POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea

Sistema di apprendimento automatico per il rilevamento di canali di comunicazione nascosti utilizzati dal malware

Relatore
ing. Cataldo Basile

Candidato
Dario Platania

Dicembre 2018

Alla mia famiglia

Ringraziamenti

Il lavoro descritto in questa monografia è stato svolto sotto la supervisione del Prof. Cataldo Basile (tutore accademico), Rotondo Simone (tutore aziendale) e dell'Ing. Stefano Rinaldi (tutore aziendale).

Si ringrazia inoltre aizoOn Consulting per avermi dato la possibilità di svolgere la tesi presso la loro sede di Torino.

Indice

Capitolo 1

Introduzione

Lo svolgimento di questa tesi si è concentrato sullo studio e l'analisi dei Covert Channel, utilizzati dai malware al fine di nascondere le proprie comunicazioni con il server di comando e controllo.

1.1 Obiettivo Tesi

Un Covert Channel, in sicurezza informatica, è un tipo di tecnica usata da un avversario per stabilire una connessione con il sistema vittima ed eseguire un *exfiltration* di dati. Il suo uso, la maggior parte delle volte, è difficile da rilevare o controllare mediante i meccanismi di sicurezza alla base dei moderni sistemi operativi ed ha come obiettivo quello di operare in maniera silente sul sistema vittima. I malware che sfruttano questa tecnica possono essere implementati sia su un singolo sistema, sia su un sistema distribuito, ma l'uso dei covert channel non è semplice poiché vi sono molti problemi come, la presenza di rumore, la coerenza del dato da estrapolare e l'implementazione di protocolli utili all'esecuzione dell'attacco. Vengono principalmente usati per tentare di nascondere l'esistenza di un canale di comunicazione tra due o più sistemi.

Esistono due diversi tipi di canali nascosti, noti rispettivamente come canali di archiviazione e canali di sincronizzazione:

1. i canali di archiviazione utilizzano un supporto di memorizzazione condivisa in cui una parte scrive e l'altra legge. Questo tipo di canali include protocolli di rete o archiviazione su disco.
2. i canali di sincronizzazione usano il tempo degli eventi per trasmettere informazioni mediante l'uso di modelli. Questo viene fatto alterando intenzionalmente risorse come la CPU o altre.

Ad oggi però la grade quantità di traffico di rete che viene prodotta ha fatto si che principalmente i protocolli di rete vengano sfruttati come mezzo per le comunicazioni nascoste, cercando di nascondere le informazioni all'interno dei campi dei vari pacchetti.

L'obiettivo della tesi è stato quello di condurre uno studio accurato sui Covert Channel e il loro uso negli attacchi informatici. Esso è stato diviso in due parti:

- nella prima parte sono stati analizzati molti gruppi di *Advanced Persistent Threat* (APT) e i relativi malware e tecniche di covert channel usate; ci si è poi concentrati sui gruppi i cui malware fanno uso del record A del DNS come mezzo di comunicazione nascosta. Per fare ciò sono state raccolte informazioni tramite la lettura di alcuni articoli e report riguardante lo stato dell'arte al momento.
- nella seconda parte, sulla base di quanto fatto nella prima fase, sono stati scritti degli algoritmi utili per la creazione di un sistema che non serva ad evitare l'attacco, ma possa segnalare tramite l'analisi del traffico di rete e l'applicazione di alcuni algoritmi di machine learning (Random Forest, Decision-Tree, Naive Bayes, K-nearest neighbors), se si è in presenza o meno di un covert channel, in modo da poter attivare i sistemi di difesa necessari.

1.2 Sommario

Nel capitolo 2 verrà descritta la Threat Intelligence, attività di analisi delle minacce finalizzata alla raccolta di informazioni, andando a vedere in particolare come vengono usati strumenti come la Pyramid of Pain, la Cyber Kill Chain e il Diamond Model. Verrà infine fatta un'analisi sui principali gruppi APT studiati, dandone una collocazione geografica, il target di attacco e i software usati.

Nel capitolo 3 verrà spiegato cos'è un covert channel e le sue modalità di uso sfruttando i protocolli di rete come HTTP e DNS. Verrà fatto anche un'approfondimento su come è possibile usare il record A del DNS per veicolare al suo interno dati, in quanto è stato l'oggetto di studio di questa tesi.

Nel capitolo 4 verrà descritto il caso studio della tesi, i principali strumenti utilizzati per la parte di detection e l'ambiente in cui essa è stata svolta, vedendo più da vicino gli strumenti usati (Bro), le famiglie e i file .pcap analizzati.

Nel capitolo 5 verrà descritta tutta la parte pratica della tesi. Sarà vista tutta la parte di progettazione e implementazione della soluzione, andando a vedere la definizione dell'apprendimento automatico, la spiegazione di come sono state scelte le varie caratteristiche da Bro per la generazione del data-set e i tipi di classificatori usati. Vengono infine per ogni algoritmo presentati anche tutti i risultati.

Nel capitolo 6 vengono valutati e confrontati tutti gli algoritmi con un ultima parte dedicata alle conclusioni e agli sviluppi futuri.

Nella parte di appendice invece vengono presentati i vari algoritmi scritti per la fase implementativa.

Capitolo 2

Threat Intelligence

Ad oggi grazie alla continua e veloce evoluzione del mondo digitale l'interazione tra individui, aziende e istituzioni per finalità sociali, finanziare o economiche ha subito un grande incremento creando però allo stesso tempo nuove opportunità per attività criminali di vario genere, portando anche a nuovi modelli di strutturazione e organizzazione della criminalità.

La Threat Intelligence nota anche come *Cyber Threat Intelligence* (CTI) include la raccolta e l'analisi di informazioni al fine di caratterizzare possibili minacce dal punto di vista tecnico, di risorse, di motivazioni e di intenti, spesso in relazione a contesti operativi specifici.

In questa capitolo saranno descritti tre modelli usati per rilevare le attività di un attaccante e l'impatto che l'attacco può avere sul sistema vittima. Verranno infine descritti gli APT sia come tipo di attacchi che come gruppi hacker.

2.1 Pyramid of Pain

La Pyramid of Pain è un diagramma utilizzato per mostrare la relazione tra i tipi di indicatori che possono essere usati per rilevare le attività di un attaccante e l'impatto che provoca il loro utilizzo nella negazione di quel determinato servizio. E' usata per l'illustrare anche l'obiettivo finale di sistemi dediti al rilevamento delle intrusioni. In Figura 2.1 è mostrata la pyramid of pain.

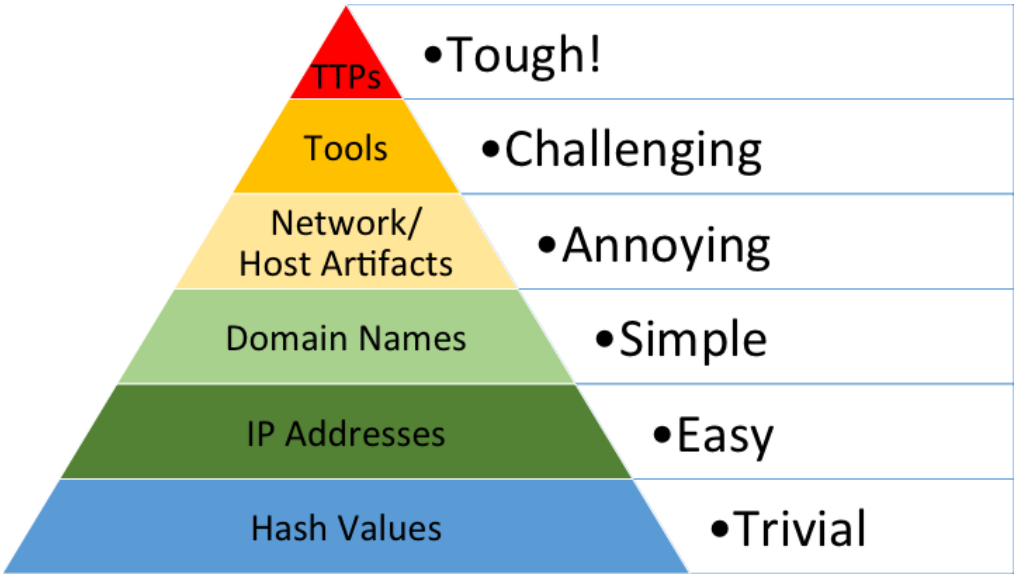


Figura 2.1. Pyramid of pain (fonte:detect-respond).

La piramide è strutturata a livelli, più si sale più è complesso ottenere quel tipo di informazione che diventa anche più importante. Andando verso l'alto diventa sempre più oneroso e difficile per l'attaccante modificare quel tipo di informazione, poiché, per esempio, è molto meno complesso modificare un indirizzo IP (utilizzando ad esempio VPN, TOR, Proxy) piuttosto che un TTPs (Tactics, Techniques and Procedures).

Tipi di Indicatori

Nella Pyramid of pain sono presenti sei tipi di indicatori che verranno adesso analizzati:

1. HASH VALUE:

   E' possibile usare hash, come ad esempio SHA1 o MD5, che corrispondono a specifici file sospetti o malevoli. Normalmente vengono impiegati per fornire riferimenti su uno specifico campione di malware o file coinvolti in un attacco. Questa informazione si trova alla base della piramide poiché è molto suscettibile anche a minimi cambiamenti e basta quindi cambiare una minima parte che l'hash generato sarà del tutto diverso da quello precedente, perdendo così il riferimento a quel determinato campione. Esistono tecniche chiamate hash fuzzy che permettono di confrontare due file o campioni che hanno delle notevoli somiglianze nel loro hash.

2. IP ADDRESSES:

   E' uno degli indicatori fondamentali in quanto un attaccante ha necessariamente bisogno di una connessione di rete al fine di effettuare l'attacco, e quindi le macchine che verranno usate avranno un indirizzo IP. Questa informazione risiede nella parte più bassa e larga della piramide poiché si possono avere un grande numero di indirizzi IP. In qualsiasi attacco visto in questa tesi, si è visto come l'attaccante cambia indirizzi IP molto frequentemente, quindi negargli l'uso di uno di questi o creare una 'lista nera', è una misura molto leggera in quanto possono essere cambiati con poco sforzo e senza rallentare di molto l'attacco.

3. DOMAIN NAMES:

   Appena sopra gli indirizzi IP troviamo i nomi di dominio. Anche loro si trovano nella parte verde della piramide poiché, rispetto agli indirizzi IP, richiedono più tempo per essere cambiati, ma non richiedono impiego di grandi risorse. Ad oggi esistono numerosi fornitori di DNS su internet, con gli standard di registrazione LAX e risulta quindi molto semplice cambiare dominio; questo passaggio richiede soltanto qualche giorno per permettere di essere utilizzabile su Internet e quindi rallenta di poco il processo d'attacco.

4. NETWORK/HOSTS ARTIFACTS:

   E' il primo indicatore il cui utilizzo può avere un impatto notevole sull'attacco. Molte volte si obbliga l'attaccante a dover interrompere l'attacco per dover riconfigurare le loro macchine. Un esempio può essere la scoperta di un determinato User Agent; se si bloccano le richieste provenienti da esso si costringe l'attaccante a capire come è stato rilevato e a cambiarlo. Il cambio può essere banale, ma non lo è il tempo impiegato dall'attaccante per lo studio del metodo di rilevazione.

5. TOOLS:

   Questo livello è giallo poiché implica all'attaccante di non poter più usare uno strumento a sua disposizione. Questo toglie tantissimo tempo e risorse sia durante la fase di attacco sia dopo, poiché è necessario trovare un software sostitutivo che adempie agli stessi compiti di quello precedentemente scoperto o addirittura, li dove l'attaccante ne sia capace, di scriverne uno nuovo. E' inoltre possibile anche la scoperta di un determinato protocollo di comunicazione che obbliga l'attaccante a dover cambiare metodo di attacco, andando a rivedere tutta la struttura utilizzata per quei fini.

6. TTPs:

   Infine all'apice della piramide troviamo le *Tactics, Techniques and Procedures* (TTP). Dal punto di vista dell'efficacia questo è il livello più alto che si può raggiungere poiché qui si

va ad intervenire direttamente sul comportamento dell'attaccante e non su gli strumenti da lui utilizzati, obbligandolo a rinunciare all'attacco o addirittura a re-inventarlo da zero.

2.2 Cyber Kill Chain

Con il termine 'Cyber Kill Chain' si definisce un modello a fasi che va ad analizzare i passaggi utilizzati da un attaccante per eseguire un attacco informatico. Questo schema può essere molto utile poiché divide i tipi di attacchi in fasi e per i difensori risulta più facile concentrarsi su uno dei singoli blocchi; inoltre lo si può usare anche come strumento di gestione per contribuire a migliorare la sicurezza della propria infrastruttura. Il modello descrive sette diverse fasi di attacco in cui ogni fase fornisce un input alla fase successiva; andandole ad analizzare più da vicino esse sono suddivise in:

1. RECONNAISSANCE:

   nel primo step l'attaccante mira ad ottenere più informazioni possibili circa la superficie di attacco esposta dal target e dunque a sua disposizione. In questa fase l'obiettivo è quello di concentrarsi sulla vittima, normalmente individui che hanno accessi privilegiati al sistema o sono in possesso di dati confidenziali. Esistono due metodi di ricognizione, passiva e attiva:

   - nel primo metodo l'attaccante cerca informazioni non correlate al dominio della vittima
   - nel secondo metodo l'attaccante cerca di utilizzare le informazioni ottenute per avere accesso a materiali confidenziali o addirittura per provare ad aggirare firewall o altri sistemi per ottenerlo.

2. WEAPONIZATION:

   nel secondo step l'attaccante, tramite le informazioni carpite al punto precedente, seleziona dei tools per cercare di creare un metodo di accesso remoto alla macchina infetta sfruttando un determinato exploit o una backdoor presente. In questa fase avviene la creazione di un payload usato poi nella fase 3. I tool più usati sono:

   - botnet: una rete composta da dispositivi infettati tramite malware e chiamati zombie e controllata da remoto da un master.
   - distributed denial of service (DDoS): attacco, tramite un elevato numero di richieste, il cui scopo è quello di saturare le risorse di un sistema o di una rete di computer che distribuisce diversi servizi con l'obiettivo di renderle irraggiungibili e inutilizzabili.
   - malware: software malevolo che viene iniettato in un sistema o in una rete. Alcuni dei più usati sono worm, virus, sniffer di pacchetti (utili per l'intercettazione di traffico su di una rete).

3. DELIVERY:

   nel terzo step l'attaccante recapita il payload (sviluppato nel passaggio precedente) al bersaglio scelto tramite due metodi, o tramite *controlled-delivery* o tramite *released-delivery*

   - nel primo metodo l'attaccante, tramite un hacking diretto, rilascia il payload malevolo.
   - nel secondo metodo l'attaccante trasmette il payload malevolo tramite varie tecniche come ad esempio:
     - phishing: invio di un messaggio di posta elettronica contenente allegati o link malevoli.
     - drive-by-download: download involontario di codice malevolo sul computer o dispositivo mobile che lascia la vittima esposta ad un possibile attacco informatico.

4. EXPLOITATION:

   nel quarto step l'attaccante si occupa di eseguire con successo il payload recapitato nel sistema della vittima per comprometterlo. Una volta che l'attaccante ha identificato la vulnerabilità nel sistema la sfrutta per portare a termine l'attacco. Durante questa fase può

succedere che l'attaccante installi un downloader per scaricare ulteriori malware da internet e aumentare l'arsenale a sua disposizione per tentare vari tipi di attacchi (es. escalation dei privilegi).

5. INSTALLATION:

   nel quinto step l'attaccante ha il compito di installare ed eseguire il malware in modo da poter aggirare i controlli di sicurezza e mantenere l'accesso nel sistema della vittima. L'installazione del malware avviene per mezzo del successo dell'exploit scelto durante la fase di weaponization, inviato durante la fase di delivery ed eseguito durante la fase di exploitation.

6. COMMAND & CONTROL

   nel sesto step l'attaccante stabilisce una connessione tra il sistema vittima e la macchina remota dal quale opera. In questo modo l'attaccante può avere un controllo persistente e un accesso continuo nell'ambiente della vittima. Il comando e controllo di una risorsa compromessa può avvenire in molteplici modi; ad oggi la maggior parte delle volte vengono usati dei covert channel che sfruttano i protocolli di rete e il cui traffico malevolo può essere scambiato come un normale traffico di rete.

7. ACTIONS ON OBJECTIVE:

   nel settimo ed ultimo anello della catena l'attaccante porta a termine l'attacco andando a colpire l'obiettivo prefissato. Per azione si intende il modo in cui l'attaccante riesce ad arrivare al suo obiettivo finale e lo raggiunge tramite l'uso di molteplici tecniche.

In Figura 2.2 è mostrata la cyber kill chain

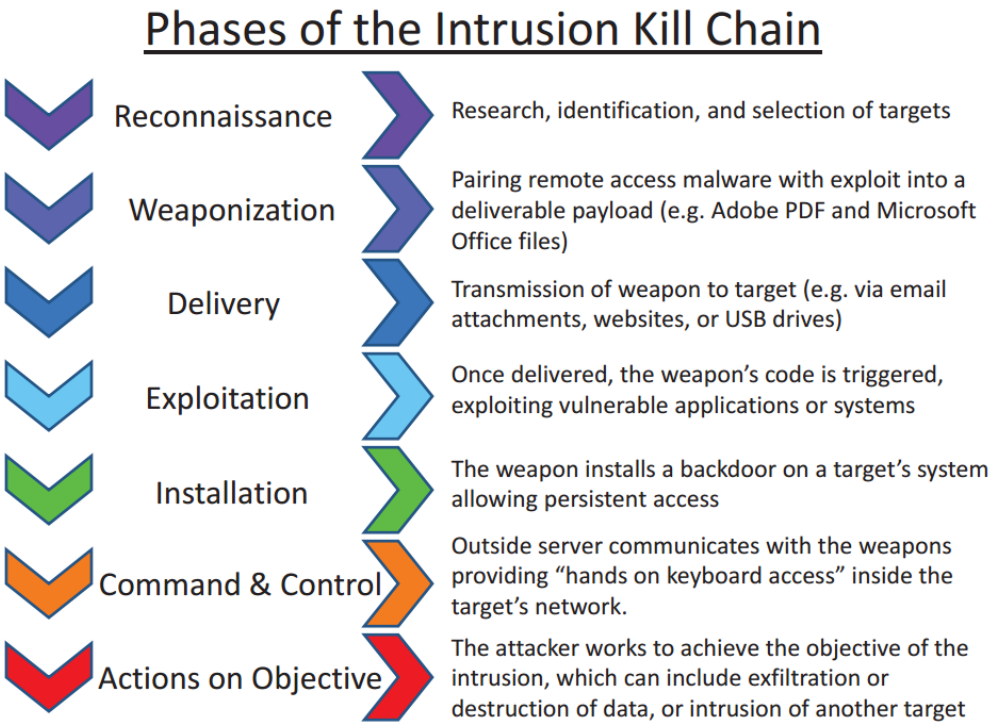


Figura 2.2. Cyber Kill Chain (fonte:wikipedia).

Tuttavia oggi secondo alcuni ricercatori lo schema appena mostrato non si rivela più un metodo totalmente affidabile vanificando magari sistemi di sicurezza basati su di essi. Un esempio possono essere gli *exploit zero day*, delle vulnerabilità ignote persino agli sviluppatori, che rendono inutile le fase di reconnaissance poiché si passa direttamente all'attacco e se l'exploit va a buon fine, l'attaccante ha avuto accesso al sistema.

Si è inoltre visto durante lo studio che le tecnologie cloud hanno introdotto nuovi punti d'accesso poiché esse si appoggiano ad infrastrutture di terze parti che sono fuori dal controllo della compagnia e queste possono rivelarsi molto dannose. Si può quindi concludere che non è più possibile focalizzarsi solo sulle aree ritenute più a rischio poiché *l'attack surface* [1] e oggi troppo estesa e quindi i vecchi metodi sono destinati ad essere rivisitati o addirittura a fallire.

Sulla base dell'ultimo step della Cyber Kill Chain, il Mitre ha definito quattro matrici, per le principali quattro piattaforme (Windows, Mac, Linux, Mobile), che forniscono una rappresentazione schematica delle possibili tecniche utilizzabili.

La matrice, detta ATT&CK Matrix [1], comprende centinaia di tecniche e può essere applicata per diversi scopi; normalmente viene usata per capire il comportamento degli attaccanti, ma è anche un buon metodo per avere una visione più ampia su ciò che viene usato per determinati attacchi. Usando questa matrice è anche possibile mappare una buona tecnica difensiva che può essere utile ad un'organizzazione come metodo per conoscere l'efficacia delle proprie tecniche difensive.

2.3 Diamond Model

In Figura 2.3 è mostrato il Diamond Model.

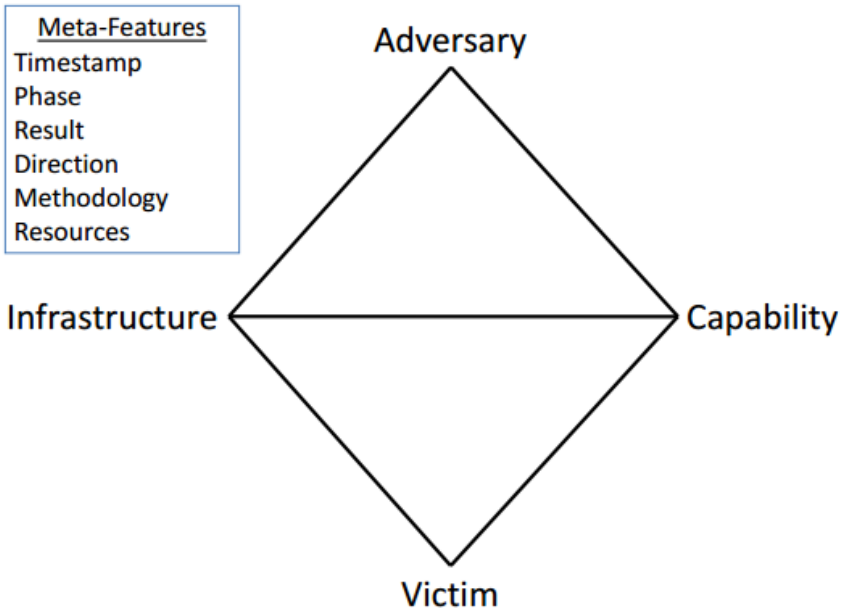


Figura 2.3. Diamond Model (fonte:cybersecportal).

E' un modello usato per il monitoraggio e l'analisi delle intrusioni informatiche e prende il nome dalla forma che creano gli elementi connessi. Serve a descrivere le relazione tra i quattro componenti principali di un'intrusione che sono:

1. avversario

[1] l'attack surface di un sistema è quella parte del sistema stesso che può essere esposta ad accesso o a modifiche di utenti non autorizzati. Tanto maggiore è la superficie, tanto più il sistema è vulnerabile.

2. capacità
3. infrastruttura
4. vittima

L'assioma principale spiega che: 'per ogni intrusione esiste un avversario che si muove verso un certo obiettivo, utilizzando una capacità dell'infrastruttura contro una determinata vittima, per arrivare ad un determinato risultato'.[2]

Con *intrusione* si definisce un'attività a tempo limitato, dove un attaccante utilizza una certa metodologia su di una infrastruttura avversaria, al fine di arrivare ad un determinato risultato.

Le caratteristiche principali di un intrusione sono: *adversary, capability, infrastructure* and *victim.* Esse sono collegate tra di loro per rappresentare le relazioni fondamentali tra le funzionalità che possono essere sfruttate per scoprire o sviluppare la conoscenza di attività dannose.

Adversary

Un avversario è una persona o un'organizzazione responsabile dell'utilizzo di una determinata *capability* per raggiungere il proprio scopo. La conoscenza dell'avversario rischia di essere 'nulla' per la maggior parte delle intrusioni, almeno fino al momento della scoperta.

Capability

Descrive gli strumenti o le tecniche utilizzate dall'attaccante durante l'intrusione. La flessibilità del modello consente la descrizione delle *capability* con sufficiente fedeltà. Qui sono inoltre comprese anche le capacità individuali di un attaccante di riuscire ad usare vulnerabilità o exploit ai fini di un attacco.[2]

Infrastructure

Descrive le strutture di comunicazione utilizzate dall'attaccante per avere il controllo sul sistema vittima e riuscire nel suo scopo (es. esfiltrazione dei dati). Alcuni esempi possono includere indirizzi IP, nomi di dominio, indirizzi e-mail ecc. Bisogna inoltre distinguere due tipi di infrastructure:

1. controllata o posseduta interamente dall'avversario.
2. controllata da un intermediario (intenzionale o involontario). Serve a offuscare l'origine e l'attribuzione dell'attività malevola dell'attaccante. Questa infrastruttura include host zombie, server di gestione temporanea del malware, nomi di dominio dannosi, account e-mail compromessi, ecc.

Victim

E' l'obiettivo dell'attaccante e colui contro il quale vengono sfruttate le vulnerabilità e usate le capability. Una vittima può essere un'organizzazione, una persona, un determinato dominio ecc.

2.4 Kill Chain e Diamond Model

La Kill Chain e il Diamond Model possono fondersi anche in un unico modello dando vita allo schema rappresentato in Figura 2.4 dove in alcune fasi della Kill Chain può essere sfruttato anche il Diamond Model per avere una visione ancora più accurata dell'attacco.

In figura viene riportato il modello con le varie fasi ed un esempio che sarà descritto di seguito:

1. L'attaccante conduce una ricerca sul web per la società vittima *Gadgets Inc.* e nei risultati di questa ricerca trova il dominio gadgets.com.

[2] In questo modello il comando e controllo è inteso come caratteristica della capability.

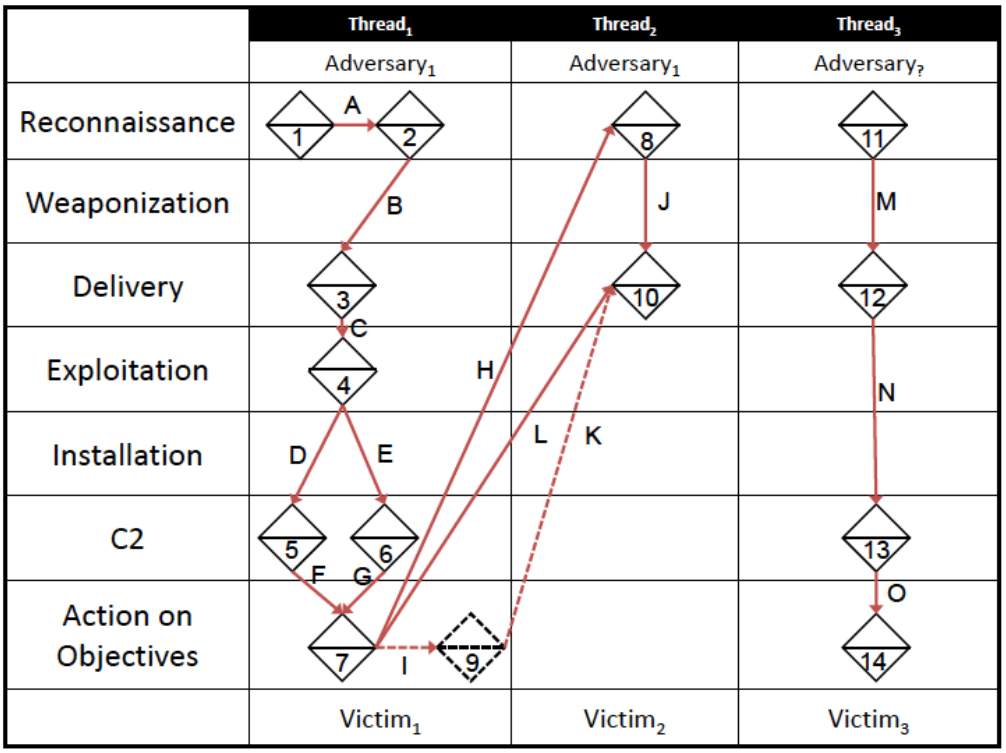


Figura 2.4. Kill Chain e Diamond Model.

2. L'attaccante utilizza il nuovo dominio scoperto per una nuova ricerca del tipo 'network administrator gadget.com' che mostra dei post su alcuni forum, con i relativi indirizzi e-mail, di utenti che affermano di essere amministratori di rete della società Gadgets Inc.

3. L'attaccante invia delle e-mail di spear phishing per un attacco di tipo trojan agli amministratori di rete di gadget.com scoperti nella fase 2.

4. Un amministratore di rete (NA1) di gadet.com apre l'allegato malevolo che esegue l'exploit, portando all'esecuzione sulla macchina di ulteriore codice di cui l'utente non si rende conto.

5. L'host dell'amministratore di rete (NA1) che ha eseguito l'exploit è ora compromesso. Viene inviato automaticamente dal malware installato sulla macchina vittima un messaggio di tipo HTTP POST a un indirizzo IP malevolo.

6. Se dal primo indirizzo IP non si riceve nessuna risposta, il malware ha un secondo indirizzo IP configurato da usare in caso di non risposta.

7. Tramite un messaggio HTTP il client comunica con il suo centro di comando e controllo.

8. Tramite il proxy installato sull'host vittima (NA1), Adversary1 esegue una ricerca sul Web e trova la società *Interesting Research Inc*, di cui fa parte la sua vittima finale.

9. L'attaccante controlla l'elenco dei contatti e-mail nella macchina vittima (NA1) per provare a trovare degli indirizzi utili della società Interesting Research Inc. per portare avanti il suo attacco. Riesce a scovare il contatto dello Chief Research Officer della società in questione.

10. L'attaccante invia a questo contatto un'e-mail di spear-phishing dall'indirizzo e-mail dell'amministratore di rete (NA1) di Gadget.Inc, inserendo un allegato malevolo come nella fase 3.

11. A questo punto vengono analizzati i server Web vulnerabili, incluso quello di Victim3 (Chief Research Officer).

12. Viene consegnato a Victim3 un exploit per una vulnerabilità scoperta durante la fase 10.
13. Viene installata una reverse shell per controllare Victim3.
14. L'avversario usa la shell remota per scaricare tutti i documenti della directory privata di Victim3.

In questo modello le attività sono divise in thread e ogni evento ha una relazione all'interno dei due modelli. Questi eventi sono connessi tra di loro tramite dei percorsi (freccie rosse) che possono essere necessari o opzionali (in questo caso potenzialmente per quell'evento ci sono più percorsi percorribili). Inoltre nel caso in cui la freccia non è tratteggiata allora l'informazione o la risorsa di quella fase è necessaria per il verificarsi dell'evento successivo.

I thread sono organizzati verticalmente in modo tale che ognuno descrivi gli eventi che un avversario esegue contro quella specifica vittima. Vengono quindi divise le fasi in thread specifici per coppia di attaccante-vittima, e come visto in questo esempio l'attaccante opera contro tre vittime (Società Gadgets Inc., amministratore di rete, Chief Research Officer) in momenti separati.

2.5 Gruppi APT

In questa sezione verranno descritti gli APT sia come tipo di malware che come cyber-criminali altamente qualificati che offrono prodotti e strumenti tra i più avanzati e sofisticati, capaci di colpire singoli individui, aziende, organizzazioni, enti governativi.

Dei gruppi studiati verrà inoltre descritto anche il loro target, la loro connotazione geografica e i software usati.

Il termine *Advanced Persistent Threat* (APT) comprende due concetti:

1. tipologia d'attacco persistente e mirato contro un determinato bersaglio
2. gruppi di cyber-criminali finanziati e ben strutturati che hanno come obiettivo operazioni di spionaggio e si occupano dei maggiori attacchi informatici a livello mondiale, spinti anche da motivazioni politiche o religiose.

Nel primo caso è stato possibile notare, durante la fase di studio, che gli attacchi di questo tipo mirano a compromettere una o più macchine contenenti specifici dati di interesse dell'attaccante. Essi sono creati per rimanere nascosti, in quanto l'obiettivo non è causare un danno o un'interruzione di qualche servizio ma l'intento è quello di rubare dati altamente riservati o segreti industriali ad aziende, enti governativi o compagnie. Riescono a rimanere nascosti poiché vengono creati quasi sempre per una specifica funzione e cancellati o sospesi una volta finita.

Per riuscire a contenere questo tipo di minacce è necessario avere degli strumenti, dediti all'analisi approfondita di file, che forniscano quasi in tempo reale un'esecuzione dell'oggetto, eseguendone tutti gli step dell'attacco e rilevando in maniera corretta eventuali azioni o componenti compromessi sull'end-point infetto. Questo permette al team di Incident Response di poter decidere in maniera rapida ed efficace le azioni di difesa da intraprendere. Si è inoltre evidenziata la difficoltà, una volta che l'APT è in atto, di discernere cosa fa parte del normale flusso operativo del sistema e cosa invece fa parte del flusso delle operazioni compiute dall'APT.

Nel secondo caso invece si è visto come per APT si intendono cyber-criminali ben organizzati e strutturati, quasi sempre finanziati da enti terzi, e dediti ad attacchi su larga scala. Allo stato attuale molti di questi gruppi, che sono stati presi in analisi, hanno iniziato a colpire i propri bersagli mediante l'uso di tecniche come i covert channel e ad organizzare degli attacchi a catena fino ad arrivare all'obiettivo prescelto.

I cyber-criminali di maggiore rilievo e con specifiche competenze appartengono prevalentemente a: Cina, America Latina, e Europa dell'Est per gli attacchi attraverso malware; Russia, Romania, Lituania, Ucraina e altri paesi dell'Est Europa per gli attacchi alle istituzioni finanziarie; Vietnam per le minacce relative all'e-commerce, e Stati Uniti d'America per gli attacchi finanziari.

Di seguito verranno riportati tutti i gruppi studiati e in particolare quelli analizzati da FireEye con un focus sui settori di attività e i software associati.

1. APT 1

Figura 2.5. APT1 (fonte:FireEye).

Gruppo di origine cinese che è stato attribuito al 2° Ufficio di presidenza del GSD e al 3° Dipartimento del PLA, comunemente noto come Unità 61398.

- Nazionalità di origine: China's People's Liberation Army (PLA), General Staff Department's (GSD)
- Altri nomi con cui sono noti: Comment Crew, Comment Group, Comment Panda
- Settore target: IT, aereospaziale, pubblica amministrazione, telecomunicazioni, energia, trasporti, navale, chimico, alimentare, istruzione.
- Software Usati: WEBC2, BISCUIT, GLOOXMAIL [3]
- Vettore d'attacco: APT1 utilizza tecniche di phishing per provare a compromettere i propri obiettivi. Le email usate contengono sia un allegato malevolo che un collegamento iper testuale ad un file malevolo. Fanno uso di email con nomi reali, quindi questo fa presupporre che ci siano state anche delle campagne di social engineering. Usano inoltre delle backdoor personalizzate in modo che se una di esse viene scoperta può facilmente essere rimpiazzata.

2. APT 2

Figura 2.6. APT2 (fonte:miguelbigueur).

- Nazionalità di origine: China's People's Liberation Army (PLA), General Staff Department's (GSD)
- Altri nomi con cui sono noti: Putter Panda, MSUpdater
- Settore target: aereospaziale, società di ricerca nello spazio, telecomunicazioni satellitari
- Software Usati: 3PARA RAT, 4H RAT, pngdowner, httpclient [4]
- Vettore d'attacco: APT2 utilizza campagne mirate di phishing per provare ad inviare emali con allegati malevoli utili ad installare le proprie backdoor.

3. APT 18

Gruppo di origine cinese che opera dal 2009 e ha preso di mira una vasta gamma di settori.

- Nazionalità di origine: Cina
- Altri nomi con cui sono noti: Threat Group-0416, TG-0416 - 2, Dynamite Panda
- Settore target: industria manifatturiera, industria sanitaria, aereospaziale e della difesa, gruppi per i diritti umani, governi

- Software Usati: Gh0st, HTTPBrowser, Pisloader, cmdhcdLoader, WIN32.ISMDOOR.GEN, WIN32.BACKDOOR.DENIS. [5]
- Vettore d'attacco: APT18 ha usato campagne di social engineering per colpire principalmente i settori farmaceutici Americani.

4. APT 19

Figura 2.7. APT19 (fonte:FireEye).

- Nazionalità di origine: Cina
- Altri nomi con cui sono noti: Codoso Team
- Settore target: settore giuridico e imprese di investimenti nel territorio Cinese.
- Software Usati: COBALT STRIKE. [6]
- Vettore d'attacco: APT19 utilizza diverse tecniche per tentare di compromettere gli obiettivi. Ha usato campagne di phishing per provare a sfruttare la vulnerabilità di Microsoft Windows descritta in CVE 2017-0199, oppure ha provato ad usare delle vulnerabilità nelle macro dei documenti di Microsoft Excel (XLSM) per caricare un payload Cobalt Strike.

5. APT 32

Figura 2.8. APT32 (fonte:FireEye).

Gruppo di origine Vietnamita sospettato di essere attivo dal 2014.

Il gruppo ha preso di mira diversi settori dei governi stranieri, con particolare interesse su coloro che esportano in Vietnam.

- Nazionalità di origine: Vietnam
- Altri nomi con cui sono noti: OceanLotus
- Settore target: imprese straniere che investono sui prodotti di consumo in Vietnam.
- Software Usati: SOUNDBITE, WINDSHIELD, PHOREAL, COBALT STRIKE, KOMPROGO, UDPos.[7]
- Vettore d'attacco: APT32 ha impiegato metodi di social engineering e phishing per permettere agli utenti di scaricare dei file con all'interno un payload malevolo.

6. APT 33

Figura 2.9. APT33 (fonte:FireEye).

Gruppo di origine iraniana sospettato di aver effettuato operazioni di spionaggio informatico dal 2013.

Il gruppo ha preso di mira le organizzazioni in diversi settori degli Stati Uniti, dell'Arabia Saudita e della Corea del Sud, con un particolare interesse nei settori dell'aviazione e dell'energia.

- Nazionalità di origine: Iran
- Settore target: aereospaziale, energia
- Software Usati: shapeshift, Dropshot, TURNEDUP, NANOCORE, NetWire, ALFA Shell. [8]
- Vettore d'attacco: APT33 ha portato avanti una campagna di phishing contro i dipendenti legati al settore del trasporto aereo. Queste email contenevano un link malevolo ad una pagina HTML (.hta). Da qui venivano poi scaricati ed installati alcuni dei malware a loro disposizione.

7. APT 34

Figura 2.10. APT34 (fonte:FireEye).

Gruppo di origine iraniana attivo dal 2014.

Ha preso di mira una vasta gamma di settori, tra cui quello finanziario, della pubblica amministrazione, dell'energia, della chimica, e delle telecomunicazioni, e ha focalizzato la propria attività all'interno del Medio Oriente. FireEye pensa che il gruppo lavori per conto del governo iraniano.

- Nazionalità di origine: Iran
- Altri nomi con cui sono noti: OilRig
- Settore target: finanza, pubblica amministrazione, energia, telecomunicazioni
- Software Usati: POWBAT, POWRUNER, BONDUPDATER, HELMINTH. [9]
- Vettore d'attacco: nella sua ultima campagna, APT34 ha sfruttato alcune vulnerabilità di Microsoft Office CVE-2017-11882 per implementare POWRUNER e BONDUPDATER.

Capitolo 3

Covert-Channel

In questo capitolo verrà affrontato l'argomento centrale della tesi, andando a focalizzare lo studio sui covert channel, facendone una descrizione e andandone a vedere le varie tipologie incontrate durante la fase di analisi. Verrà infine descritta la tipologia di comunicazione nascosta che si è scelta di prendere in studio e che sarà centrale per la parte di sviluppo.

3.1 Descrizione Covert-Channel

L'European Union Agency for Network and Information Security (ENISA) nel 'Threat Landscape Report del 2014' [10], ha indicato come sono in crescente aumento il numero di attacchi mirati a grandi compagnie per il furto di informazioni. Questi attacchi mirano alle proprietà intellettuali, ai dati delle società o alle informazioni sensibili dei governi e usano sofisticati metodi di attacco per la compromissione di sistemi e l'estrazione di dati.

In figura 3.1 è possibile osservare le fasi di un'attacco di questo tipo.

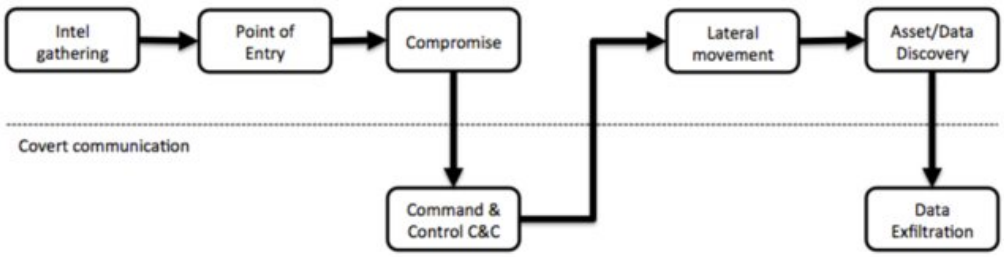


Figura 3.1. Fase di un attacco (fonte:researchgate).

Gli attaccanti cercano di rendere silente ognuno di questi passaggi per evitare di essere scoperti.

Come illustrato in figura però le fasi di comando e controllo (C2) e di data exfiltration sono quelle dove viene generato più rumore in termini di traffico di rete e quindi risultano le fasi di maggior interesse per l'analisi.

Questo genere di attacchi consente di raccogliere informazioni sulle macchine vittime, di comunicare con il malware all'interno del sistema infetto e di istruirlo per portare avanti la fase di estrazione dei dati. Il traffico generato tra il malware e il suo server C2 ha un importanza rilevante per il successo di un attacco, infatti è di interesse primario per un attaccante cercare di mascherare il più possibile queste comunicazioni.

Nell'ultimo decennio, grazie anche all'aumento del traffico di rete generato, c'è stato un incremento per quanto riguarda l'uso dei canali nascosti mediante i protocolli di rete; in particolare

HTTP e DNS sono risultati quelli più usati per veicolare informazioni nascoste durante l'attuazione di un attacco. La struttura di questi protocolli permette un gran numero di modi in cui i dati possono essere codificati al loro interno, questo perché quando sono stati standardizzati, non si è posta tanta attenzione ai problemi di sicurezza che potessero derivarne da un uso improprio.

Nella prossima sezione andremo ad analizzare l'uso dei covert channel tramite questi due protocolli.

3.1.1 Tipologie

Durante la fase di analisi si è visto come questi due protocolli siano quelli più sfruttati durante le fasi di un attacco. In questa sezione verranno analizzati nel loro uso per una comunicazione nascosta. Ci si concentrerà invece, nella prossima sezione, sull'uso del record A del DNS come mezzo di comunicazione nascosta, che è stato l'oggetto principale di studio di questa tesi.

HTTP Covert Channel

L'HyperText Transfer Protocol (HTTP) è un protocollo a livello applicativo usato come principale strumento per la trasmissione di dati sul web in una architettura client-server. Tutte le specifiche di questo protocollo vengono gestite dal World Wide Web Consortium (W3C). Il suo funzionamento prevede che il client esegua una richiesta, e il server restituisca una risposta. Normalmente il client è il browser web mentre il server è la macchina su cui risiede il sito web. Esistono quindi due tipi di messaggi HTTP: messaggi di richiesta e messaggi di risposta.

HTTP differisce dagli altri protocolli a livello 7 poiché ogni qualvolta una o più richieste sono state soddisfatte, chiude la connessione. Alcune volte però la sua natura senza stato della sessione di navigazione (*stateless*) pone agli sviluppatori dei problemi sulla conservazione dello stato dell'utente, costringendoli ad usare altri strumenti come ad esempio i cookie. [1]

Il messaggio di richiesta è composto da quattro parti:

- riga di richiesta (request line)
- sezione header (informazioni aggiuntive)
- riga vuota (CRLF: i 2 caratteri carriage return e line feed)
- body (corpo del messaggio)

Il messaggio di richiesta è composto da metodo, URI e versione del protocollo. Il metodi di richiesta sono: GET, POST, HEAD, PUT, DELETE, TRACE, OPTIONS, CONNECT.

Il messaggio di risposta è di tipo testuale ed è composto da quattro parti: riga di stato (status-line), sezione header, riga vuota (CRLF: i 2 caratteri carriage return e line feed), body (contenuto della risposta).

La riga di stato contiene al suo interno un codice a tre cifre che riporta lo stato del risultato; ne esistono di vari tipi es. 404, 200, 500 ecc, dove ognuno al suo interno riporterà il risultato di una richiesta effettuata.

La flessibilità di questo protocollo ha fatto si che venisse sfruttato anche come canale di comunicazione nascosto per il trasferimento di dati. Esistono vari modi di usare un covert channel tramite HTTP e qui di seguito saranno elencati alcuni di quelli visti durante la fase di studio.

Cookie

[1] sistema che permette un aumento di efficienza poiché il numero di connessioni attive si limita a solo quelle necessarie, diminuendo così il carico e l'occupazione sia sul client che sul server

I cookie vengono utilizzati dalle applicazioni web (lato server) che archiviano e recuperano informazioni a lungo termine riguardo un determinato client. I server inviano il cookie all'interno della risposta HTTP in modo da permettere al browser di salvarlo e inviarlo nuovamente quando vengono fatte richieste aggiuntive.

Sono composti da quattro attributi che sono: coppia *nome/valore*, *scadenza*, *modalità d'accesso* e *secure*. Di questi quattro solo il primo è obbligatorio, mentre gli ultimi tre sono opzionali. L'accesso al cookie è possibile farlo non solo dal server che lo ha generato ma anche da qualsiasi altro server che sia all'interno dello stesso dominio. Esistono servizi DNS (es. dyndns.org) che permettono di posizionare dei server in maniera arbitraria nello stesso dominio che si trova all'interno del cookie, potendo permettere così lo scambio di dati quando il client si connette. Poiché i dati sono memorizzati al suo interno, i cookie possono quindi diventare un canale di comunicazione nascosta.

Il throughput possibile utilizzando i cookie è pari a 160KB di dati, questo perché un singolo cookie può memorizzare fino a 4KB nel campo nome/valore ed essendo possibile memorizzare fino ad un massimo di 40 cookie per server sarà possibile condividere 160KB di dati. Questa tecnica risulta però poco robusta poiché è possibile cancellare i cookie (lato client) in qualsiasi momento, interrompendo di fatto la comunicazione nascosta.

Referer Tag

Nelle richieste HTTP (es. GET, POST) esiste un tag opzionale chiamato *referer*, che può essere manipolato per incorporare informazioni diverse rispetto a quelle della pagina visitata. Ha un throughput pari a 1024 byte in quanto è limitato alla lunghezza standard di un URL. Questo canale è estremamente robusto poiché i dati memorizzati nel tag referer rimangono generalmente all'interno dei proxy server e pochi dispositivi di rete interferirebbero con esso, rendendo questo canale di comunicazione altamente affidabile.

Active content

Queste tecniche sfruttano il codice eseguito sul client (es. Javascript, Flash, ActiveX) per creare canali di comunicazione nascosta tra vari script in esecuzione su diversi server e client web. Questi script potrebbero utilizzare qualsiasi tipo di meccanismo per nascondere i dati, come ad esempio l'apertura di porte TCP aggiuntive oppure l'invio di form HTML arbitrari. Ad esempio tramite Javascript è possibile creare form HTML invisibili ad un normale utente, per mandare dati ad un altro server malevolo. Il throughput varia in base al tipo di tecnica scelta; con quella descritta prima il Javascript crea e popola un form HTML nascosto e lo invia ad un server, quindi il limite di dati possibili da inviare è imposto soltanto dal protocollo HTTP. A livello di robustezza si è visto però che queste tecniche sono ad oggi poco efficaci poiché esistono molteplici plug-in o software in grado di impedire l'esecuzione di 'active content' e di interrompere quindi facilmente questo tipo di comunicazioni.

In fase di studio è stato visto come queste tecniche vengono usate da alcuni dei gruppi APT descritti sopra, i quali tramite i loro software sfruttano queste vulnerabilità del protocollo HTTP.

DNS Covert Channel

Il protocollo DNS è un sistema che viene usato per la risoluzione di nomi di host in indirizzi IP. Esiste un sistema di database distribuiti realizzato mediante server DNS. Questo protocollo ha una struttura gerarchica e una suddivisione in domini (es. it, com). Ad ogni dominio corrisponde un nameserver, al cui interno sono presenti delle informazioni sui domini di cui è responsabile. Quando non ha le informazioni riguardante il suo dominio, esso si rivolge ad altri nameserver. I server responsabili del dominio radice sono i cosiddetti *root nameservers* o *root domain*, che possiedono l'elenco dei server autoritativi di tutti i domini di primo livello (TLD) riconosciuti e lo forniscono in risposta a ciascuna richiesta.

In Figura 3.2 è possibile osservare la struttura del DNS.

Allo stato attuale si è visto come molti gruppi APT stanno sviluppando tecniche di covert channel mediante l'uso del protocollo DNS per inviare e ricevere informazioni dal server C2. Questo meccanismo risulta efficace poiché esistono diversi exploit che sfruttano la natura non

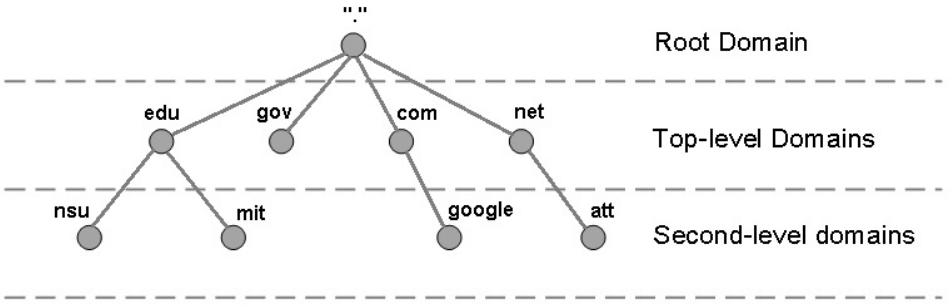


Figura 3.2. gerarchia DNS (fonte:itgeared).

protetta del DNS e la sua bidirezionalità. Molte volte in un perimetro aziendale sistemi che monitorano il traffico di rete non bloccano o controllano il traffico DNS poiché esso è necessario per un corretto funzionamento della rete. Questo espone il protocollo ad un uso improprio da parte di un attaccante, che lo usa come vettore per un attacco.

I protocolli utili alla comunicazione su internet forniscono un numero illimitato di modi in cui essi possono essere sfruttati per nascondere dati in mezzo al normale traffico di rete. Inoltre oggi, grazie alla grande quantità di banda a cui abbiamo accesso, è possibile estrarre sempre più quantità di dati.

Secondo RFC 1035 [11] un domain name può essere costituito da caratteri ASCII. Ognuno di questi caratteri è memorizzato con 8 bit ed è identificato come un ottetto. Un Fully qualified domain name (FQDN) è espresso in una sequenza di etichette.

In Figura 3.3 è mostrato un esempio per il domain name 'www.itb.ie'

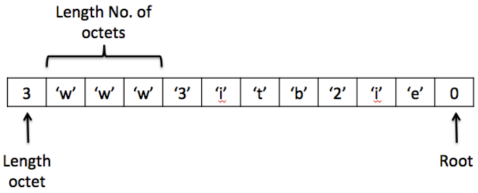


Figura 3.3. esempio FQDN.

La dimensione massima per qualsiasi domain name è pari a 255 ottetti meno il primo a sinistra e l'ultimo che indica la root, quindi in tutto la dimensione utilizzabile è pari a 253 ottetti.

Quando digitiamo sul browser un indirizzo, il browser comunica con server DNS per risolvere il nome di dominio e tradurre quel determinato URL in un indirizzo IP. I server DNS sono distribuiti e l'interrogazione avviene su questi database situati in tutto il mondo. In Figura 3.4 possiamo vedere le fasi del processo di risoluzione di un indirizzo IP per la richiesta 'www.mydomain.com'

1. il client richiede un record A al server DNS locale per la risoluzione in un indirizzo IP dell'URL www.mydomain.com.
2. il server DNS riceve la richiesta e la inoltra a uno dei 13 root server DNS.
3. il root server DNS non conosce quel determinato dominio, ma risponde con un riferimento ai server DNS di livello superiore. In questo caso il Global Top Level Domain (GTLD) responsabile del dominio .com.
4. Il server DNS locale richiede la risoluzione dell'indirizzo dal server DNS di livello superiore responsabile per il dominio .com.

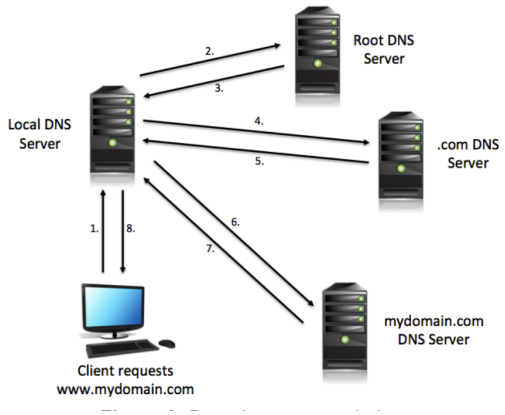


Figura 3.4. flusso query DNS.

5. il server DNS di livello superiore risponderà con un riferimento al server DNS di secondo livello che in questo caso è il server responsabile per il dominio .com.
6. il server DNS locale richiede la risoluzione dell'indirizzo dal server DNS di secondo livello (server .com).
7. in questo caso il server DNS di secondo livello è autorevole, risponderà con l'IP indirizzo dell'host.
8. Il server DNS locale invia al client la risposta.

Ad un nome DNS inoltre possono corrispondere varie informazioni, infatti esistono diversi tipi di record DNS. I principali sono:

- record A: indica la corrispondenza tra un nome e uno o più indirizzi IP

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                      ADDRESS                  |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

```
where:

ADDRESS         A 32 bit Internet address.

Hosts that have multiple Internet addresses will have multiple A
records.
```

- record AAAA: uguale al record A ma restituisce un indirizzo IPv6
- record TXT: associa ad un dominio un campo di testo arbitrario in cui può essere contenuta una descrizione o può essere utilizzata per la realizzazione di altri servizi

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
/                    TXT-DATA                   /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

```
where:

TXT-DATA        One or more <character-string>s.

TXT RRs are used to hold descriptive text.  The semantics of the text
depends on the domain where it is found.
```

- record CNAME: restituisce un alias per un dato host

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
/                     CNAME                     /
/                                               /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

```
where:

CNAME           A <domain-name> which specifies the canonical or primary
                name for the owner.  The owner name is an alias.
```

- record NS: restituisce l'IP del server DNS autorevole per il dominio interrogato

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
/                    NSDNAME                      /
/                                                 /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

```
where:

NSDNAME         A <domain-name> which specifies a host which should be
                authoritative for the specified class and domain.
```

- record MX: indica a quali server inviare la posta elettronica per quel determinato dominio

```
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                   PREFERENCE                    |
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
/                    EXCHANGE                     /
/                                                 /
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

```
where:

PREFERENCE      A 16 bit integer which specifies the preference given to
                this RR among others at the same owner.  Lower values
                are preferred.
```

Alcuni di questi record non hanno un controllo sulla dimensione massima, e questo può diventare un mezzo per consentirne un uso improprio. Ad esempio è possibile inviare richieste DNS con URL lunghi come 'http://hostname.group.mail.domain.com' ma è possibile inserire al posto di 'hostname.group.mail' qualsiasi istruzione da inviare al server di comando e controllo. Questo tipo di informazioni malevole può viaggiare sia in chiaro o può anche essere codificata.

Esistono diversi exploit che sfruttano la natura non protetta del DNS e la sua bidirezionalità.

I record più usati per inserire dati utili alla comunicazione tramite il client infetto e il centro di comando e controllo sono: il record TXT e il record A. Il record TXT consente inoltre una maggiore flessibilità nell'uso dei caratteri che possono essere codificati anche in Base64.

Durante le fasi di analisi è stato studiato il malware *pisloader* appartenente al gruppo APT18 che fa uso del record TXT. Qui in Figura 3.5 verranno dimostrati alcuni passaggi con cui gli attaccanti inseriscono informazioni nascoste per far comunicare il client infetto con il suo C2 in una normale comunicazione DNS.

In questa figura il record TXT inviato dal malware al suo server presenta una sequenza di 9 lettere maiuscole, seguite da un numero casuale, seguito a sua volta da un'altra sequenza di 7 lettere maiuscole e il dominio C2 utilizzato. Pisloader invia periodicamente una richiesta DNS all'indirizzo del server C2 che è codificato nel malware.

Il server risponde con un record TXT che può contenere vari comandi per il malware utili per raccogliere informazioni sul sistema, elencare le informazioni sui file per una directory specifica, caricare un file sulla macchina infetta e avviare una reverse shell.

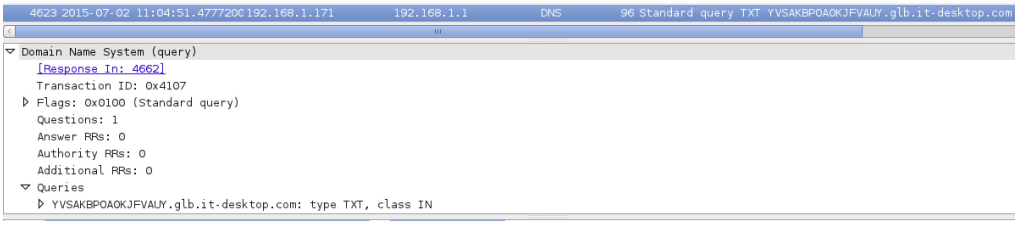


Figura 3.5. record TXT pisloader APT18 (fonte:anomali).

È possibile trasferire un massimo di 255 byte di dati utilizzando questo metodo, in questo modo il processo è più lento ma va bene per un attacco a lungo termine.

La risposta inviata dal server contiene un record TXT codificato in modo simile alla richiesta iniziale. Nella risposta, il primo byte viene ignorato e i dati rimanenti sono codificati in base32.

Un esempio di questo lo possiamo vedere in Figura 3.6:

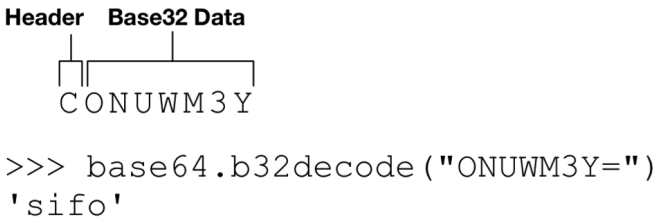


Figura 3.6. record TXT server C2 pisloader (fonte:researchcenter paloaltonetworks).

Questi sono i comandi e le loro descrizioni che il malware supporta:

- sifo - Raccogli le informazioni sul sistema delle vittime
- drive - Elenca le unità sulla macchina vittima
- lista - elenca le informazioni di uno o più file
- upload - Carica un file sul computer della vittima
- open - Crea una shell di comando

3.1.2 Record A DNS

Lo studio di tesi su questi protocolli si è concentrato infine sull'analisi dell'utilizzo, da parte di un attaccante, del record A del DNS per la comunicazione tra il malware e il suo centro di comando e controllo.

Queste tecniche permettono di inserire del carico utile all'interno dei record del DNS (es. record A), con l'obiettivo di utilizzarlo per l'esfiltrazione di dati o per il comando e controllo. L'attaccante oltre a riuscire a compromettere i sistemi o l'organizzazione destinataria dell'attacco, deve anche possedere dei domini registrati e dei server autorevoli per quei determinati domini, al fine di poter eseguire i programmi di tunneling lato server.

In una presentazione del 2012 alla conferenza dell'RSA, Ed Skoudis ha definito gli attacchi malware che sfruttano il DNS per il comando e controllo, come i nuovi attacchi più pericolosi, mentre la prima volta che l'argomento fu trattato è stato grazie a Oskar Pearson che nel 1998 portò alla luce questo problema. Da quel momento in poi sono stati creati molti software per il tunneling DNS che variano tra di loro principalmente per la codifica o per dei dettagli implementativi. Il principio alla base invece è quasi sempre lo stesso poiché è prevista una codifica dei dati che si

vogliono inviare e ricevere all'interno del payload DNS; un esempio di alto livello può essere il seguente:

- il client desidera inviare al suo C2 dei dati. Esso li inserirà all'interno del payload DNS (in questo caso usando il record A) codificandoli all'interno del nome host in questa maniera:

  MRZGS3TLEBWW64TFEBXXMYLMORUW4ZI.t.example.com

- il server ricevuti questi dati decodificherà la richiesta e invierà una risposta simile alla seguente:

  NVWW2IDPOZQWY5DJNZSQ.t.example.com

- il client, a questo punto, decodificherà i dati e li interpreterà come un comando che verrà eseguito sulla macchina vittima per portare avanti l'attacco.

In questo modo tutti i dati possono essere codificati e inviati tra il server ed il client. Può essere necessario però che sia il server ad avviare la comunicazione, ma in questo caso non può farlo direttamente; proprio per questo il client sta sempre in uno stato di polling aspettando che il server mandi dei pacchetti per far capire che è attivo e che può iniziare una comunicazione.

Ci sono anche una serie di limiti su come i dati possono essere inseriti nel payload DNS; prima di tutto devono essere immessi nel campo *hostname* che è strutturato in etichette e accetta 255 caratteri in totale. I caratteri possono essere lettere maiuscole, minuscole, numeri e simboli.

Anche la codifica ha un ruolo importante e quelle usate più frequentemente sono:

1. Base32:

   codifica a 5 bit, comunemente usata nelle richieste lato client. Sono permessi 37 caratteri (26 lettere, 10 numeri e il carattere '-'). E' possibile prendere 5 bit di dati alla volta, per cui si hanno 32 valori possibili.

2. Base64:

   codifica a 6-bit, comunemente usata nelle risposte lato server quando si usa il record 'TXT' come covert channel. Nel TXT record esiste una distinzione tra maiuscole e minuscole. Sono permessi 64 caratteri (26 lettere maiuscole, 26 lettere minuscole, 10 numeri e i caratteri '-' e '+'). E' possibile prendere 6 bit di dati alla volta.

Capitolo 4

Rilevamento di covert channel basati sul record A del DNS

4.1 Analisi del problema

In questa sezione verrà descritto il caso studio di questa tesi e i principali strumenti che sono stati utilizzati per la parte di detection.

Durante la fase di studio si è visto come molti malware oggi usano il record A del DNS come mezzo di comunicazione nascosta. Gli attaccanti preferiscono usare questa tecnica in quanto il traffico DNS in un perimetro aziendale difficilmente viene bloccato, inoltre la sua natura lo rende un canale robusto da usare poiché è difficile intercettarne il flusso o addirittura interromperlo. Il record A, così come da RFC-1035 [11], permette di inserire al suo interno uno o più indirizzi IP, facendo sì che che ad ogni richiesta possa essere trasmessa un'alta quantità di dati verso il client infetto.

4.2 Ambiente di analisi

In questa sezione saranno descritti con maggiori dettagli l'ambiente di lavoro e i tool usati per la fase di analisi.

4.2.1 Bro

Bro è un software Open-Source per l'analisi passiva del traffico di rete [12]. E' principalmente un sistema che ispeziona tutto il traffico su una porzione di rete con l'obiettivo di segnalare attività sospette. Inoltre in Bro sono presenti altri moduli, come per esempio la misurazione delle prestazioni della rete, utili per avere un quadro generale nella fase di monitoring.

In Figura 4.1 è possibile vedere la storia temporale di Bro:

Quando Bro è installato su una porzione della rete genera dei file di log che registrano l'attività che si sta svolgendo. In questi file di log vengono riportati per esempio le sessioni HTTP con l'URI richiesta, i tipi MIME, le richieste DNS o quant'altro possa essere utile ai fini dell'analisi. Bro inoltre è un software completamente personalizzabile ed estendibile tramite la scrittura di script, nonostante abbia già librerie standard con al suo interno molteplici funzionalità.

Bro può essere installato su qualsiasi hardware ed è quindi un'alternativa di basso costo a soluzioni proprietarie soprattutto perché anche lui prevede funzionalità aggiuntive come la ricerca di attività dannose. E' un intrusion detection system abbastanza diverso rispetto agli altri, poiché è sia *signature based* che *anomaly based* in quanto non si limita ad un particolare metodo di rilevamento delle intrusioni né tantomeno si basa sulle tradizionali firme come il resto dei sistemi.

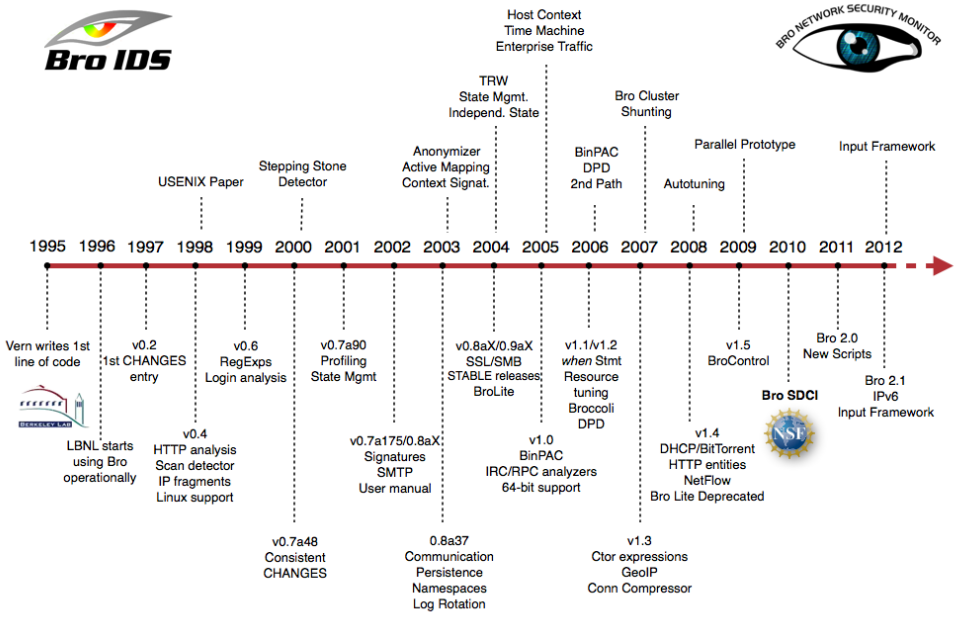


Figura 4.1. Storia di Bro (fonte:bro.org).

Esso traduce il flusso di pacchetti all'interno di una rete in una serie di *eventi* di vario genere e applica a questi script (personalizzabili dall'utente) delle azioni da compiere in risposta all'evento in ingresso. Un evento può essere qualsiasi cosa che avviene all'interno della rete analizzata come per esempio una richiesta HTTP, una connessione TCP o una query DNS. Un approccio di questo tipo è possibile grazie alla presenza di un linguaggio di dominio specifico chiamato *Bro scripting language*, che permette di realizzare degli script da associare ai vari eventi, determinando così le attività effettuate da Bro in risposta a ciò che accede nella rete in maniera molto flessibile; permette inoltre sia il rilevamento semantico per controllare se ne viene fatto un uso improprio, sia l'analisi comportamentale per il rilevamento di attività anomale.

Caratteristiche

Bro viene distribuito per i principali sistemi operativi UNIX (Linux, FreeBSD e MacOS). Ha un'interfaccia *libpcap* per la cattura dei pacchetti e per l'analisi sia in real-time che offline. Supporta anche una modalità 'Cluster' per l'implementazione su larga scala.

Durante la fase di analisi vengono supportati i principali protocolli a livello applicazione (DNS, HTTP, FTP, SMTP, SSH, SSL), vengono rilevate attività di tunnelling come per esempio Teredo e Ayiya in cui Bro analizza il contenuto come se non vi fosse nessun tunnel. Inoltre vengono usati linguaggi di script come Python, Perl o Ruby per l'inserimento di funzionalità aggiuntive da parte dell'utente.

Architettura e Funzionalità

Bro agisce principalmente come un *monitor di sicurezza*, cioè un'analizzatore di rete che ispeziona tutto il traffico che passa su un di un link, in cerca di attività sospette. Il vantaggio sta nell'usufruire dei *file di log* che vengono generati e che registrano al loro interno tutte le attività di rete o dei link analizzati. Questi log al loro interno hanno una trascrizione molto accurata di ciò che accade a livello di applicazione, come per esempio un resoconto di tutte le sessioni HTTP iniziate con tutte le URI richieste e le relative risposte del server, delle query DNS effettuate o per esempio dei certificati SSL inviati. Questi file di log possono essere utilizzati, come nel nostro

caso, anche per operazioni di elaborazione da parte di algoritmi esterni. Un ultima peculiarità di Bro, che lo differenzia dagli altri IDS, è la presenza di un linguaggio di scripting Turing completo, il Bro scripting language, che come descritto sopra permette agli utenti di inserire attività di analisi arbitrarie di qualsiasi tipo. Questo sistema rende Bro molto flessibile anche perchè di base è fornito di una vasta quantità di funzionalità predefinite, estendibili tramite codice.

L'architettura è suddivisa in due componenti principali: l'*Event Engine*, cioè il motore di eventi, e il *Policy Script Interpreter*, cioè l'interprete degli script.

In Figura 4.2 è possibile vedere l'architettura di Bro

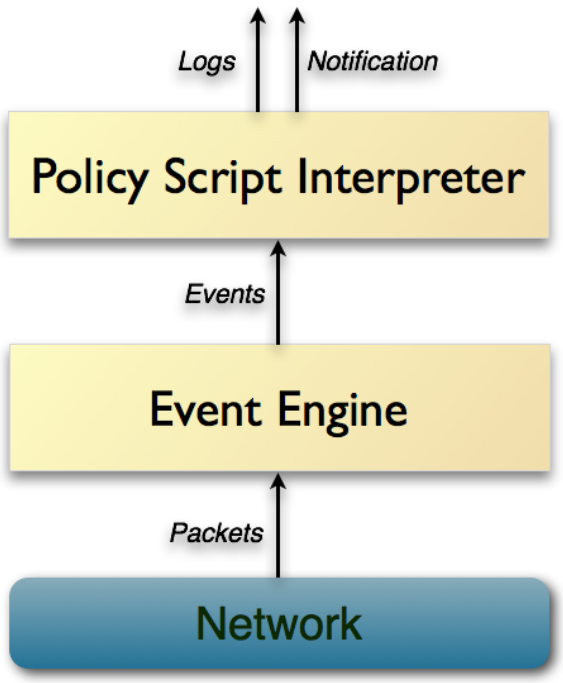


Figura 4.2. Architettura di Bro (fonte:bro.org).

L'event Engine è il componente che si occupa di ridurre il flusso di pacchetti in entrata in una serie di eventi ad alto livello. Un evento viene generato semplicemente quando qualcosa accade e riflette quindi un'attività della rete, per esempio ogni richiesta HTTP viene convertita in un evento *httprequest* che porta con sé tutta una serie di informazioni utili quali indirizzi IP, porte, URI richiesta, versione del protocollo HTTP ecc. Questi eventi però non forniscono alcuna spiegazione sul suo contenuto, come ad esempio se l'URI richiesta corrisponda ad un sito di malware già noto. Questo tipo di semantica appartiene invece al secondo componente, lo Script Interpreter, che permette di eseguire degli *event handlers* scritti con Bro scripting language. Questi script descrivono le azioni da compiere quando l'IDS rileva diversi tipi di attività e consentono quindi di definire una policy di sicurezza ad hoc per un determinato sito. Di default Bro è impostato per registrare tutti gli eventi nei file di log ma può essere configurato per mandare notifiche agli amministratori di sistema oppure lanciare un alert o una system call che richiama uno script.

File di log

Di default i log sono dei file ASCII scritti in un formato leggibile sia dalla macchina che dall'uomo, e popolati con dati per la maggior parte *connection oriented* organizzati in colonne. Essi vengono generati da uno script interno di Bro che ne definisce la struttura, specifica il protocollo o il tipo di connessione in base a ciò che è stato preso in esame. Un esempio di quanto scritto fino ad ora è la Figura 4.3 che rappresenta il file di log DNS.

ts	uid	id.orig_h	id.orig_p	id.resp_h	id.resp_p
time	string	addr	port	addr	port
CTYK0O1a1N	127.0.0.1	53962	127.0.0.1	53	udp
CTYK0O1a1N	127.0.0.1	53962	127.0.0.1	53	udp
CTYK0O1a1N	127.0.0.1	53962	127.0.0.1	53	udp
CTYK0O1a1N	127.0.0.1	53962	127.0.0.1	53	udp
CTYK0O1a1N	127.0.0.1	53962	127.0.0.1	53	udp
CTYK0O1a1N	127.0.0.1	53962	127.0.0.1	53	udp
CTYK0O1a1N	127.0.0.1	53962	127.0.0.1	53	udp
CTYK0O1a1N	127.0.0.1	53962	127.0.0.1	53	udp
CTYK0O1a1N	127.0.0.1	53962	127.0.0.1	53	udp
CTYK0O1a1N	127.0.0.1	53962	127.0.0.1	53	udp
CTYK0O1a1N	127.0.0.1	53962	127.0.0.1	53	udp
CTYK0O1a1N	127.0.0.1	53962	127.0.0.1	53	udp
CTYK0O1a1N	127.0.0.1	53962	127.0.0.1	53	udp
CTYK0O1a1N	127.0.0.1	53962	127.0.0.1	53	udp
CTYK0O1a1N	127.0.0.1	53962	127.0.0.1	53	udp

Figura 4.3. log dns.

Tutti i log di qualsiasi protocollo avranno sempre questi 3 campi:

- timestamp(ts): serve a dare una collocazione temporale all'evento in analisi
- UID: identificatore univoco della connessione
- connection 4-tuple: insieme di quattro attributi (indirizzo IP/porta origine, indirizzo IP/-porta destinazione).

Di fondamentale importanza è soprattutto l'UID, che serve per identificare tutte le attività associate ad una stessa connection 4-tuple.

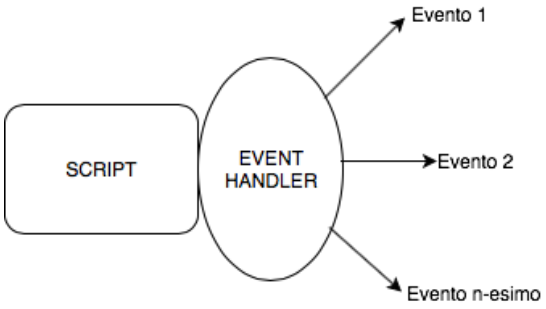
Le restanti colonne invece completano il file di log e contengono informazioni specifiche sul protocollo e sull'attività di rete a cui il log è collegato. Esistono vari tipi di file di log e quello usato in queste analisi è il file *dns.log*. Esso offre un sommario di tutta l'attività DNS rilevata. Ogni record contiene informazioni inerenti a questo protocollo come per esempio il dominio soggetto della query DNS e la corrispondente riposta da parte del server.

Event Handler

La parte di scripting di Bro dipende quasi interamente dalla gestione degli eventi generati dall'IDS mentre analizza il traffico di rete. Il sistema controlla in maniera continua il verificarsi di un evento e quando questo accade manda in esecuzione una o più parti di script, scritti appositamente per gestire quel tipo di evento.

Bro inserisce questi eventi generati in una coda detta *event queue*, e li processa secondo la modalità FIFO.

La struttura di qualsiasi script Bro è la seguente:



Dalla figura possiamo vedere come ogni script ha un *event handler* associato ad uno specifico evento, il quale contiene tutte le azioni da eseguire racchiuse all'interno di funzioni.

4.2.2 Gruppi APT

Durante la fase di studio sono stati analizzati vari gruppi e i relativi malware usati al fine degli attacchi. I mezzi con cui sono state condotte queste analisi hanno previsto quasi sempre una fase di ricerca e poi di download nelle principali piattaforme quali: hybrid analysis [13], virushare [14], packetotal [15], virustotal [16], reverse.it [17].

I malware presi in considerazioni sono tutti quelli che usano il record A del DNS come covert channel, e di cui è stato possibile reperire un campione di traffico utile ai fini della parte implementativa.

1. **APT18**:

   WIN32.BACKDOOR.DENIS

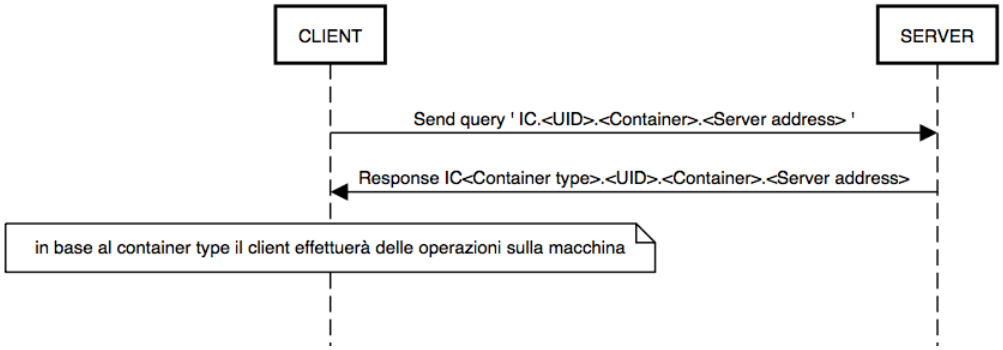
   Il protocollo di comunicazione prevede 3 step:

   (a) il malware invia una richiesta al server C2 ogni minuto per 3 volte. Se il server non risponde riproverà dopo un tempo arbitrario.

   (b) quando il server è attivo risponde al malware inviando questa query 'IC<Container type>.<UID>.com'.

   (c) il client in base al numero di container eseguirà diverse operazioni sulla macchina vittima.

   E' possibile osservare lo schema temporale del protocollo di comunicazione.



Descrizione dei campi

- *Container type*: esistono quattro tipi di container. Il malware determina qual è richiesto, in base al comando ricevuto e ai risultati dell'esecuzione come descritto in tabella 4.1:

Tipo	*Obiettivo*
1	invia informazioni sul sistema vittima
2	invia informazioni sullo stato di ricezione del file
3	invia informazioni sulla corretta ricezione del file
4	invia informazioni sulla corretta esecuzione del comando con codice 0xCB

Tabella 4.1. Container Type.

Offset	*Description*
0x0 - 0xF (0-15)	Contiene l'hostname
0x10 - 0x14 (16-20)	Contiene l'indirizzo IP utente

Tabella 4.2. Struttura UID.

- *UID*: è l'ID dell'utente che è lungo 0x20 (32) byte. È una stringa Base32 che, dopo la decodifica, ha la seguente struttura come descritto in tabella 4.2:

- *Container*: nel container vi sono i risultati dell'operazione del malware in una forma compressa. La struttura del container può variare in modo significativo a seconda del comando e della risposta. Il container è una stringa Base32 che, dopo la decodifica, ha la seguente struttura come descritto in tabella 4.3:

WIN32.ISMDOOR.GEN

Di esso non è stato possibile reperire campioni utili all'analisi del traffico prodotto, ma conoscendone il protocollo di funzionamento è stato possibile replicarlo tramite un algoritmo in python che simula la comunicazione tra il malware e il suo C2.

Il protocollo di comunicazione prevede sei step:

(a) il malware invia in polling sulla porta 3000 un messaggio di ping per vedere se il server è attivo; se dopo due tentativi non riceve nessuna risposta smetterà di provare a contattarlo e riproverà dopo un tempo arbitrario.

(b) il server ricevuto il messaggio di ping invia il messaggio di pong per far capire al malware che è attivo.

(c) il malware invia una query con il seguente hostname 'n.n.c.<ID sessione>.<Dominio server>.com'.

(d) il server ricevuta questa query controlla l'ID di sessione e procede in due modi: se non è presente nel DB, lo salva all'interno di un file SQLite per mantenerlo in memoria inserendo anche il comando che gli invierà, altrimenti se è già presente procederà soltanto con il salvataggio del comando da inviargli.

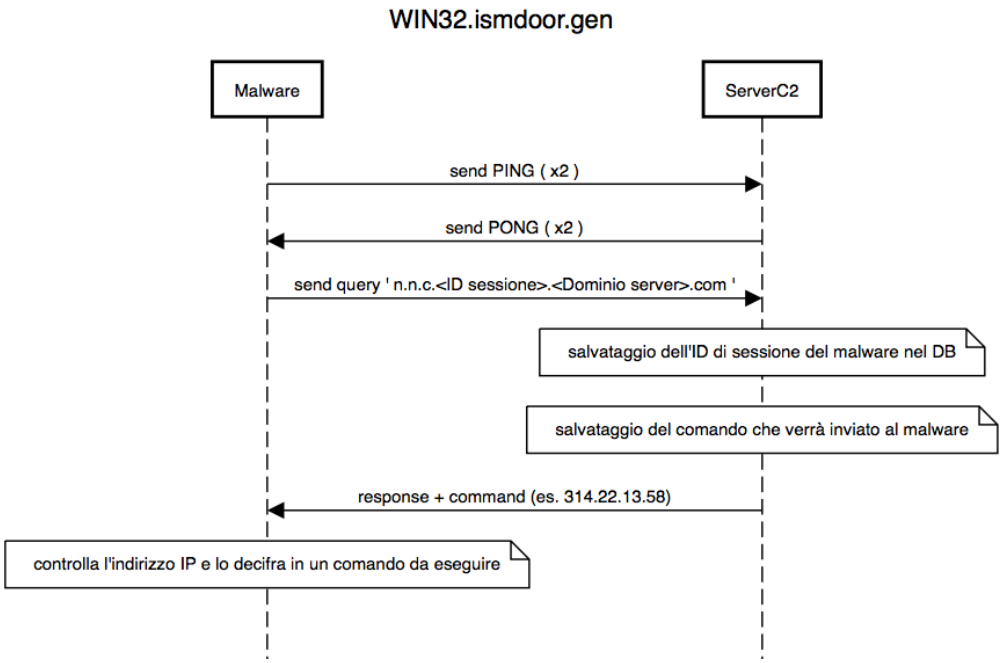
(e) il server risponde alla query e nel record A incapsula un indirizzo IP del tipo (314.22.13.58).

(f) il client ricevuta la response, controlla l'indirizzo IP e lo decifra in un comando che eseguirà sulla macchina vittima. In questo caso l'indirizzo IP 314.22.13.58 corrisponde al comando 'downfi' che permette al malware di scaricare un determinato file dal server.

E' possibile osservare lo schema temporale del protocollo di comunicazione.

Offset	*Description*
0x0 - 0x3 (0-3)	Tempo (in sec.) impiegato per creare il messaggio
0x4 - 0xB (4-11)	IACIMAOQ signature
0xC - 0x22 (12-34)	Messaggio

Tabella 4.3. Container.



Qui è riportata la tabella 4.4 con alcuni dei principali comandi usati dal malware:

Comando	*Base32*	*IP*
(invia informazioni sul client infetto)	SI	9.1.4.0
(update del sistema)	RUN	2.8.6.31
(scarica un file)	DOWNFI	314.22.13.58
(upload di un file)	UPFI	10.09.1.38
(schreenshot)	SCREEN	95.30.06.551

Tabella 4.4. Principali comandi WIN32.ismdoor.gen.

2. **APT34**:

Helminth

Il protocollo di comunicazione prevede i seguenti step:

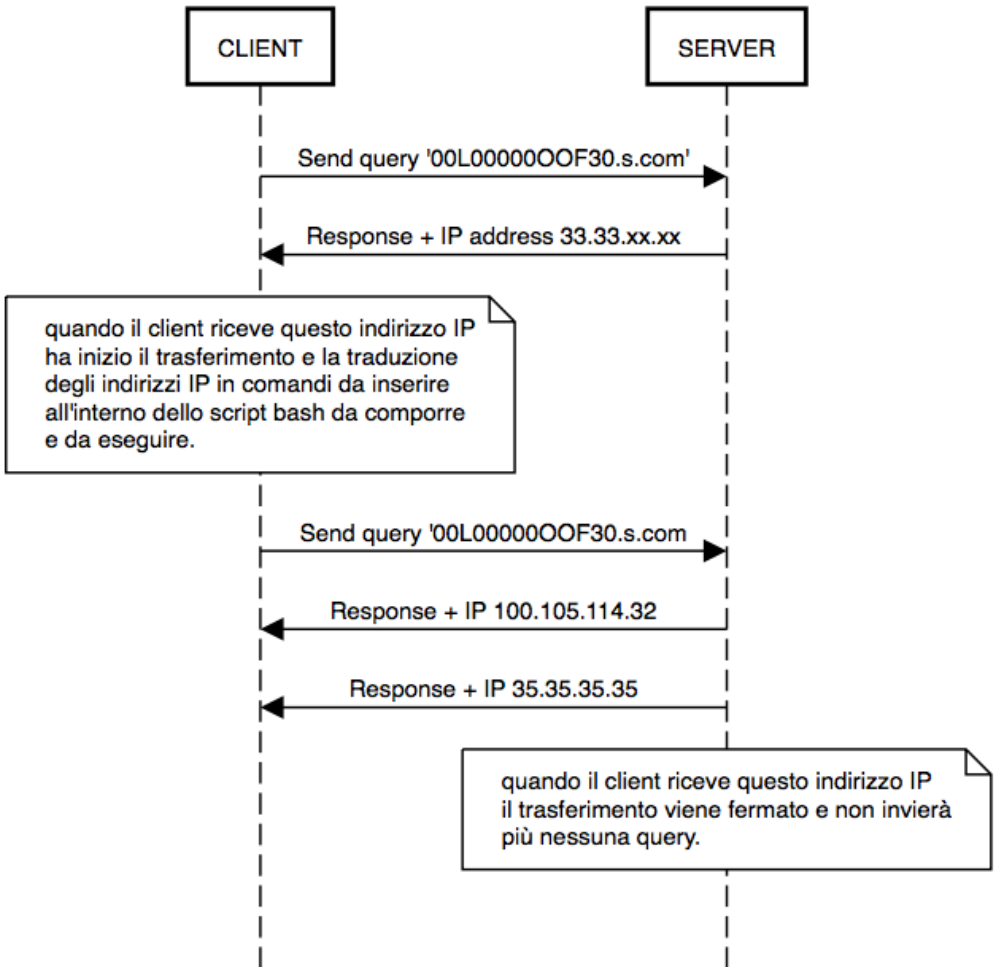
(a) il malware invia una query al server con il seguente hostname '00L00000OOF30.s.com'

(b) il server ricevuta la query invia la response incapsulando un indirizzo IP del tipo 33.33.xx.xx che sta a significare l'inizio del trasferimento.

(c) il malware invia altre query e il server risponderà con degli indirizzi IP che il client interpreterà come delle lettere da scrivere in uno script bash che comporrà il comando da eseguire sulla macchina vittima.

(d) il malware finirà di scrivere i dati per lo script quando il server invierà una response contenente l'indirizzo 35.35.35.35 che sta a significare la fine del trasferimento.

E' possibile osservare lo schema temporale del protocollo di comunicazione.



La richiesta DNS iniziale avrà la seguente struttura: '00<identifier>00<sequence number>30.s.com'.

Il server C2 risponderà alle richieste DNS con indirizzi IPv4 all'interno di record A; questi indirizzi IP saranno trattati e analizzati come dati per costruire uno script da eseguire sul sistema. Alla ricezione dell'indirizzo IP '33.33.xx.xx', lo script utilizza gli ultimi due ottetti di questo indirizzo IP per dare il nome al file batch che sarà salvato nella cartella '*tp*' creata inizialmente dal malware. Una volta ottenuto il nome del file batch, lo script continuerà a inviare richieste DNS aggiuntive e utilizzerà gli ottetti degli indirizzi IP risolvibili come caratteri da scrivere nello script batch. Lo script continua a scrivere caratteri fino a quando non riceve l'indirizzo IP '35.35.35.35' che sta a significare l'interruzione e il salvataggio dei dati e quindi l?inizio dell?esecuzione dello script.

3. **APT32**:

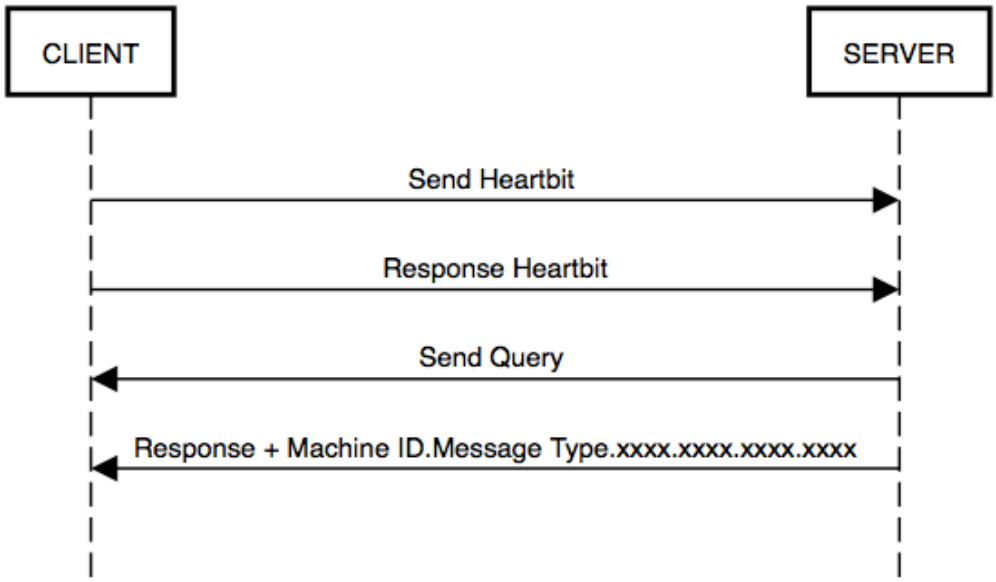
UDPos

Il protocollo di comunicazione prevede i seguenti step:

(a) il malware invia una query al suo server di comando è controllo

(b) il server ricevuta la query invia la response avente il seguente formato 'Machine ID.Message Type.xxxx.xxxx.xxxx.xxxx'.

E' possibile osservare lo schema temporale del protocollo di comunicazione.



- Machine ID: è un campo lungo sempre 15 caratteri in cui è contenuto l'ID della macchina
- Message Type: Ha una lunghezza che non supera mai i 31 caratteri.

Esistono cinque possibili valori per il campo *Message Type*:

(a) bin

(b) info

(c) ping

(d) trp

(e) note

I messaggi *bin* vengono utilizzati per trasmettere i dati raccolti dal processo *infobat.bat*.

I messaggi *ping* sono messaggi del tipo heartbeat inviati al C2 ogni 60 minuti.

I messaggi *info* sono puramente informativi e vengono inviati insieme ai messaggi di ping e possono essere utilizzati per avere informazioni sull'hostname, sull'IP del client vittima, sul sistema operativo utilizzato, sul numero di processi attivi ecc.

I messaggi *trp* servono per inviare dati convertiti convertiti in Base32 al client che saranno inseriti subito dopo il campo Message Type.

I messaggi *note* servono invece per chieder informazioni al client riguardo il nome dei processi attivi.

Il malware registra inoltre:

- il nome del processo che il client invia dopo aver ricevuto il messaggio di tipo *note* in un file chiamato sinf.dat
- il numero di processi totali con estrazione riuscita in hdwid.dat
- salva un hash del messaggio trp in udwupd.kdl, presumibilmente allo scopo di tenere traccia di ciò che è già stato inviato al server C2.
- tutti e cinque i tipi di messaggi vengono registrati nel file .dat.ID macchina prima della trasmissione.

Cobalt Strike

Cobalt Strike è un software di penetration-testing che viene definito come 'un software di simulazione completo e progettato per eseguire attacchi mirati ed emulare azioni post-exploitation'.

Le funzionalità interattive post-exploit di Cobalt Strike coprono l'intera gamma di tattiche ATT&CK, tutte eseguite all'interno di un unico sistema integrato. Inoltre Cobalt Strike può essere usato insieme ad altri strumenti noti come Metasploit Framework.

Per la trasmissione di dati tramite Cobalt Strike viene usato Beacon, un payload in grado di dare la possibilità di mandare del carico utile tra un client ed un server all'interno delle richieste DNS. E' usato quindi per una comunicazione tra il malware e il proprio centro di comando e controllo.

Cobalt Strike ha inoltre integrata un'interfaccia a linea di comando per interagire con le varie macchine su cui è installato e permette anche agli attaccanti che ne fanno uso di modificare in maniera molto semplice il modo in cui Beacon invia, all'interno del payload, del carico utile usato per le comunicazioni tra il client infetto e il centro di comando e controllo. La sua peculiarità sta nel fatto che ha la possibilità di ricevere del carico utile su un protocollo (es. DNS) e rispondere su di un altro (es. HTTP).

Come primo step viene creato un *Beacon listener* a cui bisogna assegnare un nome e scegliere la modalità beacon_dns.

A questo punto che il Beacon Listener è attivo, è possibile tramite Metasploit framework inviare all'interno del record A del dns dei dati cifrati che possano essere interpretati dal client come dei comandi da eseguire sulla macchina vittima.

4.2.3 Red Team Tool

In questa sezione sarà descritto dnscat2, strumento utilizzato per il tunnelling DNS e la creazione di una comunicazione client-server ad hoc, che simuli quella tra un malware ed il suo C2.

Caratteristiche

Dnscat2 è disponibile in due parti, client e server. E' stata usata una macchina virtuale con installato Ubuntu 16.10 per l'esecuzione di entrambe in ambiente VMware.

Il client, scritto in C, è progettato per essere eseguito su una macchina compromessa. Quando viene eseguito, in genere deve essere specificato un nome di dominio. Tutte le richieste saranno inviate al server DNS locale che le reindirizzerà al server DNS per quel dominio. Se non si dispone (come in questo caso studio) di un server DNS autorevole, è possibile utilizzare le connessioni UDP sulla porta 53 (o anche altre scelte arbitrariamente). Simuleranno del traffico dns ma avranno nella query la presenza del prefisso *dnscat2*.

Il server invece, scritto in ruby, è progettato per essere eseguito su un server DNS autorevole. Anche in questo caso, quando lo si esegue vuole specificato il nome di dominio. Quando riceve del traffico per uno dei domini specificati, tenta di stabilire con esso una connessione logica. In questa maniera viene creato un tunnel che permetterà di caricare o scaricare file, di eseguire una shell o altri comandi.

Dnscat2 prevede anche una connessione diretta, cioè il client si può connettersi direttamente all'indirizzo IP sulla nostra macchina tramite la porta UDP 53.

Setup e uso

Per compilare il client in una macchina UNIX è necessaria la presenza di make/gcc ed eseguire i seguenti step:

```
$ git clone https://github.com/iagox86/dnscat2.git
$ cd dnscat2/client/
$ make
```

Il server non bisogna compilarlo, ma essendo scritto in ruby dipende da alcune gem[1] che dovranno essere installate per soddisfare le dipendenze:

```
$ git clone https://github.com/iagox86/dnscat2.git
$ cd dnscat2 / server /
$ gem install bundler
$ bundle install
```

se si ottiene un errore quando si esegue il comando bundle install, può essere necessario eseguirlo come root.

Per l'esecuzione è necessario aprire due terminali distinti ed eseguire per primo il server e per secondo il client con i seguenti comandi:

```
$ ruby ./dnscat2.rb
$ ./dnscat
```

In generale il server va sempre eseguito per primo ed è multi threading, cioè in grado di servire più client contemporaneamente. Quando il client viene eseguito apre una sessione verso il server, connettendosi in questo caso in maniera diretta.

Per creare una comunicazione tra il client e il server e simulare un centro di comando e controllo è necessario eseguire i seguenti comandi:

lato server:

```
$ ruby ./dnscat2.rb test.com
```

in questo caso è stato assegnato un nome di dominio di prova.

lato client:

è possibile vedere se il server è attivo lanciando il comando ping dal client specificando server e dominio da pingare. Qualora il server è in ascolto su di un altra porta differente dalla 53, bisogna specificare anche la porta tramite il flag 'porta=num. di porta'.

```
$ ./dnscat --dns server=192.168.1.15,domain=test.com --ping
```

se il ping ha esito positivo allora vuol dire che il client riesce a comunicare con il suo server C2.

La sessione è vista come un collegamento virtuale tra un client e un server e viene identificata da un session ID a 16 bit. In queste sessioni come descritto in precedenza è possibile eseguire comandi, shell, caricare file, vedere lo stato della macchina vittima ecc.

In questa maniera siamo riusciti a ricreare un ambiente in cui sono stati eseguiti un client ed un server al fine di simulare una comunicazione tra un malware e il suo C2 ed il traffico generato è stato inserito insieme agli altri campioni per essere usato nella parte di detection.

[1] strumento da riga di comando per l'installazione e la gestione delle librerie e delle gem. RubyGems si integra con Ruby runtime-loader utile per trovare e scaricare le gemme da installare. Anche se è possibile utilizzare repository RubyGems privati, il repository pubblico è più comunemente usato per la gestione di esse.

Capitolo 5

Progettazione ed implementazione della soluzione

La seconda parte di tesi si concentra sullo sviluppo e sull'implementazione di una soluzione che ha come obiettivo la catalogazione del traffico DNS in arrivo su Bro. Per questa parte è stato usato sia traffico non malevolo (internet, skype, hangout, youtube, ecc.) che traffico malevolo generato dai malware visti in precedenza che usano il DNS come metodo di comunicazione nascosta. L'obiettivo è quello di riuscire, tramite algoritmi di machine learning, ad etichettare questo traffico per capire se esso sia malevolo o meno, per poi riuscire a compiere delle azioni basate su queste assunzioni.

5.1 Apprendimento automatico

L'apprendimento automatico o *machine learning*, rappresenta un insieme di metodi che utilizzano metodi statistici per migliorare progressivamente la performance di un algoritmo nell'identificare i pattern nei dati [19]. L'apprendimento automatico è strettamente legato al riconoscimento di pattern e alla teoria computazionale dell'apprendimento ed esplora lo studio di algoritmi che possano apprendere da un insieme di dati e fare delle predizioni su di essi, costruendo quindi in maniera induttiva un modello basato su dei campioni. L'apprendimento automatico ad oggi è molto usato anche nel campo della cyber security per l'individuazione di intrusioni in una rete o di attaccanti che cercano di violare dati; inoltre permette di gestire la complessità di applicazioni reali, che talvolta sono troppo complesse per poter essere modellate efficacemente.

I compiti dell'apprendimento automatico sono classificati in tre categorie, a seconda del 'feedback' disponibile al sistema di apprendimento. Queste categorie sono:

1. Apprendimento supervisionato: vengono forniti degli esempi nella forma di possibili input e i rispettivi output con l'obiettivo di estrarre una regola generale che associ l'input all'output corretto.
2. Apprendimento non supervisionato: l'algoritmo ha come obiettivo quello di trovare una struttura negli input forniti, senza che gli input vengano etichettati in alcun modo.
3. Apprendimento per rinforzo: l'algoritmo interagisce con un ambiente dinamico nel quale cerca di raggiungere un obiettivo, avendo un insegnante che gli dice soltanto se l'obiettivo è stato raggiunti.

Nel campo dell'apprendimento automatico, l'obiettivo della classificazione statistica è di usare le caratteristiche degli oggetti per identificare a quale gruppo essi appartengono. Questo tipo di classificazione è detta lineare e raggiunge questo scopo facendo una decisione sulla classificazione basata sul valore di una combinazione lineare di caratteristiche. Le caratteristiche di un oggetto sono anche conosciute come *feature value* e vengono rappresentate come un vettore detto *vettore delle caratteristiche*.

5.1.1 Dati e Pattern

In quest'ambito, dove il comportamento degli algoritmi non è pre-programmato, i dati hanno un ruolo fondamentale poiché l'algoritmo apprende dai dati stessi. E' possibile anche usare il termine *pattern* per riferirsi ai dati. Un pattern può essere un qualsiasi oggetto contenente dati e il *Pattern Recognition* è la disciplina che studia il riconoscimento di essi. Esistono sia dei pattern numerici che dei pattern categorici. Nel primo caso vi sono dei valori associati a caratteristiche misurabili, mentre nel secondo caso vi sono dei valori associati a caratteristiche qualitative e alla presenza o meno di una di esse.

5.1.2 Training Set e Test Set

Gli algoritmi di apprendimento automatico usati hanno sempre bisogno di un dataset contenente al suo interno il *Training Set* e il *Test Set*.

Il primo è l'insieme di pattern su cui addestrare il sistema, trovando il valore ottimo per i parametri. Esso spesso consiste in un vettore di input a cui è associata una risposta o una determinata classificazione. Una volta eseguito, l'algoritmo apprende, in base alla risposta o alla classificazione, quali caratteristiche discriminano gli elementi appartenenti alle differenti categorie. Una volta finita la fase di apprendimento, la correttezza dell'algoritmo sarà verificata eseguendo lo stesso su un test set.

Il secondo invece è l'insieme dei pattern su cui valutare le prestazioni finali dell'algoritmo. In questo caso non bisogna mai tarare i parametri poiché si rischia una sovrastima delle prestazioni.

In questa sezione per la creazione del training set sono state seleziona delle caratteristiche dai file di log generati da Bro e sono state poi date in input come training set agli algoritmi usati.

5.2 Knowledge Discovery

E' il processo di analisi dei dati finalizzato all'estrazione di informazioni, accurate e utili per comprendere e modellare i fenomeni del mondo reale corrispondenti ai dati per supportare i processi decisionali.

5.2.1 Generazione log di Bro

In questa sezione viene usato Bro come mezzo per la generazione dei log sul traffico di nostro interesse.

E' stato possibile dare in input dei file .pcap contenenti traffico utile; per il traffico non malevolo è stato usato wireshark come mezzo di sniffing e generazione dei file .pcap, mentre per il traffico malevolo sono stati scaricati i file .pcap dalle principali piattaforme elencate in precedenza.

Il primo step prevede l'esecuzione di Bro sulla porzione di rete in cui esso è installato. Durante questa fase resta in uno stato di loop prendendo tutto il traffico in ingresso dalla rete. La porzione di nostro interesse ai fini dell'analisi è il traffico DNS generato sulla porzione di rete sotto osservazione e quindi tramite l'event engine di Bro, ogni volta che arriva una richiesta dns, scatterà l'evento *DNS::log_dns* che produrrà i file di log che serviranno per lo step successivo.

Una volta eseguiti tutti gli script da parte del motore di Bro sul traffico analizzato, vengono generati dei file di log che al loro interno contengono tutta l'analisi svolta.

In questa fase però, per nessuno dei campioni analizzati, il centro di comando e controllo era ancora attivo, quindi anche eseguendo i malware non è stato possibile riprodurre nessun traffico utile.

Si è scelto quindi di inserire nel motore di Bro dei file .pcap relativi al traffico generato dai malware analizzati ma scaricati per mezzo della sandbox Hybrid Analisys[1].

Per eseguire e generare i file di log sono stati eseguiti i seguenti comandi in ambiente Ubuntu 16.10 su cui era installato Bro.

Dal terminale è possibile digitare i seguenti comandi:

```
$ bro -Cr win32backdoordenis.pcap

$ bro -Cr win32ismdoorgen.pcap

$ bro -Cr helminth.pcap

$ bro -Cr udpos.pcap

$ bro -Cr dnscat2.pcap

$ bro -Cr cobaltstrike.pcap
```

Con queste istruzioni vengono inseriti nel motore di Bro i file .pcap e in output (all'interno del percorso *bro/site/bro-scripts*) verranno generati, per ognuno, i file *dns_entropy.bro* contenenti le analisi del traffico dns di ciascun malware.

Sono stati inoltre generati altri 6 file .pcap contenente traffico non malevolo del tipo: traffico internet, traffico skype, traffico hangout.

5.2.2 Selezione dei dati e trasformazione

Per la creazione del training-set è stato deciso di creare un algoritmo che, per ogni richiesta DNS analizzata, prenda i dati generati da Bro, e ne estrapoli alcuni valori (label) da inserire in un file .csv che verrà fornito in input ai classificatori.

Le label scelte sono:

1. *Dimensione della richiesta*: dimensione in byte della richiesta
2. *Dimensione della risposta*: dimensione in byte della risposta
3. *Entropia Hostname*: casualità di caratteri contenuti in una stringa, strettamente collegata al numero minimo di bit necessari per rappresentarla senza errori.
4. *Percentuale di char*: percentuale dei caratteri all'interno della singola richiesta DNS
5. *Percentuale di numeri*: percentuale di numero all'interno della singola richiesta DNS
6. *Numero di livelli*: numero di sottodomini presenti nella richiesta DNS

Nel primo passo per la creazione del training-set viene preso in input, dall'algoritmo scritto in python, il file *traffic.csv*. Questo file contiene tutte le richieste DNS generate dai log di Bro sia per il traffico malevolo che per il traffico non malevolo. Da questo file l'algoritmo estrapolerà i seguenti campi:

1. Dimensione della richiesta:

[1] una sandbox è un meccanismo per eseguire applicazioni in uno spazio limitato. Solitamente è utilizzata per eseguire programmi non testati o non attendibili, non verificati o provenienti da terze parti non riconosciute (come utenti o siti web), senza rischiare di infettare il dispositivo dove viene eseguita l'applicazione

```
def size_in():

        . . .
        for index, row in df.iterrows():
            list_.insert(index, sys.getsizeof(row['QUERY']))
```

Viene letta la dimensione in byte dalla colonna 'QUERY'. I valori di questa colonna vengono salvati all'interno di una lista e poi inseriti come prima colonna nel training-set.

E' stato osservato che in alcuni malware la dimensione in byte della richiesta è maggiore di 100 e quindi questo campo può diventare un ottimo discriminante nella catalogazione del traffico, qualora anche la dimensione della risposta sia all'incirca della stessa dimensione.

2. Dimensione della risposta:

```
def size_out():

        . . .
        for index, row in df.iterrows():
            list_.insert(index, sys.getsizeof(row['ANS']))
```

Viene letta la dimensione in byte dalla colonna 'ANS'. Anche in questo caso i valori di questa colonna vengono salvati all'interno di una lista e poi inseriti come seconda colonna nel training-set.

Normalmente nella risposta che il server C2 invia al malware sono contenuti degli indirizzi IP. E' stato osservato come la dimensione in byte della risposta ha un valore elevato quando il server invia più di un indirizzo IP nella stessa risposta. In questo caso anche questo campo può essere un buon discriminante per catalogare quella determinata richiesta come traffico malevolo.

3. Entropia Hostname:

```
def entropy():

        . . .
        for index, row in df.iterrows():
            list_.insert(index, row['QUERY_ENTROPY'])
```

Il valore di entropia viene letto dalla colonna 'QUERY ENTROPY' e per ogni richiesta viene calcolata secondo la formula di Shannon:

$$H = -\sum \rho(x) log \rho(x)$$

Un dominio, la cui stringa analizzata ha bassi valori di casualità, ha un entropia minore rispetto ad un dominio casuale [2] generato da un malware. Un esempio osservato è il seguente:

- il dominio aaaaa.com ha un valore di entropia pari a 1,8.
- il dominio google.com ha un valore di entropia pari a 2,6
- il dominio dnscat.452e03aa090000000068ebc2362b07325650a4dac8a6bb6b06546e4d0b57 ha un valore di entropia pari a 4

Come si evince nell'esempio, un dominio con bassi livelli di casualità come i primi due ha un valore di entropia molto basso, mentre l'ultimo dominio ha un valore di entropia molto elevato dovuto alla casualità di caratteri che lo compongono.

[2] per dominio casuale si intendono tutti quei domini generati tramite algoritmi; questa tecnica è ad oggi molto usata da varie famiglie APT.

Questo campo è molto utile ai fini della catalogazione del traffico poiché molte delle richieste effettuate dal malware e le relative risposte del server, contengono domini molto lunghi e casuali, il che porta ad avere un valore di entropia molto elevato che può essere quindi un ottimo indicatore di traffico malevolo.

4. Percentuale di caratteri e di numeri

```
def char_percent():

        . . .
        fields = ['QUERY']

        df = pd.concat([pd.read_csv(f, usecols=fields) for f in
            allFiles], ignore_index=True)

        for column in range(144):
            array = df.iloc[column, 0]
            d = l = 0

            for c in array:
                if c.isdigit():
                    d = d + 1
                elif c.isalpha():
                    l = l + 1
                else:
                    pass
            tot = l + d
            if tot == 0 or l == 0:
                pass
            char_perc = ((l / tot) * 100)
            num_perc = ((d / tot) * 100)
```

In questa funzione come primo passo viene letta la colonna 'QUERY' che contiene le singole richieste. I valori di questa colonna vengono salvati all'interno di un'array e per ogni richiesta si procede quindi al calcolo della percentuale sia dei caratteri sia dei numeri presenti.

5. Livelli

```
def level():

   . . .

        fields = ['QUERY']

        df = pd.concat([pd.read_csv(f, usecols=fields) for f in
            allFiles], ignore_index=True)

        for column in range(144):
            a = df.iloc[column, 0]
            level_num = 0
            for i in a:
                if not i.find('.'):
                    level_num += 1
            list_.insert(column, level_num)
```

In questa funzione vengono calcolati il numero di sottodomini presenti in ogni richiesta. Anche in questo caso viene letta la colonna 'QUERY' contenente le singole richieste. Per ognuna di esse anche in questo caso i valori vengono salvati all'interno di una lista e poi inseriti come penultima colonna nel training-set.

6. Classe

```
def class():

    . . .
        class_type_nm = 0
        class_type_m = 1

        for i in range(0,149):
            if i <= 63:
                data = data.append(pd.DataFrame({'CLASS': class_type_m},
                    index=[6]), ignore_index=True)
            else:
                data = data.append(pd.DataFrame({'CLASS': class_type_nm},
                    index=[6]), ignore_index=True)
```

Infine come ultima colonna del training-set, tramite questa funzione, il traffico viene diviso in traffico malevolo e non malevolo etichettando come 0 il traffico non malevolo e come 1 il traffico malevolo. Questa distinzione viene fatta in maniera corretta poiché si conoscono a priori le singole richieste ed è quindi possibile etichettare in maniera corretta tutte le richieste sia di traffico non malevolo che di traffico malevolo.

5.2.3 Classificatori usati ed esecuzione

Per classificazione si intende la possibilità di assegnare una classe ad un pattern. Una classe non è altro che un insieme di pattern aventi proprietà comuni. Un classificatore è un sistema che ha l'obiettivo di usare le caratteristiche di un oggetto per identificare a quale classe questo appartiene. Gli input che riceve sono divisi in due o più classi e il sistema di apprendimento deve produrre un modello che assegni gli input non ancora visti a una o più di queste [20]. I classificatori scelti per questa tesi sono stati: Decision Tree [22], Random Forest [23] [24], K-Nearest Neighbors [25] [26] e Naive Bayes [27] [28]. Tramite l'uso di scikit-learn (libreria python open source di apprendimento automatico), è stato fornito in input, ai quattro classificatori, il training-set generato sopra per vederne i vari comportamenti.

Tutti gli algoritmi scritti in questa fase sono stati scritti in python versione 3 tramite l'IDE PYCharm CE, ed eseguiti in ambiente Ubuntu 16.10 e macOS High Sierra.

Il primo passo dell'algoritmo prevede una fase di pre-processing dove i dati sono stati convertiti in valori numerici, in modo da poter essere usati dalle funzione di apprendimento automatico. A questo punto gli algoritmi sono stati divisi in tre fasi: fase di apprendimento, fase di predizione e output dei risultati. Una volta istruito il classificatore nella fase di apprendimento tramite il training-set, si passa alla fase di predizione, usando il test-set contente sia traffico malevolo che traffico non malevolo. Finita anche questa parte si entra nell'ultima fase dell'algoritmo dove vengono generate le varie misure di performance, ricavate a partire dalle matrici di confusione (sistema che permette di raffigurare quali campioni sono stati predetti correttamente e quali no), che sono: accuratezza, precisione, recall ed F1-score (misura usata nel campo del recupero delle informazioni per misurare l'accuratezza della classificazione). Viene usata anche una k-fold cross validation [21], la cui funzione è quella di dividere il data-set in k parti, che a turno vengono prese come test set per poter essere predette mentre le altre k-1 finiscono all'interno del training set per poter allenare il classificatore. Questo ha permesso di diminuire il fenomeno dell'overfitting cioè quando un modello di predizione si adatta troppo ai dati osservati perché ha un numero eccessivo di parametri rispetto al numero di osservazioni.

Per tutti e 4 gli algoritmi sono stati usati due dataset. Il primo, uguale per tutti, contente un training-set di 150 campioni e un test-set di 145 campioni; nel secondo dataset invece è stato scelto di usare un training-set di 150 campioni e un test-set diverso per ogni algoritmo usato, con un numero di campioni varianti tra 100 e 150.

Le valutazioni sono state fatte andando a vedere il comportamento dei classificatori usando tutti e due i dataset.

Nel secondo caso è stato scelto questo approccio a 'campioni varianti' per valutarne il comportamento in contrapposizione anche al metodo k-fold cross validation [21].

Random Forest

Questo classificatore è un classificatore d'insieme, composto da molti alberi di decisione e dà in uscita la classe che corrisponde all'uscita delle classi degli alberi presi individualmente. La differenza sostanziale che qualifica la foresta come random è che ciascun albero opera su un sottoinsieme di istanze estratte casualmente dal training set di partenza (tale metodo di estrazione di features prende il nome di *random subspace*).

I passaggi eseguiti da questo algoritmo sono i seguenti:

- Tramite la funzione split dataset della libreria scikit-learn viene dato in input sia il training-set con la percentuale di training sia il test set.

  ```
  train_x, test_x, train_y, test_y = split_dataset(dataset, 0.3,
      HEADERS[1:-1], HEADERS[-1])
  ```

  In questo caso è stata scelta una percentuale di training set pari allo 0.3.

- Viene creato un oggetto di tipo RandomForestClassifier e viene generato il modello.

  ```
  trained_model = random_forest_classifier(train_x, train_y)

  predictions = trained_model.predict(test_x)
  ```

- Vengono generati la matrice di confusione da cui verranno estrapolate le misure di performance, il training accurancy e il test accurancy che saranno confrontate nella sezione dei risultati; inoltre per questo algoritmo è stato scelto di vedere per ogni record sia il valore predetto che quello effettivo.

  ```
  for i in range(0, 105):
      print("Actual outcome :: {} and Predicted outcome ::
          {}".format(list(test_y)[i], predictions[i]))

  print("Train Accuracy :: ", accuracy_score(train_y,
      trained_model.predict(train_x)))
  print("Test Accuracy :: ", accuracy_score(test_y, predictions))
  print("Confusion matrix", confusion_matrix(test_y, predictions))
  ```

Decision-Tree

Questo classificatore utilizza come modello predittivo un *albero di decisione* che ha il compito di creare un modello che predica il valore di una variabile di destinazione in base a diverse variabili di input. L'albero contiene per ogni variabile di input (nel nostro caso dimensione della richiesta, dimensione della risposta, entropia ecc.) un nodo interno in cui sono rappresentate le percentuali di osservazione su quell'etichetta. Questo algoritmo controlla questi valori in input per predire la classificazione in output dell'elemento analizzato. Per decidere quale attributo controllare per primo esso cerca quello che fornisce un *information gain* più elevato.

I passaggi eseguiti da questo algoritmo sono i seguenti:

- Al classificatore vengono dati in ingresso due array, X contenente tutto il training-set escluso la label scelta da predire, Y contenente solo la label scelta.

  ```
  X_train = features.drop(['CLASS'], axis=1)
  Y_train = features['CLASS']
  ```

- Viene creato un oggetto di tipo DecisionTreeClassifier e tramite il comando fit si crea il modello.

```
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X_train.values,Y_train)
```

- Viene dato in input al classificatore un altro set di dati con all'interno del nuovo traffico per vedere come si comporta il modello di predizione.

```
df = pd.read_csv(path_to_test)
X_test = df.drop(['CLASS'], axis=1)
Y_test = df['CLASS']
```

- Vengono generati l'albero di decisione, la confusion matrix da cui verranno estrapolate le misure di performance e il livello di accuratezza dell'algoritmo che saranno confrontate nella sezione dei risultati.

```
print("Prediction for Decision Tree: ",Y_prediction)

print("Accuracy:",accuracy_score(Y_test,Y_prediction))
print(" Confusion matrix ", confusion_matrix(Y_test, Y_prediction))
```

Per ciascun record è possibile inoltre navigare il decision tree per vedere il valore predetto. In base alla correttezza delle predizioni si ha un valore che è chiamato *training set error* che permette di capire la quantità di record predetti in maniera errata.

Naive-Bayes

Il teorema di Bayes rappresenta una tecnica fondamentale per la classificazione di pattern basati sul training-set. Il comportamento dell'algoritmo di Bayes sta nel fatto che il classificatore osserva l'oggetto e ne determina il tipo in base all'osservazione fatta esaminando determinate caratteristiche. La teoria bayesiana svolge un ruolo importante solo quando risultano conosciute alcune informazioni a priori sull'oggetto; se viene osservata una particolare caratteristica 'X' misurabile dell'oggetto riesco a stimare la probabilità che essa rappresenti una certa classe 'Y' a posteriori l'osservazione. Sotto questo punto di vista i classificatori bayesiani forniscono esattamente la probabilità che il vettore di dati in ingresso rappresenti la determinata classe in uscita.

Il classificatore Naive Bayes sfrutta l'ipotesi semplificativa di indipendenza degli attributi (feature) osservati: in questo caso date m variabili osservate $x_1 \dots x_m$ la probabilità che l'evento y_i si verifichi sarà:

$$p(x_1 \dots x_m | y_i) = \prod_{j=1}^{m} p(x_j | y_i)$$

I passaggi eseguiti da questo algoritmo sono i seguenti:

- Al classificatore vengono dati in ingresso due array, X contenente tutto il training-set escluso la label scelta da predire, Y contenente solo la label scelta.

```
X_train = dataset.iloc[:, :-1].values
Y_train = dataset.iloc[:, 6].values

dataset_test = pd.read_csv(path_to_test)
X_test = dataset.iloc[:, :-1].values
Y_test = dataset.iloc[:, 6].values
```

- Viene creato un oggetto di tipo GaussianNB e tramite il comando fit si crea il modello.

```
gaunb = GaussianNB()
gaunb = gaunb.fit(X_train, Y_train)
```

- Viene dato in input al classificatore un altro set di dati con all'interno del nuovo traffico per vedere come si comporta il modello di predizione, così come è stato fatto con gli algoritmi precedenti.

```
prediction = gaunb.predict(X_test)
```

- Vengono generati la matrice di confusione da cui verranno estrapolate le misure di performance, il training accurancy e il test accurancy che saranno confrontate nella sezione dei risultati.

```
confusion_matrix = confusion_matrix(Y_test, prediction)

print("Train Accuracy :: ", accuracy_score(Y_train,prediction))
print("Test Accuracy :: ", accuracy_score(prediction, Y_test))
print("Confusion matrix ", confusion_matrix)
```

K-nearest neighbors

E' un algoritmo utilizzato nel riconoscimento di pattern per la classificazione di oggetti basandosi sulle caratteristiche degli oggetti vicini a quello considerato. Un oggetto è classificato in base alla maggioranza dei voti dei suoi k vicini. In un contesto binario in cui sono presenti due classi è opportuno scegliere un k dispari per evitare situazioni di parità.

La scelta del parametro k dipende dalle caratteristiche dei dati. Generalmente è stato visto che all'aumentare di k si riduce il rumore che compromette la classificazione. In questo caso è stato usato un valore di k = 5, che ha diviso il data-set in 5 porzioni da 150 campioni ciascuno, per provare a predire una porzione per volta e facendone poi una media sulle 5 iterazioni.

I passaggi eseguiti da questo algoritmo sono i seguenti:

- Al classificatore vengono dati in ingresso due array, X contenente tutto il training-set escluso la label scelta da predire, Y contenente solo la label scelta.

```
X_train = dataset.iloc[:, :-1].values
Y_train = dataset.iloc[:, 6].values

X_train, X_test, y_train, y_test = train_test_split(X_train, Y_train,
    test_size=0.30)
```

- Viene creato un oggetto di tipo KNeighborsClassifier e tramite il comando fit si crea il modello; viene dato inoltre in input al classificatore un altro set di dati con all'interno del nuovo traffico per vedere come si comporta il modello di predizione, così come è stato fatto con gli algoritmi precedenti.

```
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
```

  Viene generata la matrice di confusione da cui verranno estrapolate le misure di performance. Inoltre è stato calcolato anche l'error rate facendo variare il valore di k tra 1 e 40. Questi valori saranno confrontati nella sezione dei risultati.

```
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))
```

```
fig = plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='red', linestyle='dashed',
    marker='o',
        markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```

5.2.4 Valutazione delle prestazioni

Dato un classificatore addestrato tramite un training set, è necessario valutarne il comportamento sul test set. Da questo confronto è possibile estrarre degli indici che permettono di valutare il classificatore e anche di confrontarne diversi tra di loro. E' stato indispensabile per valutare gli indici di prestazione usare campioni che non sono stati usati nella fase di apprendimento in modo da poter rilevare problemi come l'*overfitting* dei dati ovvero la mancata generalizzazione.

Per la valutazione delle prestazioni di ciascun algoritmo la matrice di confusione è stata usata secondo lo schema 'veri positivi/falsi positivi' e 'falsi negativi/veri negativi' ed è stato possibile partendo da essa estrarre alcuni valori di prestazione come:

- Accuratezza: è il rapporto tra il numero di predizioni corrette sul numero di predizioni totali.

$$ACC = \frac{(VP+VN)}{(VP+VN+FP+FN)}$$

- Error Rate: è il numero totale di predizioni errate sul numero totale di predizioni

$$ER = \frac{(FP+FN)}{(VP+VN+FN+FP)}$$

- Precision: è la probabilità che un positivo ritornato dal classificatore sia corretto

$$PPV = \frac{VP}{(VP+FP)}$$

- Recall (o hit rate): è la percentuale di positivi riconosciuti correttamente

$$REC = \frac{VP}{(VP+FN)}$$

- f1-score: misura l'accuratezza del test tenendo in considerazione precisione e recupero del test, dove la precisione è il numero di veri positivi diviso il numero di tutti i risultati positivi, mentre il recupero è il numero di veri posotivi diviso il numero di tutti i test che sarebbero dovuti risultare positivi (ovvero veri positivi + falsi negativi).

$$F_1 = 2 \cdot \frac{p \cdot r}{(p+r)}$$

In Figura 5.1 è rappresentato il modello usato per la matrice di confusione:

E' possibile confrontare i risultati dei 4 algoritmi in questo caso studio e vedere quale fra di essi è stato il più accurato nella predizione.

In tutti e 4 gli algoritmi è stato scelto di seguire il seguente approccio, e cioè quello di valutare prima gli algoritmi usando un dataset contente un training-set di 150 campioni e un test-set di 145 campioni e poi valutarli usando un test-set con un numero di campioni che è variato in base ai 4 casi analizzati tra 100 e 150.

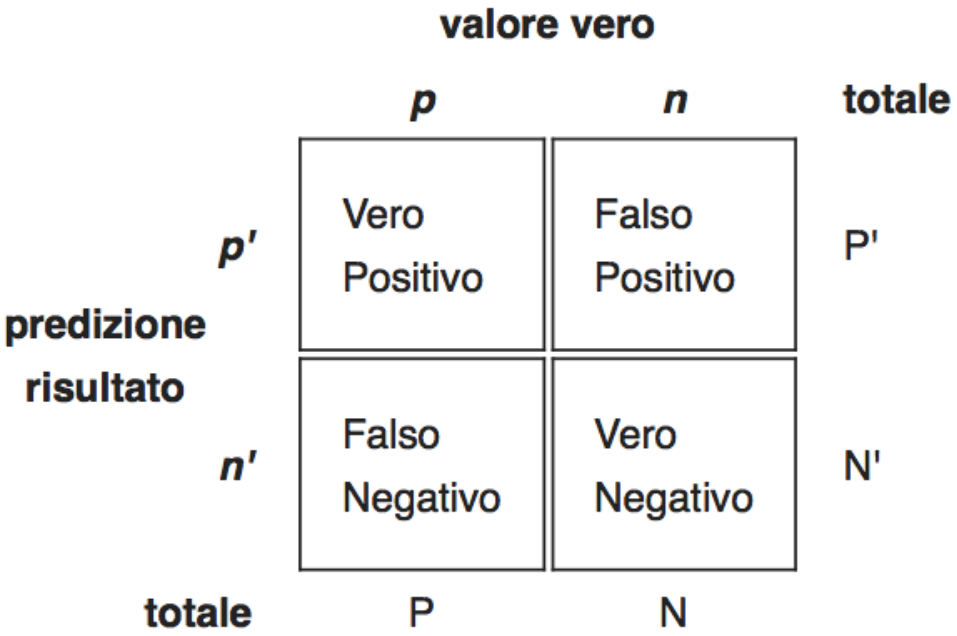


Figura 5.1. Modello matrice di confusione.

Random Forest

Usando l'algoritmo Random Forest inizialmente con un test-set di 145 campioni e in un secondo momento con un test-set di 105 campioni, l'accuratezza della previsione è stata pari al 97%; esso è stato in generale molto accurato nella predizione dei campioni in quanto, dalla matrice di confusione, è possibile vedere come ha etichettato un solo campione non malevolo come malevolo e 2 campioni malevoli come non malevoli.

La matrice di confusione relativa all'esecuzione di questo algoritmo sul nostro set di dati è mostrata in Figura 5.2.

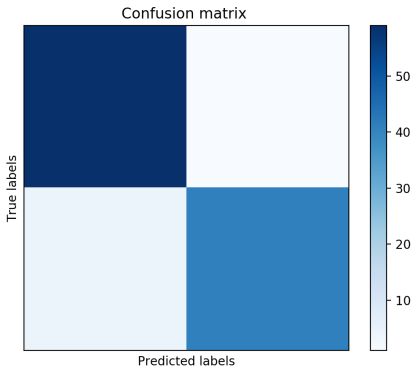


Figura 5.2. Matrice di confusione Random Forest.

I valori di essa sono i seguenti:

$$\begin{bmatrix} 59 & 1 \\ 2 & 43 \end{bmatrix}$$

Dalla matrice di confusione sono stati estrapolati i seguenti valori:

- Accuratezza:

$$ACC = \frac{(59+43)}{(59+43+1+2)=} \; 0{,}9714$$

- Error Rate:

$$ER = \frac{(2+1)}{(59+43+1+2)=} \; 0{,}028$$

- Precision:

$$PPV = \frac{59}{(59+2)=} \; 0{,}96$$

- Recall:

$$REC = \frac{59}{(59+1)} \; 0{,}098$$

- F1:

$$F_1 = 2 \cdot \frac{0{,}96 \cdot 0{,}98}{(0{,}96+0{,}98)=} \; 0{,}968$$

In tabella 5.1 sono anche riportati i primi 10 valori analizzati dall'algoritmo, il quale a video stampa il valore effettivo e quello predetto. Sono stati riportati solo i primi 10 campioni analizzati per semplicità, in quanto tutto il set è composto da 105 campioni. Da questa tabella è pero possibile vedere come l'algoritmo quasi sempre predice il risultato corretto.

Name	*Valore Effettivo*	*Valore Predetto*
github.com	0	0
dnscat.452e03aa090000000068ebc2362b07325650a4	1	1
www.google.it	0	0
www.youtube.com/watch?v=-QlaOX5w8G8	0	0
n.n.c.rax5pzzlfxwz6tte4o7sttxcjkrrk00i.dnslookupdater.org	1	1
n.n.c.oyagj1qhzukurj5pfdupggs8lbexw9dk.dnslookupdater.org	1	1
www.facebook.com	0	1
ic1.kbjvavkck5js2ucdaa2dknrxhdakqoas.gap3cwsjifbustkbj5.com	1	1
00500000huv30.winodwsupdates.me	1	0

Tabella 5.1. Tabella raffigurante i valori effettivi di ogni record e i valori predetti dall'algoritmo.

Decision Tree

Usando l'algoritmo DecisionTree con un test-set di 145 campioni si è visto che l'accuratezza della previsione è pari a circa il 71%; esso è stato meno accurato nella predizione dei campioni non malevoli, avendone etichettati 40 come malevoli; più accurato invece nella predizione dei campioni malevoli, avendone etichettato solo 1 come non malevolo.

La matrice di confusione relativa all'esecuzione di questo algoritmo sul nostro set di dati è mostrata in Figura 5.3.

I valori di essa sono i seguenti:

$$\begin{bmatrix} 65 & 39 \\ 1 & 40 \end{bmatrix}$$

Dalla matrice di confusione sono stati estrapolati i seguenti valori:

- Accuratezza:

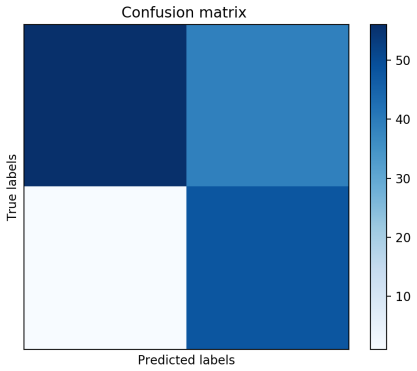


Figura 5.3. Matrice di confusione Decision Tree.

$$ACC = \frac{(65+40)}{(65+40+1+39)=} \; 0{,}714$$

- Error Rate:

$$ER = \frac{(1+39)}{(65+40+1+3)=} \; 0{,}275$$

- Precision:

$$PPV = \frac{65}{(65+1)=} \; 0{,}98$$

- Recall:

$$REC = \frac{65}{(65+39)} \; 0{,}625$$

- F1:

$$F_1 = 2 \cdot \frac{0{,}98 \cdot 0{,}625}{(0{,}98+0{,}625)=} \; 0{,}763$$

Inoltre in Figura 5.4 è mostrato anche l'albero di decisione generato dall'algoritmo e utilizzato in fase di predizione.

Naive Bayes

Usando l'algoritmo Naive Bayes inizialmente con un test-set di 145 campioni e in un secondo momento con un test-set di 149 campioni l'accuratezza è risultata pari al 91%; per quanto riguarda il risultato dell'algoritmo esso è stato molto accurato nella predizione dei campioni in quanto ha etichettato solo 1 campione non malevolo come malevolo e 11 campioni malevoli come non malevoli.

La matrice di confusione relativa all'esecuzione di questo algoritmo sul nostro set di dati è mostrata in Figura 5.5.

I valori di essa sono i seguenti:

$$\begin{bmatrix} 84 & 1 \\ 11 & 53 \end{bmatrix}$$

Dalla matrice di confusione sono stati estrapolati i seguenti valori:

- Accuratezza:

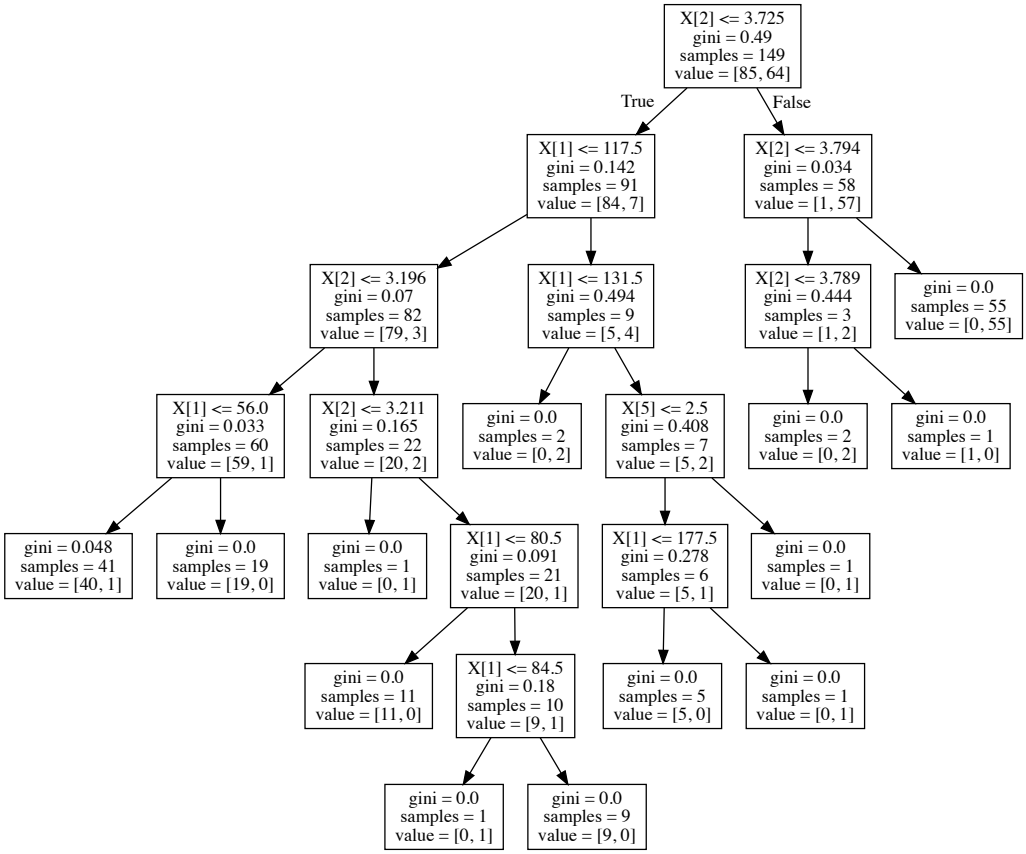


Figura 5.4. Decision Tree.

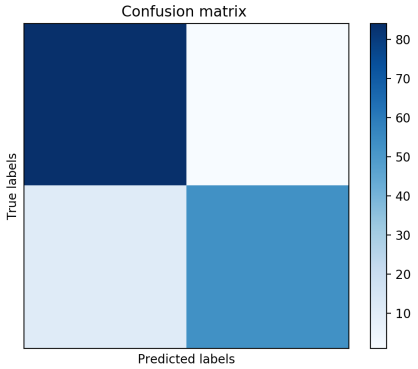


Figura 5.5. Matrice di confusione Naive Bayes.

$$ACC = \frac{(84+53)}{(84+53+11+1)=} \; 0{,}914$$

- Error Rate:

$$ER = \frac{(11+1)}{(84+53+11+1)=} \; 0{,}080$$

- Precision:

$$PPV = \frac{84}{(84+11)=} \ 0{,}88$$

- Recall:

$$REC = \frac{84}{(84+1)} \ 0{,}98$$

- F1:

$$F_1 = 2 \cdot \frac{0{,}88 \cdot 0{,}98}{(0{,}88+0{,}98)=} \ 0{,}927$$

K-Nearest Neighbors

Usando l'algoritmo K-Nearest Neighbors inizialmente con un test-set di 145 campioni e in un secondo momento con un test-set di 100 campioni l'accuratezza è risultata pari al 96%; è stato molto accurato ed efficiente nella predizione dei campioni in quanto ha etichettato solo 3 campioni non malevoli come malevoli e 1 campione malevolo come non malevolo.

La matrice di confusione relativa all'esecuzione di questo algoritmo sul nostro set di dati è mostrata in Figura 5.6.

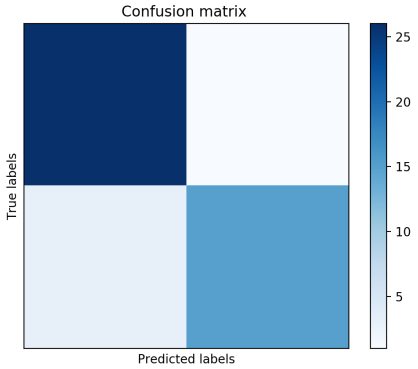


Figura 5.6. Matrice di confusione K-Nearest Neighbors .

I valori di essa sono i seguenti:

$$\begin{bmatrix} 54 & 3 \\ 1 & 42 \end{bmatrix}$$

Dalla matrice di confusione sono stati estrapolati i seguenti valori:

- Accuratezza:

$$ACC = \frac{(54+42)}{(54+42+1+3)=} \ 0{,}96$$

- Error Rate:

$$ER = \frac{(1+3)}{(54+42+1+3)=} \ 0{,}04$$

- Precision:

$$PPV = \frac{54}{(54+1)=} \ 0{,}981$$

- Recall:

$$REC = \frac{54}{(54+3)} \ 0{,}947$$

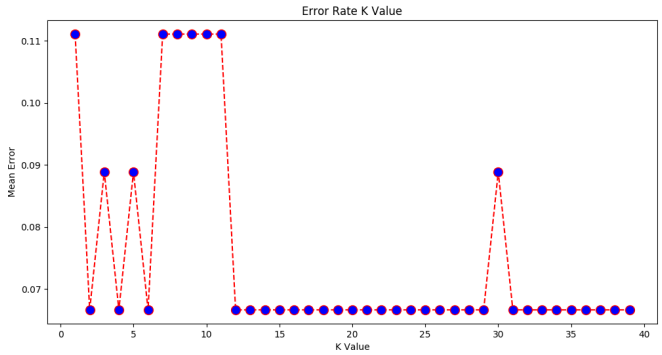


Figura 5.7. Error rate K value.

- F1:

$$F_1 = 2 \cdot \frac{0{,}98 \cdot 0{,}94}{(0{,}98+0{,}94)=} \; 0{,}959$$

Inoltre in Figura 5.7 è mostrato anche un grafico che rappresenta al variare del valore k tra 1 e 40 l'error rate.

E' possibile dal grafico vedere che nonostante varia il valore di k l'error rate rimane all'incirca pari allo 0,04 quando vengono usati valori tra 11 e 40. Si è anche visto che vi sono dei picchi dove l'error rate varia tra 0,09 e 0,11 come per esempio nel caso in cui si è usato un k pari a 10 o un k pari a 30.

Capitolo 6

Conclusioni e sviluppi futuri

6.1 Valutazioni e sviluppi futuri

Andando a guardare tutte le performance dei vari algoritmi e confrontandole tra loro si evince che, con l'uso di entrambi i dataset, i migliori algoritmi a livello di predizione in questo lavoro di tesi sono stati il Random Forest e il K-Nearest Neighbors con misure dal 96% in sù. Il meno efficiente in termini di predizione invece è stato l'algoritmo Decision Tree in quanto le sue performance non superano il 71%.

Anche con l'uso del secondo dataset, con un numero di campioni variante, si è visto che le performance rimangono invariate. Un possibile sviluppo futuro, per valutare in maniera più accurata questo approccio e la bontà nella classificazione dei modelli, può essere quello di vedere il comportamento dei vari algoritmi con l'uso di più dataset ma con un numero di campioni maggiore e confrontarne le varie prestazioni.

Sono state confrontate anche le performance in termini di tempo di esecuzione e in questo caso l'algoritmo Naive Bayes è stato il più veloce, ma questo è dovuto alla sua struttura meno complessa. Il più lento invece in questa fase è stato il K-Nearest Neighbors in quanto il calcolo delle distanze è stato computazionalmente oneroso e proporzionale alla taglia dell'insieme di dati sotto esame. Esistono degli algoritmi che in futuro possono essere applicati e provati su questo lavoro di tesi che mirano a migliorare questo aspetto, cercando principalmente di diminuire il numero di distanze da calcolare per la decisione.

Tramite le matrici di confusione, si può andare a far luce su alcuni aspetti importanti della classificazione. Infatti, tramite il codice degli algoritmi utilizzati, come mostrato nel precedente capitolo si possono stampare le matrici di confusione, che indicano per ogni classe quanti campioni sono stati classificati correttamente sul totale, e tra quelli che non sono invece stati classificati correttamente, si può vedere in quale classe sono stati erroneamente etichettati. Questo può dare indicazione su quali classi il classificatore sbaglia e provare magari ad intuirne il motivo.

Gli errori che sono presenti nelle matrici di confusione sono giustificati, poiché molte volte i campioni erano simili tra di loro, proprio per questo dei possibili sviluppi futuri su questa parte possono essere i seguenti: la prima è quella di aggiungere maggiori caratteristiche di traffico da estrapolare dai log di Bro per la generazione di un training-set che abbia più caratteristiche discriminanti per migliorare le misure di performance; la seconda invece potrebbe prevedere l'eliminazione di valori meno importanti e meno discriminanti dal training-set per vedere come si comportano le misure di performance.

In questo sviluppo futuro un ruolo fondamentale lo possono avere l'algoritmo Decision Tree e l'algoritmo Random Forest. Per tutti e due gli algoritmi si può provare a migliorare le loro prestazioni di predizione.

Nel primo casi si è visto che se l'algoritmo predice il rumore, cioè presta attenzione alle parti irrilevanti dei dati, allora è in *overfitting* andando a peggiorare le performance finali sul test set. Proprio per questo è possibile in futuro, quando viene usato questo algoritmo, fare delle scelte

accurate sul training-set. E' possibile, non sapendo a priori quali sono le variabili irrilevanti, navigare l'albero di decisione e incontrare dei casi dove:

$$X = a\ AND\ b$$

In questo caso è possibile dire che la caratteristica b è irrilevante in quei rami dell'albero dove a = 0, e quindi questo può portare non solo a migliorare la percentuale di predizione corretta dell'algoritmo ma anche ad eliminare tutte le caratteristiche che possono avere meno peso.

Nel secondo caso invece è possibile fare uno studio supplementare sull'importanza di ogni parametro, poiché per quanto riguarda l'algoritmo Random Forest viene resa disponibile una funzione che permette di aver restituiti la lista di tutti gli attributi utilizzati per la classificazione con il loro relativo peso. Questa funzione non è disponibile negli altri algoritmi usati che infatti vengono definiti di tipo *black box*, mentre il Random Forest viene definito di tipo *white box* cioè è reso trasparente all'utente il modo di procedere nella classificazione; da questo assunto è possibile in futuro decidere quali sono i parametri di classificazione con maggiore importanza e quali quelli minori, in modo da provare ad eliminarli e vedere come si comporta sia questo che gli altri algoritmi di classificazione.

Tramite le mappe di calore, si può andare a far luce su alcuni aspetti importanti della classificazione. Infatti, tramite il codice degli algoritmi utilizzati, come mostrato nel precedente capitolo si possono stampare le matrici di confusione, che indicano per ogni classe quanti campioni sono stati classificati correttamente sul totale, e tra quelli che non sono invece stati classificati correttamente, si può vedere in quale classe sono stati erroneamente etichettati. Questo può dare indicazione su quali classi il classificatore sbaglia e provare magari ad intuirne il motivo.

In tutte le performance analizzate a partire dalla matrice di confusione si è visto che con le tecniche viste finora non viene fornita nessuna informazione sul tipo di gravità dell'errore e cioè se esso sia 'lieve' o 'grave'. In futuro possono essere applicate anche tecniche come la *curva ROC* per poter studiare correttamente i rapporti fra allarmi veri e allarmi falsi.

Nonostante i risultati sembrano essere buoni per gran parte delle esecuzioni ci possono essere sicuramente dei margini di miglioramento che possono far salire ancora le misure di performance. Una miglioria potrebbe essere quella di far eseguire gli algoritmi con a turno un parametro in meno per la classificazione per andare a notare quali sono i parametri che, se tolti, non peggiorano le performance medie, in modo che possano essere eliminati. L'obiettivo quindi sarebbe quello di ridurre il numero di parametri al minimo possibile. Oppure si potrebbero tenere i parametri che hanno un peso sopra una certa soglia, ed andare ad eliminare quelli che hanno un peso irrisorio e notare se le performance salgono o comunque rimangono invariate. Questo tipo di approccio, configurabile a piacimento, potrebbe ridurre la parte computazionale di generazione degli attributi da utilizzare per ogni file .pcap, e quindi il miglioramento lo sia ha sia in termini di spazio che di tempo. Tempo che ridurrebbero anche gli algoritmi di classificazione, che ritrovandosi ad avere a che fare con meno attributi ridurrebbero sicuramente i tempi di risposta.

E' tuttavia complesso riuscire ad istruire un classificatore che riesca a classificare ogni tipo di richiesta, e quindi studi come questo servono a dimostrare come possano essere usati algoritmi di machine learning per provare a creare un sistema che non eviti l'attacco, ma che permetta di analizzare il traffico di rete e riuscire, grazie ad essi, a catalogare il traffico come malevolo o non malevolo e a dare quindi la possibilità ad un sistema di difesa di capire se si è in presenza o meno di un malware che ne sta facendo uso.

Una forte limitazione di questo lavoro di tesi è stata data anche dal fatto che sono stati raccolti un numero esiguo di campioni di traffico utile. Per poter quindi raggiungere risultati più attendibili si sarebbe dovuto incrementare il data-set, non limitandosi a qualche centinaio di campioni, per poter rendere un classificatore più robusto. Il problema si è riscontrato anche nel poter scaricare campioni di traffico utile dei rispettivi malware, in quanto molte volte o non erano presenti o venivano richieste alcune autorizzazioni non in nostro possesso.

Questo lavoro è stato provato in un ambiente controllato ma potrebbe in futuro essere inserito all'interno del perimetro aziendale di AizoOn dove si è svolta la tesi. Per quanto riguarda quindi l'applicabilità di questo lavoro di tesi, integrato in un prodotto aziendale (es. Aramis), potrebbe

essere provato con alcuni parametri rivisti e sostituiti da alcune stime, mentre altri non sono applicabili date le esigenze spazio-temporali. Ecco che quindi la sua effettiva utilità andrebbe del tutto valutata e vista dopo aver provato il lavoro all'interno del sistema aziendale.

Capitolo 7

Appendice

7.1 Manuale Utente

In questa sezione verranno descritti i passaggi per l'installazione e l'uso dei software usati in questa tesi.

7.1.1 Installazione di Bro

Passaggi da effettuare per l'installazione e l'uso del software Bro usato nella sezione 4.2.1

Prerequisiti

- Libpcap (http://www.tcpdump.org)
- OpenSSL libraries (http://www.openssl.org)
- BIND8 library
- Libz
- Python 2.6 or greater (for BroControl)

Per installare le dipendenze necessarie è possibile usare il seguente comando:

1. RPM / RedHat Linux-based:

   ```
   $ sudo yum install cmake make gcc gcc-c++ flex bison libpcap-devel openssl-devel
   ```

2. DEB:

   ```
   $ sudo apt-get install cmake make gcc g++ flex bison libpcap-dev libssl-dev python-dev
   ```

3. FreeBSD:

   ```
   $ sudo pkg install bash cmake swig bison python py27-sqlite3
   ```

Installazione

```
$ git clone --recursive git://git.bro.org/bro

$ ./configure

$ make

$ make install
```

Esempio d'uso di Bro

In questa tesi Bro è stato utilizzato per la generazione dei log dopo la sottomissione dei vari file .pcap all'interno del suo motore.

Verranno adesso descritti i comandi da eseguire per la generazione dei file di log.

- Il primo comando serve per inserire all'interno del motore di Bro il file .pcap di interesse e tramite lo script dns_entropy.bro generare il file di log.

  ```
  $ bro -Cr firefox.pcap bro/site/bro-scripts/dns_entropy.bro
  ```

- Il secondo comando serve invece per generare un file .csv a partire dal file di log.

  ```
  bro2csv -i bro_log/firefox/dns_entropy.log
  ```

A questo punto è stato generato il file .csv contenente l'analisi dns fatta da Bro. E' possibile tramite algoritmi manipolare il file per poter estrarre le colonne di nostro interesse.

7.1.2 Installazione di Python 3.x su Ubuntu 16.04

La versione di Ubuntu 16.04 ha già nativamente installato python. Per fare in modo di avere la versione aggiornata di python eseguire i seguenti comandi:

```
$ sudo apt-get update
$ sudo apt-get -y upgrade
```

Per la gestione dei pacchetti software usati(es. numpy o pandas) è possibile eseguire l'installazione dando i seguenti comandi:

```
$ sudo apt-get install -y python3-pip
$ pip3 install \textit{package name}
```

Per l'esecuzione dei vari algoritmi è stata usata la IDE python PyCharm, ma è possibile eseguire tutti gli algoritmi andando nella cartella dove sono presenti i sorgenti e digitando il comando:

```
$ python nomeeseguibile.py
```

Esempio d'uso

In questo caso un esempio d'uso con gli algoritmi scritti in python è il seguente:

- Aprire il terminale e spostarsi nella cartella dei sorgenti.
- Una volta all'interno della cartella dove sono presenti i nostri sorgenti è possibile digitare i seguenti comandi per ognuno degli algoritmi descritti:

  ```
  $ python randomforest.py

  $ python decisiontree.py

  $ python naivebayes.py

  $ python knearestneighbors.py
  ```

Quando l'esecuzione sarà terminata verranno dati in output sul terminale i risultati dell'esecuzione.

7.1.3 Installazione PyCharm su Ubuntu 16.04

Installazione OpenJDK 8

Per l'installazione di OpenJDK8 (variante open source di Java Development Kit) digitare il seguente comando:

```
$ sudo apt-get install -y openjdk-8-jre-headless
```

Installazione PyCharm

Per installare PyCharm bisogna aggiungere il repository Ubuntu Make digitando il seguente comando:

```
$ sudo add-apt-repository -y ppa:ubuntu-desktop/ubuntu-make
```

Bisogna installare il pacchetto appena scaricato digitando i seguenti comandi:

```
$ sudo apt-get update
$ sudo apt-get install -y ubuntu-make
```

Per installare PyCharm Community Version bisogna digitare il seguente comando:

```
$ umake ide pycharm
```

Quando il software è installato, l'ambiente è già pronto per poter mandare in esecuzione i sorgenti. I passaggi da seguire per l'esecuzione sono i seguenti:

- Aprire la cartella contente i sorgenti da importare in PyCharm.
- Cliccare con il tasto destro sul sorgente di nostro interesse e selezionare 'Apri con PyCharm'
- Appena il progetto verrà caricato all'interno di PyCharm, cliccare con il tasto destro sul sorgente e premere 'Run'
- Finita l'esecuzione verranno visualizzati a video i risultati.

7.2 Manuale del programmatore

In questa sezione verranno inserite tutte le parti di codice utili ai fini di questa tesi. Alla fine di ogni porzione di codice verranno descritte le funzioni usate.

7.2.1 Codice WIN32.ismdoor.gen

Codice che replica la comunicazione tra il malware WIN32.ismdoor.gen e il suo server C2 visto nella sezione 4.1.3

Server

```
import socket,glob,json,sqlite3,random
from threading import Thread

ip = 'localhost'
port = 53
readbuffer = 4096

# Create a UDP socket
sock_dns = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock_udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

#Database path
db = '/mydb'

store_domain = ""

comm_array = [
               '10.10.10.10',
               '0.0.0.0',
               '210.13.435.30',
               '92.31.10.112',
               '20.20.20.20',
               '190.234.1.8',
               '35.33.56.78',
               '130.20.5.78'
            ]


#################
def writeToJSONFile(path, fileName,domain,command):

    filePathNameWExt = './' + path + '/' + fileName + '.zone'

    #write value into json file
    text = {
        "$origin": domain,
        "$ttl": 3600,
        "soa": {
            "mname": "ns1.howcode.org.",
            "rname": "admin.howcode.org.",
            "serial": "{time}",
            "refresh": 3600,
            "retry": 600,
            "expire": 604800,
            "minimum": 86400
        },
        "ns": [
            { "host": "ns1.howcode.org." },
            { "host": "ns2.howcode.org." }
        ],
        "a": [
            { "name": "@", "ttl": 444, "value": command }
        ]}
    with open(filePathNameWExt, 'w') as fp:
        json.dump(text, fp, indent=3)
```

```
def load_zones():

    jsonzone = {}
    zonefiles = glob.glob('zones/*.zone')

    for zone in zonefiles:
        with open(zone) as zonedata:
            data = json.load(zonedata)
            zonename = data["$origin"]
            jsonzone[zonename] = data
    return jsonzone

def getflags(flags):

    byte1 = bytes(flags[:1])
    #byte2 = bytes(flags[1:2])

    #rflags = ''

    QR = '1'

    OPCODE = ''
    for bit in range(1,5):
        OPCODE += str(ord(byte1)&(1<<bit))

    AA = '1'

    TC = '0'

    RD = '0'

    # Byte 2

    RA = '0'

    Z = '000'

    RCODE = '0000'

    return int(QR+OPCODE+AA+TC+RD, 2).to_bytes(1,
        byteorder='big')+int(RA+Z+RCODE, 2).to_bytes(1, byteorder='big')

def getquestiondomain(data):

    state = 0
    expectedlength = 0
    domainstring = ''
    domainparts = []
    x = 0
    y = 0
    for byte in data:
        if state == 1:
            if byte != 0:
                domainstring += chr(byte)
            x += 1
            if x == expectedlength:
                domainparts.append(domainstring)
```

```
                domainstring = ''
                state = 0
                x = 0
            if byte == 0:
                domainparts.append(domainstring)
                break
        else:
            state = 1
            expectedlength = byte
        y += 1

    questiontype = data[y:y+2]

    return (domainparts, questiontype)

def getzone(domain):

    #qui dalla richiesta del client estrapolo il domain
    global zonedata
    global store_domain
    global last_command

    zone_name = '.'.join(domain)

    #variable contain domain name to use for storedb
    store_domain = zone_name
    print('domain: ',store_domain)
    last_command = random.choice(comm_array)
    print('last_command saved: ',last_command)
    writeToJSONFile('zones/','file',store_domain,last_command)

    zonedata = load_zones()

    return zonedata[zone_name]

def getrecs(data):
    domain, questiontype = getquestiondomain(data)
    qt = ''
    if questiontype == b'\x00\x01':
        qt = 'a'

    zone = getzone(domain)

    return (zone[qt], qt, domain)

def buildquestion(domainname, rectype):
    qbytes = b''

    for part in domainname:
        length = len(part)
        qbytes += bytes([length])

        for char in part:
            qbytes += ord(char).to_bytes(1, byteorder='big')

    if rectype == 'a':
        qbytes += (1).to_bytes(2, byteorder='big')
```

```
    qbytes += (1).to_bytes(2, byteorder='big')

    return qbytes

def rectobytes(domainname, rectype, recttl, recval):

    rbytes = b'\xc0\x0c'

    if rectype == 'a':
        rbytes = rbytes + bytes([0]) + bytes([1])

    rbytes = rbytes + bytes([0]) + bytes([1])

    rbytes += int(recttl).to_bytes(4, byteorder='big')

    if rectype == 'a':
        rbytes = rbytes + bytes([0]) + bytes([4])

        for part in recval.split('.'):
            rbytes += bytes([int(part)])
    return rbytes

def buildresponse(data):

    # Transaction ID
    TransactionID = data[:2]

    # Get the flags
    Flags = getflags(data[2:4])

    # Question Count
    QDCOUNT = b'\x00\x01'

    # Answer Count
    ANCOUNT = len(getrecs(data[12:])[0]).to_bytes(2, byteorder='big')

    # Nameserver Count
    NSCOUNT = (0).to_bytes(2, byteorder='big')

    # Additonal Count
    ARCOUNT = (0).to_bytes(2, byteorder='big')

    dnsheader = TransactionID+Flags+QDCOUNT+ANCOUNT+NSCOUNT+ARCOUNT

    # Create DNS body
    dnsbody = b''

    # Get answer for query
    records, rectype, domainname = getrecs(data[12:])

    dnsquestion = buildquestion(domainname, rectype)

    #store domain function
    storedb(db,store_domain,last_command)

    for record in records:
```

```
        dnsbody += rectobytes(domainname, rectype, record["ttl"],
            last_command)

    return dnsheader + dnsquestion + dnsbody
#################

def ping():
    try:

        # Bind the socket to the port
        server_address = (ip, 3000)
        sock_udp.bind(server_address)

        message = 'PONG'

        while True:
            data, address = sock_udp.recvfrom(readbuffer)

            #print('received %s bytes from %s' % (len(data), address))
            print('received',data)

            if data:
                sock_udp.sendto(bytes(message.encode()), address)
                print('sending',message)

    except Exception:
        print("server ping problem..close socket")
        sock_dns.close()
        exit()

def recv():
    try:
        server_address = (ip, port)
        print('starting up on %s port %s' % server_address)
        sock_dns.bind(server_address)

        while True:
            print('waiting to receive message')
            data, address = sock_dns.recvfrom(readbuffer)

            print('received %s bytes from %s' % (len(data), address))
            #print(data.decode())

            if data:
                r = buildresponse(data)
                sock_dns.sendto(r, address)
                print('risposta inviata')
                #print('sent %s bytes back to %s' % (sent, address))

    except Exception:
        print("server send problem")
        #sock_dns.close()
        #sock_udp.close()
        exit()

def createdb(db):
    try:
```

```
    # Creates or opens a file called mydb with a SQLite3 DB
        conn = sqlite3.connect(db)
        # Get a cursor object
        cursor = conn.cursor()
        # Check if table users does not exist and create it
        cursor.execute('''CREATE TABLE IF NOT EXISTS history(domain TEXT
            PRIMARY KEY, last_command TEXT)''')
        # Commit the change
        conn.commit()
        # Catch the exception
    except sqlite3.Error as e:
        # Roll back any change if something goes wrong
        conn.rollback()
        print('Create db fail')
        raise e
    finally:
        # Close the db connection
        conn.close()

def storedb(db,domain,last_command):
    try:
        conn = sqlite3.connect(db)
        cursor = conn.cursor()

        #check is element exists
        cursor.execute("SELECT * FROM history WHERE domain=?", (domain,))
        entry = cursor.fetchone()
        print("entry",entry)

        if entry is None:
            print("Entry not found..stored")
            #cursor.execute('INSERT INTO ProSolut (Col1, Col2, Col3) VALUES
                (?,?,?)', ('a', 'b', 'c'))
            cursor.execute("INSERT INTO history VALUES(?,?)",
                (domain,last_command))
            conn.commit()
            print('Insert new entry into db success')
        else:
            print('Entry found')
            cursor.execute("UPDATE history SET last_command = ? WHERE
                domain=?", (last_command,domain))
            conn.commit()
            print('Insert exist entry into db success')

    except Exception as e:
        # Roll back any change if something goes wrong
        conn.rollback()
        print('Insert into db fail')
        raise e
    finally:
        # Close the db connection
        conn.close()


"""
if __name__ == "__main__":
    ping()
```

```
    recv()
    time.sleep(5)

"""
class myClassA(Thread):
    def __init__(self):
        Thread.__init__(self)
        self.daemon = True
        self.start()
    def run(self):
        while True:
            recv()

class myClassB(Thread):
    def __init__(self):
        Thread.__init__(self)
        self.daemon = True
        self.start()
    def run(self):
        createdb(db)
        while True:
            ping()



myClassA()
myClassB()
while True:
    pass
```

- *socket.socket()*: funzione tramite la quale vengono creati i socket per l'invio di messaggi tra server e client. Al suo interno vengono passati l'indirizzo Internet composto da indirizzo IP dell'host (32 bit) e da numero di porta (16 bit) e il tipo di comunicazione.
- *socket.bind()*: funzione tramite la quale il socket sta in ascolto.
- *recvfrom()*: funzione tramite la quale vengono ricevute le richieste DNS da parte del client.
- *sendto()*: funzione tramite la quale il server invia la risposta DNS al client.
- *commarray*: array con all'interno i comandi in indirizzi ip che il server C2 invia al client e che il client memorizza in un file sqlite.
- *writeToJSONFile()*: funziona tramite la quale vengono inseriti in un file json tutti i valori utili a scrivere il file zone per simulare una comunicazione DNS.
- *getflags()*: funzione tramite la quale vengono settati i flag del protocollo DNS.
- *getzone()*: funzione che serve per estrapolare il dominio richiesto dal client. Inoltre all'interno di questa funzione viene anche memorizzato il dominio in questione all'interno di un db sqlite.
- *buildquestion()*: funzione tramite la quale il server riesce ad interpretare correttamente la richiesta effettuata dal client. In ingresso riceve il domain name e il tipo di record DNS.
- *rectobytes()*: funzione che trasforma la richiesta in byteorder.
- *buildresponse()*: funzione tramite la quale il server crea la risposta da inviare partendo dalla richiesta del client. In questa funzione vengono settati tutti i parametri utili per la creazione del corpo del messaggio DNS.

- *ping()*: funzione usata dal server per rispondere ai messaggi di ping da parte del client, che si accerta che il suo C2 è attivo e pronto per trasmettere i comandi.
- *createdb()*: funzione che serve quando il server viene avviato per creare un db sqlite in cui mantenere traccia delle comunicazioni con i vari client e i comandi inviati.
- *storedb()*: funzione usata per la memorizzazione, all'interno del file sqlite, della comunicazione con l'i-esimo client e dei comandi inviatigli.

Client

```
import socket,time,string,random,bitstring,codecs
from threading import Thread

ip = 'localhost'
port = 53
readbuffer = 4096

ran_str = ''.join(random.choices(string.ascii_letters + string.digits, k=32))
domain = "n.n.c." + ran_str.upper() + ".dnslookupdater.org"

ran_id = "0x1a2"
ran_id+=random.choice("abcdef")

# Create a UDP socket
sock_udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sock_dns = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

retval = True
#retval2 = True

##########################
def to_hex_string(x):
    """
    Encodes either a positive integer or string to its hexadecimal
        representation.
    """

    result = "0"

    if x.__class__.__name__ == "int" and x >= 0:
        result = hex(x)

        if x < 16:
            result = "0" + result[2:]

    elif x.__class__.__name__ == "str":
        result = "".join([hex(ord(y))[2:] for y in x])

    return "0x" + result



def send_message(host_name_to):
    try:
        server_address = (ip,port)
```

```
host_name_to = host_name_to.split(".")

# Construct the DNS packet consisting of header + QNAME + QTYPE +
    QCLASS.

DNS_QUERY_FORMAT = [
    "hex=id"
  , "bin=flags"
  , "uintbe:16=qdcount"
  , "uintbe:16=ancount"
  , "uintbe:16=nscount"
  , "uintbe:16=arcount"
]

DNS_QUERY = {
    "id": ran_id
  , "flags": "0b0000000100000000" # Standard query. Ask for recursion.
  , "qdcount": 1 # One question.
  , "ancount": 0
  , "nscount": 0
  , "arcount": 0
}

# Construct the QNAME:
# size|label|size|label|size|...|label|0x00


j = 0

for i, _ in enumerate(host_name_to):

    host_name_to[i] = host_name_to[i].strip()

    DNS_QUERY_FORMAT.append("hex=" + "qname" + str(j))

    DNS_QUERY["qname" + str(j)] = to_hex_string(len(host_name_to[i]))

    j += 1

    DNS_QUERY_FORMAT.append("hex=" + "qname" + str(j))

    DNS_QUERY["qname" + str(j)] = to_hex_string(host_name_to[i])

    j += 1

# Add a terminating byte.

DNS_QUERY_FORMAT.append("hex=qname" + str(j))

DNS_QUERY["qname" + str(j)] = to_hex_string(0)

# End QNAME.

# Set the type and class now.

DNS_QUERY_FORMAT.append("uintbe:16=qtype")
```

```
DNS_QUERY["qtype"] = 1 # For the A record.

DNS_QUERY_FORMAT.append("hex=qclass")

DNS_QUERY["qclass"] = "0x0001" # For IN or Internet.

# Convert the struct to a bit string.

data = bitstring.pack(",".join(DNS_QUERY_FORMAT), **DNS_QUERY)

print('query sending "%s"' % data)
sock_dns.sendto(data.tobytes(), server_address)
sock_dns.settimeout(5)

data, server_address = sock_dns.recvfrom(readbuffer)
sock_dns.settimeout(5)

# Convert data to bit string.

data = bitstring.BitArray(bytes=data)

# Unpack the receive DNS packet and extract the IP the host name
    resolved to.

# Get the host name from the QNAME located just past the received
    header.

host_name_from = []

# First size of the QNAME labels starts at bit 96 and goes up to bit
    104.
# size|label|size|label|size|...|label|0x00

x = 96
y = x + 8

for i, _ in enumerate(host_name_to):

    increment = (int(str(data[x:y].hex), 16) * 8)

    x = y
    y = x + increment


    host_name_from.append(codecs.decode(data[x:y].hex,
        "hex_codec").decode())

    x = y
    y = x + 8 # Eight bits to a byte.

# Get the response code.
# This is located in the received DNS packet header at
# bit 28 ending at bit 32.

response_code = str(data[28:32].hex)

result = {'host_name': None, 'ip_address': None}
```

```
        # Check for errors.

        if (response_code == "0"):

            result['host_name'] = ".".join(host_name_from)

            # Assemble the IP address the host name resolved to.
            # It is usually the last four octets of the DNS
            # packet--at least for A records.

            result['ip_address'] = ".".join([
                str(data[-32:-24].uintbe)
              , str(data[-24:-16].uintbe)
              , str(data[-16:-8].uintbe)
              , str(data[-8:].uintbe)
            ])

        elif (response_code == "1"):
            print("\nFormat error. Unable to interpret query.\n")

        elif (response_code == "2"):
            print("\nServer failure. Unable to process query.\n")

        elif (response_code == "3"):
            print("\nName error. Domain name does not exist.\n")

        elif (response_code == "4"):
            print("\nQuery request type not supported.\n")

        elif (response_code == "5"):
            print("\nServer refused query.\n")

        return result


    except socket.timeout:
        print("socket timeout..response not received")
        sock_dns.close()
        #return False
        exit()

    except Exception:
        print('message not send..generic Exception')
        sock_dns.close()
        #return False
        exit()


def format_hex(hexa):
    """format_hex returns a pretty version of a hex string"""
    octets = [hexa[i:i+2] for i in range(0, len(hex), 2)]
    pairs = [" ".join(octets[i:i+2]) for i in range(0, len(octets), 2)]
    return "\n".join(pairs)
##############################

def ping():
```

```
    try:
        server_address = (ip,3000)

        message = 'PING'
        # Send data
        print('heartbit sending "%s"' % message)
        sock_udp.sendto(bytes(message.encode()), server_address)
        sock_udp.settimeout(10)

        # Receive response
        data, server_address = sock_udp.recvfrom(readbuffer)
        sock_udp.settimeout(10)
        print('heartbit received "%s"' % data)
        return True

    except socket.timeout:
        print("Pong not received...server is not running...close socket")
        sock_udp.close()
        sock_dns.close()
        #exit()
        return False
"""
if __name__ == "__main__":

    while 1:
        ping()
        time.sleep(5)
        send_message(domain)
        time.sleep(5)

"""

class myClassA(Thread):
    def __init__(self):
        Thread.__init__(self)
        self.daemon = True
        self.start()
    def run(self):
        while True:
            global retval
            time.sleep(5)
            send_message(domain)

class myClassB(Thread):
    def __init__(self):
        Thread.__init__(self)
        self.daemon = True
        self.start()
    def run(self):
        global retval
        while True:
            retval = ping()
            time.sleep(2)

myClassB()
myClassA()
while retval:
```

```
    pass
```

- *socket.socket()*: funzione tramite la quale vengono creati i socket per l'invio di messaggi tra client e server. Al suo interno vengono passati l'indirizzo Internet composto da indirizzo IP dell'host (32 bit) e da numero di porta (16 bit) e il tipo di comunicazione.
- *socket.bind()*: funzione tramite la quale il socket sta in ascolto.
- *recvfrom()*: funzione tramite la quale vengono ricevute le richieste DNS da parte del client.
- *sendmessage()*: funzione tramite la quale viene creato tutto il corpo della query DNS da inviare al server C2.
- *sendto()*: funzione tramite la quale il client invia la query DNS al server.
- *settimeout()*: funzione che serve per settare un timeout che non renda la funzione di *recv* bloccante. Se il client non riceve una risposta entro un tot di tempo scelto arbitrariamente, la socket verrà chiusa.
- *tohexstring()*: funzione tramite la quale viene effettuata una codifica di un numero intero positivo o di una stringa al suo valore esadecimale.
- *ping()*: funzione che serve al client per vedere se il server C2 è attivo. Vengono scambiati dei messaggi di ping tra client e server e qualora il server non fosse attivo il client non procederà all'invio della query DNS.

7.2.2 Algoritmi Machine learning

Codice degli algoritmi di machine learning usati e visti nella sezione 6.1.3

Random Forest

```
# Required Python Packages
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import numpy as np

#File Paths

#in questo algoritmo non uso direttamente il mio file result.csv ma uso
    result_rf.csv dove i dati sono separati da virgola e senza headers
#con la funzione data_to_csv lo trasformo come result.csv e lo chiamo
    training_set.csv

INPUT_PATH = ''
path_to_training = ''
path_to_test = ''

# Headers
HEADERS = ['SIZE_IN','SIZE_OUT','ENTROPY','%CHAR','%NUM','LEVEL','CLASS']



def read_data(path):
```

```
    """
    Read the data into pandas dataframe
    :param path:
    :return:
    """
    data = pd.read_csv(path)
    return data


def get_headers(dataset):
    """
    dataset headers
    :param dataset:
    :return:
    """
    return dataset.columns.values


def add_headers(dataset, headers):
    """
    Add the headers to the dataset
    :param dataset:
    :param headers:
    :return:
    """
    dataset.columns = headers
    return dataset


def data_file_to_csv():
    """

    :return:
    """

    # Headers
    headers = ['SIZE_IN','SIZE_OUT','ENTROPY','%CHAR','%NUM','LEVEL','CLASS']
    # Load the dataset into Pandas data frame
    dataset = read_data(INPUT_PATH)
    # Add the headers to the loaded dataset
    dataset = add_headers(dataset, headers)
    # Save the loaded dataset into csv format
    dataset.to_csv(path_to_training, index=False)
    print("File saved ...!")


def split_dataset(dataset, train_percentage, feature_headers, target_header):
    """
    Split the dataset with train_percentage
    :param dataset:
    :param train_percentage:
    :param feature_headers:
    :param target_header:
    :return: train_x, test_x, train_y, test_y
    """

    # Split dataset into train and test dataset
```

```
    train_x, test_x, train_y, test_y =
        train_test_split(dataset[feature_headers], dataset[target_header],
                                                   train_size=train_percentage)

    return train_x, test_x, train_y, test_y


def handel_missing_values(dataset, missing_values_header, missing_label):
    """
    Filter missing values from the dataset
    :param dataset:
    :param missing_values_header:
    :param missing_label:
    :return:
    """

    return dataset[dataset[missing_values_header] != missing_label]


def random_forest_classifier(features, target):
    """
    To train the random forest classifier with features and target data
    :param features:
    :param target:
    :return: trained random forest classifier
    """
    clf = RandomForestClassifier()
    clf.fit(features, target)
    return clf


def dataset_statistics(dataset):
    """
    Basic statistics of the dataset
    :param dataset: Pandas dataframe
    :return: None, print the basic statistics of the dataset
    """
    print(dataset.describe())

def plot_confusion_matrix(cm, title='Confusion matrix_random_forest',
    cmap=plt.cm.Blues):
    plt.imshow(cm, cmap=plt.cm.Blues)
    plt.xlabel("Predicted labels")
    plt.ylabel("True labels")
    plt.xticks([], [])
    plt.yticks([], [])
    plt.title('Confusion matrix ')
    plt.colorbar()
    plt.show()


def main():
    """
    Main function
    :return:
    """
    data_file_to_csv()
```

```
    # Load the csv file into pandas dataframe
    dataset = pd.read_csv(path_to_training)
    # Get basic statistics of the loaded dataset
    #dataset_statistics(dataset)

    # Filter missing values
    #dataset = handel_missing_values(dataset, HEADERS[6], '?')


    print(HEADERS[1:-1])
    train_x, test_x, train_y, test_y = split_dataset(dataset, 0.3,
        HEADERS[1:-1], HEADERS[-1])


    # Train and Test dataset size details
    print("Train_x Shape :: ", train_x.shape)
    print("Train_y Shape :: ", train_y.shape)
    print("Test_x Shape :: ", test_x.shape)
    print("Test_y Shape :: ", test_y.shape)


    # Create random forest classifier instance
    trained_model = random_forest_classifier(train_x, train_y)

    #print("Trained model :: ", trained_model)
    predictions = trained_model.predict(test_x)

    for i in range(0, 105):
        print("Actual outcome :: {} and Predicted outcome ::
            {}".format(list(test_y)[i], predictions[i]))

    print("Train Accuracy :: ", accuracy_score(train_y,
        trained_model.predict(train_x)))
    print("Test Accuracy :: ", accuracy_score(test_y, predictions))
    print("Confusion matrix \n", confusion_matrix(test_y, predictions))



    np.set_printoptions(precision=2)
    fig = plt.figure()
    plot_confusion_matrix(confusion_matrix(test_y, predictions))
    # Create new directory
    output_dir = "../graphics"
    fig.savefig('{}/conf_matrix_random_forest.png'.format(output_dir))

if __name__ == "__main__":
    main()
```

- *readdata()*: funzione che prende in ingresso il data-set da leggere in formato .csv e ritorna un data-set in formato *pandas dataframe*.
- *getheaders()*: funzione che prende il data-set appena convertito e ritorna il valore di ogni colonna.
- *datafiletocsv()*: funzione che serve per la creazione del training-set e del test-set e che inserisce soltanto le colonne di nostro interesse.
- *splidataset()*: divide i data-set in training e test set.

- *randomforestclassifier()*: funzione tramite la quale viene creato il classificatore di tipo random forest.
- *plotconfusionmatrix()*: funzione tramite la quale viene creata la matrice di confusione dalla quale verranno estratti tutti i valori di prestazione del classificatore.

Decision Tree

```
#Pandas is used for data manipulation
import pandas as pd
import graphviz
from sklearn import tree
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import numpy as np

#path
path_to_training = ''
path_to_test = ''


#leggo i dati dal csv
features = pd.read_csv(path_to_training)

#in ingresso al classificatore vengono dati 2 array; X pari al tutto il set
    escluso la label scelta, Y pari alla label scelta
X_train = features.drop(['CLASS'], axis=1)
Y_train = features['CLASS']

#creo un oggetto di tipo DecisionTree e con il comando fit creo il modello
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X_train.values,Y_train)

#creo la predizione dopo aver dato il comando predict al modello
#prediction = clf.predict(X.values)

#print(prediction)

#viene dato in paso un altro set per vedere come si comporta il modello di
    predizione
df = pd.read_csv(path_to_test)
X_test = df.drop(['CLASS'], axis=1)
Y_test = df['CLASS']

Y_prediction = clf.predict(X_test)
print("Prediction for Decision Tree: \n",Y_prediction)

print("\nAccuracy:",accuracy_score(Y_test,Y_prediction))
print(" \nConfusion matrix \n", confusion_matrix(Y_test, Y_prediction))

dot_data = tree.export_graphviz(clf, out_file=None)
graph = graphviz.Source(dot_data)
graph.render('../graphics/decision_tree')

def plot_confusion_matrix(cm, title='Confusion matrix_decision_tree',
    cmap=plt.cm.Blues):
```

```
    plt.imshow(cm, cmap=plt.cm.Blues)
    plt.xlabel("Predicted labels")
    plt.ylabel("True labels")
    plt.xticks([], [])
    plt.yticks([], [])
    plt.title('Confusion matrix ')
    plt.colorbar()
    plt.show()

np.set_printoptions(precision=2)
fig = plt.figure()
plot_confusion_matrix(confusion_matrix(Y_test, Y_prediction))

# Create new directory
output_dir = "../graphics"
fig.savefig('{}/conf_matrix_decision_tree.png'.format(output_dir))
```

- *readcsv()*: funzione tramite la quale viene letto il data-set contenente i dati di training.
- *DecisionTreeClassifier()*: funzione che serve per la creazione del classificatore di tipo Decision Tree a cui vengono dati in pasto due array; uno contenente tutto il training-set escluso la label scelta da classificare e un'altro contenente soltanto i valori del training-set della label scelta.
- *predict()*: funzione tramite la quale viene effettuata la predizione sui campioni del test-set.
- *plotconfusionmatrix()*: funzione tramite la quale viene creata la matrice di confusione dalla quale verranno estratti tutti i valori di prestazione del classificatore.

Naive Bayes

```
import pandas as pd
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import numpy as np

#path
path_to_training = ''
path_to_test = ''

#leggo i dati dal csv
dataset = pd.read_csv(path_to_training)

#in ingresso al classificatore vengono dati 2 array; X pari al tutto il set
    escluso la label scelta, Y pari alla label scelta
X_train = dataset.iloc[:, :-1].values
Y_train = dataset.iloc[:, 6].values

dataset_test = pd.read_csv(path_to_test)
X_test = dataset.iloc[:, :-1].values
Y_test = dataset.iloc[:, 6].values

#create naive bayes classifier
gaunb = GaussianNB()
```

```
#train classifier with dataset
gaunb = gaunb.fit(X_train, Y_train)

# predict using classifier
prediction = gaunb.predict(X_test)

confusion_matrix = confusion_matrix(Y_test, prediction)

print("Train Accuracy :: ", accuracy_score(Y_train,prediction))
print("\nTest Accuracy :: ", accuracy_score(prediction, Y_test))
print("\nConfusion matrix \n", confusion_matrix)

def plot_confusion_matrix(cm, title='Confusion matrix_naive_bayes',
    cmap=plt.cm.Blues):
    plt.imshow(cm, cmap=plt.cm.Blues)
    plt.xlabel("Predicted labels")
    plt.ylabel("True labels")
    plt.xticks([], [])
    plt.yticks([], [])
    plt.title('Confusion matrix ')
    plt.colorbar()
    plt.show()

np.set_printoptions(precision=2)
fig = plt.figure()
plot_confusion_matrix(confusion_matrix)
# Create new directory
output_dir = "../graphics"
fig.savefig('{}/conf_matrix_naive_bayes.png'.format(output_dir))
```

- *readcsv()*: funzione tramite la quale viene letto il data-set contenente i dati di training.
- *GaussianNB()*: funzione che serve per la creazione del classificatore di tipo Naive Bayes a cui vengono dati in pasto due array; uno contenente tutto il training-set escluso la label scelta da classificare e un'altro contenente soltanto i valori del training-set della label scelta.
- *gaunb.fit()*: funzione tramite la quale viene creato il modello ed effettuato il training del classificatore.
- *gaunb.predict()*: funzione tramite la quale viene effettuata la predizione sui campioni del test-set.
- *plotconfusionmatrix()*: funzione tramite la quale viene creata la matrice di confusione dalla quale verranno estratti tutti i valori di prestazione del classificatore.

K-nearest neighbors

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
import pandas as pd

#path
path_to_training = ''
```

```
path_to_test = ''


#leggo i dati dal csv
dataset = pd.read_csv(path_to_training)

#in ingresso al classificatore vengono dati 2 array; X pari al tutto il set
    escluso la label scelta, Y pari alla label scelta
X_train = dataset.iloc[:, :-1].values
Y_train = dataset.iloc[:, 6].values


X_train, X_test, y_train, y_test = train_test_split(X_train, Y_train,
    test_size=0.30)

#scaling
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

#Training and Predictions
#5 è il valore di n_neighbors; non c'è un valore ideale ma questo è quello
    usato per iniziare

classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

#Evaluating the Algorithm
print('Confusion Matrix :\n',confusion_matrix(y_test, y_pred))
print('\nClassification_report :\n',classification_report(y_test, y_pred))

#Comparing Error Rate with the K Value

error = []

# Calculating error for K values between 1 and 40
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    error.append(np.mean(pred_i != y_test))

fig = plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',
        markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')

# Create new directory
output_dir = "../graphics"

fig.savefig('{}/k_nearest_neighbors.png'.format(output_dir))
```

```
def plot_confusion_matrix(cm, title='Confusion matrix_k_nearest_neighbors',
    cmap=plt.cm.Blues):
    plt.imshow(cm, cmap=plt.cm.Blues)
    plt.xlabel("Predicted labels")
    plt.ylabel("True labels")
    plt.xticks([], [])
    plt.yticks([], [])
    plt.title('Confusion matrix ')
    plt.colorbar()
    plt.show()

np.set_printoptions(precision=2)
fig = plt.figure()
plot_confusion_matrix(confusion_matrix(y_test, y_pred))

# Create new directory
output_dir = "../graphics"
fig.savefig('{}/conf_matrix_k_nearest_neighbors.png'.format(output_dir))
```

- *readcsv()*: funzione tramite la quale viene letto il data-set contenente i dati di training.
- *traintestsplit()*: funzione tramite la quale al classificatore vengono dati in ingresso due array, X contenente tutto il training-set escluso la label scelta da predire, Y contenente solo la label scelta.
- *KNeighborsClassifier()*: funzione tramite la quale viene creato un oggetto di tipo KNeighborsClassifier.
- *classifier.fit()*: funzione tramite la quale viene creato il modello di predizione ed effettuato il training del classificatore.
- *classifier.fit.predict()*: funzione tramite la quale viene effettuata la predizione sui campioni del test-set.
- *plotconfusionmatrix()*: funzione tramite la quale viene creata la matrice di confusione dalla quale verranno estratti tutti i valori di prestazione del classificatore. In questo algoritmo è stato fatto variare anche il valore di k tra 1 e 40 per vederne l'error rate.

Bibliografia

[1] ATT&CK MATRIX, https://attack.mitre.org/wiki/ATT&CK_Matrix/

[2] S.Caltagirone, A.Pendergast, C.Betz, "The Diamond Model of Intrusion Analysis", Jul 05, 2013, pp. 8-14, http://www.activeresponse.org/wp-content/uploads/2013/07/diamond.pdf/

[3] Software APT1 https://attack.mitre.org/wiki/Group/G0006/

[4] Software APT2 https://attack.mitre.org/wiki/Group/G0024/

[5] Software APT18 https://attack.mitre.org/wiki/Group/G0026/

[6] Software APT19 https://www.fireeye.com/current-threats/apt-groups.html#apt19/

[7] Software APT32 https://www.fireeye.com/current-threats/apt-groups.html#apt32/

[8] Software APT33 https://www.fireeye.com/current-threats/apt-groups.html#apt33/

[9] Software APT34 https://www.fireeye.com/current-threats/apt-groups.html#apt34/

[10] S.Sheridan, A.Keane, "Detection of DNS based covert channels', European Conference on Information Warfare and Security, ECCWS, Jan, 2015, pp. 2, https://arrow.dit.ie/cgi/viewcontent.cgi?article=1001&context=nsdcon/

[11] P. Mockapetris, "Domain names - implementation and specification", RFC-1035, November 1987, DOI 10.17487/RFC1035

[12] Documentazione di Bro https://www.bro.org/documentation/index.html

[13] Piattaforma hybrid-analysis https://www.hybrid-analysis.com/

[14] Piattaforma virus share https://virusshare.com/

[15] Piattaforma packet total https://packettotal.com/

[16] Piattaforma virus total https://www.virustotal.com/it/

[17] Piattaforma reverse[.]it https://www.reverse.it/

[18] Script Bro https://github.com/BrashEndeavours/bro-scripts/blob/master/dns_entropy.bro/

[19] Apprendimento Automatico https://it.wikipedia.org/wiki/Apprendimento_automatico/

[20] Xuan Dau Hoang, Quynh Chi Nguyen "Botnet Detection Based On Machine Learning Techniques Using DNS Query Data", May 18, 2018, pp. 4-10, DOI 10.3390/fi100500438

[21] S. Ray "Improve your model performance using cross validation. ", https://www.analyticsvidhya.com/blog/2018/05/improve-model-performance-cross-validation-in-python-r/

[22] Decision Tree https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html/

[23] Random Forest https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html/

[24] N. Donges "The random forest algorithm", https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd/

[25] K-nearest neighbors https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html/

[26] A. Bronshtein "A Quick Introduction to K-Nearest Neighbors Algorithm", https://medium.com/@adi.bronshtein/a-quick-introduction-to-k-nearest-neighbors-algorithm-62214cea29c7/

[27] Naive Bayes https://scikit-learn.org/stable/modules/naive_bayes.html/

[28] S. Ray "6 Easy Steps to Learn Naive Bayes Algorithm ", https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/