UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Elettronica

Tecniche di sicurezza nelle reti basate su rilevamento di malware

Network security techniques based on malware detection

Relatore:
Prof. Marco Baldi

Tesi di Laurea di:
Giuseppe Granatiero

Correlatori:
Prof. Alessandro Cucchiarelli
Prof. Luca Spalazzi

Anno Accademico 2018/2019

Ai miei genitori

Indice

Capitolo 1

Introduzione

Il numero dei dispositivi connessi ad internet è in costante aumento. L'utilizzo dei servizi online da parte di questi dispositivi porta ad un enorme volume di transazioni finanziarie, dove vengono scambiati tramite internet dati sensibili. Dall'ultimo rapporto Clusit[1] emerge un incremento degli attacchi gravi. Per attacchi gravi si intendono attacchi che portano a pesanti conseguenze ad esempio come perdite economiche, danni alla reputazione o diffusione di dati sensibili.

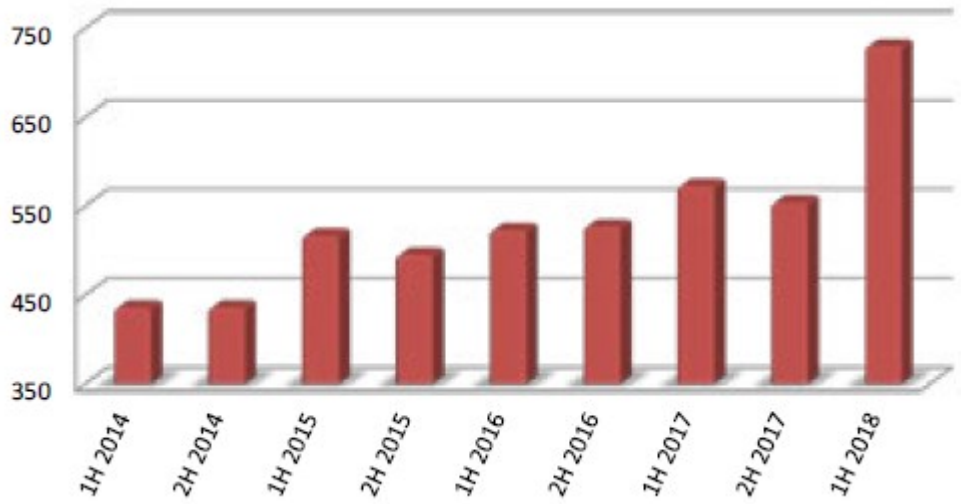

Figura 1.1: Attacchi gravi rilevati per semestre

Durante il primo semestre 2018 si sono registrati 7.595 attacchi a livello mondiale di cui 730 in Italia, constatando un incremento rispetto all'ultimo semestre 2017 del 31.77%.

ATTACCANTI PER TIPOLOGIA	2014	2015	2016	2017	2H 2017	1H 2018	Variazioni 1H 2018 su 2H 2017	Trend 1H 2018
Cybercrime	526	684	751	857	434	587	35,25%	↑
Hacktivism	236	209	161	79	34	29	-14,71%	↓
Espionage / Sabotage	69	96	88	129	55	93	69,09%	↑
Information Warfare	42	23	50	62	31	21	-32,26%	↓
TOTALE	873	1.012	1.050	1.127	554	730	+31,77%	↑

Tabella 1.1: Distribuzione degli attaccanti per tipologia

L'interesse di un attaccante può essere mosso sia dalla semplice curiosità ma soprattutto da un ritorno economico. I cyber-criminali utilizzano diversi tipi di malware per raggiungere i loro obiettivi. I malware di tipo Botnet sono considerati tra i più pericolosi. Una rete Botnet contiene Bot, che sono computer infettati da malware senza il permesso dell'utente. Il botmaster da remoto gestisce la botnet attraverso un canale di comando e controllo.
Nel lavoro di tesi svolto si illustrano le topologie, gli scopi e le tecniche adottate per stabilire una intera rete botnet. Nello specifico si analizzano le tecniche per mantenere in vita la rete tramite l'utilizzo di algoritmi di generazione automatica dei nomi di dominio (*DGA*).

Sono state sviluppate due tecniche per il rilevamento di malware:

- La prima tecnica è stata sviluppata presso l'azienda Namirial di Senigallia. Questa tecnica sfrutta un prodotto commerciale ma lo utilizza all'interno di un sistema che è stato creato durante il tirocinio per automatizzare la somministrazione e l'analisi dei malware.

- La seconda tecnica è stata sviluppata in ambito accademico. Questa tecnica si basa su algoritmi di intelligenza artificiale e ha il vantaggio di non richiedere firme, come ad esempio antivirus e altro, ma lavora con dei dataset creati da reale traffico di rete.

I DGA sono stati oggetto di altre tesi [7] [8] nelle quali è stato allenato un classificatore in grado di riconoscere se un dominio proviene o meno da DGA.
L'obiettivo di questa tesi è stato quello di capire se la classificazione basata su DGA trova riscontro ed è utile per l'identificazione del malware che fa uso di DGA. Per questo scopo è stato necessario creare un dataset dall'analisi di reale traffico di rete di computer infetti da malware di tipo botnet. I dati estrapolati provengono da una fonte certa [9] e sono stati utilizzati solamente dati contenenti traffico di rete generato da DGA.

Capitolo 2

Botnet

Il termine **botnet** deriva da roBOT NETwork e rappresenta una rete formata da dispositivi informatici collegati ad internet ed infettati da *malware*. Questi dispositivi si trovano sotto il comando di un singolo *hacker*, o un piccolo gruppo di hacker, conosciuto come **botmaster**. I dispositivi che compongono la *botnet* sono chiamati **bot** o *zombie*. Un esempio di botnet è mostrato in figura 2.1.

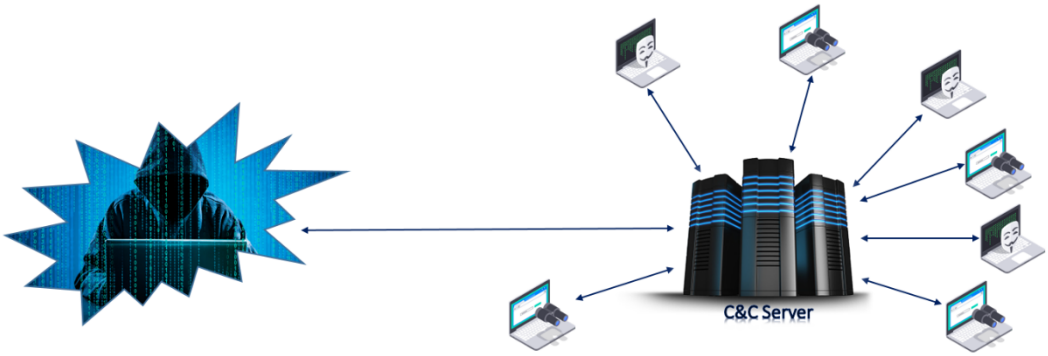

Figura 2.1: Esempio di una botnet

Tutti i bot eseguono i comandi impartiti dal botmaster attraverso il server *Command & Control* (C&C). Le botnet sono più pericolose rispetto ai comuni virus e/o malware in quanto in questo modo il botmaster è in grado di controllare le risorse di un gran numero di computer compromessi.

Quando l'attaccante riesce a infettare una nuova macchina, essa diventa a sua volta un bot e viene inserita nella rete botnet aumentandone la capacità. Una volta infettata, la macchina, continuerà a lavorare come se nulla sia successo, mentre in realtà esegue in *background* i comandi che gli vengono impartiti dal botmaster. Il botmaster ha la capacità di coordinare tutti i bot presenti all'interno della botnet e all'occorrenza sferrare attacchi sincronizzati con una notevole capacità di calcolo.

2.1 Principali applicazioni

Le botnet sono spesso utilizzate per condurre vari attacchi, che vanno da attacchi *Distributed Denial-of-Service* (DDoS), ad attacchi a *e-mail* tramite *spamming*, *keylogging*, *clic fraudolenti*, e per diffondere nuovi malware. A differenza di altri tipi di attacchi, le botnet possono essere composte da migliaia di host compromessi in grado di assemblare una enorme quantità di potenza di calcolo aggregata con l'obiettivo di eseguire una serie di attacchi contro un'ampia gamma di dispositivi. Per esempio, un botmaster può comandare ad ogni bot incluso in una botnet di lanciare email spam, effettuare una sorta di furto di carte di credito (raccolte da *keylogger* installati di nascosto), e lanciare attacchi DDoS simultaneamente contro migliaia di computer. A causa di questo, gli *hacker* sono sempre più interessati a utilizzare le botnet per massimizzare i loro guadagni finanziari.

Allo stesso tempo, il grado di distruzione causata dagli hacker utilizzando attacchi botnet è centinaia di volte più pericoloso degli attacchi tradizionali.
Le botnet possono propagarsi tramite reti-condivise, piattaforme di *file-sharing*, reti P2P (*Peer-to-Peer*) e/o *backdoor* predisposte da virus che sfruttano le vulnerabilità dei sistemi operativi.

2.2 Classificazione basata sui protocolli di rete

Affinché un botmaster sia in grado di interagire con i bot è necessario che sia stabilita una connessione tra i due. Tutte le connessioni di rete instaurate si basano su protocolli che definiscono le regole per l'interazione tra i computer sulla rete. Può essere fatta una prima classificazione delle botnet in base ai protocolli utilizzati.

2.2.1 Protocollo IRC

Il protocollo IRC è stato utilizzato nei primissimi tipi di botnet. Ogni computer infetto si connette al server IRC (master) ed attende i comandi dal suo master tramite il canale IRC. I notevoli miglioramenti nel monitoraggio di canali di questo tipo ha portato i malintenzionati ad abbandonare questo tipo di protocollo.

2.2.2 Programmi di messaggistica

Questo tipo di botnet non è particolarmente diffuso. Si basa sull'utilizzo dei canali di comunicazione forniti dai servizi di *messaggistica istantanea* (*IM*). Il problema più grande è che in questo caso i bot devono rimanere connessi alla rete ed ognuno di loro ha bisogno del proprio account di messaggistica istantanea per eseguire attività dannose. I proprietari di botnet orientate a IM dispongono di un numero limitato di account registrati, il che limita il numero di bot che possono essere online contemporaneamente.

2.2.3 Web

Questo è un tipo di botnet relativamente nuovo e in rapida evoluzione. I bot si connettono ad un *web server* tramite il protocollo *HTTP*. Attualmente viene ancora usato questo protocollo, spesso con l'ausilio della versione su *SSL* (*HTTPS*). Questo consente anche di criptare la comunicazione tra bot e botmaster, rendendo inefficace l'intercettazione del traffico di rete per il rilevamento di malware in attività.

2.3 Classificazione basata sulla topologia

Oltre alla classificazione in base ai protocolli, può essere fatta una classificazione che si basa sulla topologia delle botnet create. Il botmaster può scegliere la topologia più opportuna da attuare in base alle necessità. Esistono tre principali topologie di rete: *centralizzata*, *decentralizzata* e *non strutturata*.

2.3.1 Topologia centralizzata

Questa topologia si basa su un'architettura client – server dove i bot comunicano direttamente con il botmaster attraverso il C&C server. Il vantaggio di questa configurazione è la ridotta latenza e la consegna di pacchetti garantita, essenziale per organizzare e lanciare attacchi. Di questa topologia fanno parte le reti a *stella*, a *stella estesa* e *gerarchica*.
Nella configurazione a stella i computer sono tutti collegati direttamente al server C&C, mentre nella configurazione a stella estesa il botmaster può utilizzare server differenti per controllare la botnet. I server in questione hanno la funzione di simulare il comportamento del C&C server della stella semplice. Questo approccio presenta molti vantaggi rispetto al caso precedente, introducendo ridondanza a livello di server centrale, aumentando la scalabilità della rete e al tempo stesso una migliore gestione del traffico non più diretto esclusivamente verso il server Command & Control. Una rete multistella implica anche una struttura più complessa e una maggiore difficoltà nella sua gestione.

La topologia gerarchica deriva da quella a stella estesa, ma anziché avere più C&C server per la gestione della botnet esiste un unico server di controllo principale che utilizza dei bot come proxy. Il vantaggio è che nessun bot conosce l'intera struttura della rete, rendendo più difficile non solo il sequestro della rete compromessa, ma anche solo stimarne la reale dimensione. I bot utilizzati come proxy presentano le stesse caratteristiche del server centrale e solamente alcuni di essi sono a conoscenza dell'indirizzo che porta al botmaster.
I bot usati come proxy possono presentare problemi in termini di latenza, riducendo l'efficacia di attacchi in tempo reale. Come è possibile intuire, lo svantaggio principale dell'architettura centralizzata è la dipendenza estrema dal server principale: se il C&C server viene messo fuori uso, tutta la botnet viene compromessa. Inoltre, essendo tutti i bot connessi allo stesso punto, è molto facile da rilevare.

2.3.2 Topologia decentralizzata

Le reti strutturate in questo modo operano come una rete Peer To Peer che può essere completamente o parzialmente connessa. Le botnet in questo caso non hanno un'entità principale o un gruppo di controllo della rete, ma presentano un'architettura decentralizzata in cui non è presente un solo Command & Control server che assegna le istruzioni.
Nel primo caso un bot è connesso a tutti gli altri bot della rete, mentre nel secondo solo ad una parte.

L'aspetto fondamentale però è che ogni bot conosce solo i nodi a cui è strettamente collegato, mentre ignora completamente la restante parte.
Non avendo un punto centrale di comando, tutti i nodi sono allo stesso tempo master e slave (a differenza della topologia centralizzata, in cui erano ben identificati il master e gli slave) e possono inviare comandi come un server C&C. Questo rende la rete difficile da analizzare dall'esterno. Infatti i bot possono essere distribuiti in aree geografiche completamente diverse, e l'errore di un singolo nodo ha impatto limitato sul funzionamento dell'intera botnet.
I principali svantaggi sono dati dall'elevata complessità della struttura, che richiede l'implementazione di protocolli ad-hoc che possono garantire la connettività tra il botmaster e tutti i computer compromessi, possibilmente attraverso percorsi multi-hop. Questo comporta un'elevata latenza nella trasmissione dei messaggi, rallentando le comunicazioni.
La topologia a *maglia* è l'applicazione di questa infrastruttura.

2.3.3 Topologia non strutturata

In questo modello i bot sono completamente ignari della botnet a cui appartengono. Ogni volta che devono inviare un messaggio alla rete infetta, lo crittografano, analizzano in modo random la rete Internet e trasmettono il messaggio quando rilevano un altro bot.
Nonostante la struttura semplice, questa topologia non può

essere usata perché non sarebbe in grado di garantire la consegna effettiva dei pacchetti e sarebbe anche soggetta a latenze estremamente elevate.

2.4 Command & Control Channel

I computer infetti devono essere in grado di utilizzare una serie di metodi e canali per recuperare i comandi dal botmaster ed eseguirli. L'insieme dei protocolli e topologia di rete che soddisfano questa premessa è chiamata *infrastruttura di comando e controllo*. Nel caso più semplice questa comprende uno o più server e una rete di collegamento che funge da mezzo per trasmettere i dati dalla macchina infetta all'attaccante e viceversa. L'infrastruttura C&C può essere composta da più canali C&C, che sono definiti dai protocolli utilizzati e dalla struttura del messaggio, e da intermediari utilizzati nella comunicazione [2]. L'uso dei canali di comando e controllo è di fondamentale importanza nei recenti attacchi.
La comunicazione viene solitamente avviata dal bot, a cui segue la fase di *rallying* in cui viene cercato il punto di *rendezvous*, ovvero le coordinate a cui contattare il server principale. Chi tenta di mettere fuori uso una rete botnet concentra i suoi sforzi nel bloccare le comunicazioni del C&C come espediente per bloccare l'intera botnet. Le principali tecniche utilizzate a tal fine sono il *takeover*, in cui si prende il controllo dell'intera botnet e il *sinkholing*, in cui si interrompe semplicemente il *Channel*.

Capitolo 3

Tecniche di difesa

Dal punto di vista del botmaster, i difensori rappresentano una minaccia. È quindi fondamentale per lui comprendere appieno il modello di minaccia, in modo da costruire un'infrastruttura C&C che resista a possibili tecniche di difesa.
Prima di analizzare queste possibili contromisure, vale la pena valutare le possibilità che hanno a disposizione i difensori.
La prima possibilità considera l'evenienza che i difensori abbiano un campione della famiglia del malware utilizzato per infettare i bot. È noto che i malware moderni si propagano in modo da evitare di infettare macchine addette al controllo della sicurezza o centri di ricerca, in modo da non essere scoperti per quanto più tempo possibile. Ciononostante, è più prudente per un malintenzionato assumere che i difensori abbiano a disposizione il codice binario del malware.
Il fatto che i difensori possano avere un campione del malware non è ovviamente una minaccia di per sé, ma solleva la questione se il malware possa essere analizzato e quindi possano essere tratte informazioni sul suo funzionamento. In caso di esito positivo, queste informazioni possono essere utilizzate dai difensori per progettare adeguate contromisure.
Gli autori del malware possono utilizzare una moltitudine di

tecniche per rendere l'analisi del binario complessa o addirittura non fattibile. Alcune di queste tecniche impediscono l'analisti statica del codice binario. Queste tecniche vanno dall'azzeramento del codice alla crittografia del codice.
Altre strategie prendono di mira l'analisi dinamica del codice. Queste strategie includono, ad esempio, l'istruzione del malware a non avviarsi quando ci sono condizioni sospette come ad esempio in presenza di una macchina virtuale.
Nonostante tutto ciò, un hacker non ha nessuna garanzia che il malware progettato non venga analizzato e studiato dai difensori; quindi è più sicuro, dal punto di vista dell'attaccante, ipotizzare che il codice venga analizzato.
Da questi presupposti, i difensori possono mettere in atto due tecniche di difesa per interrompere il canale C&C di una botnet: **takeover** e **sinkholing**.

3.1 Takeover

Con il termine takeover ci si riferisce all'atto con il quale un difensore prende il controllo di una intera botnet, escludendo il botmaster dal comando. Questa operazione può essere eseguita in una varietà di modi, e si conclude con il difensore che reindirizza il canale C&C in modo che punti su alcune macchine sotto il suo controllo, invece che sul C&C server che è sotto il controllo del botmaster. In questo caso, i difensori impostano delle macchine che imitano il comportamento dei C&C server e eseguono un dirottamento dell'intero canale

C&C. Quando i bot iniziano a contattare i server sotto il controllo dei difensori, vengono istruiti in modo da comportarsi in maniera innocua, con il fine di disabilitare l'intera botnet.
In letteratura c'è un esempio ben noto di takeover avvenuto con successo [3]. Nel 2009 i ricercatori dell'Università della California a Santa Barbara sono riusciti a prendere il controllo per dieci giorni della botnet Torpig, una rete di circa 180'000 macchine compromesse, utilizzate per raccogliere credenziali private dalle vittime. I ricercatori hanno dirottato il canale C&C e hanno iniziato a comunicare direttamente con le macchine infettate dal malware, imitando il legittimo botmaster. Per dieci giorni hanno incaricato le macchine remote di non eseguire attività dannose e hanno raccolto informazioni per stimare la diffusione del malware Torpig.
Riuscire a prendere il controllo di una intera botnet è sicuramente il risultato più ambizioso che un difensore possa sperare di ottenere. Questo risultato però è condizionato dal modo in cui il malware sia progettato. L'acquisizione di una botnet è possibile solo se il malware che infetta le macchine non utilizza l'autenticazione con il botmaster prima dello scambio dei messaggi. Infatti, in questo modo i bot possono essere ingannati nel credere di essere in contatto con il loro botmaster anche se non lo sono.
Sfortunatamente per i difensori, un'implementazione corretta di un appropriato metodo di autenticazione è sufficiente a garantire che non avvenga nessun dirottamento di canale.

3.2 Sinkholing

Con il termine sinkholing ci si riferisce all'operazione con la quale i difensori impediscono al botmaster di comunicare con le macchine. In questo caso l'obiettivo del difensore è solamente quello di interrompere il canale C&C. Di solito questo avviene con l'utilizzo di una blacklist di indirizzi IP o di domini malevoli che puntano ai C&C server. Il sinkholing non richiede che i difensori imitino il botmaster, ma può essere altrettanto efficace per inibire una botnet. Dal punto di vista dell'attaccante, a differenza del caso del takeover, in questo caso non esiste un modo immediato per affrontare il problema.

3.3 Tecniche di rallying

Una tecnica di rallying è un protocollo utilizzato per stabilire la connessione, cioè il canale C&C, tra botmaster e bot. In una topologia centralizzata è responsabilità dei bot trovare i C&C server. Si descrivono tre tecniche in ordine di complessità e resilienza alle contromisure dei difensori.

Hardcoded IP Address: Il modo più semplice per indicare a un bot come connettersi a un server è quello di inserire l'indirizzo IP all'interno del malware o spedire il malware con un file esterno contenente l'indirizzo IP da contattare. Tuttavia questa tecnica è vulnerabile alla perdita di informazioni: una volta che il difensore riesce a prendere il codice binario del

malware è in grado di eseguirne il *reverse engineering* e conoscere quindi i punti di rendez-vous. Una volta acquisite queste informazioni è possibile impostare una strategia per inibire contemporaneamente tutti i server C&C e rendere l'intera infrastruttura innocua.

Hardcoded Domain Name: Un primo miglioramento consiste nel far leva sul protocollo DNS e spedire il malware con un elenco di domini da interrogare per ottenere l'indirizzo IP del server C&C. In questo modo l'attaccante è libero di spostare il server C&C in posizioni diverse e aggiornare i record DNS per consentire al dominio di puntare al nuovo indirizzo IP. Il risultato è un protocollo di comunicazione più robusto rispetto all'hardcoded IP. I difensori in questo caso devono intervenire a livello DNS e cooperare con le autorità per mandare fuori uso i nomi di dominio malevoli utilizzati dal malware. Tuttavia, si incontrano ancora dei problemi nel caso di perdita di dati: se si riesce a mettere le mani sul software ed eseguirne il reverse-engineering, si possono scoprire i domini in questione e quindi i punti di rendez-vous.

Domain Generation Algorithms: I DGA sono algoritmi utilizzati per generare automaticamente nomi di dominio, i cosiddetti *Automatically Generated Domains* (*AGD*). La maggior parte delle botnet moderne sfrutta questa tecnica nella fase di rallying: sia il botmaster che i bot producono un elenco di domini pseudo casuali, uno dei quali è registrato e funziona come punto di rendez-vous.

Dato che l'analisi del lavoro di tesi verterà principalmente su domini generati da DGA, quest'ultimi si analizzano più nel dettaglio nel paragrafo successivo.

3.4 Domain Generation Algorithms

I *Domain Generation Algorithms* (DGA) sono algoritmi che generano domini automaticamente in modo pseudo casuale. Attualmente sono la tecnica più comune per l'attuazione di meccanismi di rallying efficaci. I bot e il C&C server implementano uno stesso algoritmo per generare ogni giorno un ampio elenco di domini. Questi domini – comunemente denominati AGD (*Automatically Generated Domains*) – sono generati in modo pseudo-casuale partendo da un seme iniziale del tutto imprevedibile (ad es. il topic più di tendenza attualmente su Twitter o il risultato di una specifica ricerca fatta su Google). Soltanto uno o alcuni di questi AGD è effettivamente registrato dal bootmaster e punta al reale indirizzo IP del C&C server. I bot interrogano tutti gli AGD. Se il dominio non è registrato, il server DNS risponde con NXDOMAIN e i bot proseguono a contattare gli altri domini. Non appena il DNS fornisce una risposta valida, il bot contatta quel dominio e si stabilisce il canale C&C.

Il processo è raffigurato in figura 3.1. Il bot tenta di risolvere gli AGD fdsjfklds.info e kjhdsho.biz, senza successo; poi interroga l'AGD xyzkkk.info e ottiene l'indirizzo IP che ospita il C&C server.

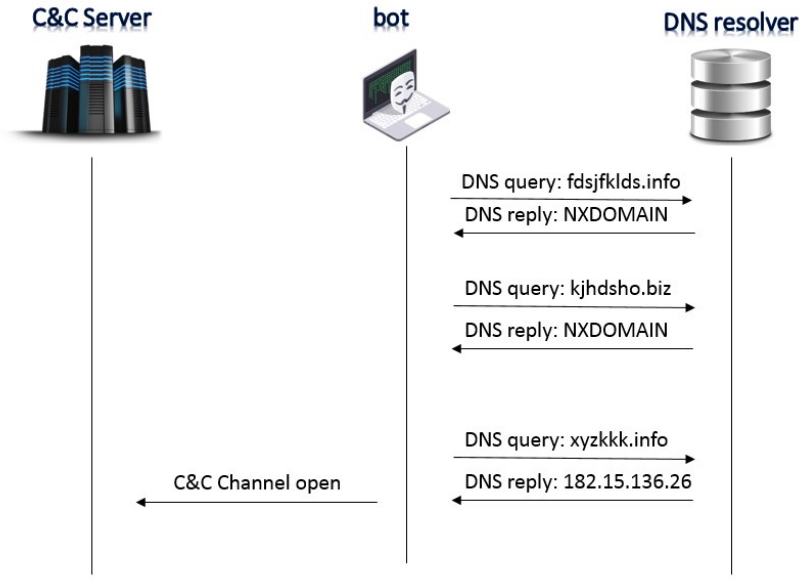

Figura 3.1: Procedura di rallying DGA

Ogni giorno vengono generati diversi AGD e il dominio di rendez-vous varia di continuo.

3.4.1 Analisi della sicurezza

Il meccanismo di rallying basato su DGA, nonostante sia facile da spiegare e da implementare, mostra un certo grado di difficoltà dal punto di vista del difensore, in quanto è molto complicato riuscire a prevedere i nomi di dominio che verranno generati.
In base alla progettazione, il codice binario del malware che implementa le funzionalità DGA perde delle informazioni che risultano insufficienti al difensore per riuscire a prevedere i futuri punti di rendez-vous. Infatti di solito vengono attuate due decisioni progettuali (eventualmente in modo combinato) che rendono robusto il sistema anche se il difensore è in grado di accedere all'intero codice sorgente del malware.

Scelta del seme casuale: I malware basati su DGA utilizzano dati esterni e dipendenti dal tempo (ad es. topic di tendenza su Twitter) per la scelta del seme iniziale dell'algoritmo di generazione di ADG. Un difensore che desideri prevedere il comportamento del bot (ad es. per divulgare i futuri domini di rendez-vous) dovrebbe sapere in anticipo il seme di partenza che verrà utilizzato. Questo ovviamente non è possibile, in quanto richiederebbe la predizione di fenomeni futuri. In generale gli autori di malware riescono a rendere impossibile la ricostruzione di condizioni realistiche di esecuzione del malware. Di conseguenza, rispondere alla domanda "quale sarà il dominio di rendez-vous domani?" è impossibile. Nemmeno l'autore del malware è in grado di rispondere a questa domanda.

Reale punto di rendez-vous: Anche nel caso di un'implementazione senza fonti esterne ed imprevedibili, è ancora impossibile per i difensori prevedere con precisione le coordinate dei futuri punti di rendez-vous. Supponendo di fare reverse engineering del binario di un malware che utilizza DGA, si troverebbero una quantità enorme di possibili domini di rendez-vous. Ad esempio, il DGA *Conficker.C* genera circa 50'000 AGD al giorno. È sufficiente che il botmaster registri solo uno di questi domini per fare in modo che i bot si connettano al C&C server, mentre i difensori, non sapendo quale sarà il dominio effettivamente registrato, dovrebbero registrarli tutti con le relative tariffe che ne conseguono.
Riassumendo, le informazioni che si riescono ad ottenere dall'analisi del binario del malware sono insufficienti per progettare qualsiasi contromisura corretta ed efficace.

3.4.2 Strategie di migrazione

Bot e botmaster si evolvono con dinamiche identiche ma disaccoppiate: eseguono entrambi lo stesso codice dipendente dal tempo e possono sincronizzarsi senza scambiarsi messaggi. Ogni giorno, infatti, bot e botmaster conoscono i domini di rendez-vous aggiornati. Il botmaster può procedere registrandone alcuni e indirizzandoli al C&C server, mentre il bot tenta di risolverli tramite query DNS. Questa procedura può essere eseguita giorno per giorno e non si prevede che le comunicazioni tra bot e C&C server vengano stabilite ogni giorno. In realtà il bot resetta quotidianamente il suo stato

operativo e non si trova mai nella condizione di non sapere come contattare il suo botmaster in futuro. Non può mai accadere che perda la comunicazione con il suo C&C server, indipendentemente da quanti giorni la comunicazione tra i due fallisca.
Dal punto di vista del botmaster, le DGA consentono un'estrema flessibilità di comunicazione. Infatti, il botmaster può spostare il suo C&C server semplicemente cambiando il mapping delle risoluzioni DNS degli AGD. In questo modo, il botmaster può migrare i suoi server eludendo eventuali blacklist di indirizzi IP create dal difensore.
In una rete basata su DGA, non vi è alcuna operazione che i difensori posso attuare per disconnettere in modo permanente un botmaster dai suoi bot.

3.4.3 Debolezze

È interessante notare che, oltre ai citati punti di forza dei malware che utilizzano DGA, esistono due importanti debolezze che possono essere sfruttate dal difensore.

Nomi di dominio random: I domini generati in modo casuale appaiono come stringhe ad alta entropia, come *sdkeeffrertj666f8e88s12las.com*, *djidsaosisoosiijsjdn399udsap.info* ecc. Invece i nomi di dominio legittimi sono pensati per essere facilmente ricordabili e pronunciabili (es. *ansa.it*, *yahoo.it*). Questi domini sono chiamati HGD (*Human Generated Domain*). Il fatto che gli AGD siano distinguibili dagli HGD è una

conseguenza del fatto che gli AGD sono creati in maniera casuale.
Gli AGD presentano una entropia elevata per due motivi:

- sono generati in maniera casuale e quindi è complicato riuscire a generare domini che siano facilmente "leggibili" in quanto sono generati da algoritmi e non sono prevedibili nemmeno dallo stesso creatore del malware;

- avendo una entropia elevata, è più probabile che i domini non siano già registrati e quindi sia possibile indirizzarli al C&C server.

Traffico NXD (*Non-Existent Domain*): I bot eseguono una quantità spropositata di query DNS, che si tramutano in risposte NXD. Ciò accade perché la maggior parte delle query DNS contatta domini non registrati. D'altra parte, gli host legittimi non hanno motivi per generare elevati volumi di query che producono a loro volta risposte NXD.
Il fatto che i DGA generino un traffico DNS così rumoroso può far scattare allarmi e rendere identificabili le macchine infette da malware. Tuttavia, la progettazione di un DGA che non generi grandi quantità di risposte NXD è impossibile. Infatti la generazione di traffico DNS rumoroso è una diretta conseguenza dell'ignoranza del malware stesso: per non generare tale traffico NXD, il malware dovrebbe conoscere a priori il dominio di rendez-vous registrato. Ma se lo facesse, facendo reverse engineering, il difensore potrebbe essere in grado di

ottenere tale informazione, venendo meno ad un punto fondamentale sul quale si basa la robustezza dei DGA.

3.5 Rilevamento di una botnet

A causa del costante aumento di attività malevole, negli ultimi anni il rilevamento e il tracciamento di botnet sono stati un interessante argomento di ricerca. Sono state proposte diverse tecniche per rilevare una botnet

3.5.1 Metodi basati su honeynet

Generalmente il metodo basato su honeynet è costituito da honeypot e honeywall [4]. Un honeypot è un sistema creato appositamente con molte vulnerabilità e senza nessun tipo di protezione. Questo viene compromesso di solito in un lasso di tempo molto breve. Per honeywall si intende il software che viene utilizzato per monitorare, raccogliere, controllare e modificare il traffico attraverso l'honeypot (ad es. Snort). Gli honeypot sono sistemi riempiti di informazioni create ad arte per apparire preziose ma alle quali un legittimo utente non accede mai. Qualsiasi accesso ad un honeypot è sospetto. Monitorando le connessioni in entrata e in uscita da un honeypot è possibile identificare altre macchine compromesse appartenenti alla stessa rete, stimare la grandezza della botnet stessa e studiarne il comportamento in un ambiente controllato.

L'utilizzo di un honeypot può tuttavia essere riconosciuto dall'attaccante che può a sua volta modificare il comportamento del bot per evitarne il riconoscimento.

3.5.2 Monitoraggio passivo del traffico di rete

Un altro metodo per rilevare una botnet consiste nel monitorare passivamente il traffico di rete generato da pc infetti. Dato che i dati all'interno del reale traffico di rete sono di vario tipo, possono essere attuate diverse tecniche basate sulla tipologia del traffico catturato. Queste tecniche possono essere raggruppate in quattro diverse tipologie: tecniche basate su caratteristiche distintive della botnet, su anomalie riscontrate nel traffico, attraverso l'analisi del traffico DNS o l'analisi basata su data-mining.

Analisi basata su caratteristiche distintive. Questa prima tipologia si basa sulla conoscenza di botnet rilevate in precedenza. La scoperta di una botnet implica infatti lo studio del suo codice alla ricerca di caratteristiche distintive, con lo scopo di produrre una sorta di firma da utilizzare in un *Intrusion Detection System* (IDS). Per "firma" si intende una raccolta di informazioni, come un indirizzo URL o una sequenza di bit, da poter confrontare in futuro per rilevare un nuovo tentativo di intrusione. Gli IDS prevedono un database in cui sono memorizzate tutte le violazioni conosciute con le loro firme caratteristiche, e inviano un messaggio di allerta ogni volta che si trovano delle corrispondenze. Il limite di questa

tecnica è che può identificare solo botnet precedentemente conosciute ed analizzate, e senza alcuna modifica.

Analisi basata su anomalie nel traffico di rete. La presenza di una botnet può indurre la rete ad avere comportamenti insoliti e non previsti. Ad esempio, gli attacchi di tipo DDoS come nel caso di Mirai generano un gran volume di traffico e aumentano la latenza della rete. Questo tipo di attacchi, insieme ad altre anomalie come l'utilizzo di porte insolite nel livello transport o un numero elevato di connessioni, possono essere utilizzati per il rilevamento di reti compromesse.

Analisi basata sullo studio del DNS. Il botmaster sfrutta la flessibilità del DNS per controllare i bot e allo stesso tempo occultarne l'identità. Per questi motivi vengono effettuate query DNS per tutta la durata della vita della botnet e possono quindi essere utilizzate per distinguere le query effettuate dai bot da quelle regolarmente effettuate dagli utenti della rete. Per questo tipo di analisi si sfruttano i *passive DNS* (*pDNS*) che hanno il compito di mantenere lo storico (log) delle query fatte al resolver DNS.

Analisi basata su data-mining. Identificare la comunicazione della botnet col C&C server è un modo efficace per rilevare attività malevole, ma non è facile separare il traffico sospetto da quello regolarmente effettuato dagli utenti della rete. Inoltre, le botnet sono costantemente aggiornate, il che può ridurre l'efficienza dei sistemi di rilevamento basati su firme e anomalie.

I ricercatori hanno quindi pensato di applicare tecniche di data-mining al traffico di rete per rilevare le comunicazioni di potenziali botnet. Tali tecniche comprendono *clustering*, *classificazione*, *regressione*, *pattern recognition* e altri algoritmi di *machine learning*.

Capitolo 4

Tecniche di difesa basate su sandbox

Il lavoro di tesi svolto si è articolato in due fasi principali.
Il primo *step* è stato svolto presso l'azienda Namirial di Senigallia, mentre la seconda parte è stata svolta presso il DII: Dipartimento di Ingegneria dell'Informazione dell'Università Politecnica delle Marche.
Durante il tirocinio ci si è prefissati l'obiettivo di catturare e analizzare del reale traffico di rete aziendale. Per individuare questo traffico si è pensato di utilizzare la *Joe Sandbox Complete* realizzata dall'azienda Joe Security LLC [5].
È stato necessario installare e configurare l'intero laboratorio di Joe Sandbox.
Su un server Ubuntu è stato installato sia il software Joe Sandbox Complete e sia il sistema FOG che permette di creare uno snapshot di ripristino del sistema sul quale verranno eseguiti i malware.
I file inviati dalla Sandbox sono stati eseguiti su una macchina fisica con sistema operativo Windows 7. Si è scelto di utilizzare una macchina fisica in quanto alcuni malware sono in grado di rilevare condizioni sospette quando questi vengono eseguiti in una macchina virtuale e mascherano i loro comportamenti.

Sono state fornite circa 9000 mail da analizzare e per automatizzare il processo di somministrazione dei file alla Sandbox è stato creato uno script in Python. [Appendice A]

4.1 Joe Sandbox

Joe Sandbox è un sistema di analisi di malware automatico che utilizza tecniche di analisi statiche e dinamiche. È capace di eseguire campioni (file binari, documenti, URL, ecc) in una macchina virtuale o in una macchina fisica che dispone di un sistema operativo installato ed è in grado di acquisire informazioni sui processi chiamati dal campione analizzato. Le informazioni acquisite sono raccolte in report di analisi disponibili in molti formati: HTML, PDF, XML, PCAP, ecc.
Come detto, la Sandbox è in grado di eseguire sia un'analisi statica che un'analisi dinamica del file somministrato. Durante l'analisi dinamica il file viene eseguito e viene monitorato tutto il suo comportamento, includendo ad esempio tutti i processi che vengono chiamati o le query DNS effettuate. L'analisi statica non esegue il file bensì esamina il codice sorgente del file e ne evidenzia eventuali anomalie che possono ricondurre a degli standard di un ipotetico file malevolo.
Entrambe le analisi presentano vantaggi e svantaggi; ad esempio l'analisi statica è limitata in quanto molti malware non sono identificabili dalla sola lettura del codice dato che spesso i dati possono essere offuscati all'interno del codice stesso. Tuttavia, una volta che un campione è stato decom-

presso, grazie all'analisi statica, è possibile valutare il completo funzionamento del programma, comprese quelle parti che non sono state eseguite a causa di alcune condizioni e/o restrizioni.
L'analisi dinamica, al contrario, riesce a monitorare solo la parte di codice che viene effettivamente eseguita.
Grazie alla combinazione dell'analisi statica e dinamica, Joe Sandbox combina i vantaggi di entrambe le analisi.
Joe Sandbox esegue i campioni in un ambiente isolato (la Sandbox). È importante avviare i malware in un ambiente isolato per evitare che l'infezione indotta volontariamente nella Sandbox si propaghi in altri sistemi. Su Joe Sandbox l'ambiente di analisi può essere una macchina virtuale o una macchina fisica con un completo sistema operativo installato.
È fondamentale avviare l'analisi dei malware su un sistema "clean" e noto, e cioè che non sia infetto ed inoltre sia in uno stato che conosciamo.

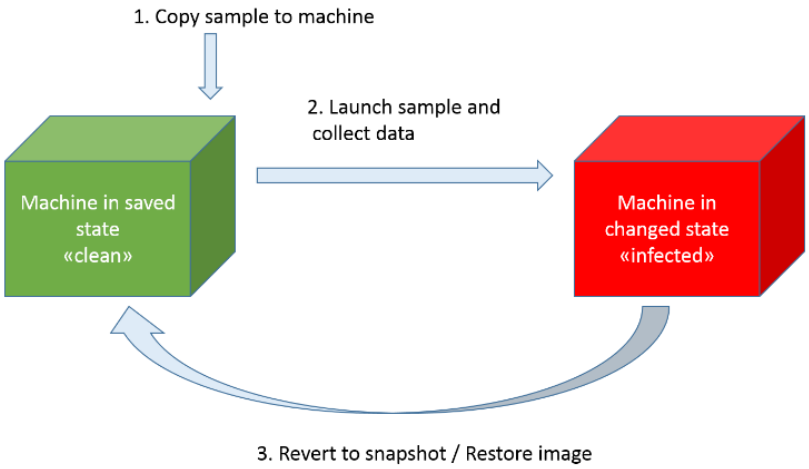

Figura 4.1: Funzionamento Joe Sandbox

In questo modo siamo sicuri del fatto che i report generati saranno frutto dell'analisi del campione inviato ed inoltre una volta finita l'analisi il sistema può essere ripristinato allo stato clean. A questo scopo viene utilizzato FOG.
FOG, "*a free computer cloning solution*", è un prodotto basato su Linux che consente di creare una copia dell'installazione del sistema operativo. FOG non implica l'impiego di alcun disco di boot dal momento che ogni operazione è da esso gestita facendo ricorso al protocollo TFTP "*Trivial File Transfer Protocol*" ed a PXE "*Preboot Execution Environment*".
PXE è una particolare metodologia che consente di effettuare l'avvio di un personal computer utilizzando le informazioni che provengono dall'interfaccia di rete ethernet, con il supporto di un server "ad hoc". FOG si propone come il server in grado di trasmettere ai sistemi connessi in LAN, i dati da utilizzare per il boot. Senza ricorrere ad alcun supporto di memorizzazione di massa, quindi, è possibile inviare – da un sistema Linux "equipaggiato" con il software FOG – le istruzioni per effettuare, ad esempio, la copia del contenuto di un'unità disco o il ripristino della stessa.
La Sandbox nella Joe Sandbox – chiamata *Analysis Machine* – è isolata, ma ovviamente deve comunicare con la macchina, ad esempio per memorizzare il campione per l'esecuzione, per acquisire i risultati di analisi, ecc. Tutto ciò è fatto sul server sul quale è installato Joe Sandbox. Il nucleo di Joe Sandbox si chiama Joeboxserver e comunica con la sandobx. Il modo con il quale Joeboxserver comunica con la sandbox dipende dal sistema operativo.

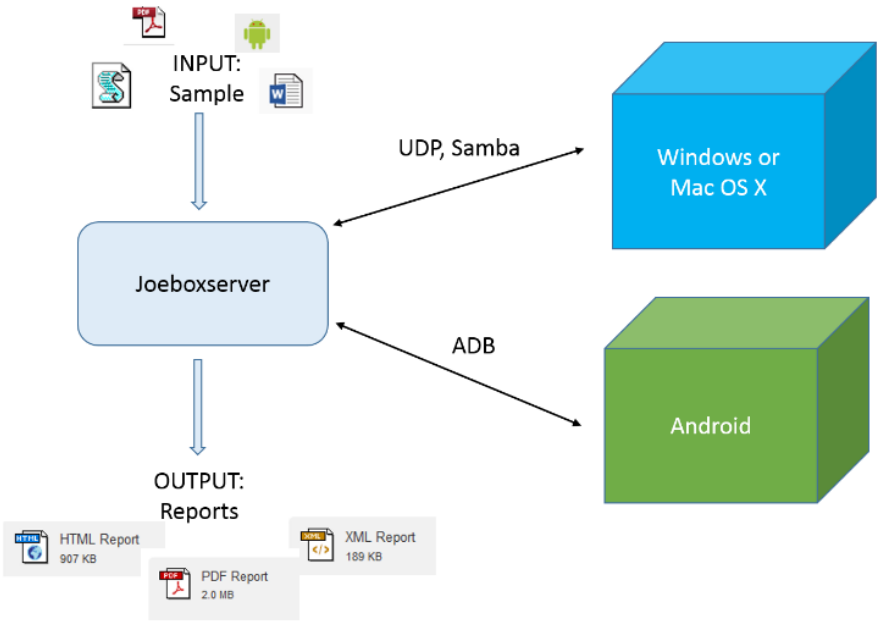

Figura 4.2: Protocolli di comunicazione Joeboxserver

Per sandbox Windows o MacOS, è usato il protocollo UDP per la comunicazione e il protocollo Samba per la condivisione di file.

4.2 Installazione del laboratorio Joe Sandbox

Joeboxserver è stato installato su un server Ubuntu 18.04 LTS.
Sono stati aggiunti i repository universe e multiverse:

```
sudo add-apt-repository multiverse
sudo add-apt-repository universe
```

Successivamente sono stati installati i software richiesti per il corretto funzionamento del software:

Mono:

```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv-keys 3FA7E0328081BFF6A14DA29AA6A19B38D3D831EF
echo "deb https://download.mono-project.com/repo/ubuntu stable-bionic/snapshots/5.14.0 main" | sudo tee /etc/apt/sources.list.d/mono.list
sudo apt update (ignore any warnings)
sudo apt install mono-complete
```

Yara: [6]

```
tar -zxf yara-3.8.1.tar.gz
cd yara-3.8.1
./bootstrap.sh
sudo apt-get install automake libtool make gcc
./configure
make
sudo make install
sudo sh -c 'echo "/usr/local/lib" >> /etc/ld.so.conf'
sudo ldconfig
```

Unrar:

```
sudo apt-get install unrar
```

4.3 Configurazione di rete

Una volta installato il Sistema operativo, si sono configurate tutte le interfacce di rete.
In figura si riporta uno schema di come le varie interfacce devono comunicare per l'utilizzo corretto di una sandbox installata su macchina fisica.

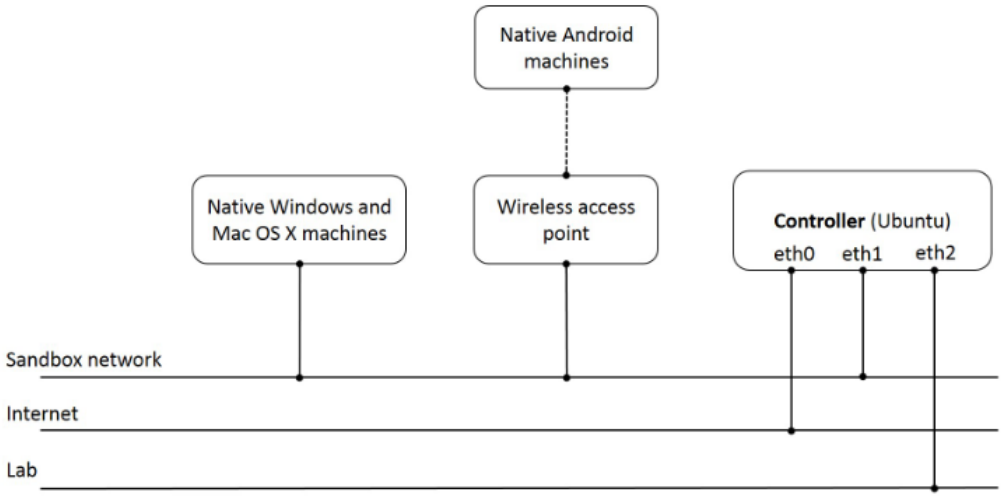

Figura 4.3: Schema di rete Sandbox

Sono presenti tre interfacce di rete:
eth0 viene configurata con un IP statico ed è la linea che è connessa ad internet.
eth1 è la connessione alla sandobx nativa
eth2 è l'interfaccia di rete di laboratorio, usata per somministrare campioni e ricevere risultati.
Le varie configurazioni vengono effettuate nel file `/etc/netplan/50-cloud-netcfg.yaml`

4.4 Installazione FOG

FOG, "*a free computer cloning solution*", è un prodotto basato su Linux che consente di creare una copia speculare delle installazioni di Windows presenti sugli altri computer connessi alla rete locale.
FOG non implica l'impiego di alcun disco di boot dal momento che ogni operazione è da esso gestita facendo ricorso al protocollo TFTP "Trivial File Transfer Protocol" ed a PXE "Preboot Execution Environment". PXE è una particolare metodologia che consente di effettuare l'avvio di un personal computer utilizzando le informazioni che provengono dall'interfaccia di rete ethernet, con il supporto di un server "ad hoc".
L'installazione di FOG viene effettuata sul server Ubuntu tramite i comandi:

```
wget  https://github.com/FOGProject/fogproject/archive/working.zip
-O /tmp/working.zip
cd /tmp
sudo apt install unzip
unzip working.zip
cd fogproject-working/bin/
sudo ./installfog.sh
```

Durante l'installazione sono stati settati tutti i parametri necessari per il corretto funzionamento del sistema come ad esempio l'IP address utilizzato per entrare nel pannello di amministrazione FOG, l'interfaccia di rete della sandbox ecc.

4.5 Configurazione Sandbox

Si è utilizzata una Sandbox nativa con sistema operativo Windows 7 a 64 bit.
Dopo aver attivato .NET Framework 3.5, si è passati all'installazione dell'ultima versione di *AutoIt* che è uno script molto leggero in grado di simulare il comportamento di un utente al pc: movimento del mouse, click su finestre ecc. Inoltre per permettere alla sandbox di eseguire correttamente i file sottomessi sono stati installati i programmi più diffusi come: Microsoft Office, Acrobat Reader, Java, Chrome ecc.
È stato abilitato il login automatico al sistema disabilitando ogni tipo di password di accesso.
Nelle configurazioni di rete della sandbox è stato inserito l'IP impostato nella eth1 sul server Ubuntu e come gateway l'IP dell'interfaccia eth0 connessa ad internet.
Infine nel BIOS è stato impostato PXE come boot primario.

4.6 Installazione Joeboxserver

Una volta scaricato il pacchetto di installazione di joeboxserver ed estratto con la password fornita in fase d'acquisto del prodotto, bisogna configurare il file di configurazione

```
package/joesandbox/server/joeboxserver.exe.config
```

In questo file è necessario impostare:
Id della macchina alla quale Joeboxserver deve connettersi: `W7`

SystemRestorer è il file dal quale andare a reperire le informazioni di ripristino del sistema: `fogc.dll`
L'*IP* della sandbox
Il *MAC address* della sandbox
Dopo aver configurato correttamente tutto il sistema, si può installare Joeboxserver

```
sudo mono joeboxserver.exe –install
```

specificando durante l'installazione che la sandbox è una macchina fisica.
Ad installazione completata, lanciando il comando di configurazione

```
sudo mono joeboxserver.exe –configure
```

verranno create le connessioni Samba tra sandbox e Joeboxserver ed inoltre verranno create anche le voci iptables che costituiscono il ponte tra la macchina sandbox e internet.
L'ultima modifica da fare alla sandbox prima di creare lo *snapshot* con FOG è inserire nella lista startup il file `jbxinit.au3` per l'esecuzione automatica dei campioni sottomessi dal Joeboxserver.
A questo punto collegandosi al pannello di amministrazione FOG è possibile creare uno snapshot della sandbox Windows attuale. Per far questo è necessario prima creare un'immagine del sistema e poi creare un task automatico che vada a impostare il ripristino della sandbox con l'immagine catturata.

Ora che tutto il sistema è stato installato e configurato correttamente è possibile somministrare i campioni alla sandbox ed ottenere i report di analisi.

4.7 Script Python

Sono state fornite circa 9000 mail PEC filtrate dal sistema di spam automatico dell'azienda Namirial.
Per poter automatizzare il processo di somministrazione dei campioni è stato creato uno script in Python avvalendosi della *RESTful Web API v2* rilasciata da JoeSandbox.
La sandbox è in grado di estrarre autonomamente gli allegati e gli URL presenti all'interno della mail e analizzarli separatamente uno alla volta. Per la scrittura dello script è stato necessario, quindi, tener presente che l'invio di un file potesse comportare la somministrazione di più campioni alla sandbox.
Quando una mail viene sottoposta all'analisi, questa viene spostata in un'altra directory dallo script. Questo è stato fatto per avere un quadro completo delle mail che sono state effettivamente analizzate e quali invece sono ancora da analizzare.
Ogni report può arrivare a pesare anche qualche centinaia di MB, ed è per questo che lo script è stato progettato in modo da tenere in memoria soltanto i report malevoli.
Quando un campione viene sottomesso all'analisi, lo stato di quel campione passa in "*running*". Se viene sottomesso un secondo campione, questo sarà nello stato "*submitted*" e passerà in "*running*" quando lo stato del campione precedente passerà in "*finished*".

Lo script analizza lo stato dell'ultimo campione sottomesso alla sandbox; se questo è finished o running sottomette un ulteriore mail, altrimenti attende.
Quando la directory contenente le mail da analizzare viene completamente svuotata, e quindi quando sono state analizzate tutte le mail, lo script rimane attivo e nel momento in cui vengono aggiunte altre mail nella directory queste saranno mandate alla sandbox per essere analizzate.
Lo script è stato lanciato in background nel server Ubuntu col comando

```
nohup Script.py &
```

ed è sempre in esecuzione.
Tutte le operazioni effettuate dallo script vengono salvate in un LOG monitorabile in tempo reale.
L'intero script è riportato nelle appendici. [Appendice A]

4.8 Report Joe Sandbox

Dopo aver completato un'analisi, la sandbox è in grado di classificare un campione come *clean*, *suspicious* o *malicious*.
Viene generato un report disponibile in molti formati: HTML, PDF, XML, PCAP, ecc.
I report generati sono molto dettagliati e tra i dati che è possibile estrapolare ce ne sono alcuni che meritano una particolare attenzione. Ad esempio è possibile osservare che in base

alla tipologia dei processi e delle chiamate eseguite dal sistema, la sandbox crea un “ragno” che va a classificare il tipo di malware presente nel file.

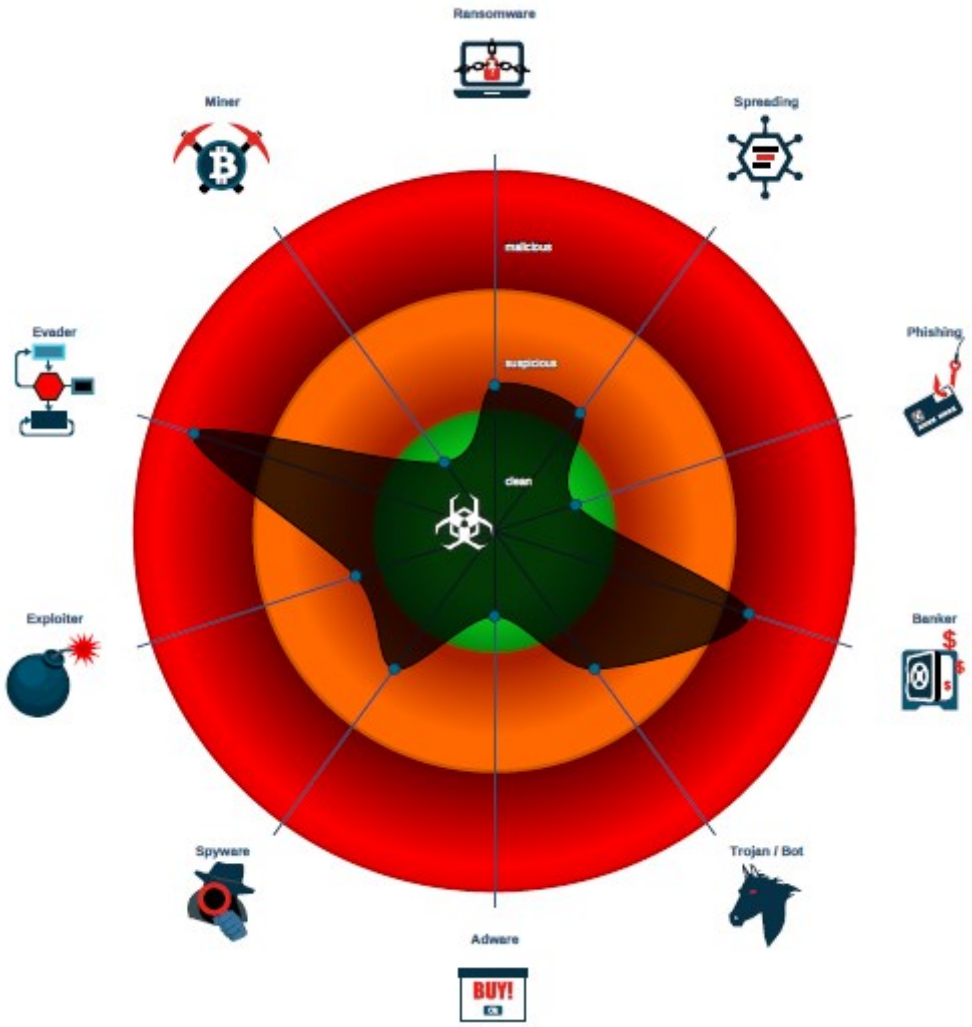

Figura 4.4: Ragno di attacco alle PEC Telecom

Inoltre sono resi palesi i domini e gli IP contattati, la loro localizzazione geografica e come questi sono stati classificati

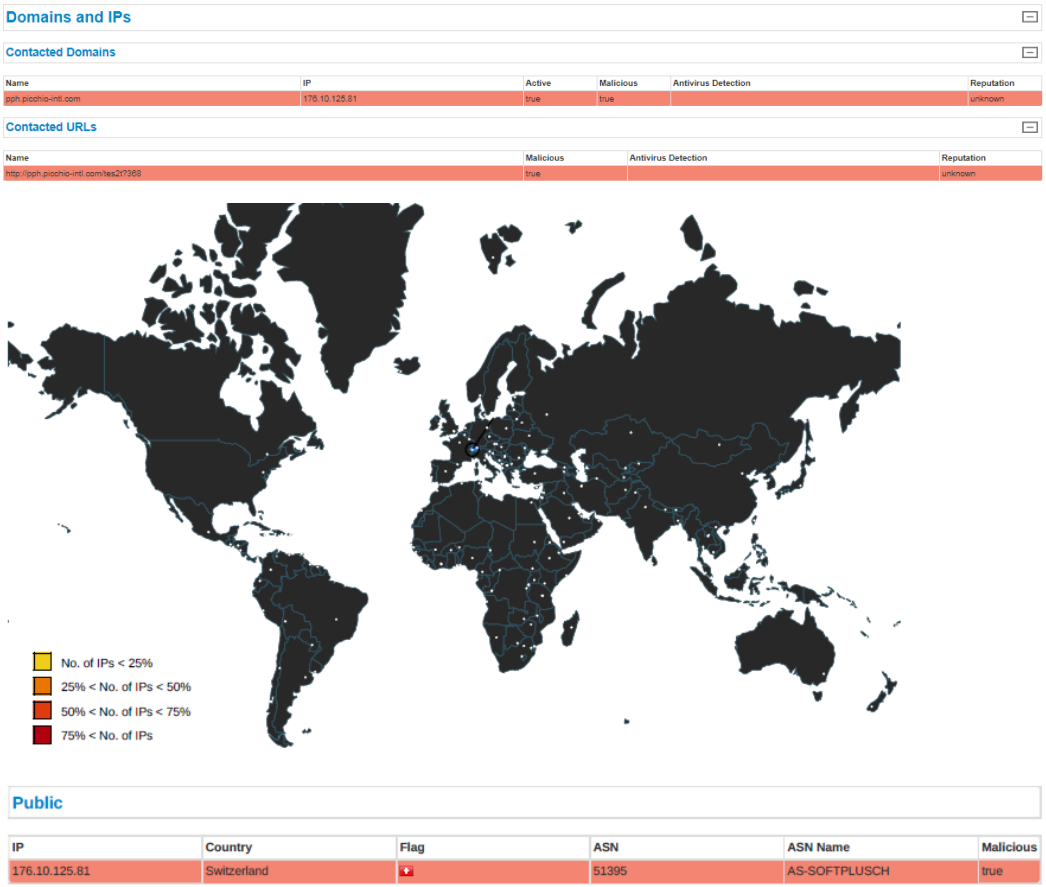

Figura 4.5: Localizzazione geografica attacco PEC Telecom

Gli screen in figura 4.4 e 4.5 sono presi dal report generato dall'analisi dell'attacco alle PEC Telecom avvenuto il 12 novembre 2018. Come si vede dalla figura, per sferrare l'attacco il malware ha contattato il dominio *pph.picchio.intl.com* con IP 176.10.125.81.
L'analisi delle 9'000 mail è durata circa due mesi.

Capitolo 5

Tecniche di difesa basate su analisi di traffico

Come detto precedentemente, la seconda parte del lavoro di tesi è stato svolto presso il dipartimento di Ingegneria dell'Informazione dell'Università Politecnica delle Marche.
L'obiettivo della tesi è quello di analizzare traffico di rete proveniente da computer infetti da malware di tipo botnet che utilizzano DGA. Tra le 9000 mail analizzate presso l'azienda Namirial non è stato possibile arrivare ad un campione consistente di traffico di rete di tipo botnet. Per questo motivo si è pensato di utilizzare traffico di rete di pc infetti reperibile online.

5.1 Machine Learning

Il *machine learning* è una branca dell'intelligenza artificiale e si occupa dello sviluppo di algoritmi di autoapprendimento per acquisire conoscenze partendo dai dati, al fine di creare modelli e fare previsioni. Partendo da alcuni dati, è possibile creare un modello e utilizzarlo per fare previsioni di nuovi dati.

Il machine learning è necessario in tutti quei casi dove è difficile prevedere e programmare a mano tutte le possibili varianti di un problema di classificazione / previsione.

5.1.1 Applicazioni

Le tecniche di machine learning sono applicate in tutti i casi in cui è necessaria un'analisi approfondita (*data mining*), a partire da una grande quantità di dati (*big data*).
Gli algoritmi si dividono in tre grandi gruppi di apprendimento, ognuno dei quali è adatto a studiare un dataset definito in un certo modo: *apprendimento supervisionato*, *apprendimento non supervisionato* e *apprendimento per rinforzo*.

5.1.2 Apprendimento supervisionato

L'obiettivo principale dell'apprendimento supervisionato è quello di elaborare delle regole a partire dalla somministrazione di campioni etichettati. Un'attività di apprendimento supervisionato con le label che fanno parte di classi discrete, viene chiamata *classificazione*. Un'altra sottocategoria di apprendimento supervisionato è la *regressione*, in cui il segnale risultante è un valore continuo.

5.1.3 Apprendimento non supervisionato

Nell'*apprendimento supervisionato* conosciamo in anticipo la risposta giusta quando formiamo il nostro modello e, nell'*apprendimento di rinforzo*, definiamo una misura di rinforzo per le azioni specifiche dell'algoritmo.
Nell'*apprendimento non supervisionato*, tuttavia, abbiamo a che fare con dati senza etichetta o dati di struttura sconosciuta. Usando tecniche di apprendimento non supervisionate, siamo in grado di esplorare la struttura dei nostri dati per estrarre informazioni significative senza la guida di una variabile di risultato nota o di una funzione di rinforzo. Tra le tecniche di apprendimento non supervisionato, la più diffusa è il *clustering*.
Il clustering è una tecnica di analisi dei dati esplorativa che ci consente di organizzare un insieme di informazioni in sottogruppi significativi (*cluster*) senza avere alcuna conoscenza preliminare delle appartenenze ai gruppi. Ogni cluster che può sorgere durante l'analisi definisce un gruppo di oggetti che condividono un certo grado di somiglianza ma che hanno poca similitudine con oggetti di altri cluster, motivo per cui il clustering viene talvolta chiamato anche "classificazione non supervisionata".

5.1.4 Metriche di valutazione

La metrica di valutazione più importante per valutare la bontà dell'algoritmo è la matrice di confusione (*Confusion Matrix*). È una tabella che racchiude la distribuzione sia delle classi assegnate inizialmente che di quelle predette dal classificatore, evidenziando sulla diagonale secondaria la ripartizione degli errori. Le righe rappresentano le etichette iniziali mentre le colonne riportano le etichette predette.

		Predicted	
		Malicious	***Benign***
Actual	***Malicious***	True Positive **(TP)**	False Negative **(FN)**
	Benign	False Positive **(FP)**	True Negative **(TN)**

Tabella 5.1: Matrice di confusione

Partendo dai risultati ottenuti dalla matrice di confusione, possono essere calcolate le metriche utilizzate per la valutazione del modello:

- *Precision*: la precision corrisponde alla capacità di un classificatore di non etichettare un campione negativo come positivo. Quindi, maggiore è la precisione, minore è il numero di falsi positivi.

$$P = \frac{TP}{TP + FP}$$

- *Recall*: La Recall (o *True Positive Rate*) è la capacità del classificatore di trovare tutti i campioni positivi. Quindi, maggiore è la recall, minore è il numero di falsi negativi.

$$R = \frac{TP}{TP + FN}$$

- *F1-score*: media armonica tra precision e recall.

$$F1 - score = \frac{2 \cdot P \cdot R}{P + R}$$

5.2 Stratosphere Lab

Come fonte di dati di traffico di rete è stato utilizzato il database messo a disposizione dal *Stratosphere Research Laboratory* [9]. Il progetto Stratosphere IPS (*Intrusion Prevention System*) è nato nell'Università di Praga come parte del lavoro del dottorato di ricerca di Sebastian García. Attualmente il progetto è finanziato dalla fondazione olandese *NLnet*.
La finalità del progetto è stata quella di creare un software gratuito che, tramite l'utilizzo di algoritmi di machine learning, fosse in grado di rilevare e prevenire specifici attacchi informatici. Nel dettaglio sono stati presi in considerazione attacchi di tipo botnet. La base di partenza del progetto sono i file di cattura di reale traffico di rete "*.pcap"*. Sono stati messi a disposizione archivi di file *.pcap* generati su macchine appositamente infettate da malware di tipo botnet.

Inoltre è stato reso disponibile anche traffico di rete di normale utilizzo di pc non infetti, utilizzato per il test degli algoritmi.
Da questi due archivi messi a disposizione sono state estrapolate le informazioni da somministrare al nostro classificatore.

5.3 Creazione delle tabelle delle query DNS

In questo lavoro di tesi si è focalizzata l'attenzione sui malware di tipo botnet che fanno uso di DGA. L'obiettivo è quello di creare una tabella con estensione *.csv* contenente le informazioni di maggiore interesse per ogni singola richiesta di domino effettuata dal pc infetto. Per automatizzare tutti i processi è stato utilizzato il linguaggio di programmazione *Python 3.7* con l'ausilio dell'ambiente *JetBrains PyCharm 2018.2.3*.
Dato che una singola tabella avrebbe avuto dimensioni ingestibili, è stata creata una tabella per ogni file *.pcap*.
Ogni tabella è formata dai seguenti campi:

- *Filename:* nome del file pcap
- *Domain name:* dominio contattato dalla query DNS
- *Malware:* nome del malware
- *DGA:* nome del DGA
- *Timestamp(sec):* tempo relativo in cui è stata effettuata la query DNS (espresso in secondi)
- *Total time(sec):* tempo totale di osservazione (espresso in secondi)

- *Alexa(YES/NO):* indica se l'indirizzo è presente o meno nella whitelist di Alexa

Una volta selezionati e scaricati i file *.pcap* provenienti da traffico di rete generato da bot infettati da malware DGA si è passati all'estrazione delle richieste di dominio (*query*) DNS. L'estrazione delle query DNS è stata effettuata sfruttando l'uso di Wireshark da terminale: *tshark*.
Le query DNS sono state memorizzate all'interno di file di testo ed eliminando i duplicati sono poi state poi confrontate con la whitelist Alexa. [Appendici B – C – D]
In un ulteriore file di testo sono stati riportati solamente i nomi di dominio che non hanno trovato corrispondenze nel confronto con la whitelist. [Appendice E]
La costruzione di queste liste è stata indispensabile per la creazione delle tabelle finali di nostro interesse in quanto si è reso il codice molto più snello. Infatti per riempire il campo *Alexa(YES/NO)* non è stato necessario confrontare ogni nome di dominio con ogni singolo nome di dominio della *whitelist* Alexa (1 Milione di domini) ma è stato sufficiente confrontare ogni nome di dominio con i nomi di dominio presenti nei file creati in precedenza (qualche decina o al massimo centinaia di domini).
Per associare il nome del malware e il nome del DGA al file in analisi si è creato un file dizionario costituito in questo modo: **nome_file, *nome_malware, *nome_DGA*
Infine, sempre con l'ausilio di *tshark*, è stato estratto il tempo di chiamata per ogni singola query DNS.

Una volta raccolti tutti i dati si sono potute popolare le tabelle contenenti tutte le query DNS e i relativi dettagli.
Sono stati analizzati 13 file contenenti traffico di rete generato da DGA. In tabella 5.2 si riporta un resoconto delle query estrapolate.

File n.	DGA	Tot. Time	Query DNS	Matching Alexa
#1	Zeus / Zbot / ZeuS	96 min	2'674	10
#2	Zeus / Zbot / ZeuS	36 ore	94'117	197
#3	Zeus / Zbot / ZeuS	30 giorni	797'007	4'481
#4	Zeus / Zbot / ZeuS	42 giorni	6'856	45
#5	Papras / Ursnif / Gozi / Snifula	30 ore	12'412	81
#6	sconosciuto	6.5 giorni	3'252	919
#7	sconosciuto	10 giorni	7'169	4'324
#8	Shiz / Shifu	11 ore	85'799	16
#9	Tinba	13 ore	218'960	5
#10	Tinba	19 ore	229'672	5
#11	Tinba	17 ore	224'943	9
#12	Tinba	2.5 giorni	677'772	8
#13	Locky	11 minuti	110	2

Tabella 5.2: Resoconto *malicious* query DNS

Allo stesso modo, prendendo i file *.pcap* del traffico di rete del normale utilizzo di pc non infetti, sono state create le tabelle contenenti le query DNS di traffico non malevolo. Sono stati analizzati 17 file e di ognuno se ne riporta il dettaglio in tabella 5.3.

File n.	Tot. Time	Query DNS	Matching Alexa
#1	115 minuti	3'025	2'678
#2	4.5 ore	848	600
#3	22 minuti	1'074	64
#4	22 minuti	1'074	64
#5	190 minuti	15'075	12'746
#6	19 minuti	8'647	7'308
#7	88 minuti	52'335	45'887
#8	133 minuti	12'341	10'403
#9	4.5 ore	5'608	4'701
#10	89 minuti	6'267	5'702
#11	6.5 ore	13'229	12'030
#12	4.5 ore	26'461	21'638
#13	49 minuti	8'340	6'499
#14	118 minuti	21'097	16'424
#15	4 ore	34'663	27'692
#16	226 minuti	40'122	31'978
#17	4 ore	54'587	42'749

Tabella 5.3: Resoconto *normal* query DNS

Capitolo 6

Risultati sperimentali

Nel paragrafo 3.4 è stato descritto in che modo i botmaster fanno uso dei DNS per gestire le botnet che utilizzano DGA. È possibile utilizzare questa conoscenza per ideare una serie di *features* che accomunano i domini generati da DGA. Questo insieme di caratteristiche può essere utilizzato per creare un classificatore per differenziare i nomi di dominio associati ad attività malevole, di seguito indicati come *malevoli,* e i restanti nomi di dominio, indicati come *benevoli*.

6.1 Features lessicali

Lo studio delle caratteristiche lessicali sulle quali poter basare lo studio per confrontare i nomi di dominio è stato oggetto del lavoro di tesi del collega Rossi *Malware identification through network traffic analysis and classification of lexicographic features* [7].
Dall'analisi dei nomi di dominio dei pDNS si sono generate 17 features. Come è stato detto in precedenza, le botnet usano algoritmi di generazione automatica di nomi di dominio che prendono come seme delle costanti impredicibili.

Essendo generati da algoritmi, i domini prodotti sono differenti da quelli inventati e registrati da una persona fisica. A prima vista è già intuitivo notare le differenze che ci sono tra *amazon.com* e *xgpbgevjlojvmvrsgx.pw*. I domini regolarmente registrati hanno spesso significati intuitivi perchè sono formati da parole di uso comune, mentre gli AGD sono insiemi di lettere non correlate tra loro.
Le features lessicali sono riportate in tabella 6.1.

Features	Descrizione
F 1 - 3	1-gram (media, varianza, deviazione standard)
F 4 - 6	2-gram (media, varianza, deviazione standard)
F 7 - 9	3-gram (media, varianza, deviazione standard)
F 10 - 12	4-gram (media, varianza, deviazione standard)
F 13	Entropia di Shannon 2LD
F 14	Numero di caratteri differenti / lunghezza totale
F 15	Quantità di cifre / lunghezza totale
F 16	Numero di consonanti / lunghezza totale
F 17	Numero di consonanti / numero di vocali

Tabella 6.1: Features lessicali

Le caratteristiche comprese tra F1 e F12 descrivono la distribuzione dei singoli caratteri (1-gram) della stringa, dei digrammi (2-gram), dei trigrammi (3-gram) e dei quadrigrammi (4-gram). Nelle prime dodici features vengono calcolate la media, la varianza e la deviazione standard per ogni n-gramma con $n \in [1, 4]$.
In principio è stata effettuata una rigorosa analisi delle frequenze per ogni n-gramma, calcolandone media, varianza e

deviazione standard, ma dato l'elevato numero di caratteristiche estratte 36^n solo per la media di ogni n-gramma, tenendo in considerazione le 26 lettere dell'alfabeto e le cifre decimali, il numero di valori estratti sarebbe stato troppo elevato per poter essere gestito dal classificatore, così si è scelto di calcolarle solo per le occorrenze degli n-grammi in modo da poter ridurre lo spazio delle features.
Ci si aspetta che, nel risultato finale, incidano particolarmente le features da F13 a F17.
L'entropia di Shannon fornisce la quantità di informazione trasportata da ogni stringa e ci si aspetta che sia tanto più alta quanto più il dominio sia di difficile comprensione.
Da F15 a F17 si studia la composizione della parola stessa. I numeri di cifre e di consonanti possono essere fondamentali per discernere correttamente i domini benevoli da quelli malevoli.

6.2 Analisi delle probabilità

Per ogni dominio contattato dalle singole query DNS incluse nelle tabelle .csv, il classificatore calcola la probabilità che questo sia malevolo o benevolo. In particolare la probabilità è 0 se il dominio è identificato al 100% come benevolo e 1 se il dominio è identificato al 100% come malevolo. Con una probabilità pari a 0,5 si è in una posizione di completa incertezza.
Ogni file estrapolato dal traffico di rete del malware contiene n query DNS ognuna con il relativo dominio contattato. L'idea

è quella di effettuare un'analisi di massima verosimiglianza LLR (Log-likelihood ratio) al variare di i.

$$L_i = log \frac{p_i}{1 - p_i} \ , \ \ con \ 1 \leq i \leq n$$

Dove:
p_i è la probabilità dell'i-esimo dominio contattato.

Se il dominio è considerato benevolo ($0 \leq p_i < 0.5$), L_i sarà negativo. Se il dominio è considerato malevolo ($0.5 < p_i \leq 1$), L_i sarà positivo. Quanto più p_i è vicina a 0 o 1, tanto più L_i sarà grande in modulo.
Invece di effettuare soltanto un'analisi puntuale di ogni singola probabilità, si vuole capire se aggiungere una '*memoria*' di δ campioni può essere utile per una migliore identificazione del malware.

$$LLR = \sum_{z=0}^{n-\delta} \sum_{i=1+z}^{\delta+z} log \frac{p_i}{1 - p_i}$$

Dove:
δ = [1, 2, 3, 4, 5, 10, 25, n]

In questo modo si fa scorrere una finestra di δ campioni lungo tutti gli n campioni. I valori contenuti all'interno della i-esima finestra vengono poi sommati.

6.3 Simulazioni

Il classificatore e le relative probabilità estratte sono state simulate con uno script Matlab. [Appendice F]
Per simulare le singole probabilità si è pensato di utilizzare una distribuzione gaussiana limitata tra 0 e 1.

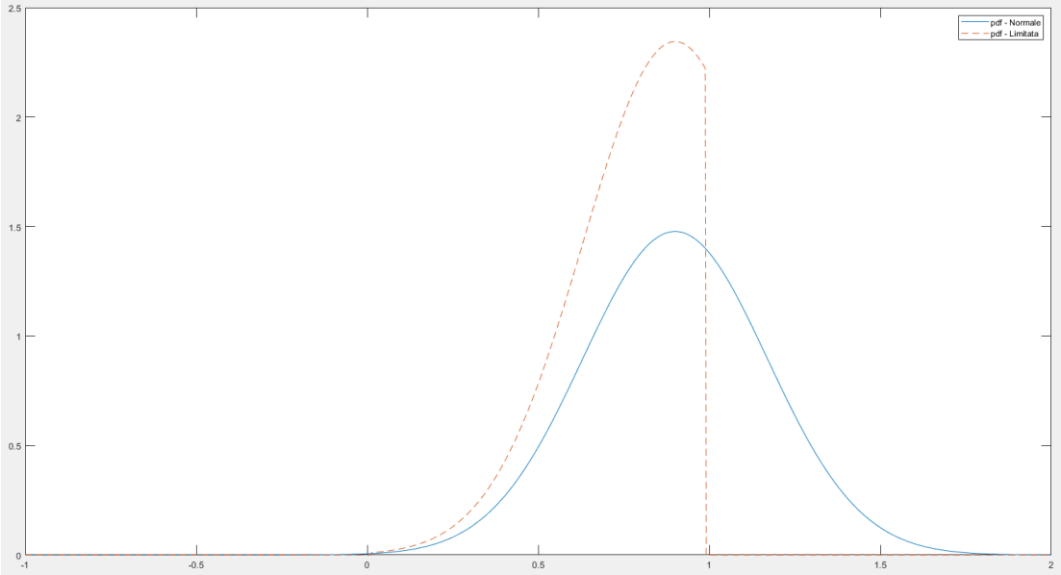

Figura 6.1: Gaussiana con assorbimento

Ad esempio, per simulare un buon classificatore al quale siano stati somministrati campioni tutti malevoli, è stata utilizzata una gaussiana con media 0.9 e deviazione standard 0.27. La media indica il valore con il quale il classificatore identifica il dominio come malevolo, la deviazione standard indica la dispersione su tale valore. In figura 6.2 si nota come con una deviazione standard di 0.27 si ha una accuratezza di identificazione dei domini malevoli di circa il 90%.

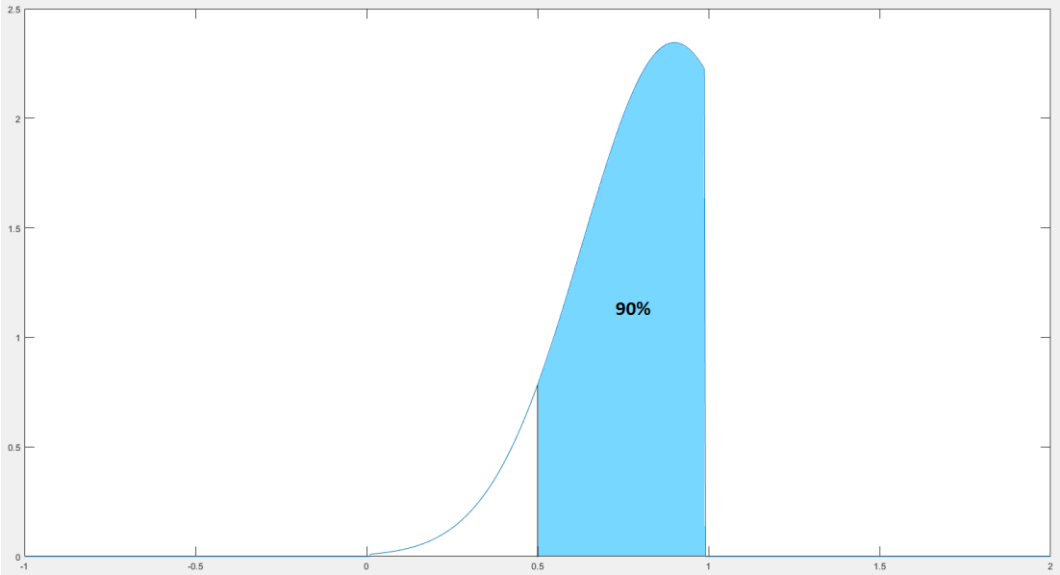

Figura 6.2: Gaussiana con accuratezza del 90%

Sono state effettuate varie simulazioni con traffico solo malevolo e con traffico misto.

Simulazione 1 – Traffico 100% malevolo

In questa simulazione è stata utilizzata una gaussiana con i seguenti valori: **m = 0.9; σ = 0.27** (accuratezza ~90%).
Sono stati generati 10'000 campioni tutti malevoli.
Il classificatore ne ha identificati 8903 come malevoli, mentre 1097 sono stati identificati erroneamente come benevoli. Applicando l'LLR con finestre da 2, 3, 4, 5, 10, 25 e 10'000 campioni si sono ottenuti i risultati visibili in tabella 6.2

LLR	TP	FN
singola p_i	8903	1097
2 campioni	9535	465
3 campioni	9803	197
4 campioni	9910	90
5 campioni	9955	45
10 campioni	10000	0
25 campioni	10000	0
10k campioni	10000	0

Tabella 6.2: TP e FN con diversa finestratura di LLR

Da questa analisi si conclude che analizzando traffico tutto malevolo, è sempre meglio utilizzare un LLR con finestratura più grande possibile. In figura 6.3 si vede che la curva dell'LLR totale è approssimabile ad una retta crescente. Questo accade perché il traffico è completamente malevolo e quindi una finestratura grande di LLR fa sì che, anche in presenza di *false negative*, risulti impossibile scendere al di sotto dello zero.

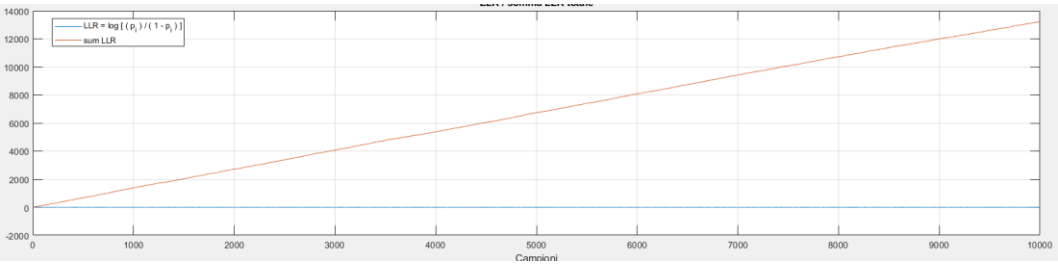

Figura 6.3: Grafico LLR totale

Per non attendere la generazione di tutto il traffico, può essere utilizzata una finestratura da 25 o 10 campioni che dà comunque ottimi risultati.

Simulazione 2 – Traffico misto con classificatore simmetrico

Alla precedente gaussiana è stata aggiunta un'altra gaussiana che simula traffico benevolo. La seconda gaussiana è speculare alla prima e quindi ha i valori: **m = 0.1**; **σ = 0.27** (accuratezza ~90%). Il traffico generato dalle due gaussiane è stato mescolato in varie percentuali. Partendo da 10% di malevolenza e arrivando fino al 90% di malevolenza sul traffico totale. I campioni totali sono sempre 10'000. In tabella 6.3 si riportano i risultati di ogni simulazione.

TP	10%	20%	30%	40%	50%	60%	70%	80%	90%
SINGOLA p_i	884	1786	2685	3551	4455	5396	6265	7125	8066
2 CAMPIONI	546	1190	1950	2719	3662	4691	5750	6920	8199
3 CAMPIONI	353	913	1569	2444	3406	4564	5771	7040	8420
4 CAMPIONI	241	698	1350	2233	3298	4487	5811	7156	8555
5 CAMPIONI	171	584	1203	2082	3152	4472	5858	7266	8639
10 CAMPIONI	41	240	772	1677	2949	4596	6179	7628	8905
25 CAMPIONI	2	29	267	1112	2823	5055	6670	7909	8996
TOTALE	0	0	0	14	2336	5998	6991	8000	8996

FP	10%	20%	30%	40%	50%	60%	70%	80%	90%
SINGOLA p_i	972	855	767	647	534	455	314	237	106
2 CAMPIONI	810	1047	1256	1356	1358	1257	1097	798	467
3 CAMPIONI	627	997	1334	1487	1608	1598	1431	1089	658
4 CAMPIONI	467	909	1277	1514	1714	1769	1611	1290	751
5 CAMPIONI	342	758	1237	1583	1830	1942	1783	1418	830
10 CAMPIONI	97	423	853	1476	2001	2389	2294	1730	955
25 CAMPIONI	10	62	315	1126	2165	2970	2748	1948	998
TOTALE	0	0	0	16	2214	3994	2992	1999	999

Tabella 6.3: Resoconto analisi: $m_1 = 0.1$; $m_2 = 0.9$; $\sigma = 0.27$

I grafici dei valori riportati nelle tabelle sono in figura 6.4 e 6.5.

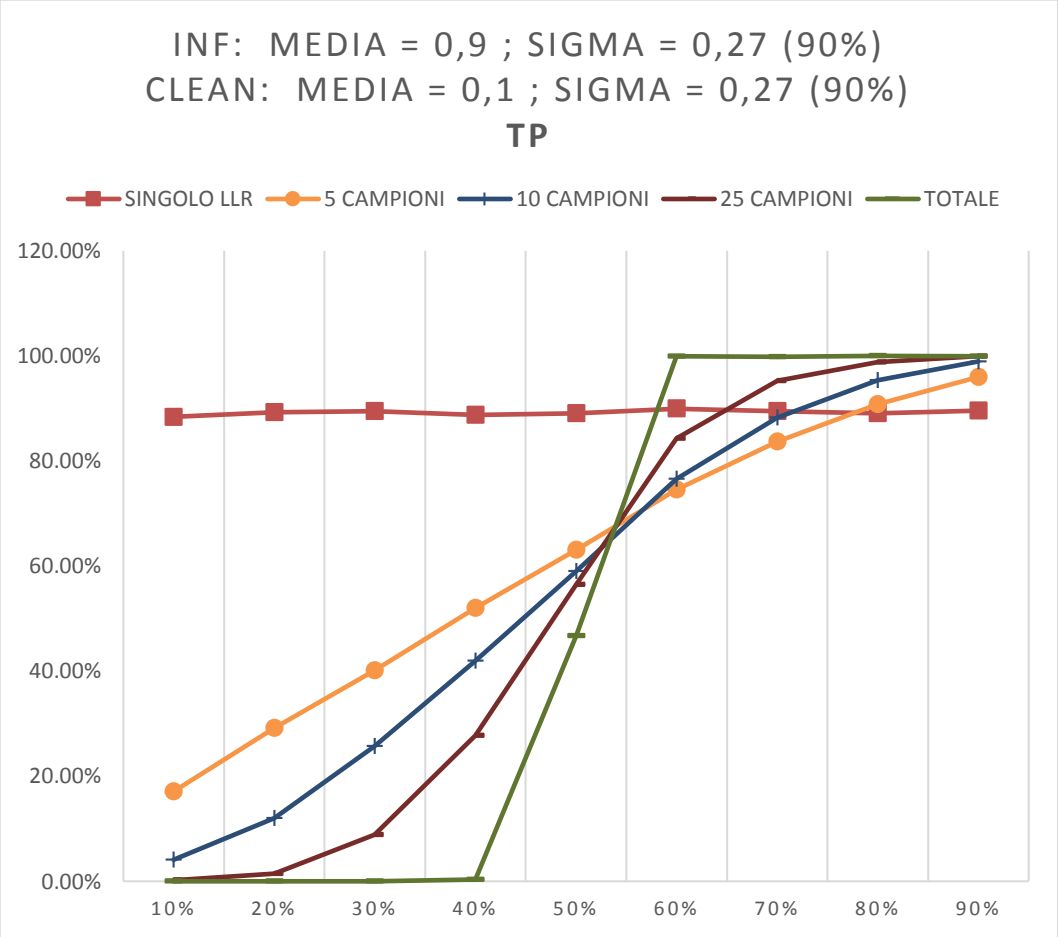

Figura 6.4: *True Positive*: $m_1 = 0.1$; $m_2 = 0.9$; $\sigma = 0.27$

L'asse delle ordinate esprime la percentuale di TP in relazione ai domini malevoli o FP in relazione ai domini benevoli. Sull'asse delle ascisse si riporta la percentuale di domini realmente malevoli nel mix di traffico.

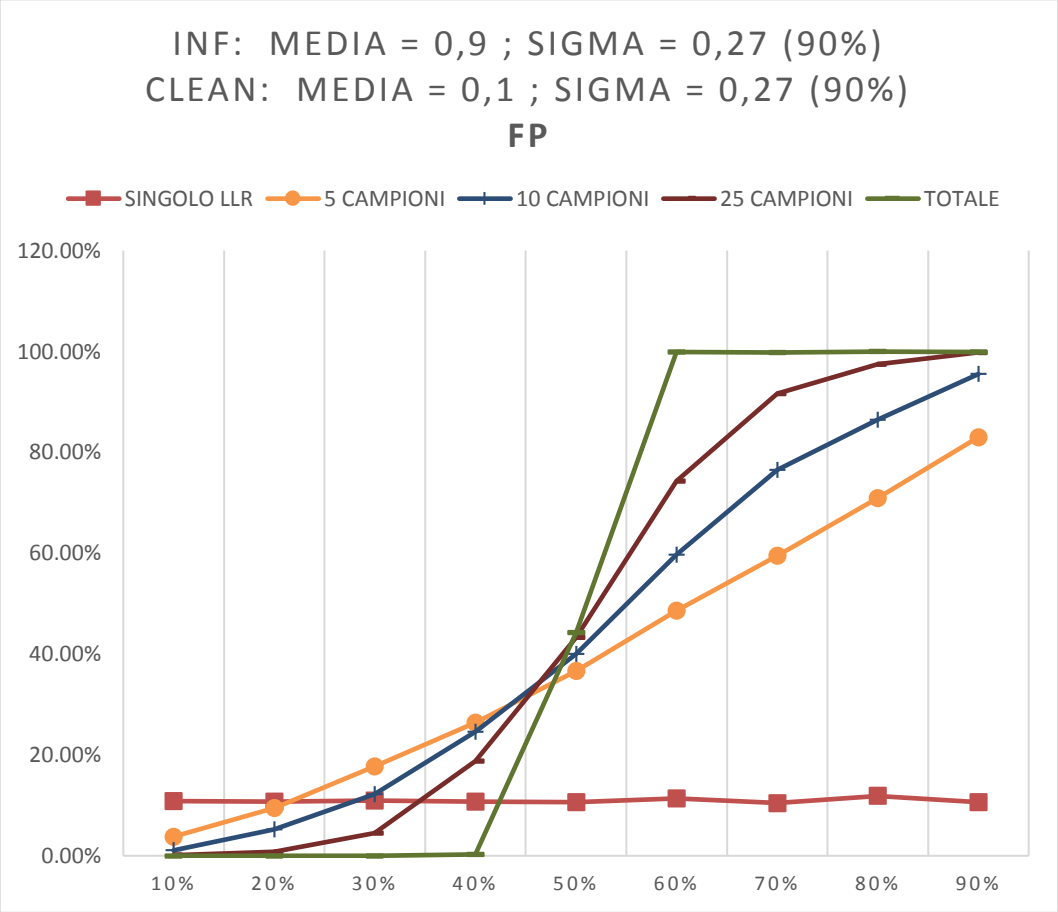

Figura 6.5: *False Positive*: $m_1 = 0.1$; $m_2 = 0.9$; $\sigma = 0.27$

Dai grafici si nota che fino a quando il traffico malevolo è minore di circa il 70% del traffico totale, è sempre meglio utilizzare una rilevazione puntuale delle singole probabilità e non utilizzare nessun tipo di finestratura LLR. Infatti una rilevazione singola dà le migliori performance dal punto di vista dei TP. Per quanto riguarda i FP è vero che le finestrature danno performance migliori, ma hanno una rilevazione di TP troppo bassa in quell'intervallo.

Questo significa che quando il traffico è prevalentemente benevolo, le finestrature di LLR non sono in grado di rilevare il malware. Quando la percentuale di traffico malevolo all'interno del traffico totale supera il 70%, si può prendere in considerazione una finestratura LLR a discapito ovviamente di un aumento anche dei FP.
Dato che solitamente un pc infetto da DGA inizia a produrre tante query DNS, e quindi la percentuale di traffico malevolo diventa predominante, si può pensare di utilizzare una analisi tramite finestratura LLR per sopperire a eventuali mancanze del classificatore.

Simulazione 3 – Traffico misto con classificatore asimmetrico

In questo caso è stato simulato un classificatore allenato meglio sui domini benevoli (m = 0.1; σ = 0.27 ~90%) e meno bene su quelli malevoli (m = 0.7; σ = 0.22 ~80%). Questo è ragionevole dato che i dataset di domini benevoli sono in numero maggiore rispetto ai dataset di domini malevoli creati da DGA. L'analisi è stata effettuata come nel caso precedente e i risultati sono riportati in tabella 6.4 e nelle figure 6.6 e 6.7.

TP	10%	20%	30%	40%	50%	60%	70%	80%	90%
SINGOLA p_i	785	1569	2410	3215	4065	4852	5648	6414	7159
2 CAMPIONI	398	935	1599	2350	3168	4083	5127	6258	7472
3 CAMPIONI	229	622	1234	1965	2805	3802	4991	6265	7709
4 CAMPIONI	129	446	1039	1686	2555	3655	4936	6340	7882
5 CAMPIONI	98	336	849	1508	2417	3565	4904	6454	8020
10 CAMPIONI	9	88	377	1004	1986	3258	5074	6893	8515
25 CAMPIONI	0	1	63	396	1267	3077	5423	7642	8962
TOTALE	0	0	1	8	13	899	7000	8000	9000

FP	10%	20%	30%	40%	50%	60%	70%	80%	90%
SINGOLA p_i	948	855	775	696	508	462	333	242	104
2 CAMPIONI	626	824	984	1072	1047	937	823	658	358
3 CAMPIONI	430	670	900	1115	1145	1132	1023	911	483
4 CAMPIONI	290	519	838	1107	1218	1258	1207	1038	593
5 CAMPIONI	185	426	746	1052	1221	1342	1338	1127	669
10 CAMPIONI	16	115	376	775	1176	1440	1636	1468	853
25 CAMPIONI	0	5	57	339	872	1564	2046	1841	986
TOTALE	0	0	2	5	5	557	2999	2000	1000

Tabella 6.4: Resoconto analisi: $m_1 = 0.1$; $m_2 = 0.7$; $\sigma_1 = 0.27$; $\sigma_2 = 0.22$

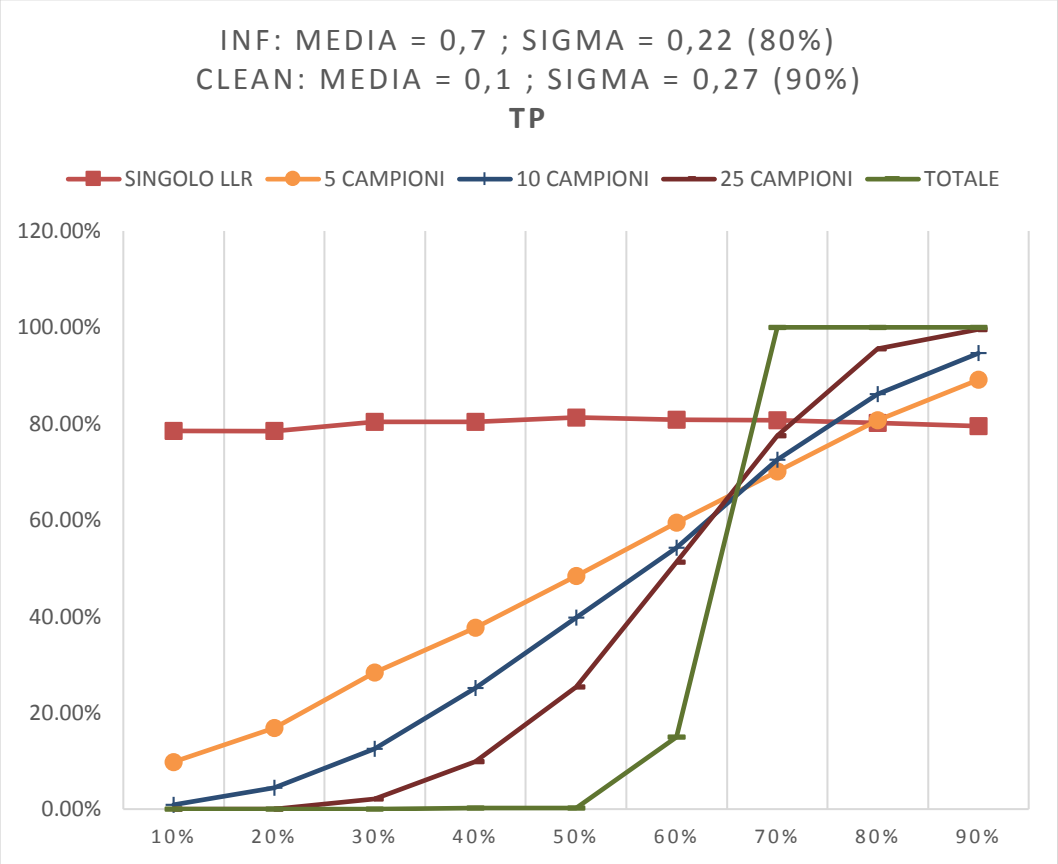

Figura 6.6: *True Positive*: $m_1 = 0.1$; $m_2 = 0.7$; $\sigma_1 = 0.27$; $\sigma_2 = 0.22$

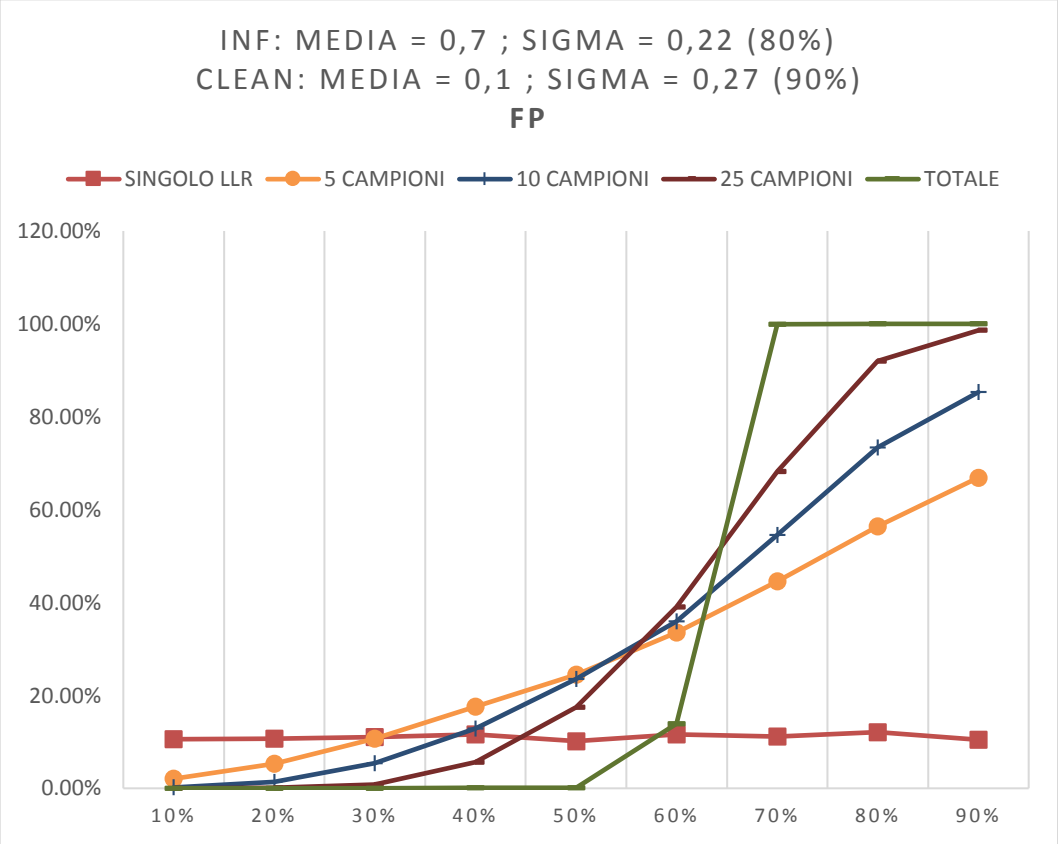

Figura 6.7: *False Positive*: $m_1 = 0.1$; $m_2 = 0.7$; $\sigma_1 = 0.27$; $\sigma_2 = 0.22$

Anche in questo caso le considerazioni sono analoghe al caso precedente. Per traffico prevalentemente benevolo la migliore soluzione è utilizzare un rilevamento delle singole probabilità date dal classificatore. Quando il traffico malevolo supera il 70% del traffico totale si può prendere in considerazione una finestratura sempre maggiore a discapito di un aumento dei FP.

Ricordando la formula della *precision*: $P = \frac{TP}{TP+FP}$

Si ottiene sempre la *precision* migliore con una rilevazione puntuale delle singole probabilità, ma se si avesse la necessità di aumentare la percentuale di TP a discapito dei FP, un buon compromesso potrebbe essere una finestratura da 5 campioni. Infatti con questa finestratura, quando il traffico malevolo supera l'80% del traffico totale, si riescono ad individuare circa il 90% dei domini malevoli, anche se il classificatore ha un'accuratezza di circa l'80%.

Anche in presenza di un classificatore non allenato in maniera molto performante sulla rilevazione dei domini malevoli, grazie all'ausilio del LLR si riesce a rilevare il malware che fa uso di DGA, in quanto il traffico malevolo diventerebbe predominante rispetto a quello benevolo e una qualsiasi finestratura maggiore di 5 campioni farebbe scattare l'allarme.

Capitolo 7

Conclusioni e sviluppi futuri

Lo scopo di questa tesi è stato quello di capire se la classificazione basata su DGA trova riscontro ed è utile a riconoscere malware che usano DGA.
Sono stati creati i dataset benevoli e malevoli da somministrare al classificatore, ma non avendo ancora a disposizione i dati reali estrapolati dal classificatore, si sono effettuate delle simulazioni. Dalle simulazioni è emerso che se una macchina è infetta da un malware di tipo DGA, è possibile effettuare un'analisi di massima verosimiglianza per ottenere percentuali di TP più alte. Quando una macchina è infetta da malware DGA il traffico è prevalentemente malevolo. In questo caso, a prescindere da quanto il classificatore sia stato allenato bene sul rilevamento di domini malevoli, utilizzare finestre maggiori di 5 campioni di LLR aiuta a migliorare la percentuale di *true positive* a discapito di un aumento anche dei *false positive*. Se il traffico malevolo scende al di sotto della soglia del 70% sul traffico totale, è sempre meglio utilizzare una rilevazione puntuale delle singole probabilità p_i stimate dal classificatore.
Confrontando questa tecnica di rilevazione con la tecnica tramite sandbox, si possono fare alcune considerazioni.
Il classificatore è più rapido e non ha la necessità di appoggiarsi su altri software come ad esempio antivirus; la sandbox

invece necessita di circa 15 minuti per effettuare l'analisi di un singolo malware e ha la necessità di tutta una infrastruttura fisica (*honeypot, server*). Di contro però, la sandbox, oltre a individuare il malware, è in grado di farne un'analisi approfondita. Al momento il classificatore è allenato per rilevare solamente domini creati da DGA, mentre la sandbox può rilevare qualsiasi tipo di malware. Una idea potrebbe essere quella di utilizzare il classificatore come sistema di protezione *real-time* e successivamente la sandbox come sistema di analisi approfondita del malware.
Il lavoro è in continua evoluzione e gli sviluppi futuri non mancano.

- I dataset benevoli e malevoli sono pronti per essere sottoposti al classificatore in modo da poter confrontare i risultati ottenuti nelle simulazioni con i dati reali del classificatore.

- È possibile aumentare le dimensioni del dataset in modo da rendere il più performante possibile il classificatore.

- Si possono ricercare altri metodi matematici in grado di migliorare la percentuale di *true positive* anche quando il traffico malevolo scende al di sotto del 70% del traffico totale o che siano in grado di limitare i *false positive* su traffico prevalentemente malevolo.

Elenco delle Figure

Elenco delle Tabelle

Bibliografia

[1] Clusit - Rapporto 2018 sulla Sicurezza ICT in Italia.
URL: https://clusit.it/wp-content/uploads/download/Rapporto_Clusit_2018_aggiornamento_settembre.pdf

[2] J. Gardiner, M. Cova, and S. Nagaraja, *Command & Control: Understanding, Denying and Detecting*.
URL: http:// arxiv.org/abs/1408.1136

[3] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. *Your botnet is my botnet: Analysis of a botnet takeover*, 2009.

[4] Xiaonan Zang, Athichart Tangpong, George Kesidis and David J. Miller, CSE Dept Technical Report on "*Botnet Detection through Fine Flow Classification*" Report No. CSE11-001, Jan. 31, 2011.

[5] JoeSecurity, URL: https://www.joesecurity.org/

[6] Yara Get Started, URL: https://yara.readthedocs.io/en/latest/gettingstarted.html

[7] Nicola Rossi, *Identificazione di malware tramite analisi di traffico di rete e classificazione di caratteristiche lessicografiche*, tesi magistrale, Dipartimento di Ingegneria dell'Informazione Università Politecnica delle Marche, a.a. 2016-2017, relatore Franco Chiaraluce.

[8] Luca Vitelli, *Identificazione di malware tramite analisi di traffico di rete e classificazione di caratteristiche protocollari,* tesi magistrale, Dipartimento di Ingegneria dell'Informazione Università Politecnica delle Marche, a.a. 2016-2017, relatore Marco Baldi.

[9] Stratosphere Research Laboratory: IPS project. URL: `https://www.stratosphereips.org/`

[10] Pedro Marques da Luz. *Botnet Detection Using Passive DNS*. 2013/2014.

Appendici

Appendice A

Script Python Sandbox

```
import os
import shutil
import subprocess
import json
import time
import jbxapi

#time to stringa unita
now = (str(time.ctime()))
now = now.replace(" ", "-")
now = now.replace(":", ".")
#file di log
out_file = open("LOG__" +now+ ".log", "w")
print("LOG__" + str(now) + ".log")
print(str(time.ctime()) + " : Inizio\n")
out_file.write(str(time.ctime()) + " : Inizio\n")

while (len(os.listdir('/home/csec/pecvirus')) == 0):
    time.sleep(600)

#loop con controllo sulla directory
while (len(os.listdir('/home/csec/pecvirus')) > 0):

    #lista file analizzati dalla sandbox presenti nel db
    list_json = json.loads(subprocess.check_output('jbxapi list',
shell=True).decode('utf-8'))
    #webid ultimo file nel db sandbox
    str_webid = list_json[0]['webid']
    print(str(time.ctime()) + ' : File analizzati nella Sandbox
presenti nel db=>> ' + str_webid + '\n')
    out_file.write(str(time.ctime()) + ' : File analizzati nella
Sandbox =>> ' + str_webid + '\n')
    #stato ultimo file nel db sandbox
    info_json = json.loads(subprocess.check_output('jbxapi info '
+ str_webid, shell=True))
    print(str(time.ctime()) + ' : E-mail da analizzare = ' + str(
        len(os.listdir('/home/csec/pecvirus'))) + '\n')
    out_file.write(str(time.ctime()) + ' : E-mail da analizzare =
' + str(
        len(os.listdir('/home/csec/pecvirus'))) + '\n')
```

```
    status = (info_json['status'] == 'finished') or
(info_json['status'] == 'running')
    print(str(time.ctime()) + ' : Sandbox pronta a ricevere file
=>> ' + str(status) + '\n')
    out_file.write(str(time.ctime()) + ' : Sandbox pronta a ri-
cevere file =>> ' + str(status) + '\n')

    if status == True:

        #elenco file da analizzare
        lista_file = os.listdir('/home/csec/pecvirus')

        file, ext = os.path.splitext('/home/csec/pecvirus/' +
lista_file[0])
        check = (ext == '.rdv') or (ext == '.esito')

        if check == True:
            # sposto il file inviato in un'altra directory
            shutil.move('/home/csec/pecvirus/' + lista_file[0],
'/home/csec/mail_analizzate/' + lista_file[0])

        else:
            # cancello report clean
            for i in range(1, 10):
                WebidCheck = list_json[i]['webid']
                JsonCheck = json.loads(subprocess.check_out-
put('jbxapi info ' + WebidCheck, shell=True))

                if (JsonCheck['runs'][0]['detection'] == 'clean'):
                    json.loads(subprocess.check_output('jbxapi de-
lete ' + WebidCheck, shell=True))
                    print(str(time.ctime()) + ' : Report ' +
WebidCheck + ' cancellato\n')
                    out_file.write(str(time.ctime()) + ' : Report
' + WebidCheck + ' cancellato\n')

            time.sleep(30)
            list_json = json.loads(subprocess.check_output('jbxapi
list', shell=True).decode('utf-8'))

            # cancello report che non genera traffico
            for i in range(1, 10):
                WebidCheck = list_json[i]['webid']
                JsonCheck = json.loads(subprocess.check_out-
put('jbxapi report ' + WebidCheck, shell=True))

                if (JsonCheck['analysis']['contacted']['ips'] is
None):
                    json.loads(subprocess.check_output('jbxapi de-
lete ' + WebidCheck, shell=True))
                    print(str(time.ctime()) + ' : Report ' +
WebidCheck + ' cancellato, non genera traffico\n')
                    out_file.write(str(time.ctime()) + ' : Report
' + WebidCheck + ' cancellato, non genera traffico\n')
```

```
            # aggiungo estensione .eml al file da inviare
            os.rename('/home/csec/pecvirus/' + lista_file[0],
                      '/home/csec/pecvirus/' + lista_file[0] +
'.eml')
            try:
                # invio file.eml
                subprocess.check_output(
                    'jbxapi submit /home/csec/pecvirus/' +
lista_file[0] + '.eml', shell=True)
                out_file.write(str(time.ctime()) + ' : File ' +
lista_file[0] + '.eml inviato alla Sandbox\n')
                print(str(time.ctime()) + ' : File ' +
lista_file[0] + '.eml inviato alla Sandbox\n')

            except subprocess.CalledProcessError:
                # gestione errore: nessun file da analizzare
                out_file.write(str(time.ctime()) + ' : Nessun file
da analizzare nella mail\n' + lista_file[0] + '.eml')
                print(str(time.ctime()) + ' : Nessun file da ana-
lizzare nella mail\n' + lista_file[0] + '.eml')

            time.sleep(10)
            #sposto il file inviato in un'altra directory
            shutil.move('/home/csec/pecvirus/' + lista_file[0] +
'.eml',
                        '/home/csec/mail_analizzate/' + lista_file[0]
+ '.eml')

    if status == False:
        print(str(time.ctime()) + ' : Waiting...\n')
        out_file.write(str(time.ctime()) + ' : Waiting...\n')
        time.sleep(120)

    while (len(os.listdir('/home/csec/pecvirus')) == 0):
        time.sleep(600)

print(str(time.ctime()) + ' : Fine!\n')
out_file.write(str(time.ctime()) + ' : Fine!\n')
```

Appendice B

Script per l'estrazione delle query DNS

```
import os
import subprocess
import time


#time to stringa unita
now = (str(time.ctime()))
now = now.replace(" ", "-")
now = now.replace(":", ".")

#file di log
out_file = open("D:\\TESI\\log_PCAP\\LOG__" +now+ ".log", "w")
print("LOG__" + str(now) + ".log")
print(str(time.ctime()) + " : INIZIO\n")
out_file.write(str(time.ctime()) + " : INIZIO\n")


def find_url_dns(file_pcap):
    try:
        subprocess.check_output('tshark -r D:\\TESI\\PCAP\\'
                                + file_pcap + ' -T fields -e
dns.qry.name -Y "dns.flags.response eq 0" > D:\\TESI\\PCAP_txt\\'+
file_pcap +'.txt', shell=True)

    except subprocess.CalledProcessError:
        print(str(time.ctime()) + ">> " + file + "\n")
        out_file.write(str(time.ctime()) + ">> " + file + "\n")



lista_file = os.listdir('D:\\TESI\\PCAP\\')

for file in lista_file:
    print(str(time.ctime()) + " : Inizio analisi file: "+ file +
"\n")
    out_file.write(str(time.ctime()) + " : Inizio analisi file: "+
file + "\n")
    find_url_dns(file)
    print(str(time.ctime()) + " : Fine analisi file: " + file +
"\n")
    out_file.write(str(time.ctime()) + " : Fine analisi file: " +
file + "\n")

print(str(time.ctime()) + " : FINE")
out_file.write(str(time.ctime()) + " : FINE")
```

Appendice C

Script per eliminare i duplicati

```
import os

lista_file = os.listdir('D:\\TESI\\PCAP_txt\\')


def remove_duplicates(file):

    content = open('D:\\TESI\\PCAP_txt\\' + file, 'r').readlines()
    content_set = set(content)
    cleandata = open( 'D:\\TESI\\PCAP_txt_no_dup\\no_dup_' + file,
'w')

    for line in content_set:
        cleandata.write(line)


for file in lista_file:
    remove_duplicates(file)
```

Appendice D

Script per il confronto con la whitelist Alexa

```
import os

lista_file = os.listdir('D:\\TESI\\PCAP_txt_no_dup\\')


def malw_url(file):

    file1 = open('D:\\TESI\\top-1m.txt', 'r').readlines()
    file2 = open('D:\\TESI\\PCAP_txt_no_dup\\' + file, 'r').read-
lines()
    file3 = open('D:\\TESI\\PCAP_txt_no_alexa\\' + file +
'_no_alexa.txt','a')
    print(str('Analisi file: '+file + '\n'))

    for line2 in file2:
        flag = 0
        for line1 in file1:
            if line1 == line2:
                flag = 1
        if flag == 0:
            file3.write(line2)


for file in lista_file:
    malw_url(file)
```

Appendice E

Script per creare le tabelle

```
import os
import subprocess
import time
import csv

# time to stringa unita
now = (str(time.ctime()))
now = now.replace(" ", "-")
now = now.replace(":", ".")

# file di log
out_file = open("D:\\TESI\\pcap_DGA_log\\LOG__" + now + ".log",
"w")
print("LOG__" + str(now) + ".log")
print(str(time.ctime()) + " : INIZIO\n")
out_file.write(str(time.ctime()) + " : INIZIO\n")


def domain_table(file_pcap):

    # malware name e dga name
    malware = file_pcap.split("_")
    with open('D:\\tesi\\malware-DGA.csv') as name:
        name_read = csv.reader(name)
        for riga in name_read:

            if riga:
                if riga[0] == malware[0]:
                    malware_name = riga[1]
                    dga_name = riga[2]

    # download domain name e timestamp
    try:

        print(str(time.ctime()) + " : Download domain name e
timestamp\n")
        out_file.write(str(time.ctime()) + " : Download domain
name e timestamp\n")

        subprocess.check_output('tshark -r D:\\TESI\\pcap_DGA\\'
                                + file_pcap + ' -T fields -e
_ws.col.Time -e dns.qry.name -Y "dns.flags.response eq 0" >
D:\\TESI\\pcap_DGA_txt\\' + file_pcap + '.txt', shell=True)

    except subprocess.CalledProcessError:

        print(str(time.ctime()) + ">> " + file + "\n")
```

```
        out_file.write(str(time.ctime()) + ">> " + file + "\n")

    # save domain name e timestamp
    with open('D:\\TESI\\pcap_DGA_txt\\' + file_pcap + '.txt',
'r') as in_file:

        print(str(time.ctime()) + " : Save domain name e
timestamp\n")
        out_file.write(str(time.ctime()) + " : Save domain name e
timestamp\n")

        stripped = (line.strip() for line in in_file)
        lines = (line.split("\t") for line in stripped if line)

        with open('D:\\TESI\\pcap_DGA_txt\\' + file_pcap + '.csv',
'w') as txt_file:
            writer = csv.writer(txt_file)
            writer.writerows(lines)

    # append table
    with open('D:\\TESI\\pcap_DGA_txt\\' + file_pcap + '_ta-
ble.csv', 'a') as out_final:
        writer = csv.writer(out_final)

        print(str(time.ctime()) + " : Start table\n")
        out_file.write(str(time.ctime()) + " : Start table\n")

        # total time
        with open('D:\\TESI\\pcap_DGA_txt\\' + file_pcap + '.csv')
as out_timestamp:
            lettore = csv.reader(out_timestamp, delimiter=",")

            for riga in lettore:
                if riga:
                    total_time = riga[0]

        # matching Alexa
        with open('D:\\TESI\\pcap_DGA_txt\\' + file_pcap + '.csv')
as out_timestamp:
            lettore = csv.reader(out_timestamp, delimiter=",")

            print(str(time.ctime()) + " : Start Alexa matching\n")
            out_file.write(str(time.ctime()) + " : Start Alexa
matching\n")

            for riga in lettore:
                print(str(time.ctime()) + " : Time " + riga[0] + "
\\ " + total_time + "\n")
                flag = 0
                with open(''D:\\TESI\\PCAP_txt_no_alexa\\' + file
+ '_no_alexa.txt') as malware_url:
                    lettore_malware_url = csv.reader(malware_url)

                    for line in lettore_malware_url:
```

```
                        if line:
                            if line[0] == riga[1]:
                                flag = 1

                    # No matching with Alexa
                    if flag is 0:
                    writer.writerow([file_pcap, riga[1], mal-
ware_name, dga_name, riga[0], total_time, "YES"])

                    # mathing with Alexa
                    if flag is 1:
                    writer.writerow([file_pcap, riga[1], mal-
ware_name, dga_name, riga[0], total_time, "NO"])


lista_file = os.listdir('D:\\TESI\\pcap_DGA\\')

for file in lista_file:

    # intestazione table
    with open('D:\\TESI\\pcap_DGA_txt\\' + file + '_table.csv',
'a') as out_final:
        writer = csv.writer(out_final)
        writer.writerow(
            ('filename', 'domain name', 'malware', 'DGA',
'timestamp(sec)', 'total time(sec)', 'alexa(YES/NO)'))

    print(str(time.ctime()) + " : Inizio analisi file: " + file +
"\n")
    out_file.write(str(time.ctime()) + " : Inizio analisi file: "
+ file + "\n")

    domain_table(file)

    print(str(time.ctime()) + " : Fine analisi file: " + file +
"\n")
    out_file.write(str(time.ctime()) + " : Fine analisi file: " +
file + "\n")

print(str(time.ctime()) + " : FINE")
out_file.write(str(time.ctime()) + " : FINE")
```

Appendice F

Script Matlab - analisi LLR

```
clear all

%media e sigma
media = 0.7; sigma = 0.22;
media_infected = 0.7; sigma_infected = 0.46;
media_clean = 0.1; sigma_clean = 0.27;

%pdf
pd = makedist('Normal',media,sigma);
pd_infected = makedist('Normal',media_infected,sigma_infected);
pd_clean = makedist('Normal',media_clean,sigma_clean);

%pdf limitate
t = truncate(pd,0.01,0.99);
t_infected = truncate(pd_infected,0.01,0.99);
t_clean = truncate(pd_clean,0.01,0.99);

x = linspace(-1,2,1000);

%grafici pdf traffico normale
%-----------------------------------------------
% figure('Name','pdf','NumberTitle','off');
% plot(x,pdf(pd,x))
% hold on
% plot(x,pdf(t,x),'LineStyle','--')
% legend('pdf - Normale','pdf - Limitata')
% hold off
%-----------------------------------------------

%grafici pdf macchina infetta
%-----------------------------------------------
%figure('Name','pdf_infected','NumberTitle','off');
%plot(x,pdf(pd_infected,x))
%hold on
%plot(x,pdf(t_infected,x),'LineStyle','--')
%legend('pdf - Normale','pdf - Limitata')
%hold off
%-----------------------------------------------

%grafici pdf macchina pulita
%-----------------------------------------------
%figure('Name','pdf_clean','NumberTitle','off');
%plot(x,pdf(pd_clean,x))
%hold on
%plot(x,pdf(t_clean,x),'LineStyle','--')
```

```
%legend('pdf - Normale','pdf - Limitata')
%hold off
%-----------------------------------------------


%TRAFFICO SINGOLO
%------------------------
% r = random(t,10000,1);
% for i = [1:length(r)]
%    r_list(i,1) = r(i,1);
%    r_list(i,2) = 1;
% end
% fprintf ('\nDomini maligni: %d\n' , length(r))
%------------------------

%TRAFFICO MISTO
%----------------------------------------
L_infected = 6000;
L_clean = 4000;
fprintf ('\nDomini maligni: %d\nDomini benigni: %d' , L_infected,
L_clean)
infected = random(t_infected,L_infected,1);
for i = [1:L_infected]
    infected_list(i,1) = infected(i,1);
    infected_list(i,2) = 1;
end

clean = random(t_clean,L_clean,1);

for i = [1:L_clean]
    clean_list(i,1) = clean(i,1);
    clean_list(i,2) = 0;
end


r_temp = [infected_list;clean_list];
r_list = r_temp(randperm(length(r_temp)),:);
for i = [1:length(r_list)]
    r(i,1) = r_list(i,1);
end
%----------------------------------------

L = length(r);

n = 1:L;

malicious = 0;
benign = 0;

TP = 0;
FP = 0;
TN = 0;
```

```
FN = 0;
for i = [1:L]


    if r(i,1) > 0.5
        malicious = malicious+1;
        if r_list(i,2) == 1
            TP = TP + 1;
        end
        if r_list(i,2) == 0
            FP = FP + 1;
        end

    else
        benign = benign+1;

        if r_list(i,2) == 1
            FN = FN + 1;
        end
        if r_list(i,2) == 0
            TN = TN + 1;
        end

    end
end

prob_mal = malicious/(malicious+benign);

fprintf ('\nIl classificatore identifica:\nDomini maligni: %d  (%d
TRUE POSITIVE e %d FALSE POSITIVE)\nDomini benigni: %d  (%d TRUE
NEGATIVE e %d FALSE NEGATIVE)\n\n' , malicious , TP,FP, benign ,
TN,FN)

figure('Name','r','NumberTitle','off');
histogram(r)


for i = [1:L]
    LLR(i,1) = log((r(i,1))/(1 - r(i,1)));
end

sum(1,1) = LLR(1,1);

for i = [2:L]
    sum(i,1) = sum(i-1,1)+LLR(i,1);
end


%Finestra da 2 campioni
delta2 = zeros([2 1]);
sum2 = zeros([L 1]);
```

```
for i = [1:L]
    delta2(2,1) = LLR(i,1);

    for j = [1:2]
        sum2(i,1) = sum2(i,1) + delta2(j,1);
    end

    delta2(1,1) = delta2(2,1);

end

%Finestra da 3 campioni
delta3 = zeros([3 1]);
sum3 = zeros([L 1]);

for i = [1:L]
    delta3(3,1) = LLR(i,1);

    for j = [1:3]
        sum3(i,1) = sum3(i,1) + delta3(j,1);
    end

    for j = [1:2]
        delta3(j,1) = delta3(j+1,1);
    end

end

%Finestra da 4 campioni
delta4 = zeros([4 1]);
sum4 = zeros([L 1]);

for i = [1:L]
    delta4(4,1) = LLR(i,1);

    for j = [1:4]
        sum4(i,1) = sum4(i,1) + delta4(j,1);
    end

    for j = [1:3]
        delta4(j,1) = delta4(j+1,1);
    end

end

%Finestra da 5 campioni
delta5 = zeros([5 1]);
sum5 = zeros([L 1]);

for i = [1:L]
```

```
    delta5(5,1) = LLR(i,1);

    for j = [1:5]
        sum5(i,1) = sum5(i,1) + delta5(j,1);
    end

    for j = [1:4]
        delta5(j,1) = delta5(j+1,1);
    end

end

%Finestra da 10 campioni
delta10 = zeros([10 1]);
sum10 = zeros([L 1]);

for i = [1:L]
    delta10(10,1) = LLR(i,1);

    for j = [1:10]
        sum10(i,1) = sum10(i,1) + delta10(j,1);
    end

    for j = [1:9]
        delta10(j,1) = delta10(j+1,1);
    end

end

%Finestra da 25 campioni
delta25 = zeros([25 1]);
sum25 = zeros([L 1]);

for i = [1:L]
    delta25(25,1) = LLR(i,1);

    for j = [1:25]
        sum25(i,1) = sum25(i,1) + delta25(j,1);
    end

    for j = [1:24]
        delta25(j,1) = delta25(j+1,1);
    end

end


%medie e alert
media2 = 0;
alert2 = 0;
med_alert2 = 0;
```

```
TP2 = 0;
FP2 = 0;

for i = [1:L]
    media2 = media2 + sum2(i,1);
    if sum2(i,1) > 0
        med_alert2 = med_alert2 + sum2(i,1);
        alert2 = alert2 + 1;

        if r_list(i,2) == 1
            TP2 = TP2 + 1;
        end
        if r_list(i,2) == 0
            FP2 = FP2 +1;
        end

    end
end
%media2 = media2/L;
med_alert2 = med_alert2/alert2;
%fprintf ('\nMedia con finestra da 2 campioni: %d\n' , media2)
fprintf ('FINESTRA 2 CAMPIONI:  %d ALERT di cui %d TRUE POSITIVE e
%d FALSE POSITIVE\n' , alert2 , TP2, FP2)


media3 = 0;
alert3 = 0;
med_alert3 = 0;
TP3 = 0;
FP3 = 0;
for i = [1:L]
    media3 = media3 + sum3(i,1);
    if sum3(i,1) > 0
        med_alert3 = med_alert3 + sum3(i,1);
        alert3 = alert3 + 1;
        if r_list(i,2) == 1
            TP3 = TP3 + 1;
        end
        if r_list(i,2) == 0
            FP3 = FP3 +1;
        end
    end
end
%media3 = media3/L;
med_alert3 = med_alert3/alert3;
%fprintf ('\nMedia con finestra da 3 campioni: %d\n' , media3)
fprintf ('FINESTRA 3 CAMPIONI:  %d ALERT di cui %d TRUE POSITIVE e
%d FALSE POSITIVE\n' , alert3 , TP3, FP3)


media4 = 0;
```

```
alert4 = 0;
med_alert4 = 0;
TP4 = 0;
FP4 = 0;
for i = [1:L]
    media4 = media4 + sum4(i,1);
    if sum4(i,1) > 0
        med_alert4 = med_alert4 + sum4(i,1);
        alert4 = alert4 + 1;
         if r_list(i,2) == 1
            TP4 = TP4 + 1;
        end
        if r_list(i,2) == 0
            FP4 = FP4 +1;
        end
    end
end
%media4 = media4/L;
med_alert4 = med_alert4/alert4;
%fprintf ('\nMedia con finestra da 4 campioni: %d\n' , media4)
fprintf ('FINESTRA 4 CAMPIONI:  %d ALERT di cui %d TRUE POSITIVE e
%d FALSE POSITIVE\n' , alert4 , TP4, FP4)


media5 = 0;
alert5 = 0;
med_alert5 = 0;
TP5 = 0;
FP5 = 0;
for i = [1:L]
    media5 = media5 + sum5(i,1);
    if sum5(i,1) > 0
        med_alert5 = med_alert5 + sum5(i,1);
        alert5 = alert5 + 1;
        if r_list(i,2) == 1
            TP5 = TP5 + 1;
        end
        if r_list(i,2) == 0
            FP5 = FP5 +1;
        end
    end
end
%media5 = media5/L;
med_alert5 = med_alert5/alert5;
%fprintf ('\nMedia con finestra da 5 campioni: %d\n' , media5)
fprintf ('FINESTRA 5 CAMPIONI:  %d ALERT di cui %d TRUE POSITIVE e
%d FALSE POSITIVE\n' , alert5 , TP5, FP5)


media10 = 0;
alert10 = 0;
med_alert10 = 0;
TP10 = 0;
```

```
FP10 = 0;
for i = [1:L]
    media10 = media10 + sum10(i,1);
    if sum10(i,1) > 0
        med_alert10 = med_alert10 + sum10(i,1);
        alert10 = alert10 + 1;
        if r_list(i,2) == 1
            TP10 = TP10 + 1;
        end
        if r_list(i,2) == 0
            FP10 = FP10 +1;
        end
    end
end
%media10 = media10/L;
med_alert10 = med_alert10/alert10;
%fprintf ('\nMedia con finestra da 10 campioni: %d\n' , media10)
fprintf ('FINESTRA 10 CAMPIONI: %d ALERT di cui %d TRUE POSITIVE e
%d FALSE POSITIVE\n' , alert10 , TP10, FP10)


media25 = 0;
alert25 = 0;
med_alert25 = 0;
TP25 = 0;
FP25 = 0;
for i = [1:L]
    media25 = media25 + sum25(i,1);
    if sum25(i,1) > 0
        med_alert25 = med_alert25 + sum25(i,1);
        alert25 = alert25 + 1;
        if r_list(i,2) == 1
            TP25 = TP25 + 1;
        end
        if r_list(i,2) == 0
            FP25 = FP25 +1;
        end
    end
end
%media25 = media25/L;
med_alert25 = med_alert25/alert25;
%fprintf ('\nMedia con finestra da 25 campioni: %d\n' , media25)
fprintf ('FINESTRA 25 CAMPIONI: %d ALERT di cui %d TRUE POSITIVE e
%d FALSE POSITIVE\n' , alert25 , TP25, FP25)


%media25 = 0;
alert_tot = 0;
%med_alert25 = 0;
TP_tot = 0;
FP_tot = 0;
for i = [1:L]
    %media25 = media25 + sum25(i,1);
```

```
    if sum(i,1) > 0
        %med_alert25 = med_alert25 + sum25(i,1);
        alert_tot = alert_tot + 1;
        if r_list(i,2) == 1
            TP_tot = TP_tot + 1;
        end
        if r_list(i,2) == 0
            FP_tot = FP_tot +1;
        end
    end
end
%media25 = media25/L;
%med_alert25 = med_alert25/alert25;
%fprintf ('\nMedia con finestra da 25 campioni: %d\n' , media25)
fprintf ('LLR TOTALE:           %d ALERT di cui %d TRUE POSITIVE e
%d FALSE POSITIVE\n' , alert_tot , TP_tot, FP_tot)


n = 1:L;

figure('Name','probabilità','NumberTitle','off');
plot(n,r,'r-')
grid on
legend('Probabilità  p_i')
title('Probabilità  p_i')
xlabel('Campioni')

figure('Name','LLR','NumberTitle','off');
title('LLR')
subplot(2,1,1)
plot(n,LLR,n,sum)
grid on
legend('LLR = log [ ( p_i ) / ( 1 - p_i ) ]','sum LLR')
title('LLR / somma LLR totale')
xlabel('Campioni')
subplot(2,1,2)
plot(n,sum5,n,sum10,n,sum25)
grid on
legend('Finestra 5 campioni','Finestra 10 campioni','Finestra 25
campioni')
title('somma LLR con finestre da 5, 10 e 25 campioni')
xlabel('Campioni')
```