.it
legit,alexa,monitoruldevrancea.ro
legit,alexa,luondo.nl
legit,alexa,imagecolorpicker.com
legit,alexa,theshiftnews.com
legit,alexa,ionio.gr
legit,alexa,businesscycle.com
legit,alexa,vub.ac.be
legit,alexa,tableau.com
legit,alexa,igsd.org,
legit,alexa,topendsports.com
legit,alexa,sunrisehouse.com
dga,necurs,mhdgicdq.nu
dga,necurs,nlqhhanwmyqfn.eu
dga,necurs,estgmkelcpwqcmdbyfe.so
dga,necurs,hqfkbreitujvxtdsyi.sc
dga,necurs,jaltiorswxklv.to
dga,necurs,cqpytndpxcod.ir
dga,necurs,euamsrhiulsofg.co
dga,necurs,dcovdqywtwhhylhnfocn.com
dga,necurs,npbdxqiexponbiiikfhw.tv
dga,necurs,kdgaqfavfyyorp.tv
dga,necurs,nhanlxdsbtxhmx.tj
dga,necurs,lvomylxkxuaxavnbs.tj
dga,necurs,lrnaoeprpjpsvurhk.in
dga,necurs,pmsjrijhnr.im
dga,necurs,qmhvjif.ru
dga,necurs,jxlennpqpxxbbl.de
dga,necurs,oicxmrntjwtg.so
```

Nonostante sia stato necessario creare il dataset per la rete LSTM.MI e in generale per immagazzinare tutti i domini utilizzati per il test, il MLP necessitava poi di dividere le famiglie in file singoli da posizionare all'interno della cartella /dataset_collapse.

Capitolo 6

Estrazione features e configurazione

Le features sulle quali vengono poi testati i classificatori sono estratte a partire dai bigrammi e dai trigrammi. I bigrammi sono le sequenze di due lettere all'interno di una singola parola, i trigrammi sono invece le sequenze di tre lettere. All'interno del primo script viene eseguita l'operazione di calcolo delle distribuzioni dei vari bigrammi per ogni blocco del dataset. Per ogni dominio del blocco viene utilizzato solamente il SLD (Secondary-Level-Domain). Viene utilizzato come riferimento per il calcolo delle distribuzioni un dizionario Python contenente tutti i bigrammi, esso è stato generato attraverso l'utilizzo di 38 caratteri: tutte le lettere dell'alfabeto, i numeri e i due segni di punteggiatura tra i quali il trattino alto e il trattino basso; tutti questi caratteri sono combinati tra loro, generando 1444 bigrammi e per ognuno è presente un contatore che indicherà la sua distribuzione. Tale dizionario viene importato da un file json, in modo da rendere più semplice la modifica e l'aggiunta di nuovi bigrammi, senza dover modificare il codice.
Analogamente alle distribuzioni di bigrammi, anche i trigrammi vengono calcolati utilizzando come riferimento per il calcolo delle distribuzioni un dizionario python. Esso contiene 54872 trigrammi e per ogni trigramma è presente un contatore che indicherà la sua distribuzione. Tale dizionario viene importato da un file json. Inizialmente la funzione calcola tutti i trigrammi per ogni SLD che vengono inseriti all'interno di un array. Ad es. per il termine "google" i trigrammi sono: 'goo', 'oog', 'ogl', 'gle'. Dopodiché per ogni trigramma viene incrementato il suo contatore nel dizionario Python; dopo aver eseguito tutte queste operazioni per ogni SLD del blocco si procede al calcolo dell'intera distribuzione. In output lo script produce un file json con la distribuzione calcolata.

6.1 Kullback-Leibler Divergence

La Kullback-Leibler Divergence, o semplicemente K-L Divergence, è una misura non-simmetrica della "distanza"tra due distribuzioni di probabilità P e Q.In particolare, la K-L Divergence di Q da P, indicata con $D_{KL}(P||Q)$, è la misura dell'informazione persa quando Q è usata per approssimare P. La divergenza (o distanza) tra due distribuzioni è quindi data da:

$$\sum_{i=1}^{n} P(i) log \frac{P(i)}{Q(i)} \tag{6.1}$$

dove n è il numero dei possibili valori contenuti nelle distribuzioni. La distribuzione di probabilità P rappresenta la distribuzione di test mentre la distribuzione Q rappresenta la distribuzione di riferimento. Nel nostro caso, verranno prese come distribuzioni di riferimento Q tutte le distribuzioni di bigrammi e trigrammi di ogni blocco di dominio associato alle 25 DGA e alla whitelist del dataset. Si andrà quindi, mediante la K-L Divergence, a calcolare la "distanza" di ogni singolo dominio presente in ogni blocco del dataset con le distribuzioni di riferimento, andando così ad ottenere per ogni dominio n features, dove n è il numero di distribuzioni di riferimento.

6.2 Jaccard Index

L'indice di Jaccard, o Jaccard Index, è un indice statistico utilizzato per misurare la somiglianza tra un insieme noto di componenti e una distribuzione di test. È definito come:

$$JI(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{6.2}$$

dove A e B sono due SLD, rappresentati nel nostro caso da bigrammi e trigrammi.

Per il calcolo della Jaccard Index viene utilizzato solo il Secondary Level Domain: vengono estratti i bigrammi di ogni dominio del blocco del dataset (A) e viene calcolato il Jaccard Index con i bigrammi dei nomi di dominio presenti nelle varie famiglie della blacklist e nella whitelist. Al termine del confronto tra un dominio in e tutti quelli di un blocco, i risultati ottenuti vengono sintetizzati calcolando la loro media. Il procedimento è analogo per i trigrammi.

6.3 Configurazione e lancio dei moduli

Ricapitolando, nella lista qui sotto possiamo vedere i comandi lanciati per effettuare due test di classificazione sul dataset composto da Domain Name derivanti da traffico reale in configurazione "full domain" (ovvero con il dominio completo, punto compreso), utilizzando un classificatore basato su Multi-layer-perceptron pre-addestrato su dati di training vecchi di un anno. Il primo test aveva l'obiettivo di vedere le performance del modello su una classificazione binaria, ovvero dividendo i domini in 2 gruppi, domini benevoli o malevoli (whitelist e blacklist).
Il secondo test verteva invece sull'identificare la famiglia DGA di appartenenza dei domini catalogati come malevoli.

Elenco comandi da terminale

```
1- python3 primo_blocco.py -dorig ./originali -n_white 1 -dirbloc ./
   function/primo_blocco -dat ./MIO_OUTPUT/dataset_collapse -pdis ./
   MIO_OUTPUT/distribuzioni

2- python3 secondo_blocco.py -dat ./MIO_OUTPUT/dataset_collapse -dor
   ./MIO_OUTPUT/setAtrain -pdis ./MIO_OUTPUT/Distribuzioni_training -
   outkl ./output_KL -outji ./output_JI $NOME_DGA &

3- python3 terzo_blocco.py -dat ./MIO_OUTPUT/dataset_collapse -outkl
   ./output_KL -outji ./output_JI -outm ./output_merge

4- python3 quarto_blocco.py -dat ./MIO_OUTPUT/dataset_collapse -outm
   ./output_merge -outs ./Set -ntr 1

5- python3 quinto_blocco.py -set ./Set -atr set_A_train -ats
   set_A_test -out ./Set_norm -shuf

6- python3 sesto_blocco.py -set ./Set_norm -ats set_A_test_shuf_norm
   -feat ./function/sesto_blocco/features.txt -out ./
   Set_custom_binary -mod $TIPO_CLASSIFICAZIONE -tld

7- python3 settimo_blocco.py -set ./Set_custom_ml -conf ./function/
   settimo_blocco -ats set_A_test_shuf_norm_Multiclass_tld_custom  -
   cl mlp -mod $TIPO_CLASSIFICAZIONE -tld -pmd ./MIO_OUTPUT/modelli/
   binario_con_TLD/saved_model_mlp_fold_1.pkl -out ./out_class
```

Il secondo modulo è stato lanciato 27 volte, inserendo ogni volta una diversa famiglia di DGA al posto di $NOME_DGA quanto i comandi da lanciare erano 27 (uno per ogni famiglia di DGA). Il modello pre-addestrato viene caricato nel modulo sette passando il path all'argomento '-pmd', inoltre è stato lanciato due volte, cambiando il tipo di classificazione da binaria a multiclasse al secondo lancio tramite il parametro $TIPO_CLASSIFICAZIONE.

Capitolo 7

Risultati sperimentali

Sotto sono presenti i risultati di ogni classificatore, in cui va notato che per la classificazione multiclasse, per la maggior parte delle classi la precision e le altre statistiche danno zero poiché ovviamente i domini di test non contenevano alcun dominio di quelle classi di DGA.

```
Load test set A (minutes) : 0.36897369225819904
Test on: set_A_train_shuf_norm_Binary_tld_customTesting (minutes)
   0.0995976448059082

Class report:
              precision    recall  f1-score   support

           0    0.68506   0.65500   0.66969     29649
           1    0.37632   0.40874   0.39186     15100

   micro avg    0.57190   0.57190   0.57190     44749
   macro avg    0.53069   0.53187   0.53078     44749
weighted avg    0.58088   0.57190   0.57594     44749

Matrice di confusione =
TN: 19420  FP: 10229
FN: 8928  TP: 6172
```

Listato 7.1: Risultati per il Multi-layer-Perceptron in classificazione binaria

```
Testing (minutes)0.394775036971

Class report:
              precision    recall  f1-score   support

       white     0.9896    0.9835    0.9866     30001
       black     0.9677    0.9795    0.9735     15106
```

```
8
9     micro avg     0.9822    0.9822    0.9822     45107
10    macro avg     0.9786    0.9815    0.9801     45107
11 weighted avg     0.9823    0.9822    0.9822     45107
12
13 F1_score Precision Recall dataset
14               macro     micro
15 f1_score   0.980054  0.982176
16 precision  0.978647  0.982176
17 recall     0.981506  0.982176
18
19 Overall accuracy = 0.9821757155208726
20
21 Confusion Matrix:
22        white  black
23 white  29507    494
24 black    310  14796
25 Accuracy for each class:
26          white 0.984
27          black 0.979
28
29
30
31 True positive: 44303
```

Listato 7.2: Risultati per la rete LSTM-MI in classificazione binaria

```
1 Load test set A (minutes) : 0.35798555612564087
2 Test on: set_A_test_shuf_norm_Multiclass_tld_customTesting (minutes)
     0.06909266312917074
3 Class report:
4                precision    recall  f1-score   support
5
6         white    0.63979   0.89321   0.74555     29648
7     conficker    0.00000   0.00000   0.00000         0
8       corebot    0.00000   0.00000   0.00000         0
9          gozi    0.00000   0.00000   0.00000         0
10       kraken    0.00000   0.00000   0.00000         0
11      murofet    0.00000   0.00000   0.00000         0
12       necurs    0.14943   0.00087   0.00172     15000
13       nymaim    0.00000   0.00000   0.00000         0
14     padcrypt    0.00000   0.00000   0.00000         0
15       pushdo    0.00000   0.00000   0.00000         0
16       pykspa    0.00000   0.00000   0.00000         0
17       qadars    0.00000   0.00000   0.00000         0
18        ramdo    0.00000   0.00000   0.00000         0
19       ramnit    0.00000   0.00000   0.00000         0
```

```
      rovnix    0.00000   0.00000   0.00000         0
    suppobox    0.00000   0.00000   0.00000         0
       symmi    0.00000   0.00000   0.00000         0
       tinba    0.00000   0.00000   0.00000         0
     vawtrak    0.00658   0.01980   0.00988       101

   micro avg    0.59212   0.59212   0.59212     44749
   macro avg    0.04188   0.04810   0.03985     44749
weighted avg    0.47399   0.59212   0.49456     44749

Overall accuracy = 0.5921249636863394
Accuracy for each class:
        white 0.893
      conficker nan
      corebot nan
      gozi nan
      kraken nan
        murofet nan
        necurs 0.001
        nymain nan
        padcrypt nan
        pushdo nan
      pykspa nan
        necurs nan
        qadars nan
      ramdo nan
        ramnit nan
        rovnix nan
        suppobox nan
        symmi nan
      tinba nan
        vawtrak 0.02
```

Listato 7.3: Risultati per il Multi-layer-Perceptron in classificazione multiclasse

```


Testing (minutes)0.391864180565

Class report:
              precision    recall  f1-score   support

      necurs     0.9977    0.8251    0.9032     15000
       alexa     0.9896    0.9835    0.9866     30001
    dromedan     0.0000    0.0000    0.0000         6
     vawtrak     0.7500    0.8700    0.8056       100

```

```
   micro avg     0.9913    0.9305    0.9599     45107
   macro avg     0.6843    0.6697    0.6738     45107
weighted avg     0.9916    0.9305    0.9583     45107

F1_score Precision Recall dataset
              macro     micro
f1_score   0.112306  0.930476
precision  0.114053  0.930476
recall     0.111611  0.930476

Overall accuracy = 0.9304764227281797

[27 rows x 27 columns]
Accuracy for each class:
       murofet nan
        pykspa nan
      padcrypt nan
        ramnit nan
       ranbyus nan
         simda nan
         ramdo nan
      suppobox nan
          gozi nan
        qadars nan
         symmi nan
         tinba nan
        rovnix nan
        fobber nan
         alexa 0.984
       corebot nan
        matsnu nan
       vawtrak 0.87
        necurs 0.825
        pushdo nan
  cryptolocker nan
      dircrypt nan
        emotet nan
        kraken nan
        nymaim nan
     conficker nan
      dromedan 0.0

True positive: 41971
```

Listato 7.4: Risultati per la rete LSTM-MI in classificazione multiclasse

Ecco le tabelle riassuntive con la comparazione dei risultati:

Tabella 1		Classificazione binaria			
		Precision	Recall	$f1_{score}$	Support
LSTMI.MI	white	0.9896	0.9835	0.9866	30001
	black	0.9677	0.9795	0.9735	15106
	weighted avg	0.9823	0.9822	0.9822	45107
	micro avg	0.9822	0.9822	0.9822	45107
	macro avg	0.9786	0.9815	0.9801	45107
MLP	white	0.68506	0.65500	0.66969	29649
	black	0.37632	0.40874	0.39186	15100
	weighted avg	0.58088	0.57190	0.57594	44749
	micro avg	0.57190	0.57190	0.57190	44749
	macro avg	0.53069	0.53187	0.53078	44749

Tabella 2		Classificazione multiclasse			
		Precision	Recall	$f1_{score}$	Support
LSTMI.MI	white	0.9896	0.9835	0.9866	30001
	necurs	0.9977	0.8251	0.9032	15000
	dromedan	0.0000	0.0000	0.0000	6
	vawtrak	0.7500	0.8700	0.8056	100
	micro avg	0.9913	0.9305	0.9599	45107
	macro avg	0.6843	0.6697	0.6738	45107
	weighted avg	0.9823	0.9822	0.9822	45107
MLP	white	0.63979	0.89321	0.74555	29648
	necurs	0.14943	0.00087	0.00172	15000
	vawtrak	0.00658	0.01980	0.00988	101
	micro avg	0.59212	0.59212	0.59212	44749
	macro avg	0.04188	0.04810	0.03985	44749
	weighted avg	0.47399	0.59212	0.49456	44749

Nella tabella 1 è rappresentato il confronto tra LSTM.MI e MLP in classificazione binaria. Sono presenti le statistiche spiegate precedentemente nel capitolo 3.1 calcolate per domini malevoli e benevoli. Nella tabella 2 invece saranno indicate invece le famiglie di DGA sulle quali è stato eseguito il test. Per esempio, la tabella indica che il LSTM.MI ha una precision del 98% nel riconoscimento dei domini white, e del 96% per i domini black. Il termine Support indica il numero di domini presenti nel dataset di test per una determinata classe e ovviamente nel caso delle medie (micro, macro, weighted avg), il valore indica la somma di tutti i domini presenti nel dataset.

Le micro e macro medie (per qualsiasi metrica) sono leggermente diverse e la loro interpretazione differisce. Una macro-media calcolerà la metrica in modo indipendente

per ogni classe e quindi prenderà la media (quindi trattando tutte le classi allo stesso modo), mentre una micro-media aggregherà i contributi di tutte le classi per calcolare la metrica media. In una configurazione di classificazione multi-classe, la micro-media è preferibile se si sospetta che ci possa essere uno squilibrio tra le classi(in inglese multiclass-imbalance, vale a dire che si possono avere molti più esempi di una classe rispetto ad altre). Per illustrare il motivo, prendiamo ad esempio la precisione

$$Pr = \frac{TP}{TP + FP} \tag{7.1}$$

ricordando che TP e FP sono rispettivamente i True Positives e i False Positives. Immaginiamo di avere ad esempio un sistema di classificazione multi-classe One-vs-All (esiste solo un output di classe corretto) con quattro classi e i seguenti numeri durante il test:

- Classe A: 1 TP e 1 FP
- Classe B: 10 TP e 90 FP
- Classe C: 1 TP e 1 FP
- Classe D: 1 TP e 1 FP

Si può vedere facilmente che, mentre $Pr_A = Pr_C = Pr_D = 0.5$, $Pr_B = 0.1$ Verrà quindi calcolata questa macro-media:

$$Pr = \frac{0.5 + 0.1 + 0.5 + 0.5}{4} = 0.4 \tag{7.2}$$

La micro-media invece sarà:

$$Pr = \frac{1 + 10 + 10 + 1}{2 + 100 + 2 + 2} = 0.123 \tag{7.3}$$

Questi sono valori abbastanza diversi tra loro. Intuitivamente, nella macro-media la precisione "buona" (0,5) delle classi A, C e D contribuisce a mantenere una precisione generale "decente" (0,4). Sebbene questo sia tecnicamente vero, è un po 'fuorviante, poiché un gran numero di esempi non sono classificati correttamente. Questi esempi corrispondono principalmente alla classe B, quindi contribuiscono solo 1/4 alla media nonostante costituiscano il 94,3% dei dati del test. La micro-media acquisirà adeguatamente questo squilibrio di classe e ridurrà la media di precisione complessiva a 0,123 (più in linea con la precisione della classe dominante B (0,1). Per ragioni computazionali, a volte può essere più conveniente calcolare le medie delle classi e poi macro-mediarle. Se è noto che lo squilibrio di classe è un problema, ci sono diversi

modi per aggirarlo. Uno è quello di riportare non solo la macro-media, ma anche la sua deviazione standard (per 3 o più classi). Un altro è calcolare una macro-media ponderata, in cui ogni contributo di classe alla media è ponderato dal numero relativo di esempi disponibili per essa.

Capitolo 8

Conclusioni e sviluppi futuri

È evidente la differenza di performance tra la rete LSTM-MI e il multi-layer-perceptron, con la prima che sfiora un'accuracy del 99% e la seconda che nel migliore dei casi, ovvero per la classificazione binaria, si avvicina al 70%.
Le reti neurali non sono completamente comprensibili dal punto di vista umano, tant'è che, ad oggi, sono considerabili una scatola nera, ovvero un modello matematico che riesce a trovare pattern nascosti nei dati "senza spiegazione". Esse infatti si basano su dei parametri, chiamati pesi, che non hanno un significato fisico, ma astratto. Per alcune applicazioni nelle quale è fondamentale fare un debug dell'intero processo, le reti neurali sono un'arma a doppio taglio, in quanto non permettono di carpire quali errori sono stati fatti durante il processo di predizione. Nel nostro esperimento possiamo fare però delle ipotesi.
È importante far notare come il MLP in precedenti lavori si sia comportato meglio di LSTM.MI, questo probabilmente a causa del fatto che nel test precedente i dati di test coprivano tutte le famiglie DGA con le quali erano stati addestrati, avendolo tra l'altro testato su famiglie nuove rispetto ai dati di training. I dati reali del GARR utilizzati sono invece fortemente sbilanciati e quindi MLP, non predisposto agli sbilanciamenti, sembra soffrirne di più a differenza della rete LSTM-MI che, nonostante peggiori le sue prestazioni, riesce a compensare.

I risultati, considerando il task particolarmente impegnativo a causa dell'aver utilizzato dati reali e temporalmente lontani dai dati di training e quindi con i seed dei DGA diversi, motivano futuri sviluppi del lavoro, ad esempio cercando di bilanciare meglio le classi nei dataset di test e di training e usando dati di training, reali o sintetici, più omogenei rispetto ai dati di test.

Appendice

```
"""Generatore di whitelist e blacklist di domini.

Lo script contiene una serie di strumenti per creare una lista di domini
a partire dal traffico della rete Garr. Attribuisce un tag ai domini presenti
nel Majestic Million e nel Bambenek DGA Feed.
Nel caso di domini dell'intersezione con il MM il tag àsar "white".
Nel caso di domini dell'intersezione con il BF il tag àsar il nome della famiglia
del DGA.

Lo script richiede che il modulo 'wget' sia correttamente installato nell'environment
di Python3 nella macchina in cui viene eseguito.
Per installarlo su sistemi UNIX da terminale:
    "pip3 install wget"

Lo script richiede che il modulo 'tqdm' sia correttamente installato nell'environment
di Python3 nella macchina in cui viene eseguito.
Per installarlo su sistemi UNIX da terminale:
    "pip3 install tqdm"

Lo script òpu essere eseguito sia come main che importato da altri moduli e contiene
le seguenti funzioni:

    * confrontaSet : confronta un set di log GARR in una cartella con i BF in un'altra cartella e
        con un file del MM. Scrive i risultati in un file output.
    * confrontaUnFile : confronta un file del log GARR con il file corrispondente del BF e il file
        del MM. Scrive i risultati in un file output.
    * aggiornaMm : aggiorna il file del MM.
    * inizializzaOut : crea o reinizializza il file di output.
```

```
29      * caricaMm : carica i domini del MM su di un dizionario con
30          chiave il dominio e valore un oggetto con ùpi informazioni
31      * caricaBf : carica i domini del BF su di un dizionario con
32          chiave il dominio e valore un oggetto con ùpi informazioni
33      * caricaOut : carica i domini àgi analizzati presenti nell'output
34          su una struttura dati composta da una lista di due dizionari:
35          uno per i benevoli, uno per i malevoli
36      * getLogFiles : fornita la cartella dei log, fornisce i log
37          presenti al suo interno
38      * familyFromDescr : fornita la descrizione di un dominio DGA
39          presente nel BF, ne estrapola la famiglia
40
41  @author: David Caprari per D.I.I.@UnivPM
42  @author: Luca Mannini per D.I.I.@UnivPM
43  """
44
45  import csv
46  import lzma
47  import os
48  import wget
49  from tqdm import tqdm
50
51  class GarrLog:
52      """Rappresenta gli elementi presenti nelle linee del Log del Garr
53
54      Contiene elementi inattivi che è possibile attivare
    decommentandoli.
55      In questo caso è necessario modificare il resto del codice, in
    particolare
56      la funzione confronta_log_mm().
57      ...
58
59      Attributes
60      ----------
61      date : str
62          NON ATTIVO
63          Data della richiesta
64      ext_ip_hashcode : str
65          NON ATTIVO
66          Hashcode dell'indirizzo che effettua la ricerca
67      internal_ip : str
68          NON ATTIVO
69          Indirizzo IP interno al DNS che risolve la richiesta
70      internet_protocol : str
71          NON ATTIVO
72          protocollo IP tra TCP o UDP
```

```
    in_out_connection : str
        NON ATTIVO
        tipo di connessione IN o OUT
    domain : str
        Dominio web che il DNS deve risolvere
    resolved_ip : str
        NON ATTIVO
        Dominio risolto o NXDOMAIN
    resolved_port : str
        NON ATTIVO
        Porta risolta
    """

    def __init__(self,
                 #date
                 #ext_ip_hashcode
                 #internal_ip
                 #internet_protocol
                 #in_out_connection
                 domain,
                 #resolved_ip
                 #resolved_port
                 ):
        """
        Parameters
        ----------
        date : str
            NON ATTIVO
            Data della richiesta
        ext_ip_hashcode : str
            NON ATTIVO
            Hashcode dell'indirizzo che effettua la ricerca
        internal_ip : str
            NON ATTIVO
            Indirizzo IP interno al DNS che risolve la richiesta
        internet_protocol : str
            NON ATTIVO
            protocollo IP tra TCP o UDP
        in_out_connection : str
            NON ATTIVO
            tipo di connessione IN o OUT
        domain : str
            Dominio web che il DNS deve risolvere
        resolved_ip : str
            NON ATTIVO
            Dominio risolto o NXDOMAIN
```

```
119         resolved_port : str
120             NON ATTIVO
121             Porta risolta
122         """
123 
124         #self.date = date
125         #self.ext_ip_hashcode = ext_ip_hashcode
126         #self.internal_ip = internal_ip
127         #self.internet_protocol = internet_protocol
128         #self.in_out_connection = in_out_connection
129         self.domain = domain
130         #self.resolved_ip = resolved_ip
131         #self.resolved_port = resolved_port
132 
133 class MajesticMillion:
134     """Rappresenta gli elementi presenti nelle linee del
    MajesticMillion
135 
136     Contiene elementi inattivi che è possibile attivare
    decommentandoli.
137     In questo caso è necessario modificare il resto del codice, in
    particolare
138     la funzione caricaMm().
139     ...
140 
141     Attributes
142     ---------
143     globalrank : str
144         NON ATTIVO
145         rank globale del dominio
146     tld_rank : str
147         NON ATTIVO
148         rank tld del dominio
149     domain : str
150         dominio web tra il primo milione di domini al mondo con ùpi
    sottoreti
151     tld : str
152         NON ATTIVO
153         tld
154     ref_sub_nets : str
155         NON ATTIVO
156         numero sotto reti a cui si riferisce il dominio
157     ref_ips : str
158         NON ATTIVO
159         numero di IP a cui si riferisce il dominio
160     idn_domain : str
```

```
161         NON ATTIVO
162         dominio idn
163     idn_tld : str
164         NON ATTIVO
165         tld idn
166     prev_global_rank : str
167         NON ATTIVO
168         rank globale del dominio nel periodo precedente
169     prev_tld_rank : str
170         NON ATTIVO
171         rank tld del dominio nel periodo precedente
172     prev_ref_sub_nets : str
173         NON ATTIVO
174         numero sotto reti a cui si riferisce il dominio nel periodo
    precedente
175     prev_ref_ips : str
176         NON ATTIVO
177         numero di IP a cui si riferisce il dominio nel periodo
    precedente
178     """
179
180     def __init__(self,
181                  #global_rank,
182                  #tld_rank,
183                  domain,
184                  family,
185                  #tld,
186                  #ref_sub_nets,
187                  #ref_ips,
188                  #idn_domain,
189                  #idn_tld,
190                  #prev_global_rank,
191                  #prev_tld_rank,
192                  #prev_ref_sub_nets,
193                  #prev_ref_ips
194                  ):
195         """
196         Parameters
197         ----------
198         globalrank : str
199             NON ATTIVO
200             rank globale del dominio
201         tld_rank : str
202             NON ATTIVO
203             rank tld del dominio
204         domain : str
```

```
205             dominio web tra il primo milione di domini al mondo con
    ùpi sottoreti
206         tld : str
207             NON ATTIVO
208             tld
209         ref_sub_nets : str
210             NON ATTIVO
211             numero sotto reti a cui si riferisce il dominio
212         ref_ips : str
213             NON ATTIVO
214             numero di IP a cui si riferisce il dominio
215         idn_domain : str
216             NON ATTIVO
217             dominio idn
218         idn_tld : str
219             NON ATTIVO
220             tld idn
221         prev_global_rank : str
222             NON ATTIVO
223             rank globale del dominio nel periodo precedente
224         prev_tld_rank : str
225             NON ATTIVO
226             rank tld del dominio nel periodo precedente
227         prev_ref_sub_nets : str
228             NON ATTIVO
229             numero sotto reti a cui si riferisce il dominio nel
    periodo precedente
230         prev_ref_ips : str
231             NON ATTIVO
232             numero di IP a cui si riferisce il dominio nel periodo
    precedente
233         """
234
235         #self.global_rank = global_rank
236         #self.tld_rank = tld_rank
237         self.domain = domain
238         self.family = family
239         #self.tld = tld
240         #self.ref_sub_nets = ref_sub_nets
241         #self.ref_ips = ref_ips
242         #self.idn_domain = idn_domain
243         #self.idn_tld = idn_tld
244         #self.prev_global_rank = prev_global_rank
245         #self.prev_tld_rank = prev_tld_rank
246         #self.prev_ref_sub_nets = prev_ref_sub_nets
247         #self.prev_ref_ips = prev_ref_ips
```

```
248
249 class BambenekFeed:
250     """Rappresenta gli elementi presenti nelle linee del
    BambenekDGAFeed
251
252     Contiene elementi inattivi che è possibile attivare
    decommentandoli.
253     In questo caso è necessario modificare il resto del codice, in
    particolare
254     la funzione caricaBf().
255     ...
256
257     Attributes
258     ---------
259     domain: str
260         Stringa con il dominio malevolo
261     family: str
262         Stringa con la famiglia del dominio malevolo
263     date : str
264         NON ATTIVO
265         Data di aggiunta al Bambenek Feed
266     link : str
267         NON ATTIVO
268         Link alla documentazione della famiglia
269     """
270
271     def __init__(self,
272                  domain,
273                  family
274                  #date,
275                  #link
276                  ):
277         """
278         Parameters
279         ---------
280         domain: str
281             Stringa con il dominio malevolo
282         family: str
283             Stringa con la famiglia del dominio malevolo
284         date : str
285             NON ATTIVO
286             Data di aggiunta al Bambenek Feed
287         link : str
288             NON ATTIVO
289             Link alla documentazione della famiglia
290         """
```

```
291
292         self.domain = domain
293         self.family = family
294         #self.date = date
295         #self.link = link
296
297 def confrontaSet(folder_bf, folder_log, filepath_mm, filepath_out):
298     """Corpo dello script: esegue intersezione tra Log, BF, MM.
299
300     Produce in output su filepath_out un file .csv con colonna dei
    domini e colonna della famiglia.
301
302     Parameters
303     ----------
304     folder_bf : str
305         Stringa con la folder con i Feed Bambenek
306     folder_log : str
307         Stringa con la folder con i log. (default= 'garr/')
308     filepath_out : str
309         Stringa contenente il filepath del .csv con i domini della
    whitelist
310     filepath_mm : str
311         Stringa contenente il filepath del .csv con i domini del MM
312
313     Returns
314     -------
315     analyzed_traffic : list
316         analyzed_traffic[0] : dict
317             Contiene i domini àgi nella whitelist
318         analyzed_traffic[1] : dict
319             Contiene i domini àgi nella blacklist
320     """
321     print("\nInizializzazione...\n")
322
323     mm_list = caricaMm(filepath_mm) #Carica MM
324     analyzed_traffic = caricaOut(filepath_out) #Carica àgi analizzati
325
326     log_files = getLogFiles(folder_log) #Prende lista dei file nella
    cartella
327     num_files = len(log_files) #Se ne ricava il numero
328
329     for count in tqdm(range(num_files), dynamic_ncols=True,
    bar_format="{l_bar}{bar}| {n_fmt}/{total_fmt} [ETA {remaining}]"):
     #tqdm per la barra di progresso
330     #for count in tqdm(range(num_files)): #per utilizzare la barra
    tqdm standard
```

```
331         date = log_files[count].lstrip("pdns_").rstrip(".log.xz")
    [:10]
332
333         filepath_bf = folder_bf + date + ".txt.xz"  #compila la
    stringa di posizione + nome del file Bambenek Feed
334                                                     #ad esempio:
    blacklist/2000_01_01.txt.xz
335         filepath_log = folder_log + log_files[count] #compila la
    stringa di posizione + nome del log
336
337         try:
338             bf_list = caricaBf(filepath_bf)
339         except (FileNotFoundError): #Se non è presente il relativo
    file bambenek...
340             print("\nNon è stato trovato il feed Bambenek relativo a"
    , log_files[count] ,
341                   "\nIl file non àverr analizzato.")
342             continue
343
344         analyzed_traffic = confrontaUnFile(filepath_log, filepath_out
    , mm_list, bf_list, analyzed_traffic) #Esegue le intersezioni sul
    file
345
346     print("Job concluso!")
347     return analyzed_traffic
348
349 def confrontaUnFile(filepath_log, filepath_out, mm_list, bf_list,
    analyzed_traffic):
350     """Confronta un'ora del log del Garr con i domini àgi in mm_list,
     bf_list e analyzed_traffic
351
352     Scorre il file compresso del log alla ricerca di domini in
    mm_list e bf_list.
353     Se trova dei domini sia nel log che in mm_list che non sono in
    analyzed_traffic
354     aggiorna analyzed_traffic e scrive i nuovi domini nel file al
    filepath_out.
355     Se trova dei domini sia nel log che in bf_list che non sono in
    analyzed_traffic
356     aggiorna analyzed_traffic e scrive i nuovi domini nel file al
    filepath_out.
357
358     Parameters
359     ----------
360     filepath_log : str
```

```
361         Filepath completo log di un'ora del garr (formato: ../
    pdns_yyyy_mm_dd_hh.log.xz)
362     filepath_out : str
363         Stringa contenente il filepath del .csv con i domini della
    whitelist
364     mm_list : dict
365         Dizionario con chiave i domini contenuti nel Majestic Million
366     bf_list : dict
367         Dizionario con chiave i domini contenuti nel Bambenek Feed e
    value un obj
368         che contiene info quali la famiglia del DGA
369     analyzed_traffic : dict
370         Dizionario con chiave i domini àgi contenuti nella whitelist
371 
372     Returns
373     -------
374     analyzed_traffic : list
375         analyzed_traffic[0] : dict
376             Contiene i domini àgi nella whitelist
377         analyzed_traffic[1] : dict
378             Contiene i domini àgi nella blacklist
379     """
380 
381     with lzma.open(filepath_log, mode='rt') as garr_feed:
382 
383         line = garr_feed.readline()
384         gdomain = GarrLog(domain=line.split(';')[5].rstrip('.'),
385                           #resolved_ip=line.split(';')[7] #Esempio
    per aggiungere altri parametri
386                           ) #Carica la prima linea del log
387         output_file = open(filepath_out, mode='a')
388         output = csv.writer(output_file)
389 
390         while line:
391 
392             ### Confronto ###
393             gdomain.domain = line.split(';')[5].rstrip('.') #Carica
    linea del log
394             domain = gdomain.domain
395 
396             if domain in mm_list: #Se il dominio sta nella lista del
    MM
397                 if domain not in analyzed_traffic[0]: #Se il dominio
    non è àgi stato analizzato
398                     analyzed_traffic[0].update({domain :
    MajesticMillion(domain = domain,
```

```
399
            family = "white")}) #Appende i nuovi dati al dizionario in
    RAM
400                     output.writerow([domain , "white"]) #Appende i
    nuovi dati al file di testo"""
401
402             else: #Se non sta in MM potrebbe essere malevolo
403
404                 #Questo else accelera di molto l'esecuzione dato che
    le chiamate ai domini del MM
405                 #sono infinitamente di ùpi rispetto a quelle ai
    domini malevoli.
406                 #Inoltre, èpoich non si ha intersezione tra MM e BF,
    funziona.
407
408                 if domain in bf_list: #Se il dominio sta nella lista
    del BF
409                     if domain not in analyzed_traffic[1]: #Se il
    dominio non è àgi stato analizzatogarr
410                         analyzed_traffic[1].update({domain :
    BambenekFeed(domain = domain,
411
      family = bf_list[domain].family)}) #Appende i nuovi dati al
    dizionario in RAM
412                         output.writerow([domain , bf_list[domain].
    family]) #Appende i nuovi dati al file di testo
413
414             line = garr_feed.readline()
415
416     return  analyzed_traffic
417
418 def aggiornaMm(filepath_mm):
419     """Aggiorna il file del Majestic Million (MM) dopo aver rimosso
    quello obsoleto.
420
421     Parameters
422     ----------
423     filepath_mm : str
424         Stringa contenente il filepath del vecchio e del nuovo MM
425     """
426     try:
427         os.remove(filepath_mm) #Cancella l'obsoleto
428     except:
429         pass
430     url = "http://downloads.majestic.com/majestic_million.csv"
431     wget.download(url, filepath_mm) #Scarica il nuovo
```

```
432     print("\nMajestic Million aggiornato!\n")
433 
434 def inizializzaOut(filepath_out):
435     """Inizializza il file di output (default: output.csv).
436 
437     Cancella il precedente file e appone in cima le intestazioni
    delle colonne:
438     Domain, Family
439 
440     Parameters
441     ----------
442     filepath_out : str
443         Stringa contenente il filepath del .csv con i domini della
    whitelist
444     """
445 
446     output_file = open(filepath_out, mode='w')
447     output = csv.writer(output_file)
448     output.writerow(["Domain" , "Family"]) #Appone le intestazioni
    delle colonne
449     output_file.close()
450 
451 def caricaMm(filepath_mm):
452     """Carica su di un dizionario i domini del MM
453 
454     (Decommentando e modificando parti del codice è possibile
    portarsi dietro
455     ùpi informazioni riguardo il dominio del MM preso in esame)
456 
457     Parameters
458     ---------
459     filepath_mm : str
460         Stringa contenente il filepath del .csv scaricato dal
    Majestic Million
461 
462     Returns
463     -------
464     mm_list : dict
465         Dizionario con:
466             key : str
467                 dominio del MM
468             value : obj
469                 oggetto contenente informazioni sul dominio
470     """
471 
472     with open(filepath_mm) as csvfile:
```

```
473         csvfile.readline() #Skip della prima riga
474         line = csvfile.readline()
475         mm_list = {line.split(',')[2] : MajesticMillion(domain=line.
    split(',')[2],
476                                                         family="
    white"
477                                                         #
    global_rank=line.split(',')[0] #Esempio per altri parametri
478                                                         )}
479         count = 0
480         while line and count <= 1000000: #Fino al primo milione del
    MM
481             mm_list.update({line.split(',')[2] : MajesticMillion(
    domain=line.split(',')[2],
482
    family="white"
483                                                                 #
    global_rank=line.split(',')[0] #Esempio per altri parametri
484                                                                 )
    })
485             line = csvfile.readline()
486             count += 1
487     return mm_list
488
489 def caricaBf(filepath_bf):
490     """Carica su di un dizionario i domini del BF
491
492     (Decommentando e modificando parti del codice è possibile
    portarsi dietro
493     ùpi informazioni riguardo il dominio del BF preso in esame)
494
495     Parameters
496     ---------
497     filepath_mm : str
498         Stringa contenente il filepath del .csv scaricato dal
    Majestic Million
499
500     Returns
501     -------
502     bf_list : dict
503         Dizionario con:
504             key : str
505                 dominio del bf
506             value : obj
507                 oggetto contenente informazioni sul dominio
508     """
```

```
509
510     with lzma.open(filepath_bf, mode='rt') as xzfile:
511         line = xzfile.readline()
512         for x in range(14):
513             line = xzfile.readline()
514         bf_list = {}
515         while line:
516             bf_list.update({line.split(',')[0] : BambenekFeed(domain=
    line.split(',')[0],
517                                                               family=
    familyFromDescr(descr = line.split(',')[1])
518                                                               )})
519             line = xzfile.readline()
520     return bf_list
521
522 def caricaOut(filepath_out):
523     """Carica su di una lista di due dizionari i domini àgi
    analizzati
524
525     Parameters
526     ---------
527     filepath_out : str
528         Stringa contenente il filepath del .csv con i domini della
    whitelist
529
530     Returns
531     -------
532     white : dict
533         Dizionario dei àgi benevoli con
534             key : str
535                 Dominio
536             value : str
537                 Family
538     black : dict
539         Dizionario dei àgi malevoli con
540             key : str
541                 Dominio
542             value : str
543                 Family
544     """
545
546     white = {} #Inizializza diz
547     black = {} #Inizializza diz
548     with open(filepath_out, 'r') as csvfile:
549         csvreader = csv.reader(csvfile, delimiter= ',')
550         not_first_line = False
```

```
551         for row in csvreader:
552             if not_first_line == False: #Skip prima riga
553                 not_first_line = True
554             else:
555                 if row[1] == "white":
556                     white.update({row[0] : row[1]}) #Key:dominio
    Value:family
557                 else:
558                     black.update({row[0] : row[1]}) #Key:dominio
    Value:family
559     return [white, black]
560 
561 def getLogFiles(folder_log):
562     """Fornita la cartella dei log restituisce la lista dei file di
    log al suo interno.
563 
564     Parameters
565     ----------
566     folder_log : str
567         Stringa con la folder dei log. (default= 'garr/')
568 
569     Returns
570     -------
571     log_files : list
572         Lista ordinata per data dei file di log.
573     """
574 
575     log_files = [f for f in os.listdir(folder_log) if os.path.isfile(
    os.path.join(folder_log, f))]
576     log_files.sort()
577     return log_files
578 
579 def familyFromDescr(descr):
580     """Estrae la famiglia dalla descrizione del dominio del Bambenek
    Feed
581 
582     Parameters
583     ----------
584     descr : str
585         Lo split del Bambenek Feed contenente la descrizione del
    dominio
586 
587     Returns
588     -------
589     family : str
590         La famiglia del dominio
```

```
591     """
592     protofamily = descr
593     if protofamily.split(" ")[3] == "Cryptolocker":
594         family = "cryptolocker"
595         return family
596     if protofamily.split(" ")[3] == "Post":
597         family = "ptgoz"
598         return family
599     if protofamily.split(" ")[0] == "P2P":
600         family = "p2pgoz"
601         return family
602     if protofamily.split(" ")[0] == "Volatile":
603         family = "volatile cedar / explosive"
604         return family
605 
606     family = descr.split(" ")[3]
607     return family
608 
609 if __name__ == '__main__':
610     """main
611 
612     Codice che viene richiamato se lo script viene eseguito come
    modulo principale.
613 
614     Assegna i valori di default ai vari filepath e folderpath.
615 
616     Inizializza l'output.
617 
618     Scarica il MM se necessario.
619 
620     Esegue confrontaSet, il modulo principale.
621     """
622 
623     filepath_mm = "whitelist/majestic_million.csv" #MODIFICARE SE SI
    VUOLE CAMBIARE PATH E NOME DEL MM
624     filepath_out = "output.csv" #MODIFICARE SE SI VUOLE CAMBIARE PATH
     E NOME DELL'OUTPUT
625     folder_bf = "blacklist/" #MODIFICARE SE SI VUOLE CAMBIARE PATH E
    NOME DELLA CARTELLA CON I FEED BAMBENEK
626 
627     if os.path.exists(filepath_out):
628         choice1 = input("Vuoi creare un nuovo file di output e
    cancellare il precedente? (s/n)\n")
629         if choice1 == 's':
630             inizializzaOut(filepath_out)
631         else:
```

```
632             print("I nuovi nomi di dominio verranno aggiunti ai
    precedenti.")
633     else:
634         inizializzaOut(filepath_out)
635 
636     if os.path.exists(filepath_mm): #Check della presenza del MM
637         choice2 = input("\nVuoi aggiornare il .csv del Majestic
    Million con l'ultimo disponibile? (s/n)\n")
638         if choice2 == 's':
639             aggiornaMm(filepath_mm)
640     else:
641         print("\nScarico il file del Majestic Million...")
642         aggiornaMm(filepath_mm) #Lo scarica se assente
643 
644     folder_log = input("\nInserire cartella dei log da analizzare: (
    default= 'garr/')\n")
645     if folder_log == "":
646         folder_log = "garr/"
647 
648     analyzed_traffic = confrontaSet(folder_bf, folder_log,
    filepath_mm, filepath_out) #Chiamata al modulo principale
```

Listato 8.1: Script finale per la creazione del dataset, whitelist + blacklist

Bibliografia

[1] Wikipedia, botnet.

[2] Wikipedia, domain generation algorithm.

[3] Navin Dhinnesh ADC and Sundareswaran N. Botnet life cycle and topologies. 119, 2018.

[4] Gianluca Albanese. Integrazione di moduli funzionali in un'architettura software per la sperimentazione di tecniche di rilevamento di malware. 2019.

[5] Pedro Marques da Luz. Botnet detection using passive dns. 2014.

[6] Riconoscimento di malware tramite analisi di features lessicali dei nomi di dominio nel traffico DNS. Riconoscimento di malware tramite analisi di features lessicali dei nomi di dominio nel traffico dns. 2019.

[7] Duc Tran, Hieu Mac, Van Tong, Hai Anh Tran, and Linh Giang Nguyen. A LSTM based framework for handling multiclass imbalance in DGA botnet detection. *Neurocomputing*, 275:2401–2413, January 2018.