UNIVERSITÀ
POLITECNICA
DELLE MARCHE

Facoltà di Ingegneria

Laurea in Ingegneria Informatica e dell'Automazione

Classificazione di Sequenze di Richieste DNS per il Riconoscimento di Macchine Infette

DNS Query Sequence Classification for Detection of Infected Machines.

Autore della tesi: **Simone Salvoni**

Relatore: **Prof. Luca Spalazzi**

A.A. 2020/2021

Indice

1 Introduzione

La sicurezza informatica è un tema ampio e costantemente in evoluzione. Con il passare degli anni ci si accorge sempre di più di quanto sia qualcosa che nessuno può ignorare, in quanto oramai pervade qualunque campo della nostra società, dalle aziende più piccole, alle multinazionali, agli stati. I danni che i cyber-criminali possono causare sono tanti e potenzialmente molto gravi, ed è per questo che bisogna armarsi al meglio delle proprie possibilità per cercare di contrastarli. E, parallelamente a questo, la ricerca scientifica e tecnica deve continuare ad avanzare per scoprire e poi sfruttare strumenti sempre più funzionali da poter mettere in campo contro questo potente nemico.
Gli strumenti a disposizione dai cyber-criminali sono moltissimi, e questa tesi ne prende in considerazione uno in particolare: le botnet basate su una classe di algoritmi che prende il nome di *Domain Generations Algorithms*, o *DGA*. Dopo aver diffuso malware che sfruttano tale algoritmo, i criminali possono sfruttare i servizi DNS per controllare le neo-formate botnet e compiere un gran numero di azioni malevole diverse.
Come si può facilmente intuire dalla imponente diffusione di botnet che sfruttano i DGA, questo è un argomento fortemente discusso e dibattuto dalla comunità scientifica, che nel tempo ha proposto numerosi metodi per contrastare questi malware, cercando continuamente di rimanere al passo con la costante evoluzione che inevitabilmente questi hanno.

1.1 Obiettivi e struttura della tesi

Questa tesi ha due scopi principali. Per prima cosa verrà analizzato il tema dei malware basati su DGA: verranno esaminati il loro funzionamento e le motivazioni che hanno portato al loro grande successo. Verranno inoltre presentate alcune metodologie che la comunità scientifica ha nel tempo presentato per contrastare questa forte tipologia di malware. In secondo luogo verrà presentato un classificatore con lo scopo di rilevare macchine infette a partire dall'analisi del traffico di rete fornito dal GARR, con la dovuta premessa che per problemi riscontrati durante il periodo di raccolta dei dati non è stato possibile effettivamente testare il classificatore proposto.

La tesi è strutturata nel seguente modo:

- Nel capitolo 2 verranno analizzati i malware basati sui DGA. Verrà presentato il servizio DNS e come può essere utilizzato per lo sfruttamento di una botnet. Verranno quindi presentate le evoluzioni che esse hanno avuto, e perché un numero sempre maggior di esse ha incominciato a sfruttare gli algoritmi di generazione di nomi di dominio, il cui funzionamento verrà esaminato.

- Nel capitolo 3 verranno presentate alcune tecniche e strumenti proposti dalla comunità scientifica negli scorsi anni per contrastare le botnet. Verranno analizzate per prima cosa alcune tecniche retroattive, sottolineando anche il motivo per cui risultano sempre meno efficaci, per poi passare allo studio di alcune tecniche in tempo reale, molto più adatte per combattere i DGA.

- Nel capitolo 4 verranno date delle basi teoriche sulle reti neurali, necessarie per comprendere il funzionamento del classificatore presentato successivamente. Verranno fornite le basi del loro funzionamento, mostrando anche le limitazioni delle reti feed-forward che hanno portati all'introduzione delle reti ricorsive, e quindi delle reti LSTM.

- Nel capitolo 5 verrà esposto il classificatore di macchine host infette. Ne verrà esaminata la struttura, i dati che necessita e i risultati che fornisce. Verranno quindi anche presentate le problematiche nell'ottenimento del dataset che hanno impedito il test della rete nei tempi predisposti.

- Infine, nel capitolo 6 verranno dati le considerazioni finali su ciò che viene presentato in questa tesi, ponendo particolare attenzione su ciò che potrà in futuro essere fatto per continuare e migliorare questo lavoro.

2 Malware basati su DGA

2.1 Nomi di dominio e DNS

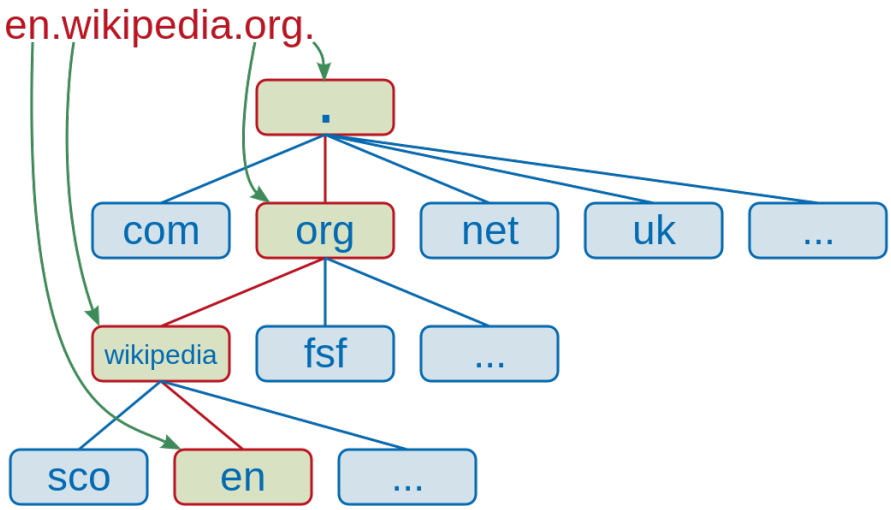

Figura 1: Struttura gerarchica dei nomi di dominio [1]

Tramite Internet dispositivi di ogni genere possono comunicare fra loro, scambiandosi informazioni da ogni parte del mondo. Per poter identificare i vari dispositivi che fanno parte di questa enorme rete viene associato a ciascun nodo, o *host*, un indirizzo numerico, che prende il nome di indirizzo IP, dove IP sta per *Internet Protocol*. Ciò permette una forma di riconoscimento biunivoca da parte di due entità che vogliono mettersi in contatto tramite Internet.
Questo indirizzo però non rappresenta qualcosa di facilmente utilizzabile da parte dell'utente medio. Difatti, se bisognasse specificare questo indirizzo ogni qualvolta che un utente desiderasse visitare un sito web, la navigazione in rete risulterebbe molto più complessa. Gli utenti sarebbero costretti a ricordarsi, o segnarsi, l'indirizzo IP di ogni sito web di loro interesse. Ciò ovviamente risulterebbe a dir poco scomodo, e se la situazione fosse stata questa Internet non si sarebbe mai potuto sviluppare e diffondere al livello attuale. Se è stato possibile è solamente grazie al *DNS*, ovvero al *Domain Name System*.
Il DNS è un servizio che permette ad un utente di riferirsi ad una risorsa web tramite uno specifico *nome di dominio*, che è mappato ad un certo indirizzo IP. Il nome di dominio è una stringa di testo come la seguente: "example.com".

Quello che è generalmente definito come nome di domino in realtà è una combinazione di un nome di dominio, inteso in senso stretto, e un *TLD*, o *Top Level Domain*, scritta da sinistra verso destra in ordine crescente di gerarchia [2]. I livelli gerarchici sono separati da un punto. I nomi di dominio sono inoltre generalmente preceduti da un *host name*, che nel caso dei siti web è "www", che indica l'appartenenza del sito al world wide web. Quindi per accedere alla pagina web in questione l'utente dovrà semplicemente richiedere l'accesso alla pagina "www.example.com", la quale è mappata ad un particolare indirizzo IP con cui l'utente si metterà in comunicazione, ma che non deve conoscere.
Questa mappatura tra nome e indirizzo numerico, che non è necessariamente biunivoca in quanto ad ogni indirizzo IP possono essere legati più nomi di dominio, rappresenta una facilitazione enorme per l'utente medio, dato che è molto più semplice ricordarsi o riconoscere un nome di dominio, che è generalmente una stringa di testo sensata e comprensibile, rispetto a una serie di cifre. È quindi chiaro quanto fondamentale sia il DNS per l'utilizzo di Internet.

2.2 Botnet

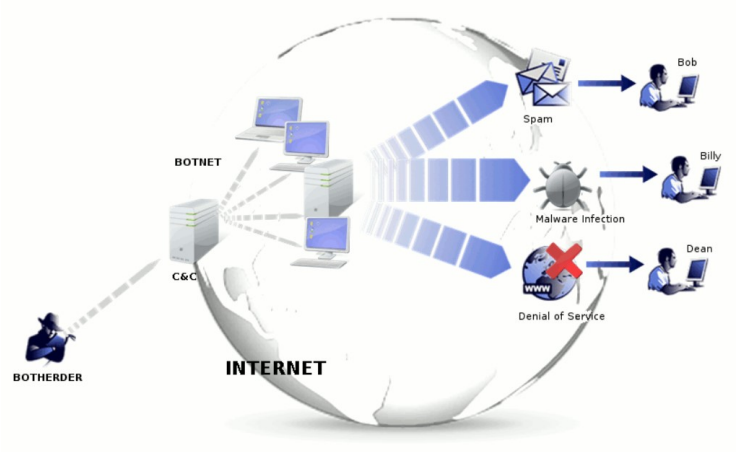

Figura 2: Una botnet, controllata da un server C&C, viene sfruttata per eseguire varie tipologie di crimini [3]

Data l'importanza e l'utilità del DNS era purtroppo inevitabile che diventasse anche uno strumento utilizzato da numerosi cyber-criminali per eseguire i loro crimini. Ne sono un esempio lampante le *botnet*.

Una botnet è una collezione di macchine infette da un malware, che prendono il nome di bot [4]. Come spiegano Bilge et al. [5], una macchina trasformata in bot è uno strumento molto utile, in quanto il bot non si accorgere di essere tale, e quindi può essere utilizzato dai malintenzionati senza problemi. Quando un dispositivo viene infettato e reso un bot esso si mette in attesa dell'arrivo di comandi da remoto, e una volta ricevuti si comporta di conseguenza, eseguendo gli ordini. I comandi provengono dal cosiddetto "*bot master*", generalmente un server (o più server) detto *Command & Control*, o *C&C*. Tramite le botnet vengono eseguiti numerosi crimini, dallo spam di email ad attacchi DDOS (Distributed Denial of Service attacks).

La connessione a questo bot master chiaramente avviene tramite la rete Internet. Tuttavia, connettersi al server specificando direttamente il suo indirizzo IP, da inserire necessariamente all'interno del malware, renderebbe la botnet fortemente vulnerabile: basterebbe scoprire questo indirizzo IP e blacklistarlo per rendere la botnet inutilizzabile. È quindi necessario l'utilizzo di un nome di dominio legato all'IP del server C&C. In questo modo i bot possono eseguire delle richieste ai servizi DNS sfruttando tale nome di dominio, venendo così reindirizzati al bot master che distribuisce loro le istruzioni. E tutto questo, plausibilmente, senza che gli utenti possessori dei bot se ne accorgano. Sfruttando il DNS i cyber-criminali possono senza problemi cambiare IP nel caso questo venga scoperto, mantenendo lo stesso nome di dominio di fronte al cambiamento di server. Tuttavia, in questo caso il problema del single point of failure è stato spostato dall'IP al nome di dominio. Rimane in ogni caso più sicuro, ma se le autorità competenti scoprono questo nome di domino, possono inserirlo in una blacklist, rendendo inutilizzabile il malware fino ad una sua modifica strutturale.

Per superare questa limitazione sono nate botnet che sfruttano tecniche più complesse. Una famiglia di queste è quella delle *Botnet P2P*, ovvero *peer to peer*. Con questa tipologia di botnet non è presente un unico punto centralizzato con cui tutti i bot comunicano, ma questi possono comunicare fra di loro. Possono comportarsi sia come Client, sia come Server [6]. Le comunicazioni si distribuiscono quindi fra i vari nodi della rete. Se un bot viene rilevato e cade, la botnet rimane operativa, e risalire al bot master in questo caso è più complesso.

Ciò detto, anche le botnet P2P hanno le loro limitazioni, principalmente per quanto riguarda la loro difficoltà a essere implementate e mantenute [7]. Dal desiderio di unificare la semplicità delle botnet classiche con la sicurezza che offrono le botnet P2P nascono le botnet basate sui DGA.

2.3 DGA

I *DGA*, o *Domain Generation Algorithm*, rappresentano una famiglia di algoritmi il cui scopo è quello di generare casualmente, a partire da un certo seed, un grande numero di possibili nomi di dominio. La tipologia di seed utilizzata dipende dall'implementazione scelta durante la costruzione del malware. Per esempio, Conficker sfrutta la data corrente in UTC, mentre Torpig sfrutta qualcosa di più peculiare, ovvero i topic in trend su Twitter [8]. Anche la tipologia di nome di dominio costruito varia da malware a malware: alcuni costruiscono stringhe di testo casuali e ci inseriscono un TLD alla fine, mentre altri, più sofisticati, come Kraken, combinano parole e suffissi effettivamente esistenti nella lingua inglese per costruire nomi leggibili [8].
Gli step che deve prendere il controllore della botnet sono dunque semplici: dopo aver fatto diffondere il malware basta registrare solamente un nome di dominio tra tutti quelli generabili, o al più un numero limitato di essi, e il gioco è fatto. I bot cominceranno a eseguire giornalmente un grande numero di richieste DNS basandosi su questi nomi di domino automaticamente generati. Chiaramente gran parte di questi riceveranno come risposta NXDOMAIN, che viene restituita quando un nome di dominio non è associato a nessun indirizzo IP. Ma quando verrà eseguita la richiesta per il corretto nome di domino il bot riuscirà a collegarsi al server C&C. Questa tecnica che prevede la generazione di una grandissimo numero di nomi di dominio e un successivo insieme di richieste DNS, la maggior parte delle quali non risalirà a nulla, è detta *domain fluxing*.
Un'altra caratteristica fondamentale di questa tipologia di botnet è la validità limitata dei nomi di domino. Difatti, sia la lista generata dai DGA, sia quindi il nome di dominio effettivamente registrato dal cyber-criminale, hanno una durata molto limitata. Se cambia il seed, cambia il pool dei nomi di domino costruiti e da cui scegliere.
Tutte queste caratteristiche rendono i DGA degli strumenti estremamente peri-

colosi nelle mani di malintenzionati, in quanto permettono di avere delle botnet estremamente agili e resilienti. Per le autorità competenti non basta più localizzare un nome di dominio o un indirizzo IP malevolo. L'IP è facilmente modificabile, esattamente come per le botnet più classiche, ma questo è specialmente vero per i nomi di domino, dato che, anche restringendo il campo su quei pochi nomi prodotti che effettivamente si mappano all'IP del server C&C, essi hanno una vita molto breve, e non appena c'è un cambio di seed il malware è di nuovo in attività senza problemi.

3 Metodi di identificazione proposti

Con la sempre maggiore diffusione delle botnet è diventato sempre più imperativo sviluppare delle metodologie per poter riconoscere e quindi contrastare questa minaccia.

3.1 Tecniche di riconoscimento retroattive

Inizialmente, e per parecchio tempo, gran parte delle ricerche sono state svolte per costruire metodologie di riconoscimento e classificazione retroattive. Le soluzioni classiche difatti si basano principalmente sul reverse engineering del malware e sul blacklisting [4].
Il reverse engineering può risultare particolarmente efficace, dato che la conoscenza della struttura e del funzionamento del malware permettete di conoscere in anticipo tutti i nomi di dominio che verranno in futuro generati dal DGA in questione, permettendo così di bloccare ogni richiesta DNS contenete quei nomi. Chiaramente però questo approccio è fortemente limitato da un fattore: la necessità di avere fra le mani un campione di tale malware. Questo non è quasi mai per nulla semplice, per cui affidarsi esclusivamente al reverse engineering non è assolutamente una strada percorribile. In più, se per caso il cyber-criminale modifica il malware e lo redistribuisce può cominciare a costruire una nuova botnet, rendendo così le precedenti conoscenze sul malware inutilizzabili.
Per questi motivi il blacklisting basato sull'analisi di traffico di rete è stato per molto tempo la metodologia più comunemente sfruttata per affrontare una minaccia come quella delle botnet. Come è facilmente intuibile, con blacklisting si intende l'inserimento di un particolare nome di dominio o indirizzo IP all'interno di una blacklist, in modo tale che sia possibile riconoscere una richiesta di connessione a tale nome o IP, e quindi bloccarla immediatamente. Questo, di fatto, equivale a mettere offline la botnet, dato che i bot si ritrovano incapaci di comunicare con il proprio bot master.
Riconoscere nomi di dominio all'interno dell'oceano di richieste DNS giornalmente eseguite in tutto il mondo però è tutt'altro che un'impresa semplice. Difatti questo campo è stato soggetto a molteplici studi e ricerche per molto tempo, e così sono state proposte numerose metodologie di riconoscimento. Di

seguito ne sono presentate alcune.

Perdisci et al [9] presentarono *FluxBuster*, un sistema passivo di riconoscimento di fenomeni malevoli di domain fluxing. Per fare ciò, per prima cosa vengono raggruppate le richieste DNS che richiedono lo stesso nome di dominio in un certo intervallo di tempo. Da questi gruppi è possibile recuperare informazioni interessanti su queste richieste, tra cui il numero di esse in un intervallo di tempo scelto o i vari indirizzi IP risolti che si legano ai nomi di dominio. I ricercatori consigliano un tempo di almeno qualche ora, in modo da poter raccogliere informazioni sufficienti. Successivamente questi gruppi vengono filtrati, per escludere tutte quelle richieste che sono molto probabilmente non collegate al fenomeno del fluxing. A questo punto si utilizza un albero decisionale, sfruttando le informazioni raccolte dai gruppi creati, per decidere se i vari fenomeni di fluxing rilevati sono o meno malevoli.

Bilge et al. [5] proposero *EXPOSURE*, un algoritmo che sfrutta varie proprietà per costruire un classificatore di domini DGA. Tra queste proprietà abbiamo, per esempio, delle caratteristiche temporali come il volume di richieste per un particolare dominio. I ricercatori infatti si sono accorti che un segnale di presenza di domain fluxing è un'improvvisa crescita nel numero di richieste seguita da un'improvvisa decrescita. Tra le altre proprietà controllate ce ne sono anche alcune delle risposte DNS, tra cui la locazione degli indirizzi IP risolti e il numero di indirizzi IP legati ad un particolare nome di dominio. Vengono anche studiate le caratteristiche sintattiche dei nomi di dominio.

Queste e molte altre tecniche simili sviluppate nel tempo hanno tutte però un grosso problema che va a scontrarsi con la sempre maggiore efficienza delle botnet a DGA: la classificazione retroattiva non è adatta ad applicazioni immediate di contrasto. Per essere efficaci richiedono numerose ore per raggruppare un dataset sufficiemente ricco e studiarlo [4]. E con la rapidità dei DGA nel creare una nuova lista di nomi di domino con cui eseguire il domain fluxing, questa lentezza è inaccettabile. E questo, di ritorno, è la principale causa per cui il blacklisting, per lo meno da solo, non è efficace per contrastare le botnet moderne. Per adattarsi alle evoluzioni di questi malware è stato necessario sviluppare quindi delle modalità di riconoscimento immediate, che permettano di scovare i nomi di domino DGA proprio mentre vengono eseguite le richieste DNS.

3.2 Tecniche di riconoscimento in tempo reale

Come precedentemente detto, per poter contrastare le caratteristiche dei DGA è necessaria velocità nel riconoscimento dei nomi di dominio malevoli, in modo da poter prontamente bloccare richieste DNS che li contengono. Questo non è eseguibile con i tempi richiesti dagli algoritmi retroattivi precedentemente discussi: sono necessari dei classificatori che sul momento siano capaci di riconoscere nomi pericolosi. Un modo particolarmente utile per costruire un classificatore di questo tipo è sfruttare il machine learning: si vogliono utilizzare strumenti che, preso in ingresso un nome di dominio, siano in grado di dirci se si tratta di un nome di dominio legit o se è frutto di un DGA. Anche per questa tipologia di classificazione sono stati negli anni proposte varie soluzioni. Di seguito ne sono presentate alcune.
Schiavoni et al [10] proposero un sistema di rilevamento basato su alcune caratteristiche linguistiche dei nomi di domino presi in input: nello specifico si parla di:

- Rapporto di caratteri significativi. Con questo si intende il rapporto di caratteri all'interno del nome di dominio che formano una parola di senso compiuto in lingua inglese. Nomi costruiti da DGA tendono ad avere tale valore piuttosto basso. Per esempio, in "*facebook*", supponendo che il numero minimo di caratteri che sono considerabili come una parola sia 3, troviamo sia "face" sia "book" come parole sensate. 8 caratteri su 8 risultano quindi facenti parte di parole di senso compiuto, il che ci dà un valore di questo rapporto pari ad 1, il massimo. Al contrario in una stringa come *"pub03str"* abbiamo solamente la parola "pub", dunque 3 caratteri su 8.

- Punteggio di normalità degli n-grammi. Con questo parametro si cerca di classificare la pronunciabilità dei nomi di dominio, partendo dal presupposto che i nomi costruiti tramite DGA siano generalmente non pronunciabili (in realtà questo varia da malware a malware). Si vuole quantificare quanto esso aderisce alla fonetica di una certa lingua (in questo caso l'inglese). Con n-grammi intendiamo sottostringhe del nome di dominio composte da n caratteri. L'algoritmo estrae questi n-grammi e controlla la loro occor-

renza in un dizionario. Più alto è il punteggio ottenuto più "legit" risulta essere il nome.

Pur essendo un metodo di riconoscimento real-time, è limitato dal basarsi così tanto sulle caratteristiche della lingua inglese. Inoltre è solo una parte di un algoritmo più complesso: difatti, una volta che un domino viene classificato come DGA viene passato ad uno step successivo per eseguire dei raggruppamenti di questi domini.

Antonakasis et al. [7], similmente al caso precedente, introdussero un sistema misto che permette di riconoscere la presenza di una botnet, composto da una prima parte retroattiva e una seconda in real time. Chiamarono questo sistema *Pleiades*. Per prima cosa l'algoritmo ricerca grossi cluster di richieste DNS che restituiscono NXDOMAIN, che hanno feature sintattiche simili e che provengono da macchine potenzialmente compromesse. Questo è fatto poiché per via della tecnica del domain fluxing, come si è precedentemente spiegato, vengono eseguite giornalmente un altissimo numero di richieste DNS su nomi di dominio generati pseudo-randomicamente, ma solamente pochi di questi sono nomi effettivamente mappati a degli IP: tutti gli altri restituiranno NXDOMAIN. Dopodiché, per ogni host considerato infetto da un bot si tenta di risalire al server C&C che li controlla. E questa è la sezione che sfrutta un modello di machine learning detto *Hidden Markov Model*, o *HMM*, che prende in ingresso nomi di dominio sospetti (da nuove chiamate DNS) che *non* hanno restituito NXDOMAIN e restituisce un punteggio, che rappresenta quanto un dato nome di dominio è probabilmente generato da un DGA.

Woodbridge et al. [11] sono stati i primi ad introdurre un classificatore puramente in tempo reale, basato su reti neurali LSTM. Questo classificatore si suddivide in due parti, un classificatore binario e uno multi-classe. Di conseguenza il processo di classificazione è composto da due stadi:

- Si passa un nome di dominio, precedentemente trasformato in una sequenza di interi, al classificatore binario. Esso restituirà quindi una singola predizione, ovvero se quel nome di dominio è considerato o meno un nome generato da un DGA.

- A questo punto se il nome è considerato legit ci si ferma. Se invece il nome

di dominio viene etichettato come malevolo, viene passato al secondo classificatore. Questo classificatore cerca di capire quale famiglia di malware ha più plausibilmente generato quel nome di dominio malevolo. L'input di questa seconda rete è lo stesso del precedente, ma in questo caso si hanno tanti output quante sono le classi di malware che stiamo considerando. Tra tutti questi output viene selezionato quello di valore maggiore, che indica la predizione più sicura.

La rete è addestrata sfruttando un dataset composto da un grande numero di richieste DNS che sono state etichettate tramite due fonti principali: la classifica dei nomi di dominio più richiesti di Alexa [12], che contiene 1 milione di nomi, e i feed giornalieri di nomi di dominio generati da DGA ottenuti da Bambenek [13]. I ricercatori hanno confrontato la performance di questo classificatore con quella del modello HMM presentato da Antonakasis et al., e hanno riscontrato che questo classificatore si comporta meglio dell'HMM, specialmente per quanto riguarda la classificazione multi-classe, per cui non sono stati nemmeno riportati i risultati dell'HMM perché non sufficientemente soddisfacenti (anche perché, come riportano loro stessi, l'HMM era stato costruito per lavorare con un numero molto limitato di famiglie di malware, mentre in questo caso ne sono state utilizzate più di 30).

Infine, Tran et al. [4] hanno proposto una modifica al classificatore precedente, per migliorarne le prestazioni. Infatti, come sottolineano i ricercatori, il classificatore costruito da Woodbridge et al. ha un difetto importante: come ogni rete LSTM è fortemente sensibile allo sbilanciamento all'interno del dataset utilizzato per il training. Infatti, in un dataset raccolto da reali richieste DNS per forza di cose il numero di richieste legit sarà sensibilmente superiore a quelle DGA. È comune trovarsi di fronte a un rapporto nell'ordine di 1:1000. E anche considerando esclusivamente le famiglie di malware, è inevitabile che si abbiano molti più esempi di una certa famiglia rispetto ad un'altra, magari perché meno comune o più recente. Ciò ha conseguenze poco desiderabili: se infatti la rete addestrata con tali dataset risulta particolarmente capace di riconoscere nomi di dominio legit da quelli malevoli, con tassi di riconoscimento di olte il 90% e una presenza di falsi positivi nell'ordine di 10^{-4}, è nella seconda classificazione che si presentano dei problemi. Infatti, se il numero di elementi appartenenti

ad una classe di malware è particolarmente scarso all'interno del dataset, tale classe risulta di difficile riconoscimento da parte della rete. Ciò che i ricercatori hanno proposto quindi è di introdurre nel meccanismo di apprendimento dei parametri di *costo* associati a ciascun nome di dominio generato da DGA nel dataset, in modo che sia tanto maggiore tanto più rara è la sua famiglia. Questo fa in modo che questi elementi del dataset abbiano un peso maggiore durante il training. Grazie inoltre al fatto che la classificazione ha due step, si evita anche un problema che si avrebbe con esclusivamente il classificatore multi-classe *cost-sensitive*. Infatti, dando più peso alle classi meno prevalenti si rischia di ridurre l'accuratezza del riconoscimento di quelle più prevalenti, e nello specifico dei nomi di dominio benigni. Etichettare un nome di dominio del genere come malevolo rischierebbe di bloccare richieste DNS completamente innocue, con tutti gli inconvenienti che ne conseguono. Ma grazie alla prima classificazione, quella binaria, il problema è fortemente ridotto. Infatti in questo step tutti i nomi di domino DGA sono categorizzati in un'unica classe (per l'appunto, DGA), il che riduce di molto lo sbilanciamento del dataset. Ciò permette in questo stadio di mantenere un alto tasso di riconoscimento di nomi di dominio legit, per poi invece introdurre tutti i benefici dell'approccio cost-sensitive nella seconda classificazione, nella quale entrano in gioco solo i nomi di dominio che sono stati classificati come provenienti da un DGA.

La rete presentata in questa tesi, che verrà analizzata successivamente, sfrutta proprio questo classificatore, e nello specifico la parte binaria, per il suo primo livello di classificazione.

4 Generalità sulle reti neurali

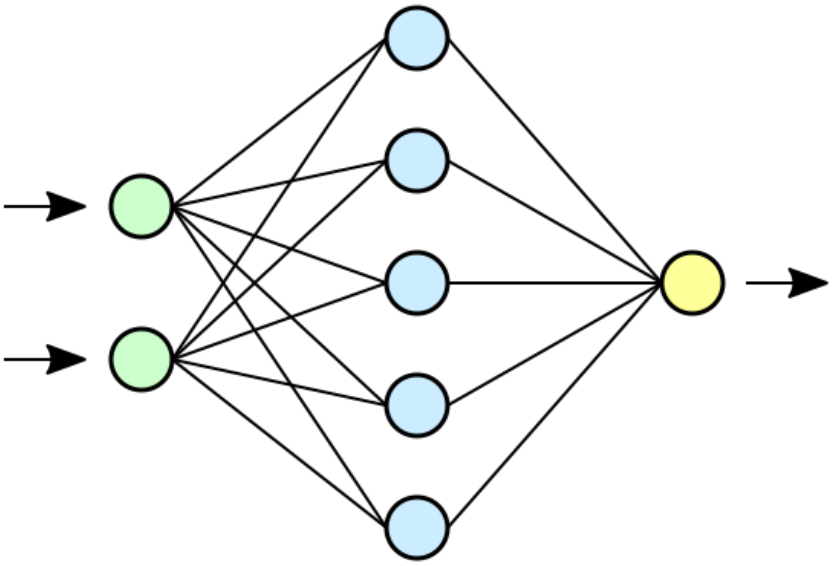

Figura 3: Struttura di un semplice rete neurale [14]

Prima di presentare il classificatore fulcro di questa tesi conviene presentare alcune basi teoriche sulle reti neurali, in modo da renderne poi la comprensione più semplice.

4.1 Basi sulle reti neurali e reti feed-forward

Una rete neurale è una struttura matematica composta da nodi, o neuroni, collegati fra loro. I nodi sono raggruppati in layer. Le reti neurali più semplici hanno solo due layer: il primo è quello di input, che contiene i dati in ingresso, mentre il secondo è quello il cui output è il vero e proprio output della rete. Reti più complesse (quasi tutte nelle applicazioni reali) hanno anche altri layer intermedi, detti layer nascosti, il cui output è passato al layer successivo.
Ogni nodo (esclusi quelli del primo layer, che semplicemente contengono i dati iniziali) prende in input un valore, lo passa in una funzione non lineare, detta funzione di attivazione, e restituisce il risultato come output. Questo è essenzialmente il funzionamento delle reti neurali più basilari, le reti *feed-forward*. Nelle reti feed-forward i collegamenti vanno tutti in un'unica direzione: dall'input

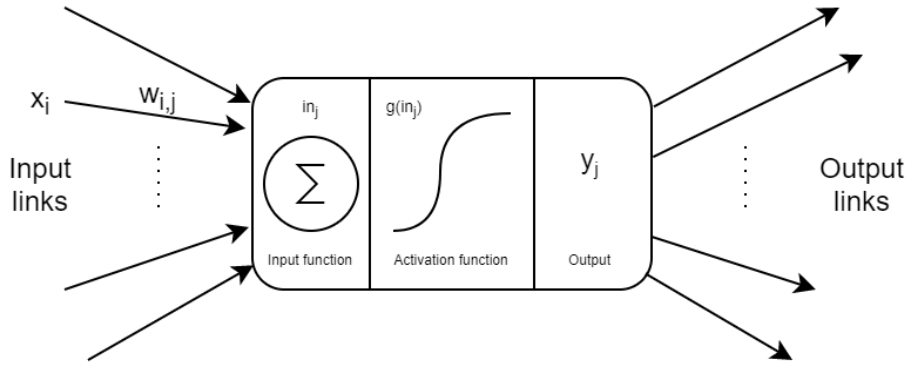

Figura 4: Modello matematico di un neurone, basato sull'illustrazione presente nel libro "Artificial intelligence: a modern approach", di Stuart Russel e Peter Norvig [15]

all'output. Di seguito verrà analizzato il funzionamento delle reti feed-forward. Ogni input di un nodo è una combinazione lineare di alcuni degli output del layer precedente. I coefficienti delle varie combinazioni lineari sono detti i pesi della rete. Se indichiamo quindi con in_j l'input del nodo j, con $w_{i,j}$ il peso relativo al collegamento fra il nodo i del layer precedente e il nodo j, e con x_i l'ouput del nodo i, allora possiamo scrivere:

$$in_j = \sum_{i=1}^{n} w_{i,j} x_i$$

Se a questo punto indichiamo la funzione di attivazione con $g(in_i)$, allora l'output y_j del nodo è:

$$y_j = g(\sum_{i=1}^{n} w_{i,j} x_i)$$

Spesso si aggiunge anche un termine di bias alla sommatoria, arrivando quindi a poter generalmente scrivere l'ouput di un nodo come:

$$y_j = g(w_{0,j} + \sum_{i=1}^{n} w_{i,j} x_i)$$

Possiamo anche rappresentarlo in forma matriciale, indicando con W il vettore dei pesi e con X il vettore dei valori in input:

$$y_j = g(w_{0,j} + W^T X)$$

Per quanto riguarda la funzione di attivazione, essa può essere una funzione di step, della forma $g(x) = \begin{cases} 0 & \text{se } |x| < T \\ 1 & \text{se } |x| >= T \end{cases}$, oppure, più comunemente, una cosiddetta *funzione logistica*, spesso una *arcotangente* o un sigmoide ($f(x) = \frac{1}{1+e^{-x}}$) [15].

Allenare una rete neurale significa trovare il valore dei pesi (e dei bias, spesso inclusi nell'insieme dei pesi) che minimizzano la *funzione di costo*, che generalmente è una funzione della media della differenza fra gli output della rete e i valori che noi vorremmo che avesse. Una funzione di costo usata comunemente per reti con output reali e continui (si parla in questo caso di problemi di regressione) è:

$$J(W) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f_i(X, W))^2$$

dove y_i è l'i-esimo valore che vogliamo che la rete produca, mentre $f_i(X; W)$ è l'i-esimo output prodotto dalla rete. Nel caso invece di reti il cui output è un valore tra zero e uno, che corrisponde generalmente ad una probabilità (in questo caso si parla di problema di classificazione), una funzione di costo comunemente utilizzata è:

$$J(W) = -\frac{1}{n}\sum_{i=1}^{n}(y_i \log f_i(X, W) + (1 - y_i)\log(1 - f_i(X, W)))$$

Il processo di allenamento della rete si basa sul dare in pasto alla rete il proprio data set (partendo da dei valori arbitrari dei pesi), comparare l'output prodotto con i valori reali, modificare i pesi in modo da ridurre $J(W)$ e quindi ripetere il processo fintanto che, idealmente, non si è raggiunto un minimo della funzione di costo. Un tipico algoritmo utilizzato per questo è la *discesa del gradiente*, o in inglese *gradient descent*, secondo cui ad ogni iterazione si calcola il gradiente di $J(W)$ rispetto ai pesi e lo si sfrutta per modificarli:

$$W \leftarrow W - \eta\frac{\partial J(W)}{\partial W}$$

η è un parametro chiamato tasso di apprendimento, che può essere costante o variabile durante il processo. Generalmente viene sfruttato un tasso di apprendimento di valore decrescente nel tempo per evitare problemi di oscillazione attorno ai minimi locali di $J(W)$.

Calcolare le derivate della funzione di costo rispetto ai pesi non è necessariamente immediato, poiché spesso non sono espliciti i legami fra di essi. Per poter calcolare il gradiente viene quindi sfruttata una tecnica che viene definita *back propagation*, o *propagazione all'indietro*. Di fatto questa tecnica non è altro che un'applicazione della regola della catena per le derivazioni. Consideriamo la seguente semplicissima rete:

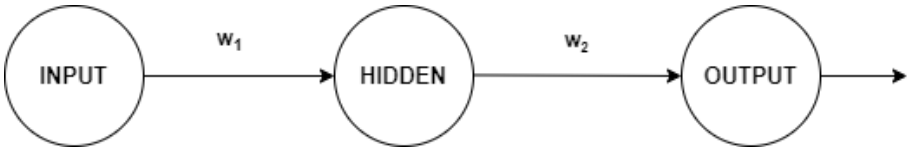

Figura 5: Questa immagine e l'esempio seguente sono basati su quello mostrato in [16]

Sono presenti solo un nodo di input, uno nel layer nascosto e uno di ouput. Di conseguenza i pesi sono due: w_1 e w_2. Se indichiamo la funzione di costo con $J(w_1, w_2)$, allora il gradiente è: $\nabla J = (\frac{\partial J}{\partial w_1}, \frac{\partial J}{\partial w_2})$. Per eseguire il calcolo conviene indicare esplicitamente la relazione fra la funzione di costo e l'output y, che a sua volta dipende dai pesi. Si può fare applicando la sopracitata regola della catena. Per $\frac{\partial J}{\partial w_2}$ è immediato, poiché è semplice sapere il legame fra l'output e l'ultimo insieme di pesi (qua solamente w_2):

$$\frac{\partial J}{\partial w_2} = \frac{\partial J}{\partial y} \cdot \frac{\partial y}{\partial w_2}$$

Per i pesi interni è meno ovvio, poiché i legami fra l'output e i pesi interni non sono espliciti, ma sono comunque presenti. Ciò che però è noto è il legame fra l'output e la funzione di attivazione del nodo intermedio, che possiamo indicare con z. A sua volta è noto il legame fra la funzione di attivazione e il peso w_1. E quindi, sfruttando due volte la regola della catena possiamo scrivere:

$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w_1}$$

In questo modo è stato possibile esplicitare il legame fra la funzione di costo e tutti i pesi, cosa che ci permette di calcolare il gradiente di J.
Con l'aumento dei nodi e dei layer l'essenza della back propagation non cambia: la regola della catena andrà applicata per ogni peso della rete, in maniera ricorsiva per ogni layer.

Poiché per reti relativamente grandi il calcolo del gradiente può risultare particolarmente pesante computazionalmente, spesso si preferisce una variante della discesa del gradiente, definita stocastica, per cui il gradiente del costo è calcolato solamente in un certo numero di punti e non su tutta la funzione.

4.2 Reti ricorsive

Le reti feed-forward hanno una forte limitazione, che le rendono poco adatte in alcune applicazioni molto importanti: non posseggono uno stato (escludendo i pesi). Per questo motivo non possono integrare il concetto di "tempo", che è importante in situazioni in cui è presente una sequenzialità fra gli input, ovvero quando ogni input dipende anche dai precedenti. Alcuni esempi di applicazioni in cui questo è importante possono essere:

- Eseguire predizioni di completamento di una frase: la scelta di una parola in una frase è senza dubbio collegata alle parole precedentemente utilizzate.
- Il riconoscimento vocale: per lo stesso motivo precedente, se una rete deve interpretare che cosa rappresenta un suono emesso da una persona è importante tenere in considerazione anche ciò che è stato detto prima
- L'auto-completamento di melodie a partire da una certa porzione di musica: è chiaro come la scelta di quali note inserire dipende dalla sequenza iniziale data in input

Per questo motivo sono state introdotte anche delle versioni più complesse di reti neurali, che prendono il nome di *reti neurali ricorsive*, o *RNN*, che sta ovviamente per *Recursive Neural Network*.

Nelle reti ricorsive vengono introdotti come nodi della rete celle di memoria in cui è possibile mantenere uno stato tra le iterazioni. Lo stato diventa quindi un input: l'output della cella al *tempo t* assume la forma $y_t = f(x_t, h_{t-1})$. Questo è ciò che genera la ricorsione nella rete. Ad ogni iterazione ovviamente lo stato subisce un update esattamente come i pesi, secondo una funzione del tipo $h_t = g(x_t, h_{t-1})$, che funge anche da funzione di attivazione. Una tipica funzione di update dei pesi è:

$$h_t = \tanh(W_{hh}^T h_{t-1} + W_{xh}^T x_t)$$

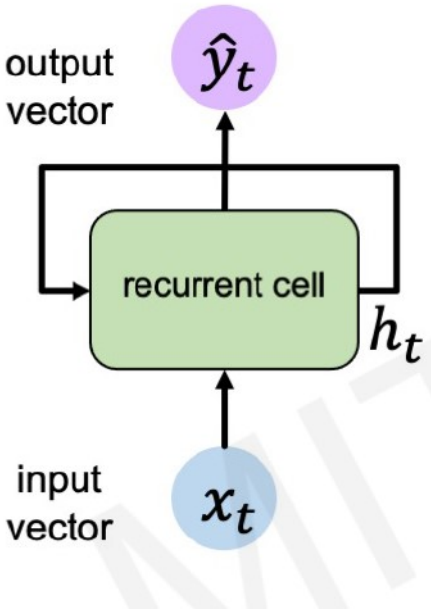

Figura 6: In un neurone ricorsivo lo stato all'iterazione precedente è un input all'iterazione successiva [16]

e a questo punto l'output del nodo di una rete ricorsiva al tempo t sarà:

$$y_t = W_{hy}^T h_t$$

dove W_{hh}, W_{xh} e W_{hy} sono le rispettive matrice di pesi.
Durante l'allenamento quindi verrà calcolata la funzione di costo per ogni step temporale, verranno sommati i costi per trovare quella totale e a quel punto verranno modificati i pesi opportunamente sfruttando la back propagation. Va però sottolineato che per le reti ricorsive la back propagation assume una forma più complessa, che prende il nome di *Back Propagation Through Time*, o *BPTT*. Diversamente da quello che accadeva per le reti feed-forward, in questo caso ci stiamo anche spostando indietro nel tempo (da qui il nome). Questo significa che la propagazione all'indietro deve anche ripercorrere gli step temporali. Ciò inevitabilmente incrementa di molto il numero di calcoli da eseguire.
Ed è proprio per questa ragione che iniziando ad implementare modelli di reti ricorsive ci si accorge che è molto comune l'insorgere di un problema chiamato *scomparsa del gradiente* (*vanishing gradient* in inglese). Per via delle lunghe catene di operazioni necessarie, i gradienti, e di fatto quindi lo stesso output

delle RNN, tendono a decadere esponenzialmente, tendendo così a 0 [11]. Più raro, ma comunque presente, è il caso opposto, ovvero *l'esplosione del gradiente*, per cui i risultati tendono a infinito. Questo a sua volta causa un problema parecchio grave: le reti RNN faticano a catturare dipendenze a lunga distanza temporale, che è a dir poco contraddittorio con la loro idea di base. Per questa sgradevole caratteristica le reti RNN di per sé non sono davvero utilizzabili in applicazioni reali.

4.3 Reti LSTM

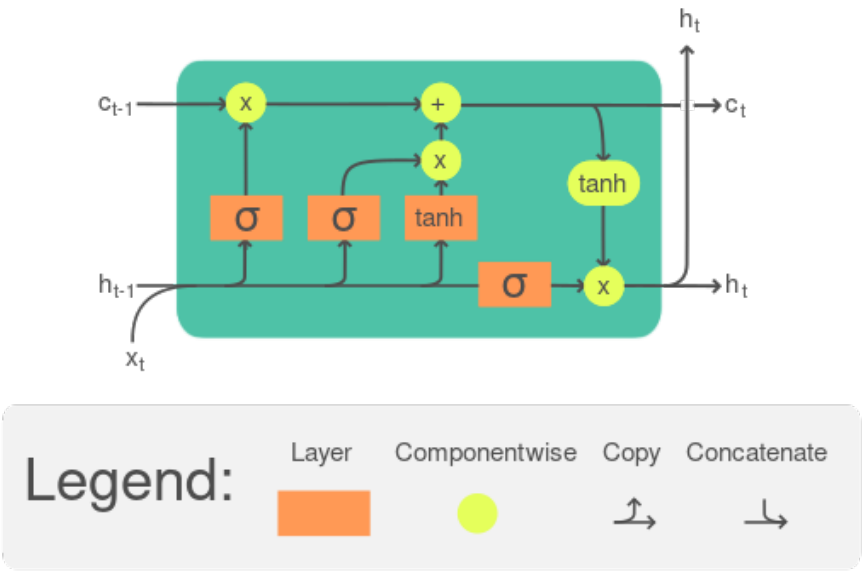

Figura 7: Un esempio di blocco di memoria LSTM, in cui sono segnati i gate utilizzati per controllare le informazioni che vengono passati tra gli step [17]

Per risolvere il problema della scomparsa del gradiente nasce una forma più complessa di rete neurale ricorsiva, che prende il nome di *LSTM*, ovvero *Long-Short Term Memory*. L'idea alla base di questa evoluzione delle reti RNN è sfruttare un'unità ricorsiva più complessa che possiede dei gate con i quali è possibili controllare quali informazioni vengono passate agli step successivi. Questo maggiore controllo permette di ridurre notevolmente il problema della

scomparsa del gradiente, e di conseguenza di costruire reti ricorsive con capacità molto migliori di catturare dipendenze a lunga distanza.
Nello specifico, ogni cella LSTM ha uno stato che può essere *letto, dimenticato o resettato* tramite una serie di gate programmabili [11]. Questo stato quindi può essere sia mantenuto tra gli step temporali tramite la ricorsione, sia essere modificato. Possiamo generalmente identificare tre gate:

- L'input gate è capace di modulare l'input che arriva nella cella sfruttando delle funzioni non lineari come una sigmoide (che trasforma l'input in un numero tra 0 e 1) o una tangente iperbolica (che trasforma l'input in un numero tra -1 e 1).
- Il forget gate sfrutta le stesse funzioni del precedente, ma lavora sulla connessione ricorsiva.
- Infine l'output gate permette di modulare l'uscita del nodo, che poi si propagherà ai layer successivi, scegliendo quanto vogliamo che lo stato lo influenzi.

Per esempio, modulando l'input a 0 e la ricorsione a 1 possiamo mantenere lo stato costante. Facendo l'opposto invece lo stato della cella viene sovrascritto dall'input. Infine, modulandoli entrambi a 0 lo stato viene a sua volta resettato a 0.
Tramite questo controllo è possibile mitigare di molto il problema della scomparsa del gradiente, rendendo così le LSTM una versione delle RNN effettivamente sfruttabile.

5 Materiali e metodi

5.1 Obiettivi

Le varie tecniche presentate nel capitolo 3 sono focalizzate sul riconoscimento di nomi di dominio malevoli. Se la tecnica di riconoscimento è retroattiva, lo scopo è principalmente blacklistare i nomi malevoli identificati, mentre le tecniche in tempo reale sono molto utili per poter bloccare le richieste DNS fortemente sospette e ridurre così sensibilmente la pericolosità delle botnet. A questo punto le strade percorribili sono due: cercare di risalire al server, o ai server, il cui IP si lega a questi nomi di dominio, in modo tale che le forze dell'ordine possano intervenire, oppure tentare di studiare e riconoscere le macchine host da cui partono queste richieste, per poterle quindi categorizzare come infette da un qualche tipo di bot. Il classificatore presentato in questo capitolo della tesi ha proprio questo secondo scopo: dato un insieme di richieste DNS è costruito per riconoscere quando una particolare macchina host è infetta, avvertendo l'utente a schermo, che poi potrà decidere come comportarsi di conseguenza.
Come verrà analizzato più nel dettaglio successivamente, per problemi di costruzione del dataset nei tempi prefissati purtroppo non è stato effettivamente possibile testare in tempo il classificatore proposto. Per cui quella che seguirà sarà essenzialmente una presentazione prettamente teorica.

5.2 Dataset utilizzati

Come sottolineato nel capitolo precedente, non è stato possibile compiere esperimenti completi sul classificatore in tempo. Questo perché la raccolta dei dati non ha dato i risultati sperati, come verrà successivamente analizzato nel capitolo 5.4. Per costruire il dataset l'idea era quella di sfruttare il traffico di rete del GARR [18], la rete nazionale che si dedica alle strutture scolastiche, universitarie e di ricerca. Sono stati raccolti dati giornalmente, a partire dal 21 Luglio 2021 fino all'11 Settembre 2021, per tutte le fasce orarie della giornata. I log raccolti contengono una richiesta DNS per riga, e forniscono varie informazioni su di essa, come l'IP della macchina da cui è partita (hashato, per motivi di privacy), il nome di dominio richiesto, l'IP risolto e il timestamp della richiesta. Contengono quindi tutti i dati che potevano servire per l'allenamento di questa

rete, e per lo più anche strutturati molto comodamente. Il problema è nato di fronte alla costruzione del dataset etichettato effettivamente da utilizzare. I vari nomi di dominio all'interno di questi log infatti vanno raccolti ed etichettati come benigni o malevoli (o al più scartati nel caso fosse impossibile riconoscerli). Questo processo sfrutta due ulteriori fonti:

- Il Majestic Million [19] come fonte dei nomi di dominio sicuramente benigni. Al suo interno troviamo il milione di domini con il maggior numero di sotto-reti di provenienza. Il file più recente utilizzato per l'etichettatura è del 9 Settembre 2021.

- I feed di Bambenek [13] per raccogliere i nomi di dominio generati da DGA da loro confermati. La raccolta di questi feed è cominciata più tardi del preventivato, a causa di un problema di ottenimento delle licenze. La raccolta è partita il 26 Agosto 2021, ed è terminata l'11 Settembre 2021.

L'etichettatura poi viene eseguita con uno script prodotto da David Caprari e Luca Mannini [20], che confronta i vari nomi di dominio con queste due fonti.
Tramite questo dataset è quindi possibile allenare il primo livello del classificatore. Con la prima rete allenata, successivamente, si può creare un nuovo dataset, sfruttando *nuovi* log del GARR (per evitare di usare gli stessi dati usati per il training del primo livello, che ci darebbe predizioni non accurate all'effettiva precisione della rete), che contiene i gruppi di predizioni del primo classificatore da usare per l'allenamento del secondo classificatore. Ora però è necessario sottolineare una cosa importante. Anche in questo caso per poter etichettare questo secondo dataset si riutilizzano i feed giornalieri di Bambenek: se una delle richieste di un gruppo contiene un nome di domino malevolo, l'IP da cui sono partite le richieste di quel gruppo è segnato come infetto. Questo però *non* rappresenta un'effettiva sicurezza di infezione: ciò rende le predizioni di questo classificatore più incerte di quanto i controlli eseguiti alla fine del training potrebbero suggerire.
Infine, di fronte all'impossibilità di costruire dataset con un numero sufficiente di nomi di dominio malevoli, per poter per lo meno eseguire l'addestramento del classificatore di primo livello è stato sfruttato un ulteriore dataset, che contiene più di 300.000 nomi di dominio catalogati come legit o provenienti da DGA, e in

questo secondo caso, da quale famiglia di malware. Questo dataset non contiene le ulteriori informazioni necessarie per il secondo livello, ma per il primo è più che sufficiente, anche grazie al fatto che è stato accuratamente costruito per ridurre il più possibile il problema dello sbilanciamento all'interno del dataset: il rapporto fra nomi legit e nomi costruiti da DGA è all'incirca 1:1.

5.3 Struttura e uso del classificatore

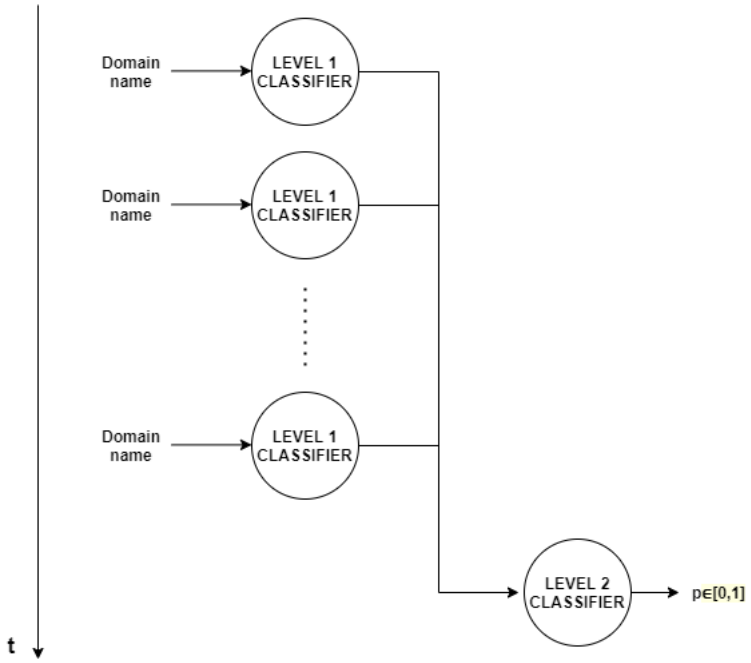

Figura 8: Modello di funzionamento del classificatore

Il processo di classificazione è suddiviso in 3 parti, sintetizzabili in questo modo:

- Per prima cosa le richieste DNS vengono raggruppate per indirizzo IP. La dimensione di questi gruppi non è fissata: va scelta. Da ora in poi la dimensione sarà indicata genericamente come n. I gruppi vengono creati sempre con richieste provenienti dallo stesso giorno. Questo è fatto per tentare di riunire richieste malevoli che fanno parte dello stesso fenomeno di domain fluxing.

- Poi le varie richieste dei gruppi che sono stati creati vengono date in pasto alla prima rete neurale, il classificatore binario proposto da Tran et al. in [4]. Questa classificazione è ripetuta per n volte prima di passare la lista degli output ottenuti al secondo livello di classificazione.

- Questo insieme di predizioni, che di fatto è una lista di valori fra 0 e 1, sono gli input del secondo livello di classificazione. Vengono quindi passati ad una seconda rete neurale, che restituisce infine come output un valore che rappresenta se l'host da cui sono partite le richieste DNS di quel gruppo è presumibilmente infetto oppure no.

Andiamo ora più nel dettaglio nell'analisi della struttura e del funzionamento del classificatore, che è implementato sfruttando il linguaggio di programmazione Python.

5.3.1 Lettura e pre-processamento dei dati

I dati provenienti dai log del GARR contengono più informazioni del necessario, quindi in lettura vengono estratte solo quelle necessarie: nome di dominio e indirizzo IP hashato. Il fatto che l'indirizzo IP sia criptato tramite una funzione di hash per motivi di privacy non è un problema per i nostri scopi, dato che verranno usati semplicemente per raggruppare le richieste provenienti dallo stesso host. Ovviamente per fare ciò bisogna supporre che i fenomeni di collisione nel processo di hashing siano sufficientemente rari da essere trascurabili. Un'altra informazione necessaria per l'utilizzo di questo classificatore è la data in cui è stata eseguita la richiesta. All'interno dei log del GARR è presente un timestamp per ogni richiesta, tuttavia non è necessario estrarlo poiché il giorno e la fascia oraria sono segnati nei nomi dei file. Non sono necessarie informazioni temporali più precise per i nostri scopi, ma in caso lo fossero modificare il codice per salvare anche il timestamp non è particolarmente complicato. Analogamente, se non fosse possibile ricavare la data di creazione del log dal nome del file, andrà estratto anche il timestamp che indica quando la richiesta è stata eseguita, e ricavare da quello ciò che serve. Se si sta usando la rete pre-allenata è sufficiente ottenere il giorno del mese, mentre per eseguire il training, se non si ha a disposizione un dataset già etichettato, è necessario conoscere anche l'anno

e il mese per poter efficientemente eseguire l'etichettatura del proprio dataset, così come spiegato nel capitolo 5.2. In ogni caso, questo fortunatamente non richiede modifiche particolarmente complesse nel codice.
Per eseguire il raggruppamento si sfrutta una lista di IP nella quale vengono inseriti, *giorno per giorno*, gli indirizzi IP rilevati. Viene quindi creato anche un dizionario, che conterrà come chiavi gli indirizzi IP, e come valori le liste dei domini richiesti da tali indirizzi IP.
Si ciclano quindi tutti i log a disposizione, aggiornando adeguatamente le due strutture dati precedentemente descritte. Per ogni log si controlla se si è passati ad un giorno del mese diverso. In questo caso si controlla il contenuto del dizionario, scorrendolo n richieste alla volta. Ed è a questo punto che viene eseguita la classificazione.

5.3.2 Primo livello di classificazione

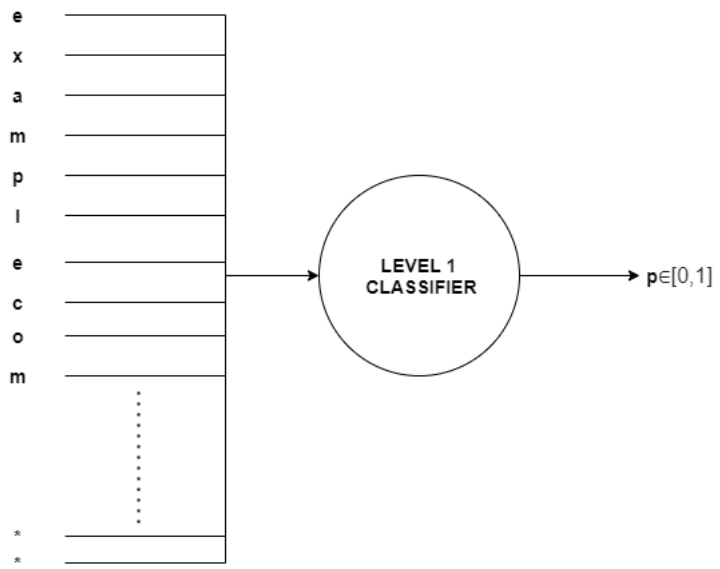

Figura 9: Rappresentazione del funzionamento del primo livello di classificazione, usando il nome di dominio "example.com".

Il primo classificatore è una rete neurale LSTM con un layer di input, un layer nascosto di celle LSTM, e un layer di output. La funzione di attivazione scelta è il sigmoide, la funzione di costo è invece la *binary cross-entropy*, che è

la funzione di costo per i problemi di classificazione presentata nel capitolo 4.1. Il layer di input prende in ingresso N interi, ognuno dei quali corrisponde ad un carattere. N è un numero arbitrario, che però va scelto sufficientemente grande da poter rappresentare tutti i nomi di dominio presenti nel dataset. Poiché l'ingresso della rete deve essere necessariamente un numero infatti, ogni nome di dominio, dopo aver rimosso tutti i punti, viene trasformato in una lista di interi. Viene successivamente eseguito del padding su tale sequenza per ottenere un input di dimensione costante. Nella figura 9 vengono segnati come input i caratteri e non gli interi per facilitare la comprensione del funzionamento.
L'output della rete è un numero tra 0 e 1, che può essere interpretato come la probabilità che il nome di dominio in input sia o meno proveniente da un DGA: 1 rappresenta la sicurezza che sia un nome malevolo, 0 che invece sia legit.

5.3.3 Secondo livello di classificazione

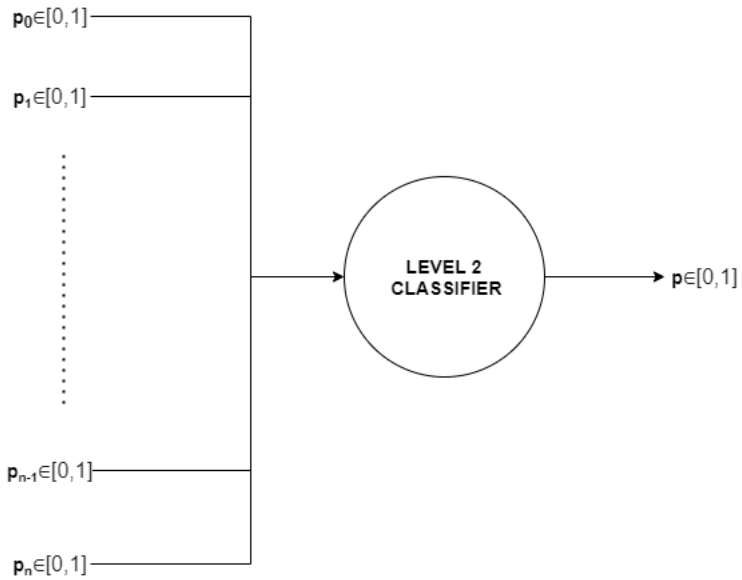

Figura 10: Rappresentazione del funzionamento del secondo livello di classificazione.

Dopo aver ripetuto l'esecuzione del primo livello di classificazione per n volte, la lista delle n predizioni viene passata al secondo classificatore. Anche questo è costituito da una rete neurale LSTM, con la stessa struttura del precedente:

un layer di input, un layer nascosto di celle LSTM e un layer di output. Analogamente, anche le funzioni di attivazione e di costo sono le medesime. Gli input sono n numeri compresi tra 0 e 1, mentre l'output è a sua volta un numero compreso da 0 e 1. In questo caso questo valore può essere interpretato come la probabilità che la macchina da cui sono partite le n richieste DNS rappresentate dall'input sia o meno infetta da un bot: 1 indica la sicurezza di infezione, 0 l'opposto.
Una volta dato in input una lista di predizioni del primo livello, questo classificatore produrrà il suo output. Se una macchina host viene rilevata come infetta, l'utente viene avvisato con un messaggio a schermo, altrimenti la computazione continua con il prossimo input.

5.3.4 Addestramento delle reti

Come precedentemente analizzato, non è stato possibile allenare il classificatore di secondo livello per mancanza di un dataset reale con sufficiente presenza di nomi di dominio riconosciuti dai feed di Bambenek come generati da DGA. Perciò non verranno presentati dati sull'efficienza del secondo livello di classificazione, in quanto incalcolabile.
Ciò che però è possibile mostrare è l'efficienza del primo livello di classificazione, ovvero del classificatore binario di Tran et al.. Sfruttando il terzo dataset presentato nel capitolo 5.2 non ci sono stati problemi nell'esecuzione del suo allenamento. Questo è stato eseguito via Google Colab, un servizio tramite il quale Google mette a disposizione a chiunque dell'hardware per far girare in remoto il proprio codice. È possibile tramite Colab anche selezionare acceleratori hardware: GPU e soprattutto, in questo caso, TPU. Tramite le TPU di Google i tempi di addestramento delle reti neurali si riducono sensibilmente.
Ci sono vari indicatori sfruttabili per rappresentare l'efficacia che una rete ha raggiunto dopo l'allenamento. Per prima cosa abbiamo l'*accuratezza* (*accuracy* in inglese), che è definita come:

$$Accuratezza = \frac{Numero\ di\ predizioni\ corrette}{Numero\ di\ predizioni\ totali} = \frac{TP + TN}{TP + TN + FP + FN}$$

dove TP, TN, FP e FN sono rispettivamente il numero di veri positivi, veri negativi, falsi positivi e falsi negativi. Nei problemi di classificazione in cui

non è presente sbilanciamento all'interno del dataset questo è generalmente sufficiente, ma qualora il dataset fosse sbilanciato l'accuratezza non è più una misura veritiera dell'efficacia della rete, in quanto non tutte le classi hanno lo stesso peso per questa misura [4]. Per questo vengono introdotti altri indicatori, più adatti in queste situazioni. Per questi indicatori ulteriori va inoltre tenuto in considerazione che possono essere analizzati in due modi: si parla di *micro-averaging* e *macro-averaging*. Nel secondo caso i valori vengono mediati per il numero di classi. Questo permette di avere risultati più accurati in presenza di sbilanciamento. Sfrutteremo l'indice m per indicare il micro-averaging e l'indice M per il macro-averaging. Possiamo a questo punto definire tre indicatori per entrambe le modalità.

La *precisione* (*precision* in inglese) è definita come:

$$Precisione_m = \frac{TP}{TP + FP} \qquad Precisione_M = \frac{\frac{TP}{TP+FP}}{Numero\ di\ classi}$$

Il *recall* è definito come:

$$Recall_m = \frac{TP}{TP + FN} \qquad Recall_M = \frac{\frac{TP}{TP+FN}}{Numero\ di\ classi}$$

Infine, l'*F1-score* è una media armonica (il reciproco della media aritmetica) delle due misure precedenti, prese o in micro-averaging o in macro-averaging:

$$\text{F1-score} = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}}$$

Di seguito sono riportati i risultati in macro-averaging ottenuti con il training del classificatore binario:

F1 Score	0.9664438207800686
Recall	0.9664444444444444
Precision	0.9664791237694859
Accuracy	0.9664444444444444

Poiché lo sbilanciamento del dataset per la classificazione binaria tende a non essere esageratamente pesante, la differenza fra micro-averaging e macro-averaging in realtà non è particolarmente grande. È stato scelto di riportare comunque i

dati con macro-averaging in quanto, come riportato nell'articolo di Tran et al. [4], rappresenta la misura migliore in caso di aggiunta futura di un livello di classificazione multiclasse. Tale possibilità verrà discussa nel capitolo 6.
Vengono di seguito poi mostrati i risultati di Tran et al. in macro-averaging così come riportati nel loro articolo:

F1 Score	0.9843
Recall	0.9840
Precision	0.9842

Purtroppo risulta mancante l'accuracy, che i ricercatori hanno deciso di omettere e che dunque non è possibile confrontare con i risultati precedenti.

5.4 Problematiche riscontrate nella costruzione dei dataset

Come accennato precedentemente, il tentativo di costruzione dei dataset a partire dai log del GARR non è andato a buon fine. E questo perché non è stato possibile rilevare un numero sufficienti di nomi di dominio generati da DGA per rendere questo un dataset utilizzabile.
Per prima cosa il processo di etichettatura è stato eseguito secondo le modalità predefinite dallo script di Caprari e Mannini: per ogni *giornata* i log vengono confrontati con un dizionario di nomi di dominio benigni, costruito dalla lista di Majestic, e con un dizionario di nomi malevoli, costruito dal feed Bambenek *di quel giorno*. Quando si passa ad un giorno successivo il dizionario di nomi DGA viene ricostruito. Questo è un approccio sensato, in quanto, come analizzato nel capitolo 2.3, un nome di dominio costruito da un DGA ha una validità temporale molto limitata, per cui è altamente improbabile che un nome valido un giorno fosse valido anche il precedente o sia valido anche il successivo. Poiché i feed Bambenek sono stati raccolti a partire dal 26 Agosto 2021, in questo caso sono stati sfruttati solamente i log del GARR a partire dallo stesso giorno. Purtroppo però, di fronte a più di 30.000 nomi di dominio riconosciuti come facenti parte della lista del Majestic, solamente *due* sono stati riconosciuti come malevoli: *"ks1n8yamb.ru"* e *"7mb0ayr6.ru"*, entrambi facenti parte della famiglia di malware *"Dromedan"*. Da dati raccolti da un traffico rete giornaliero

reale ci si può aspettare un forte sbilanciamento, ma qua è fin troppo elevato, essendo nell'ordine di 1:15.000.

Ben consapevole del fatto che per via del funzionamento dei DGA il confronto giorno per giorno è effettivamente l'approccio più adatto, si è comunque tentato di eseguire un etichettatura più completa, confrontando *tutti* i log del GARR ottenuti fino a quel momento con un dizionario costruito da *tutti* i feed Bambenek. L'aggiunta di un mese intero di log del GARR ha dunque circa raddoppiato la lista di nomi di dominio legit riconosciuti nella lista di Majestic, arrivando così a quasi 70.000, ma purtroppo il numero di nomi riconosciuti come malevoli non è aumentato.

Le motivazioni per questa mancanza di domini riconosciuti dal GARR possono essere vari. Il GARR gestisce un sottoinsieme della rete Internet, che però non è un sottoinsieme davvero rappresentativo del traffico medio di rete. Come precedentemente detto infatti, il GARR gestisce le reti universitarie e dei centri di ricerca. Per prima cosa si può quindi supporre che chi sfrutta frequentemente queste reti sia più attento dell'utente medio per quanto riguarda la sicurezza informatica, il che, plausibilmente, riduce sensibilmente le possibilità di infezione da parte di malware (questo però non significa *assolutamente* che le infezioni non possano avvenire, *anzi*). Inoltre, va sicuramente anche considerato il periodo di raccolta dei dati: il periodo estivo. In questo periodo infatti il numero di utenti che si collegano a queste reti è senza dubbio sensibilmente minore. Inoltre ancora, va tenuto in considerazione che spesso sono presenti anche delle misure di sicurezza che tendono a bloccare il traffico di rete sospetto. Tutto ciò può aver portato ad un traffico di rete quasi privato di richieste DNS per nomi di dominio malevoli, per lo meno riconosciuti da Bambenek.

6 Conclusioni e Sviluppi futuri

In questa tesi abbiamo presentato un classificatore, basato su una coppia di reti neurali LSTM, con lo scopo di riconoscere calcolatori infetti facenti parte di botnet a partire da del traffico di rete. Come spiegato però, il lavoro su questo classificatore non è andato come inizialmente previsto. Per cui molto è ancora da fare in futuro, più di quanto inizialmente immaginato. Per prima cosa infatti, sicuramente, il classificatore va testato. Senza un dataset utilizzabile non è stato possibile eseguire l'allenamento, come spiegato precedentemente, e dunque verificare l'efficienza reale del modello proposto. È quindi prioritario il raccoglimento di dati più utili, o dalla stessa fonte usata finora o da una nuova fonte. Due possibili nuovi fonti utilizzabili al posto del GARR sono le seguenti:

- Il sito `ieee-dataport.org` fornisce dataset già etichettati che sono scaricabili e sfruttabili. Uno di questi è il "TI-2016 DNS DATASET" [21], che contiene reale traffico DNS di 10 giorni, con anche delle feature etichettate.

- Sul sito `https://www.stratosphereips.org/` troviamo vari dataset sfruttabili, tra cui quello di nostro interesse: il dataset CTU-13 [22], reso disponibile dagli autori dell'articolo "An empirical comparison of botnet detection methods" [23]. Tale dataset contiene traffico rete di botnet mischiato con traffico normale e di background.

In ogni caso, una volta ottenuto un dataset soddisfacente ci sono molteplici test eseguibili, tra i quali sono evidenziabili i seguenti:

- Per prima cosa si possono testare differenti strutture per il modello del secondo livello. Nella versione qua proposta è stata presa la stessa struttura del classificatore del primo livello, ma non c'è stato modo di controllare se sia stata una scelta azzeccata e se ci siano opzioni migliori

- Una volta finalizzata la struttura, ci sono vari esperimenti da poter eseguire. Primo fra tutti è il test della grandezza dei raggruppamenti. Si può testare quindi come varia l'efficienza del classificatore modificando il numero di predizioni del primo livello che vengono inserite nel gruppo che diventa l'input per il secondo livello di classificazione. Per fare ciò chia-

ramente va modificata la struttura del secondo livello: vanno cambiati i numeri di input, e perciò va ogni volta rieseguito l'allenamento.

- Un altro esperimento interessante può essere studiare il comportamento del classificatore modificando la tipologia di input del secondo livello. Ora come ora vengono raggruppati direttamente gli output del primo livello, che sono numeri compresi fra zero e uno. È possibile però rendere questi input esclusivamente 0 e 1, dove lo 0 rappresenta un nome malevolo e l'1 un nome legit. Chiamando x_1 l'output del classificatore e x_2 ciò che inseriamo nei raggruppamenti, possiamo semplicemente quindi definire x_2 come:

$$x_2 = \begin{cases} 0 & \text{se } x_1 < 0.5 \\ 1 & \text{se } x_1 >= 0.5 \end{cases}$$

- Infine, una volta eseguiti tutti i test desiderati si potrà passare ad implementare un'evoluzione di questo classificatore. Attualmente il classificatore sfrutta esclusivamente la componente binaria del classificatore di Tran et al. [4]. Ma è sicuramente possibile sostituire il classificatore binario con quello multi-classe, oppure inserire quest'ultimo come livello intermedio, trasformando il classificatore in un classificatore a tre livelli. Nel primo caso si avrebbe una computazione più veloce, ma eliminando la classificazione binaria si rischia di incorrere nei problemi derivanti dall'approccio cost-sensitive analizzati nel capitolo 3.2. Nel secondo caso invece si evitano questi problemi, ma si rallenta la computazione, che per un classificatore che deve lavorare in tempo reale potrebbe risultare problematico. Indipendentemente dall'approccio scelto, si avrebbe un classificatore ad n output, uno per i nomi legit e uno per ogni classe di malware con cui si vuole lavorare. Questo classificatore quindi, propriamente allenato, potrebbe diventare capace di dirci non solo se una macchina host è infetta, *ma anche da che tipologia di malware.*

Riferimenti bibliografici

[1] R. Tilmann, "https://creativecommons.org/licenses/by-sa/2.5, via Wikimedia Commons, convertito in formato PNG (CC BY-SA 2.5)," accessed: 2021-10-06. [Online]. Available: https://commons.wikimedia.org/wiki/File:DNS_schema.svg

[2] zitrax, "DNS for Rocket Scientists," accessed: 2021-10-06. [Online]. Available: http://www.zytrax.com/books/dns/

[3] JeroenT96, "CC BY-SA 4.0 https://creativecommons.org/licenses/by-sa/4.0, via Wikimedia Commons," accessed: 2021-10-06. [Online]. Available: https://commons.wikimedia.org/wiki/File:Botnet2.gif

[4] D. Tran, H. Mac, V. Tong, H. A. Tran, and L. G. Nguyen, "A lstm based framework for handling multiclass imbalance in dga botnet detection," *Neurocomputing*, vol. 275, pp. 2401–2413, 2018.

[5] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "Exposure: Finding malicious domains using passive dns analysis." in *Ndss*, 2011, pp. 1–17.

[6] H. R. Zeidanloo, A. B. A. Manaf, R. B. Ahmad, M. Zamani, and S. S. Chaeikar, "A proposed framework for p2p botnet detection," *International Journal of Engineering and Technology*, vol. 2, no. 2, p. 161, 2010.

[7] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon, "From throw-away traffic to bots: detecting the rise of dga-based malware," in *21st {USENIX} Security Symposium ({USENIX} Security 12)*, 2012, pp. 491–506.

[8] S. Yadav, A. K. K. Reddy, A. N. Reddy, and S. Ranjan, "Detecting algorithmically generated domain-flux attacks with dns traffic analysis," *IEEE/Acm Transactions on Networking*, vol. 20, no. 5, pp. 1663–1677, 2012.

[9] R. Perdisci, I. Corona, and G. Giacinto, "Early detection of malicious flux networks via large-scale passive dns traffic analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 9, no. 5, pp. 714–726, 2012.

[10] S. Schiavoni, F. Maggi, L. Cavallaro, and S. Zanero, "Phoenix: Dga-based botnet tracking and intelligence," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2014, pp. 192–211.

[11] J. Woodbridge, H. S. Anderson, A. Ahuja, and D. Grant, "Predicting domain generation algorithms with long short-term memory networks," *arXiv preprint arXiv:1611.00791*, 2016.

[12] ""does alexa have a list of its top-ranked websites?"," accessed: 2021-10-06. [Online]. Available: https://support.alexa.com/hc/en-us/articles/200449834-Does-Alexa-have-a-list-of-its-top-ranked-websites-

[13] "Bambenek consulting - master feeds," accessed: 2021-10-06. [Online]. Available: https://faf.bambenekconsulting.com/feeds/

[14] Dake and Mysid, "CC BY 1.0 https://creativecommons.org/licenses/by/1.0, via Wikimedia Commons, convertito in formato PNG," accessed: 2021-10-06. [Online]. Available: https://commons.wikimedia.org/wiki/File:Neural_network.svg

[15] S. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Prentice Hall, 2002.

[16] A. Amini and A. Soleimany, "MIT 6.S191: Introduction to deep learning, © Alexander Amini and Ava Soleimany," accessed: 2021-10-06. [Online]. Available: IntroToDeepLearning.com

[17] G. Chevalier, "CC BY-SA 4.0 https://creativecommons.org/licenses/by-sa/4.0, via Wikimedia Commons, convertito in formato PNG," accessed: 2021-10-06. [Online]. Available: https://commons.wikimedia.org/wiki/File:LSTM_Cell.svg

[18] "Consortium GARR," accessed: 2021-10-06. [Online]. Available: https://www.garr.it/it/

[19] "Majestic Million," accessed: 2021-10-06. [Online]. Available: https://majestic.com/reports/majestic-million

[20] D. Caprari and L. Mannini, "Data-collector di domini sicuramente benevoli e sicuramente malevoli," https://gitlab.com/christian.morbidoni/garr-bambenek-data-collector, 2019.

[21] M. Singh, M. Singh, and S. Kaur, *TI-2016 DNS dataset.* IEEE Dataport, 2019. [Online]. Available: https://dx.doi.org/10.21227/9ync-vv09

[22] "CTU-13 Dataset," accessed: 2021-10-06. [Online]. Available: https://www.stratosphereips.org/datasets-ctu13

[23] S. Garcia, M. Grill, J. Stiborek, and A. Zunino, "An empirical comparison of botnet detection methods," *computers & security*, vol. 45, pp. 100–123, 2014.