POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Analisi intelligente di traffico di rete

Relatori
prof. Antonio Lioy
dott. Daniele Canavese
dott. Leonardo Regano

Candidato
Michele Maiullari

Anno accademico 2018-2019

A mia madre, che mi ha sempre sostenuto ed incoraggiato ad andare avanti

A mia sorella, che mi ha strappato un sorriso anche nei momenti difficili

† Al mio caro zio Pino

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Nell'ultimo decennio sono stati fatti notevoli progressi nella capacità di raccogliere grandi quantità di dati e memorizzarli su dispositivi aventi elevate capacità di archiviazione. In particolare, una grande porzione di tali dati deriva dal traffico scambiato in rete. Secondo le previsioni dell'azienda di networking Cisco [1], il traffico di rete sta crescendo e continuerà ad aumentare, come mostrato in Figura 1.1. Il numero di dati raccolti in modo continuo è talmente elevato che non saremmo in grado di analizzarli senza l'aiuto di algoritmi di machine learning e di data mining.

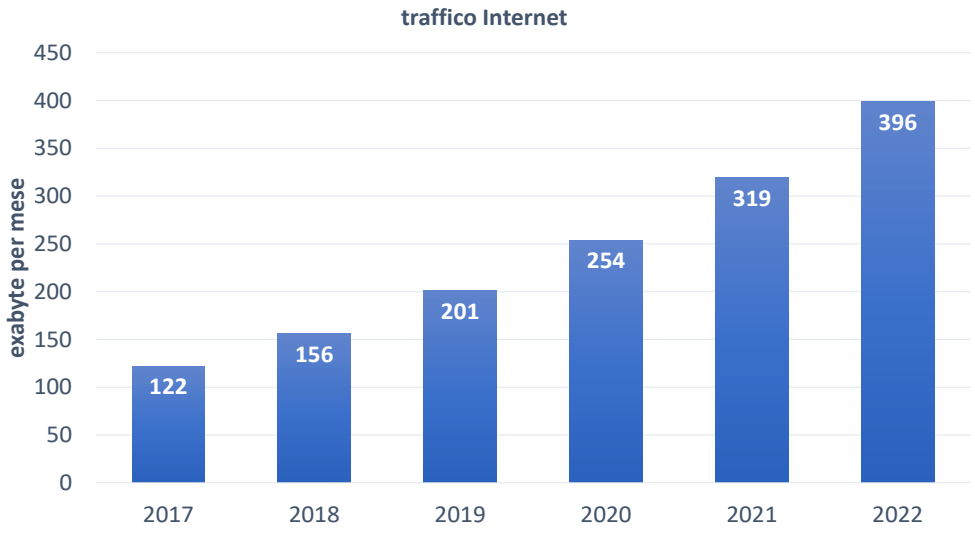

Figura 1.1: Previsione sull'aumento del traffico Internet nel quinquennio 2017-2022 (fonte: cisco.com).

Un altro aspetto da considerare è come avviene l'analisi di questi dati. La *DPI (Deep Packet Inspection)*, che nell'ambito di questa tesi assume il significato di analisi del traffico di rete mediante l'ispezione del payload dei segmenti TCP, è una pratica la cui legalità dipende dal grado di trasparenza e di libertà di espressione con la quale viene effettuata [2]. L'articolo 8 della Convenzione Europea per i Diritti dell'Uomo (*European Convention on Human Rights, ECHR*[1]) tutela il diritto alla privacy, ovvero il diritto a mantenere privato qualcosa che afferisce alla propria sfera personale[2]. Pertanto, qualora sia necessario attuare la DPI del traffico di rete per ragioni non di vitale importanza, è obbligatorio chiedere il consenso esplicito degli utenti della rete.

[1] https://www.coe.int/en/web/human-rights-convention

[2] https://dictionary.cambridge.org/dictionary/english/privacy

Sorge quindi spontaneo l'interrogativo su come sia possibile realizzare delle attività di sicurezza informatica (ad esempio la protezione di una LAN da parte di un amministratore di rete), richiedendo il consenso di utenti, una parte dei quali, pur esigua, potrebbe non concederlo sia per ragioni legittime (il diritto alla privacy) che per intenti criminali. Inoltre, anche se non ci fossero questioni etiche e legali nell'utilizzare la DPI, ci sarebbe il problema di come svolgere delle attività di sicurezza informatica nel caso in cui il traffico sia cifrato, ad esempio mediante TLS; in tal caso la DPI risulta pressoché impossibile da attuare senza impattare pesantemente sulle performance della rete. La cifratura delle connessioni di rete è ormai una realtà: a fine del 2018, si è stimato che più del 72% del traffico di rete sia cifrato [3]. Per queste ragioni, al giorno d'oggi, utilizzare la DPI non è più una scelta valida.

Un'alternativa alla DPI, è estrarre statistiche di traffico TCP dall'intestazione dei segmenti. In questo modo, oltretutto, non c'è alcuna distinzione tra analizzare traffico in chiaro e traffico cifrato, dal momento che entrambe le tipologie di traffico utilizzano gli stessi protocolli di livello di trasporto.

L'approccio appena descritto, è quello utilizzato nella parte implementativa di questo lavoro di tesi. In particolare, lo scopo di questa dissertazione è cercare, avvalendosi di alcune tecniche supervisionate di *Machine Learning (ML)*, di creare modelli che, in base ad alcune statistiche TCP estratte dal traffico di rete, permettano di individuare il traffico malevolo cifrato ed in chiaro.

La prima minaccia per la sicurezza informatica affrontata in questo lavoro di tesi, è stata il *cryptojacking*, ovvero una forma malevola di mining di criptovaluta (o *cryptomining*) che viene eseguita senza il consenso dell'utente. In particolare sono state considerate due varianti di tale minaccia:

- *web-based cryptojacking:* il cryptojacking viene effettuato mentre l'utente visita un sito web;
- *cryptojacking tramite malware:* il cryptojacking avviene ad opera di un malware che ha infettato il dispositivo dell'utente.

Inoltre, nonostante il cryptomining sia una pratica legale, può essere anche utilizzato in maniera illegale. Si pensi ad esempio al dipendente di una azienda che ruba l'energia elettrica, eseguendo un software di mining sul suo computer aziendale. Per questa ragione è utile anche identificare il traffico generato da software di cryptomining.

La tematica del cryptojacking è diventata sempre più sentita nell'ambito della sicurezza informatica, a causa della diffusione del fenomeno delle criptovalute e la crescita dei relativi interessi economici che hanno spinto i criminali a sperimentare nuove forme di guadagni illeciti. Basti pensare che, a febbraio 2019, la capitalizzazione di mercato di tutte le criptovalute (il cui numero stimato è 2071) supera i 135[3] miliardi di dollari. Ad impressionare è anche il ritmo di nascita di nuove valute digitali: a marzo 2018, infatti, il numero di criptovalute circolanti era 1658 [4]. Pertanto, in meno di un anno, sono nate più di 400 nuove valute digitali.

È stato analizzato il traffico generato dal software XMR-Stak per individuare il traffico di cryptomining, il traffico generato dagli script di Coinhive per individuare il traffico di web-cryptojacking ed il traffico generato dal worm MadoMiner, al fine di individuare il traffico di cryptojacking effettuato da malware.

Il classificatore migliore è riuscito ad ottenere i seguenti risultati:

- AUC[4] pari a 99.33% nell'individuare traffico generato dal software di cryptomining e potenziale cryptojacking;
- AUC pari a 92.27% nell'individuare traffico di potenziale web-cryptojacking;
- AUC pari a 99.27% nell'individuare traffico di cryptojacking generato dal malware.

[3] https://coinmarketcap.com/

[4] L'AUC è una statistica che indica la percentuale di osservazioni classificate correttamente da un modello anche in caso di data set sbilanciati.

La seconda minaccia affrontata è stata quella causata dalle botnet, ovvero un insieme di dispositivi infettati (anche chiamati *bot*) che agiscono per conto di un attaccante che perpetra azioni malevoli, come ad esempio attacchi DDoS e furto di informazioni sensibili (es. dati bancari). Le botnet rappresentano un problema molto rilevante nel panorama della sicurezza informatica. Basti pensare che, secondo la relazione della società di sicurezza informatica Verizon [5], nel 2018 sono state scoperte 43000 brecce[5] dovute a malware che avevano lo scopo di prendere il controllo di dispositivi rendendoli dei bot.

I classificatori di traffico di botnet sono stati ottenuti analizzando il traffico generato da:

- Pybot un programma open source che permette di costruire una botnet sulla macchina su cui viene eseguito;
- UFONet uno strumento che, sfruttando una vulnerabilità software, è in grado di lanciare attacchi DDoS tramite una botnet.

Il classificatore migliore è riuscito ad ottenere, analizzando il canale di C&C (Command-and-Control) tra master e bot, i seguenti risultati:

- AUC pari a 99.22% per la botnet Pybot;
- AUC pari a 99.58% per la botnet UFONet.

Il lavoro di tesi si articola nei seguenti capitoli:

- nel Capitolo 2 saranno approfonditi e confrontati alcuni strumenti per l'estrazione di statistiche TCP;
- nel Capitolo 3 si descriverà cosa s'intende per cryptomining ed in cosa consistono gli attacchi di web-cryptojacking e di cryptojacking attuato da malware;
- nel Capitolo 4 saranno descritte le botnet Pybot ed UFONet ed i relativi attacchi;
- nel Capitolo 5 saranno descritte e confrontate alcune tecniche supervisionate di ML;
- nel Capitolo 6 saranno addestrati, confrontati e testati diversi classificatori ottenuti tramite tecniche supervisionate;
- nel Capitolo 7 saranno descritti alcuni lavori di ricerca riguardanti la classificazione di traffico di cryptojacking e di traffico di C&C generato da botnet;
- nel Capitolo 8 saranno tratte le conclusioni da questo lavoro di tesi;
- infine in Appendice saranno forniti alcuni dettagli sul processo di mining, sul protocollo utilizzato per minare criptovaluta, sul codice scritto nell'implementazione della tesi e su come utilizzarlo.

[5] Una breccia [5] consiste nel rendere note informazioni personali ad entità non autorizzate a seguito di un incidente informatico.

Capitolo 2

Statistiche di traffico di rete

In questo capitolo saranno analizzati e confrontati alcuni strumenti open source in grado di estrarre delle statistiche sui flussi TCP, anche protetti mediante TLS, presenti nelle catture di traffico di rete memorizzate in file *pcap (Packet CAPture)*.

Il capitolo si articola nelle seguenti sezioni:

- in primo luogo saranno approfonditi il protocollo TCP (Sezione 2.1) e il protocollo TLS (Sezione 2.2);
- in seguito, poiché la maggior parte di questi strumenti utilizza la libreria libcap, essa sarà brevemente descritta nella Sezione 2.3;
- infine, nella Sezione 2.4, saranno descritti e confrontati gli strumenti per l'estrazione delle statistiche.

2.1 Il protocollo TCP

Il *TCP (Trasmission Control Protocol)* [6, 7] è un protocollo di livello di trasporto della pila ISO/OSI [8] (mostrata in Figura 2.1), le cui specifiche originarie sono riportate nel RFC 0793 [9].

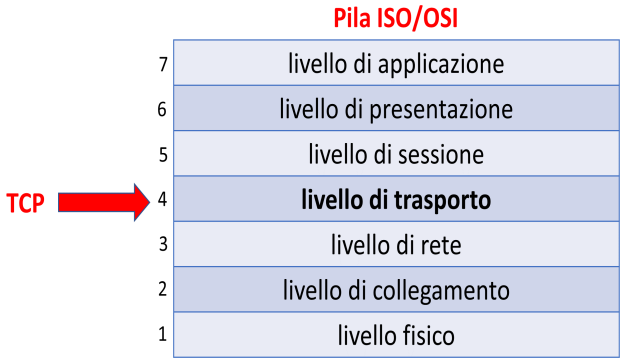

Figura 2.1: Pila ISO/OSI.

L'intestazione di un segmento TCP è mostrata in Figura 2.2 ed i suoi campi saranno descritti nelle prossime sezioni.

Alcune tra le caratteristiche del protocollo sono:

- full-duplex: ogni connessione è costituita da due flussi di dati unidirezionali che fluiscono contemporaneamente in direzioni opposte;

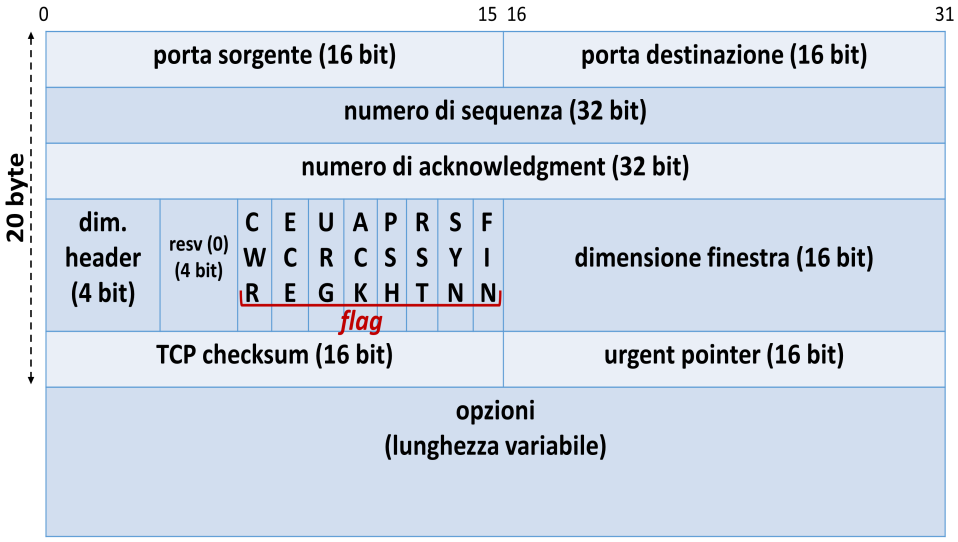

Figura 2.2: Header TCP.

- orientato al byte: TCP non interpreta i dati contenuti nel payload dei pacchetti, ma essi sono visti come un flusso di byte da trasportare;
- orientato alla connessione (Sezione 2.1.1);
- affidabile (Sezione 2.1.2), ovvero con elevata probabilità che i dati trasmessi siano ricevuti inalterati dal destinatario.

2.1.1 Protocollo orientato alla connessione

Il TCP è un protocollo orientato alla connessione: due nodi di rete, per poter utilizzare TCP, devono prima stabilire una connessione end-to-end. Pertanto il TCP non supporta le comunicazioni broadcast o multicast [7].

Una connessione TCP è univocamente determinata da quattro informazioni: indirizzo IP sorgente e porta sorgente, indirizzo IP destinazione e porta destinazione. Inoltre, poiché il TCP è full-duplex, è possibile distinguere all'interno della connessione due flussi unidirezionali: il primo flusso dal client (host che ha iniziato la connessione) verso il server, il secondo flusso in direzione opposta.

Un *socket* è una API (Application Programming Interface) standard tramite cui le applicazioni comunicano con il livello trasporto. La connessione tra due socket TCP viene stabilita tramite un protocollo denominato *three-way handshake*:

1. il client (l'host che inizia la connessione), invia un segmento con il flag SYN impostato, la porta destinazione ed un numero di sequenza iniziale pseudo-casuale;
2. il server risponde con un segmento con flag SYN e ACK impostati;
3. il client invia un ACK come riscontro per il segmento SYN inviato dal server.

La modalità standard per chiudere una connessione consiste nell'invio di quattro segmenti:

1. una delle applicazioni coinvolte nella connessione, invia un segmento di FIN;
2. l'altra applicazione invia in risposta un segmento di ACK, seguito da un segmento di FIN;
3. infine viene inviato un segmento di ACK come riscontro del segmento di FIN ricevuto.

É possibile anche terminare bruscamente una connessione, ad esempio a causa di un errore, tramite l'invio di un segmento con flag RST impostato.

2.1.2 Affidabilità del TCP

Il TCP per garantire l'affidabilità, ovvero l'elevata probabilità che il destinatario riceva correttamente i dati inviati, utilizza diversi meccanismi.

In primo luogo, esso utilizza i *numeri di sequenza*. Un numero di sequenza rappresenta l'offset del primo byte trasportato nel pacchetto corrente rispetto al byte iniziale del flusso. Il ricevente, osservando dei numeri di sequenza non consecutivi, può desumere la perdita dei pacchetti e quindi richiederne la ritrasmissione oppure, osservando dei numeri di sequenza duplicati, può dedurre l'arrivo di segmenti duplicati e dunque scartarli.

Inoltre il TCP, per informare il mittente della corretta trasmissione, utilizza l'ACK cumulativo: esso è un segmento con flag ACK impostato e numero di acknowledgment N, che indica la corretta ricezione di tutti gli $N-1$ byte inviati in precedenza. Nel caso di segmenti di ACK senza payload, si parla di *pure ACK*.

2.1.3 La finestra TCP

La *finestra TCP* (o *TCP window*) è definita nel RFC 0793 [9] come:

> il numero di byte che il mittente può inviare prima di ricevere un riscontro.

La dimensione di questa finestra viene modificata dinamicamente tramite il campo *dimensione finestra* nell'header TCP. La finestra TCP calcolata in base a tale valore viene anche chiamata *rwnd (receive window)*.

Tramite l'utilizzo del campo *dimensione finestra*, si realizza un meccanismo di controllo di flusso, spiegato nel seguito. Dal momento che una comunicazione basata su TCP è costituita da un flusso continuo di byte, è necessario che gli host coinvolti nella comunicazione usino dei buffer per disaccoppiare la velocità di trasmissione dei dati, dalla velocità con cui i dati vengono elaborati dalla applicazione che sta utilizzando il TCP. Lo spazio libero all'interno del buffer può ridursi, qualora l'applicazione non sia in grado di elaborare i dati alla stessa velocità con la quale sono stati ricevuti. In questo caso il ricevente, diminuendo il valore del campo dimensione finestra (che nel caso limite può assumere valore zero), comunica al mittente di rallentare l'invio di nuovi dati.

2.1.4 Timeout di ritrasmissione (RTO)

Il *RTO (Retrasmission TimeOut)* è il tempo che intercorre prima che avvenga la ritrasmissione di un segmento di cui non è stato ricevuto l'ACK. Esso viene calcolato sulla base delle misure del *RTT (Round Trip Time)* che rappresenta il tempo che intercorre tra l'istante in cui si invia un segmento e quando viene ricevuto il riscontro corrispondente.

Negli anni sono state sviluppate diverse tecniche per il calcolo del RTO: di seguito viene approfondita quella descritta all'interno del primo RFC del TCP [9].

Il RTO viene calcolato nel seguente modo:

$$RTO = SRTT + 4 \cdot (rttvar)$$

in cui:

- *SRTT (Smoothed RTT estimator):* è la media mobile pesata ed esponenziale delle misure di RTT dei segmenti del flusso ($a = 0.125$ [10]):

  $$SRTT = a \cdot SRTT_{precedente} + (1 - a) \cdot RTT_{campione}$$

  in cui $RTT_{campione}$ è la misura del RTT del nuovo segmento, utilizzata per aggiornare SRTT;

- *rttvar:* è il termine che tiene conto della varianza dei campioni di RTT misurati ($b = 0.25$ [6]):

$$rttvar = (1 - b) \cdot rttvar_{precedente} \; + \; b \cdot |RTT_{campione} - SRTT|$$

Nel calcolo del RTO è necessario considerare l'algoritmo di Karn [11], che affronta il problema del calcolo del RTT in caso di ritrasmissioni. In tali casi, non è possibile sapere se il riscontro ricevuto, in base al quale avviene il calcolo del RTT, si riferisca al segmento ritrasmesso oppure al segmento originale (ad esempio, il segmento originale giunge a destinazione subito dopo che è avvenuta la sua ritrasmissione, a causa della congestione della rete). In quest'ultimo caso, la misura del RTT risulta falsata. Per tale motivo, nel calcolo di SRTT, non bisogna considerare i campioni di RTT ottenuti a seguito di ritrasmissioni. D'altro canto, ignorando le ritrasmissioni nel calcolo di SRTT, il RTO non potrebbe essere modificato opportunamente, per tenere conto, ad esempio, di una situazione di congestione della rete. Per tale motivo, in caso di ritrasmissioni in successione, viene usato il meccanismo del *backoff factor*: il valore di RTO viene raddoppiato (e non calcolato a partire dal valore di SRTT e di rttvar), fino a quando non si riceve un ACK per un segmento non ritrasmesso.

2.1.5 Opzioni dell'header TCP

Il campo *opzioni* dell'header TCP ha dimensione variabile tra 0 e 40 byte e permette di specificare diverse opzioni. Esse sono espresse nel seguente modo [6]: il primo byte specifica il tipo di opzione, il byte successivo, presente solo se l'opzione ha lunghezza maggiore del singolo byte, indica la lunghezza espressa in byte. Alcune tra le opzioni disponibili nell'intestazione TCP sono:

- *MSS (Maximum Segment Size) [9]*: tramite questa opzione, gli host coinvolti nella connessione TCP possono notificare alla rispettiva controparte, la dimensione massima dei segmenti che sono in grado di gestire;
- *WSCALE (Window Scale option) [12]*: tale opzione permette di esprimere il valore del campo *Dimensione Finestra* su 30 bit anziché 16 grazie all'utilizzo di un fattore di scalamento (*scale factor (sc)*). La dimensione della finestra viene calcolata nel modo seguente:

$$\text{dim\_finestra\_scalata} = \text{dim\_finestra} \cdot 2^{sc}$$

- *TSOPT (TimeStamps OPTion) [12]*: tramite questa opzione, gli attori coinvolti in una connessione TCP possono scambiarsi i timestamp di invio e di ricezione dei segmenti. Comparando i due timestamp, il calcolo del singolo campione di RTT è più accurato e di conseguenza lo sarà anche quello del SRTT;
- *SACK (Selective ACK) e SACK-Permitted [13]*: i partecipanti alla connessione possono notificare tramite l'opzione *SACK-Permitted* che sono in grado anche di gestire ACK selettivi (SACK, Selective ACK) ovvero ACK che contengono i numeri di sequenza dei segmenti ricevuti correttamente. Il mittente può quindi dedurre, osservando il numero di sequenza dei segmenti inviati e di quelli contenuti nel SACK, quali byte ritrasmettere.

2.1.6 Controllo di congestione

Il controllo di congestione, descritto originariamente all'interno del RFC 5681 [14], ha lo scopo di evitare di saturare la rete con il traffico TCP generato dagli host.

Il TCP considera un evento di perdita di un pacchetto come sintomo di congestione della rete. Per evitare tale condizione è necessario modificare la TCP window, non solo in base ai valori del campo *dimensione finestra* (*rwnd*), ma anche sulla base della finestra di congestione *cwnd*. Pertanto deve valere la seguente relazione:

$$window = min(rwnd, cwnd)$$

L'algoritmo classico [14] per il calcolo di cwnd prevede che, all'apertura della connessione, la cwnd valga 1 MSS (Sezione 2.1.5) e la finestra di congestione cresca esponenzialmente (fase di *slow start*) fino al verificarsi di un evento che indichi la congestione della rete. I possibili eventi sono:

- *scadenza del RTO*: in questo caso viene settato un valore soglia pari alla metà della cwnd corrente, la cwnd riparte da 1 MSS e la fase di slow start ricomincia. Quando viene raggiunto il valore soglia, il TCP entra nella fase di *congestion avoidance*, ovvero la fase in cui la cwnd cresce linearmente (e non esponenzialmente) per cercare di evitare l'insorgere di una nuova situazione di congestione della rete;
- *fast retrasmit*: esso consiste nella ricezione consecutiva di tre ACK duplicati. Alla ricezione del quarto ACK, avviene la ritrasmissione senza attendere la scadenza del RTO. Quando si verifica tale evento, il valore soglia viene settato a metà del valore corrente e la finestra di congestione viene incrementata di 1 MSS per ogni ACK che è stato ricevuto (fase di *fast recovery*). Alla ricezione dell'ACK per il segmento ritrasmesso, il TCP entra nuovamente nella fase di *congestion avoidance*.

2.2 Protocollo TLS

In questa tesi non saranno estratte statistiche riguardanti il protocollo TLS, perché ciò equivarrebbe ad eseguire una DPI (descritta nel Capitolo 1). Ciononostante, TLS è diffuso ed il suo uso influenza il pattern dei protocolli a più alto livello (ad esempio HTTPS [15] è la versione sicura di HTTP, che utilizza TLS). Per tali ragioni si è scelto di approfondirlo in questa sezione.

TLS (Transport Layer Security) [6, 16, 17] è un protocollo che si colloca tra il livello di trasporto e quello applicativo. In particolare esso giace al di sopra di un protocollo di trasporto affidabile come il TCP (Sezione 2.1) e garantisce, per il traffico di rete scambiato durante una o più connessioni TCP, le seguenti proprietà di sicurezza:

- riservatezza: consiste nel garantire il controllo di quali informazioni sensibili possano essere raccolte e di quale entità sia autorizzata a farlo;
- autenticazione: consiste nella verifica della veridicità del valore di un attributo dichiarato da una entità di sistema;
- integrità dei dati: è la garanzia che i dati non siano stati modificati o distrutti in modo non autorizzato.

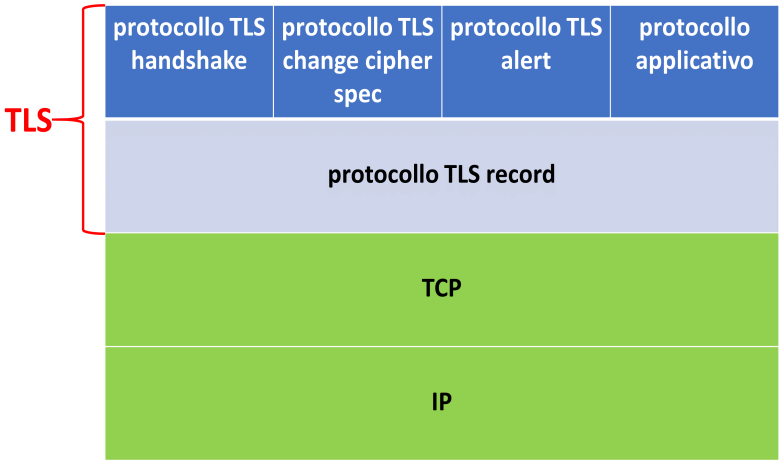

Figura 2.3: Pila protocollare di TLS.

La pila protocollare di TLS, mostrata in Figura 2.3, è costituita da un livello intermedio chiamato protocollo *TLS record* (Sezione 2.2.2) e dai seguenti protocolli di alto livello:

- *TLS change cipher spec*: protocollo specifico di TLS descritto nella Sezione 2.2.3;
- *TLS alert*: protocollo specifico di TLS descritto nella Sezione 2.2.4;
- *TLS handshake*: protocollo specifico di TLS descritto nella Sezione 2.2.5;
- *protocolli applicativi*: uno dei diversi protocolli applicativi, come ad esempio HTTPS.

2.2.1 Connessioni e sessioni TLS

Una connessione TLS è un flusso di dati temporaneo tra il client ed il server. Ogni connessione è associata ad un'unica sessione TLS.

Una sessione TLS è un collegamento logico tra il client ed il server che può essere costituito da una o più connessioni. Essa viene creata tramite uno scambio di messaggi del protocollo TLS handshake. La Figura 2.4 evidenzia le relazioni che sussistono tra connessioni e sessioni TLS.

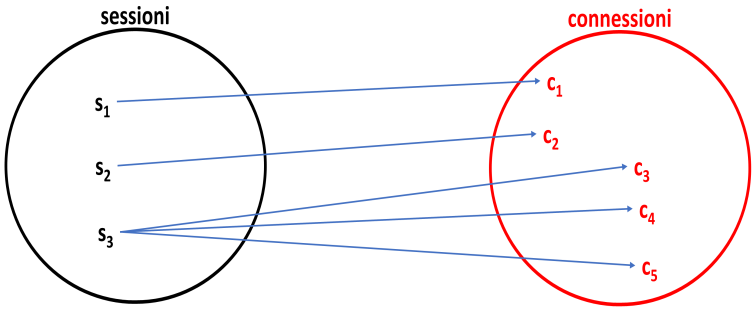

Figura 2.4: Relazioni tra sessioni e connessioni TLS.

Lo stato di una connessione è definito da una serie di parametri tra cui [17]:

- *ID sessione*: è l'identificativo di sessione, scelto dal server. Tramite questo parametro, è possibile controllare se in precedenza sia stata già aperta una sessione TLS ed evitare di rinegoziare i parametri di sessione, riducendo il numero di messaggi di handshake;
- *algoritmo di compressione*: algoritmo utilizzato nel protocollo TLS record (Sezione 2.2.2);
- *suite di cifratura*: l'insieme di algoritmi di cifratura simmetrica (ad esempio AES, Advanced Encryption Standard [18]), algoritmo di calcolo del keyed-digest[1] (ad esempio HMAC-SHA-256 [19]) e algoritmo di scambio chiavi (ad esempio Diffie-Hellman, DH [20]) concordati tramite il protocollo di TLS handshake;
- *pre-master secret e master secret*: il pre-master secret è un segreto che, quando viene creata una nuova sessione, viene condiviso tramite crittografia asimmetrica tra il client e il server. A partire dal pre-master secret, viene generato il master secret utilizzato, insieme al *client random* e al *server random*, per derivare il vettore di inizializzazione (IV)[2] e le chiavi per il keyed-digest e la cifratura simmetrica;
- *certificato/i*: certificato X.509 [21] del server e opzionalmente del client.

Anche una connessione ha un proprio stato, definito dai seguenti parametri:

[1] Il keyed-digest è una sequenza di caratteri, ottenuti dall'applicazione su alcuni dati (es. un file) di un algoritmo di hash crittografico che richiede come parametro una chiave crittografica.

[2] Il vettore di inizializzazione (IV, Initialization Vector) è un parametro, casuale ed impredicibile, in ingresso ad alcuni algoritmi di cifratura simmetrica a blocchi (es. CBC).

- *server random e client random*: sono dei numeri casuali scelti rispettivamente dal server e dal client, utilizzati insieme al master secret, per generare ad ogni connessione, le chiavi di cifratura simmetrica (chiave cifratura client-server e chiave cifratura server-client), le chiavi per il MAC[3] (chiave MAC client-server e chiave MAC server-client) e i vettori di inizializzazione IV (IV client-server e IV server-client);

- *numeri di sequenza*: sono dei contatori, aggiornati separatamente lato client e lato server, per il numero di messaggi scambiati durante la connessione. Essi sono utili per la protezione contro attacchi *replay*, ovvero attacchi di duplicazione o cancellazione dei messaggi scambiati.

2.2.2 Protocollo TLS record

Il protocollo TLS record viene utilizzato per garantire la riservatezza e l'integrità dei dati scambiati, ed è costituito dalle seguenti fasi:

1. fase iniziale in cui i dati applicativi sono divisi in frammenti;

2. fase opzionale di compressione senza perdita (*lossless*) di ciascun frammento;

3. fase di calcolo del keyed-digest di ciascun frammento;

4. fase di cifratura: dopo un'eventuale operazione di padding[4], si applica un algoritmo di cifratura simmetrica per garantire la riservatezza dei dati trasmessi.

Infine, prima che il record TLS possa essere inviato in rete, viene premessa un'intestazione che comprende i seguenti campi:

- *Content Type (1 byte)*: indica il protocollo di livello superiore che viene trasportato, ovvero uno dei quattro protocolli di livello superiore mostrati in Figura 2.3;

- *Major version (1 byte)*: ad esempio, nel caso di TLSv1.2, la major version è 1;

- *Minor version (1 byte)*: ad esempio, nel caso di TLSv1.2, la minor version è 2;

- *Dimensione dei dati (2 byte)*: indica la lunghezza in byte dei dati, eventualmente compressi, escludendo la lunghezza del keyed-digest.

2.2.3 Protocollo TLS change cipher spec

I messaggi di questo protocollo, mostrati in Figura 2.5 sotto il nome *modifica cipher spec*, vengono utilizzati per aggiornare la suite di cifratura utilizzata durante una sessione.

2.2.4 Protocollo TLS alert

Il protocollo TLS alert viene utilizzato quando si verifica una condizione anomala, come ad esempio un keyed-digest non corretto, il fallimento della fase di handshake, il fallimento della procedura di decompressione di un record e diversi altri. All'interno dei messaggi di TLS alert è anche specificato il livello di gravità dell'errore avvenuto: qualora il livello sia *fatal*, ovvero grave, la connessione corrente viene immediatamente terminata.

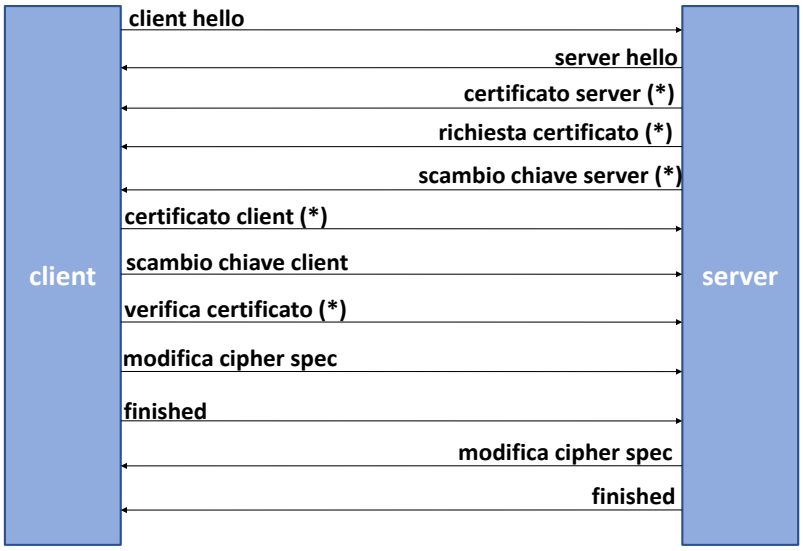

Figura 2.5: Messaggi scambiati durante il protocollo di handshake (il simbolo (∗) indica che il messaggio è opzionale).

2.2.5 Protocollo TLS handshake

Il protocollo TLS handshake viene utilizzato per aprire una sessione TLS. Di seguito verranno descritti i tipici messaggi scambiati nel protocollo, mostrati in Figura 2.5.

I messaggi scambiati nel protocollo di handshake sono:

client hello: questo messaggio è costituito dai seguenti campi:

- *versione*: la versione di TLS più aggiornata supportata dal client;
- *random client*: è costituito da 28 byte generati in modo casuale e dal timestamp della loro creazione. A partire da questo campo vengono generate le chiavi per diverse tipologie di algoritmi come descritto nella Sezione 2.2.1;
- *ID sessione*: un valore non nullo indica che il client vorrebbe riprendere la sessione TLS già aperta, identificata da tale valore;
- *suite di cifratura*: la lista delle suite di cifratura supportate dal client, in ordine di preferenza;
- *metodo di compressione*: la lista degli algoritmi di compressione supportati dal client.

server hello: tale messaggio contiene i seguenti campi:

- *versione*: la versione di TLS più aggiornata supportata dal server e quella più vecchia supportata dal client (indicata nel messaggio di Hello precedente);
- *random server*: ha la stessa funzione del campo random client;
- *ID sessione*: contiene un nuovo valore scelto dal server oppure, qualora il campo ID sessione inviato dal client sia non nullo, una replica di tale valore;
- *suite di cifratura*: la suite di cifratura scelta dal server tra quelle proposte dal client;
- *metodo di compressione*: l'algoritmo di compressione scelto dal server tra quelli proposti dal client.

[3]Il MAC (Message Authentication Code) è un keyed-digest utilizzato per l'autenticazione dei dati.

[4]Il padding è una tecnica che permette di rendere la dimensione dei dati, su cui viene applicato una tipologia di algoritmo di cifratura simmetrica a blocchi (es. CBC), un multiplo della dimensione del singolo blocco.

certificato server: il server si autentica al client inviando il proprio certificato X.509 [21].

Se il certificato fornito contiene l'estensione *key usage* valida solo per la firma, allora il server userà dei meccanismi effimeri di scambio chiavi come ad esempio *DH effimero* [6]. Il termine effimero indica che una nuova chiave segreta viene generata al volo ad ogni nuova connessione. In tali casi, il certificato sarà usato dal client per verificare la firma del server applicata ai parametri scambiati tramite i meccanismi effimeri;

richiesta certificato: tramite questo messaggio opzionale, il server può richiedere al client di autenticarsi. In questo messaggio vengono specificati i seguenti parametri:

- *tipo certificato*: ad esempio RSA [22] solo a scopo di firma, RSA per DH effimero e diversi altri;
- *CA (Certification Authority) fidate*: contiene i DN[5] delle CA considerate fidate dal server.

scambio chiave server: questo messaggio viene utilizzato qualora il server scelga di usare meccanismi effimeri. Ad esempio, se il server utilizza DH effimero, questo messaggio contiene i parametri pubblici e la loro firma;

certificato client: il client invia il proprio certificato X.509, che deve soddisfare le richieste del messaggio *richiesta certificato* del server;

scambio chiave client: tramite questo messaggio, il client invia le informazioni necessarie affinché il server possa generare il pre-master secret. Ad esempio, se il server ha utilizzato DH effimero, tramite questo messaggio il client invia i propri parametri pubblici di DH;

verifica certificato: se il client ha inviato un certificato che ha capacità di firma, questo messaggio contiene una verifica della sua validità. In particolare, esso contiene la firma applicata all'hash ottenuto dal master secret e dai messaggi di handshake precedenti;

modifica cipher spec (server/client): questo messaggio appartiene al protocollo TLS change cipher spec (Sezione 2.2.3) e viene utilizzato per rendere effettivo l'uso della nuova suite di cifratura concordata;

finished (server/client): questo messaggio è il primo ad utilizzare la suite di cifratura concordata nei messaggi precedenti. Esso contiene un keyed-digest, calcolato su tutti i messaggi di handshake inviati in precedenza e sul master secret, che ha lo scopo di garantire l'autenticazione e l'integrità dei dati. In questo modo si ottiene la protezione da alcuni tipi di attacchi tra cui, ad esempio, quello di *cyphersuite rollback* [23].

2.3 La libreria libpcap

La libreria libpcap è utilizzata dalla maggior parte degli strumenti di analisi di traffico approfonditi in questo capitolo, motivo per cui sarà descritta in questa sezione.

Libpcap[6] [24] è una libreria che fornisce un'interfaccia per interagire con i moduli kernel deputati alla cattura e al filtraggio del traffico di rete. Grazie a questa libreria è anche possibile gestire file con estensione *pcap (Packet CAPture)*.

L'architettura software originale per il *packet filtering*, utilizzata in *BSD Berkeley Software Distribution* e successivamente inglobata con alcune modifiche e migliorie nel kernel di diverse distribuzioni Linux, è mostrata in Figura 2.6.

[5] Il DN (Distinguish Name) è un identificativo univoco di una entità all'interno dei certificati X.509.

[6] http://www.tcpdump.org/

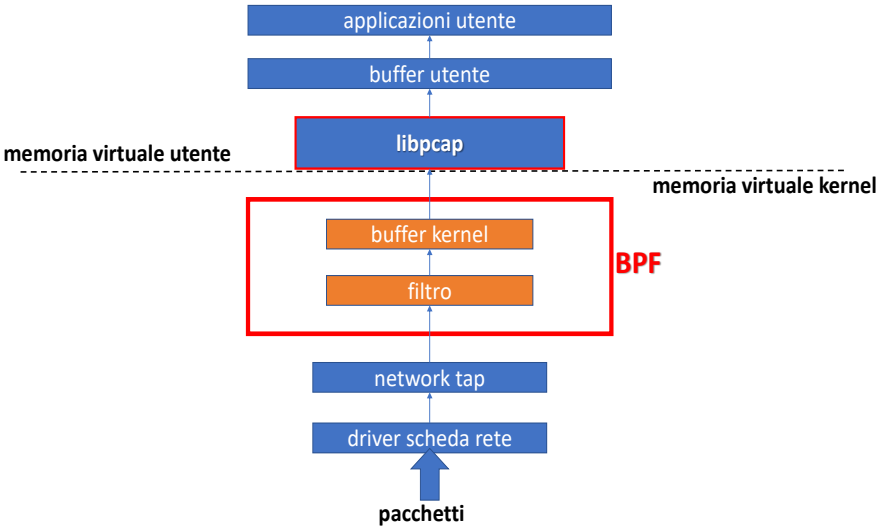

Figura 2.6: Architettura BSD per il filtraggio dei pacchetti.

Il cuore dell'architettura software per il filtraggio dei pacchetti è il *BPF (Berkeley Packet Filter)*[7]. Esso è un modulo kernel costituito da una CPU virtuale, che ha il compito di applicare un filtro, introdotto dall'utente, sul traffico di rete.

Il BPF può interagire passivamente con il flusso di pacchetti provenienti dalla scheda di rete grazie al *network tap*, un modulo che esegue una copia dei pacchetti prima di farli proseguire nello stack protocollare di rete.

I pacchetti che soddisfano il filtro vengono memorizzati all'interno di un buffer nel kernel.

La libreria libcap fornisce le funzioni per fare operazioni di I/O da e verso il buffer del kernel e trasferire i pacchetti letti/scritti all'interno del buffer utente. Tale buffer può essere acceduto da un'applicazione utente di analisi di traffico di rete.

2.4 Gli strumenti di estrazione di statistiche di rete

In questa sezione saranno descritti e comparati alcuni strumenti per estrarre statistiche di traffico di rete.

2.4.1 Tshark

Tshark[8] è uno strumento open source multi-piattaforma di analisi e cattura di traffico di rete. Rappresenta la versione a linea di comando di Wireshark[9].

Questo strumento si basa sulla libreria libpcap (o winpcap[10] se viene installato ed eseguito sotto Windows) ed è in grado sia di catturare traffico di rete in tempo reale che di analizzare catture di traffico salvate in formato pcap.

Alcune tra le funzioni offerte da Tshark sono:

[7] https://www.kernel.org/doc/Documentation/networking/filter.txt

[8] https://www.wireshark.org/docs/man-pages/tshark.html

[9] https://www.wireshark.org

[10] https://www.winpcap.org/

- tradurre il filtro digitato dall'utente, nel linguaggio assembler utilizzato dalla CPU virtuale BPF (Sezione 2.3);
- visualizzare i pacchetti catturati in modo human-friendly, evidenziando i protocolli di rete utilizzati.

Tshark è anche in grado di ricostruire i flussi TCP ed estrarre, per ognuno di essi, le seguenti statistiche:

- numero di pacchetti totali scambiati durante la connessione;
- numero totale di byte;
- numero di pacchetti scambiati nella direzione client-server;
- numero di byte scambiati nella direzione client-server;
- numero di pacchetti scambiati nella direzione server-client;
- numero di byte scambiati nella direzione server-client;
- istante di inizio della connessione corrente rispetto all'istante di inizio della cattura;
- durata della connessione;
- throughput del flusso client-server;
- throughput del flusso server-client.

2.4.2 Tcpflow - TCP flow recorder

Tcpflow[11] è uno strumento [25] open source creato nel 1999, che è in grado di ricostruire i flussi TCP sia di traffico on-line che di traffico salvato in file pcap.

Inoltre, dal momento che Tcpflow utilizza la libreria libpcap, è possibile opzionalmente scrivere filtri, anche particolarmente complessi, per filtrare il traffico da analizzare.

Il principale campo di utilizzo di Tcpflow è quello dell'analisi forense di traffico rete [25]. Per tale motivo è stato progettato per raccogliere il maggior numero possibile di informazioni, con la consapevolezza che spesso non è possibile effettuare nuovamente la cattura dello stesso traffico.

Qualora sia stato utilizzato il protocollo applicativo HTTP [26] o SMTP [27], Tcpflow è anche in grado di estrarre e salvare su disco i file scambiati durante la connessione.

Infine, lo strumento in analisi crea un file *DFXML (Digital Forensics XML)*[12] in cui sono riportate diverse informazioni tra cui anche le seguenti statistiche sui flussi TCP analizzati:

- timestamp dell'inizio del flusso;
- timestamp di fine del flusso;
- indirizzo IP sorgente;
- indirizzo IP destinazione;
- indirizzo MAC sorgente;
- indirizzo MAC destinazione;
- porta sorgente;

[11] https://github.com/simsong/tcpflow

[12] https://github.com/dfxml-working-group/dfxml_schema

- porta destinazione;
- numero di pacchetti;
- numero di byte.

Opzionalmente, Tcpflow può creare un file PDF (Portable Document Format) in cui vengono realizzati una serie di grafici a barre che mostrano i dati ricevuti nel tempo, gli indirizzi IP e le porte sorgente che hanno inviato più traffico e gli indirizzi IP e le porte destinazione che hanno ricevuto maggior traffico.

2.4.3 Joy

Joy[13] è un software scritto in C, sviluppato da Cisco[14] e rilasciato sotto licenza BSD. Esso utilizza la libreria libpcap ed è in grado di analizzare traffico in tempo reale o leggere file pcap di catture precedenti.

Dal traffico analizzato, Joy ricostruisce i flussi TCP ed estrae alcune statistiche.

Il risultato dell'analisi viene salvato in un oggetto *JSON (JavaScript Object Notation)* [28] (un esempio è mostrato nel Listato 1). Qualora siano stati utilizzati i protocolli HTTP, TLS o DNS [29], questo strumento è in grado di memorizzare i rispettivi metadati: header HTTP, certificati TLS e nomi di richieste DNS.

Joy considera terminato un flusso e quindi lo memorizza nel file JSON, se dura più di 30 secondi ($expire\_type = a$) oppure non viene osservato traffico per un periodo di 10 secondi ($expire\_type = i$).

Il Listato 1 mostra l'oggetto JSON creato per un flusso TCP, con le relative statistiche commentate.

```
{
        "sa": "130.192.1.122",              // indirizzo IP sorgente
        "da": "51.144.104.161",             // indirizzo IP destinazione
        "pr": 6,                            // IP protocol number (6 = TCP)
        "sp": 41676,                        // porta sorgente TCP
        "dp": 9999,                         // porta destinazione TCP
        "bytes_out": 233,                   // byte inviati da "sa" a "da"
        "num_pkts_out": 5,                  // pacchetti inviati da "sa" a "da"
        "time_start": 1541534563.512119,    // tempo di inizio del flusso in secondi
        "time_end": 1541534738.737051,      // tempo di fine del flusso in secondi

        "packets": [            // array di informazioni sui pacchetti
        {
                "b": 240,       // numero di byte nel payload TCP
                "dir": ">",     // direzione: sa -> da
                "ipt": 0        // inter-packet time: ms trascorsi da time_start
        },
        {
                "b": 320,       // byte nel payload TCP
                "dir": "<",     // direzione: da -> sa
                "ipt": 4076     // inter-packet time: ms trascorsi da ultimo pacchetto
        },
        ...
        ],
        "expire_type": "i"
}
```

Listato 1: Oggetto JSON che rappresenta un flusso estratto dallo strumento Joy.

[13] https://github.com/cisco/joy

[14] https://www.cisco.com

2.4.4 Captcp: TCP Analyzer for PCAP Files

Captcp[15] è uno strumento open source, scritto in Python[16], che è in grado di ricostruire da file pcap le connessioni TCP. Di contro, non consente di analizzare traffico in tempo reale.

Lo strumento in questione è costituito da diversi moduli, tra i quali vi è il modulo di estrazione di statistiche. Esso è in grado di mostrare a video oppure salvare in un file CSV [30], le seguenti statistiche per flusso:

- numero di byte trasmessi a livello di collegamento;
- numero di byte trasmessi a livello di rete;
- numero di byte trasmessi a livello di trasporto;
- numero di byte trasmessi a livello di applicazione;
- numero di byte ritrasmessi;
- numero di pacchetti ritrasmessi;
- percentuale di byte ritrasmessi rispetto a quelli inviati a livello di trasporto;
- numero di segmenti di ACK.

2.4.5 Scapy

Scapy[17] è una libreria scritta in Python che permette di creare, inviare, catturare e analizzare del traffico di rete.

È molto utilizzato nell'ambito della sicurezza informatica: permette di creare software per testare la sicurezza della rete, simulare attacchi o eseguire delle semplici scansioni.

Scapy considera ogni pacchetto come un oggetto di una classe e pertanto richiede di essere allocato nell'area di memoria heap; per evitare di saturare tale area di memoria, Scapy non è indicato per l'analisi di file pcap di grandi dimensioni. Inoltre, lo strumento in questione non permette l'estrazione automatica di statistiche: è necessario scrivere delle apposite funzioni in base alla statistica che si vuole calcolare. Nel Listato 2 è mostrato un esempio di utilizzo di Scapy per calcolare il numero di pacchetti che trasportano un segmento TCP.

```
import scapy.all as sc
packets = sc.rdpcap('cattura.pcapng')

num_packet_tcp = 0
for packet in packets:
        if sc.TCP in packet:
                num_packet_tcp += 1
print('Il numero di pacchetti contenenti un segmento TCP è: ', num_packet_tcp)
```

Listato 2: Esempio di uso di Scapy per calcolare il numero di pacchetti contenenti un segmento TCP.

[15] http://research.protocollabs.com/captcp/

[16] https://www.python.org/

[17] https://scapy.net/

2.4.6 Tstat

Tstat[18] [31] è uno strumento open source di analisi passiva di traffico di rete, sviluppato dal gruppo di ricerca TNG (Telecommunication Network Group) del Politecnico di Torino a partire dal 2000. È stato scritto, a partire dallo strumento *tcptrace*[19], in ANSI C. Anche grazie all'utilizzo di un linguaggio di basso livello come il C, Tstat è in grado di analizzare traffico di rete a velocità di diversi Gbps.

Tstat è in grado di estrarre statistiche sia da traffico in tempo reale, avvalendosi della libreria libpcap, sia di leggere file con diverse estensioni, tra cui pcap. Inoltre, esso è in grado di estrarre delle statistiche per pacchetto, per flusso TCP o UDP e per flussi aggregati. In questa tesi, verranno considerate le prime due tipologie di statistiche e saranno analizzati i flussi TCP. Dal momento che una connessione TCP è costituita da due flussi unidirezionali ed opposti, nella documentazione di Tstat, viene usata la seguente terminologia:

- *C2S (Client-to-Server):* è il flusso TCP che dal client (ovvero l'host che per primo ha iniziato il three-way handshake) fluisce verso il server;
- *S2C (Server-to-Client):* è il flusso TCP che dal server fluisce verso il client.

L'analisi del traffico viene condotta seguendo lo stack TCP/IP:

1. ogni pacchetto viene analizzato dal modulo IP che calcola alcune statistiche come ad esempio bit rate, lunghezza del pacchetto e diverse altre;
2. risalendo lo stack, vengono calcolate ed aggiornate diverse statistiche sui flussi TCP come il numero totale di byte e dei pacchetti, il RTT (Round-Trip-Time), il throughput e alcune altre;
3. È possibile arrivare al livello applicativo e cercare di classificare il traffico in base all'applicazione.

L'output dell'analisi è costituito da un insieme di file testuali formattati (file di log), in cui ogni riga è associata ad una connessione ed ogni colonna rappresenta una statistica. I file di log TCP, generati da Tstat, sono:

- `log_tcp_complete`: in questo file vengono raggruppate le connessioni TCP che sono terminate entro uno dei timer (approfonditi nel seguito), tramite uno scambio di segmenti FIN/ACK o tramite un segmento di RST. Di default Tstat scarta automaticamente tutte le connessioni in cui non è stato rilevato un handshake completo o parziale;
- `log_tcp_nocomplete`: in questo file vengono raggruppate sia le connessioni TCP che non appaiono correttamente terminate all'interno dei file pcap sia le connessioni TCP per cui è scaduto uno dei seguenti timer:
  1. `TCP_Singleton_time` (default 10 secondi): timer che viene impostato con il primo segmento di SYN trasmesso e scade se non vengono osservati dati nella connessione per un periodo maggiore o uguale a 10 secondi;
  2. `TCP_Idle_Time` (default cinque minuti): timer impostato ogni volta che viene trasmesso un nuovo segmento. Esso scade se il periodo che intercorre tra due segmenti successivi è maggiore o uguale a cinque minuti.

[18] http://tstat.polito.it/

[19] http://www.tcptrace.org

Tstat è in grado di estrarre complessivamente, senza considerare le statistiche riguardanti il livello applicativo, 101 statistiche[20]. Nell'ambito di questa tesi, ai fini dell'addestramento dei classificatori, saranno considerate solo 31 statistiche appartenenti al gruppo di quelle *core*, ovvero estratte per qualsiasi flusso TCP; ciò è necessario perché ogni campione deve avere lo stesso numero di attributi per far parte dell'insieme di dati utilizzati per addestrare i classificatori, descritti nel Capitolo 5. L'elenco completo delle 31 statistiche è disponibile nella Sezione 6.2. Alcune delle rimanenti statistiche, rilevanti ai fini statistici ma disponibili solo per flussi TCP terminati entro la scadenza dei timer, sono consultabili nell'Appendice A.

2.4.7 Confronto tra gli strumenti utilizzati

Tutti gli strumenti testati sono open source e disponibili per Linux. A parte Captcp, tutti gli strumenti analizzati sono in grado sia di filtrare traffico in tempo reale che di leggere catture di traffico pcap. Inoltre, a parte Scapy, tutti gli strumenti analizzati sono in grado di estrarre alcune statistiche in modo automatico, ovvero senza la necessità di programmarle.

Tshark ha il pregio, rispetto agli altri strumenti analizzati, di riconoscere un numero elevato di protocolli di rete. Di contro, il numero di statistiche che è in grado di estrarre in modo automatico è ridotto se paragonato a quelle di Tstat.

Tcpflow è utilizzato prevalentemente nell'ambito dell'analisi forense di traffico di rete: per questo motivo eccelle nell'accuratezza di raccolta delle informazioni (ad esempio i file scambiati durante le connessioni) piuttosto che nell'estrazione di statistiche.

Joy ha il pregio di salvare i dati derivanti dall'analisi in formato JSON, che risulta maggiormente human-readable rispetto ai linguaggi utilizzati da altri strumenti. Di contro, come gli strumenti citati in precedenza, non eccelle nel numero di statistiche estratte.

Nella Tabella 2.1, è mostrato il numero di statistiche estratte per un flusso TCP, in automatico, da ogni strumento. Inoltre sono state omesse dal computo l'indirizzo MAC sorgente/destinazione, l'indirizzo IP sorgente/destinazione e la porta sorgente/destinazione, poiché, nell'ambito di questa tesi, risultano poco rilevanti ai fini statistici.

strumento	*numero statistiche*
Tstat	31
TShark	10
Captcp	8
Tcpflow	4
Joy	4

Tabella 2.1: Confronto del numero di statistiche per strumento.

Dalla Tabella 2.1 risulta evidente che il numero di statistiche estratte da Tstat è di gran lunga superiore rispetto a quello degli altri strumenti.

In definitiva, dal momento che avere un numero elevato di misure risulta fondamentale per creare un insieme di dati di partenza abbastanza ampio da permettere un accurato addestramento dei classificatori, Tstat è lo strumento utilizzato nella parte implementativa di questa tesi.

[20] http://tstat.polito.it/measure.shtml

Capitolo 3

Cryptomining e cryptojacking

3.1 Introduzione

Il *cryptomining* [32] o mining di una criptovaluta, consiste nella validazione, all'interno di una rete peer-to-peer[1], di transazioni di moneta virtuale e nella loro successiva aggiunta all'interno di un nuovo blocco della blockchain. Il miner è colui che esegue questo procedimento, ovvero è un nodo partecipante alla rete peer-to-peer della moneta digitale.

Una *criptovaluta* è una moneta scambiata nel mondo digitale, che viene emessa e controllata tramite il consenso distribuito all'interno di una rete peer-to-peer. Al giorno d'oggi esistono diverse centinaia di criptovalute: a febbraio 2019, si è stimato che il loro numero fosse 2071[2]. La prima criptovaluta della storia è stata il bitcoin (BTC) [32], nata nel 2008 e documentata nell'articolo "Bitcoin: A peer-to-peer electronic cash system" [34], scritto da un gruppo di sviluppatori sotto lo pseudonimo di Satoshi Nakamoto.

La *blockchain*[3] [35] è utilizzata per creare una base dati digitale condivisa all'interno di un rete pubblica peer-to-peer. Il suo compito principale è quello di registrare le transazioni di monete virtuali. Per assolvere tale scopo, la blockchain possiede alcune caratteristiche:

- è pubblica, ovvero ogni nodo partecipante alla rete di scambio della criptovaluta ne possiede una copia;
- è memorizzata in chiaro in modo che ogni nodo possa verificare, indipendentemente, la validità delle transazioni avvenute.

Queste due proprietà risultano fondamentali in un sistema che non prevede l'esistenza di un'entità centrale di controllo, al fine di garantire che l'ammontare di valuta digitale spesa sia stata emessa in precedenza e che non si possa verificare il problema del *double spending*, ovvero di spendere due volte la stessa somma di criptovaluta. Per quanto riguarda la struttura dati, la blockchain è costituita da un insieme di blocchi inseriti in ordine cronologico e tra loro concatenati (da questo aspetto deriva il nome di "blockchain", ovvero catena di blocchi).

La frequenza con cui avviene il mining di un nuovo blocco, viene controllata mediante la modifica della difficoltà di un puzzle crittografico [35], anche chiamato *algoritmo proof-of-work*. Il primo miner che riesce a risolverlo, ottiene una ricompensa in criptovaluta. Negli ultimi anni, a causa dell'esplosione del fenomeno delle criptomonete, si è assistito ad un aumento considerevole

[1] Una rete peer-to-peer (P2P) [33] è un sistema distribuito, ovvero costituito da dispositivi isolati (ad esempio i nodi di una rete) che comunicano tra loro tramite un protocollo. Ogni nodo di rete condivide le risorse hardware e può agire contemporaneamente sia da client che da server.

[2] https://coinmarketcap.com/

[3] https://www.merriam-webster.com/dictionary/blockchain

dei nodi deputati al mining e di conseguenza ad un inasprimento della *hashing race*, ovvero della competizione per riuscire a minare un nuovo blocco e ricevere la ricompensa. Ciò ha causato la nascita di *mining pool*, ovvero associazioni di miner che uniscono le loro risorse computazionali per vincere l'hashing race, dividendosi poi il profitto in base alle risorse che ognuno ha impiegato. Nella Sezione 3.2 verrà introdotto Stratum, un protocollo utilizzato nelle comunicazioni tra i miner ed il mining pool. In questa tesi, Stratum è stato adoperato per il mining di Monero e quindi, dalla sua analisi, sono state estratte le statistiche utilizzate per l'addestramento dei classificatori.

Monero (XMR)[4] è una criptovaluta, nata nell'aprile del 2014, con i seguenti obiettivi:

- garantire l'anonimato nelle transazioni delle criptovalute;
- rendere eseguibile il processo di mining su hardware general-purpose. Quest'ultimo aspetto rende XMR particolarmente adatta[5] per il cryptojacking.

Il termine *cryptojacking* [36] è utilizzato per indicare il mining di una criptovaluta senza il consenso del possessore del dispositivo su cui viene eseguito tale processo. Il cryptojacking sarà approfondito nelle seguenti sezioni:

- nella Sezione 3.4 verrà discussa la problematica del web-cryptojacking;
- nella Sezione 3.5, verrà descritto MadoMiner, un malware per il mining di XMR.

Alcuni approfondimenti riguardanti il meccanismo di funzionamento della blockchain, il processo di mining e le caratteristiche peculiari di XMR, sono collocati nell'Appendice B.

3.2 Il protocollo Stratum

Stratum[6][7] [37, 38] è un protocollo che utilizza il formato JSON-RPC [28] e che si basa su TCP. Esso supporta anche TLS (Sezione 2.2). É utilizzato nelle comunicazioni tra i miner e il pool. Al momento della scrittura di questa tesi, non esiste uno standard per l'implementazione di questo protocollo: per tale motivo ne possono esistere molteplici varianti omonime.

Stratum si basa sul protocollo WebSocket [39], il quale permette di creare comunicazioni bidirezionali client-server più efficienti rispetto a quelle supportate dal protocollo HTTP. L'efficienza nelle comunicazioni bidirezionali è molto importante nel processo di mining, poiché il mining pool deve essere in grado di inviare notifiche ai miner (ad esempio notifiche sulla presenza di un nuovo job[8]).

Questo protocollo viene approfondito dal momento che le catture analizzate di traffico di web-cryptojacking (Sezione 3.4) utilizzano Stratum con TLS, mentre le catture derivanti dal malware MadoMiner (Sezione 3.5) utilizzano Stratum in chiaro.

3.2.1 Struttura dei messaggi

Esistono tre tipologie di messaggi:

- *messaggi di richiesta*: sono messaggi che richiedono una risposta da parte del server;

[4] https://www.getmonero.org/

[5] Per approfondire, si veda l'Appendice B.4.

[6] https://en.bitcoin.it/wiki/Stratum_mining_protocol

[7] https://github.com/ctubio/php-proxy-stratum/wiki/Stratum-Mining-Protocol

[8] Il termine *job*, in questo contesto, indica la porzione di calcolo computazionale per il mining di un nuovo blocco, che il miner deve effettuare per ottenere la ricompensa.

- *messaggi di risposta*: sono i messaggi di risposta a richieste precedenti;
- *messaggi di notifica*: sono messaggi che non richiedono alcuna risposta.

Ogni messaggio di richiesta e di notifica è costituito dai seguenti campi:

- *identificatore (id)*: è un identificatore univoco, aggiornato in modo indipendente lato client e lato server; pertanto può esistere una richiesta inviata dal server che ha il medesimo identificativo di una richiesta inviata dal client ma non possono esistere due richieste con medesimo id, provenienti dalla stessa entità. L'identificativo presente in una richiesta, viene replicato nella risposta corrispondente. All'interno di un messaggio di notifica, il campo id assume valore *null*;
- *metodo (method)*: è una stringa che identifica il tipo di messaggio;
- *parametri (params)*: è una lista variabile di parametri, come ad esempio l'identificativo del miner, l'identificativo del job e diversi altri.

Ogni messaggio di risposta è costituito dai seguenti campi:

- *identificatore (id)*: assume il medesimo significato visto in precedenza;
- *risultato (result)*: è il risultato dell'operazione richiesta. Può essere uno qualsiasi dei tipi di dato supportati da JSON [28] (string, list, array, eccetera);
- *errore (error)*: può assumere diversi valori, in base all'errore verificatosi. Alcuni tra i più comuni sono:
  - 21: *Job not found*, ovvero identificativo del job non esistente;
  - 24: *Unauthorized worker*, ovvero fase di autorizzazione fallita;
  - 25: *Not subscribed*, ovvero fase di sottoscrizione fallita.

3.2.2 Analisi dei messaggi

I tipici messaggi scambiati nel protocollo Stratum sono rappresentati in Figura 3.1; i dettagli della loro struttura, in formato JSON, sono riportati nell'appendice C.

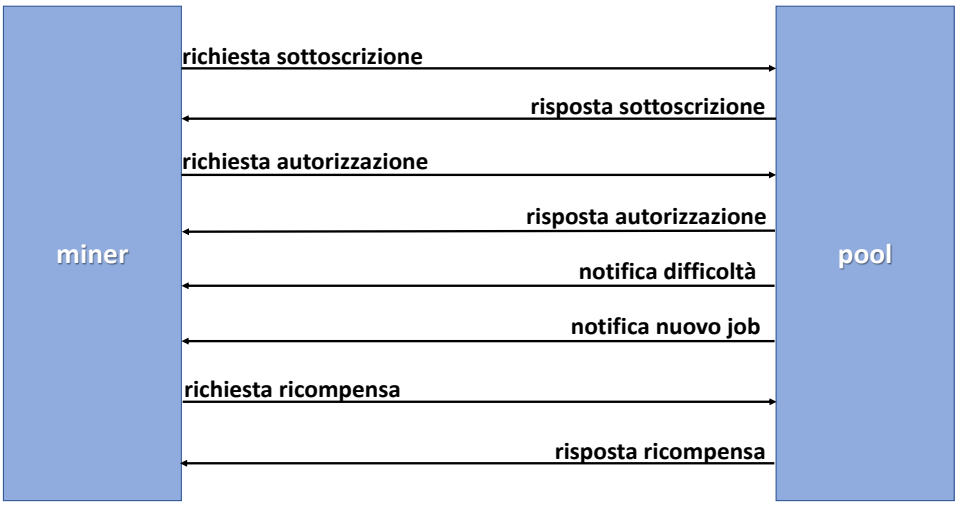

Figura 3.1: Tipici messaggi del protocollo Stratum.

La descrizione dei messaggi è la seguente:

richiesta sottoscrizione: messaggio di richiesta, tramite cui il miner si registra al mining pool;

risposta sottoscrizione: messaggio di risposta del mining pool alla richiesta precedente. Tale messaggio contiene i seguenti campi:

- *id connessione*: identificativo univoco della connessione;
- *extranonce1*: numero casuale, unico per connessione, utilizzato per comporre la transazione coinbase[9];
- *dimensione extranonce2*: dimensione dell'extranonce2, ovvero un numero casuale utilizzato all'interno del processo di mining (Appendice B.3).

richiesta autorizzazione: dopo la registrazione, il miner si autentica al mining pool. I parametri di questo messaggio sono l'identificativo del miner (ad esempio username) e, opzionalmente, la password;

risposta autorizzazione: risposta del mining pool alla richiesta di autorizzazione da parte del miner. Il campo *result* all'interno del messaggio può assumere o valore *true* se l'autorizzazione è avvenuta con successo, o valore *false* in caso contrario;

notifica difficoltà: completata con successo la fase di autorizzazione, il mining pool invia una notifica della difficoltà minima che il miner deve essere disposto ad accettare per partecipare al pool di mining;

notifica nuovo job: notifica inviata dal mining pool quando è disponibile un nuovo job.

Il messaggio in questione contiene diversi campi, tra cui:

- *id job*: identificativo del job. Verrà utilizzato in seguito, quando il miner reclamerà la ricompensa per il suo lavoro;
- *coinbase1*: parte iniziale della transazione coinbase;
- *coinbase2*: parte finale della transazione coinbase;
- *opzione clean-job*: se l'opzione contiene il valore *true*, il miner deve immediatamente interrompere il job che sta calcolando ed iniziare quello inviato all'interno di questo messaggio di notifica.

richiesta ricompensa: il miner, una volta che ha completato il cryptomining, invia la richiesta per ottenere la ricompensa. La ricompensa (anche chiamata *miner share*) è costituita dalla quota, rispetto all'ammontare totale di criptovaluta presente nella transazione coinbase, calcolata in base alla difficoltà del lavoro di mining svolto.

Il messaggio in analisi contiene diversi campi, tra cui:

- *id miner*: identificativo del miner;
- *id job*: identificativo del job;
- *extranonce2*: extranonce2 calcolato dal miner durante il processo di mining;
- *nonce*: parametro calcolato, insieme a extranonce2, durante il processo di mining (Appendice B.3).

risposta ricompensa: se il mining pool verifica con successo la richiesta inviata in precedenza, invia al miner la ricompensa. Il campo *result* di tale messaggio può assumere valore *true* se la verifica è andata a buon fine, o *false* in caso di esito negativo.

[9] La transazione coinbase [32] è la prima transazione contenuta in ciascun blocco della blockchain. Tale transazione è deputata ad immettere nuova moneta all'interno della rete di scambio della criptovaluta, sotto forma di ricompensa nei confronti dei miner che hanno completato con successo il processo di mining. (Per approfondire, si rimanda alla Appendice B.2.1)

3.3 XMR-Stak

XMR-Stak[10] è un software che permette di minare diverse criptovalute che utilizzano l'algoritmo CryptoNight (Appendice B.4.3), tra cui Monero. Esso utilizza il protocollo Stratum in chiaro o cifrato con TLS per le comunicazioni con il mining pool.

XMR-Stak possiede un'interfaccia HTTP che permette di monitorare da remoto l'hash rate prodotto dal mining di Monero, oltre che i risultati ottenuti e le statistiche sulla connessione. É inoltre possibile ottimizzare il processo di mining, in base all'hardware utilizzato. Le opzioni disponibili sono:

1. *CPU only:* opzione da scegliere se non si ha a disposizione una GPU per il mining;
2. *NVIDIA/AMD only:* opzione da scegliere in caso di presenza di una o più GPU. Il software è in grado di rilevare in automatico il produttore della scheda video.

Questo è il software che è stato utilizzato, nella parte implementativa della tesi, per generare una porzione del traffico Stratum, sia cifrato che in chiaro, per l'addestramento dei classificatori (Capitolo 5). Tale scelta deriva dal fatto che molti malware integrano al loro interno XMR-Stak o software simili per effettuare il cryptojacking sui dispositivi infettati.

3.4 Cryptojacking tramite web browser

Il *web-cryptojacking* o *drive-by mining* o *browser-based mining* [36, 40, 41, 42] indica la pratica illegale di utilizzare le risorse computazionali degli utenti per il mining di una criptovaluta, mentre si visita una pagina web.

Secondo *Eskandari et al.* [36], il browser-based mining è ad oggi la tipologia di cryptojacking più diffusa.

L'idea del mining tramite browser è nata pressoché in concomitanza con la diffusione del bitcoin ma questa pratica era stata abbandonata a causa dell'esigenza di eseguire il mining su hardware specifico. La diffusione di Monero con le sue caratteristiche di poter essere eseguito su macchine general-purpose e di garantire l'anonimato, ha fatto rifiorire il web-browser mining. Questa tesi è anche avvalorata dall'interesse di ricerca[11], tramite Google, dei termini *browser mining* e *Monero* come mostrato in Figura 3.2.

3.4.1 Descrizione dell'attacco

Al fine di una migliore comprensione dell'attacco di web-cryptojacking, è utile descrivere brevemente le seguenti tecnologie per il web:

- *Web Workers*[12]: sono delle primitive di programmazione utilizzate all'interno di codice JavaScript per creare dei thread di background;
- *Wasm (WebAssembly)*[13]: è un linguaggio di basso livello simile all'Assembler, che può essere eseguito dai browser e che risulta più efficiente rispetto a JavaScript. È possibile generare codice Wasm a partire da codice di più alto livello, come ad esempio funzioni in linguaggio C per il calcolo dell'hash crittografico. Inoltre, Wasm è interoperabile con JavaScript;
- *asm.js*[14]: è un sottoinsieme di JavaScript, ideato per permettere l'esecuzione di programmi,

[10] https://github.com/fireice-uk/xmr-stak

[11] L'interesse di ricerca rappresenta la frequenza con cui gli utenti hanno ricercato un termine tramite Google: un interesse di ricerca prossimo a 100 indica che, in un determinato periodo di tempo, si è verificata la massima frequenza di ricerca del termine in questione.

[12] https://www.w3.org/TR/workers/

[13] https://www.w3.org/TR/wasm-web-api-1/

[14] http://asmjs.org/spec/latest/

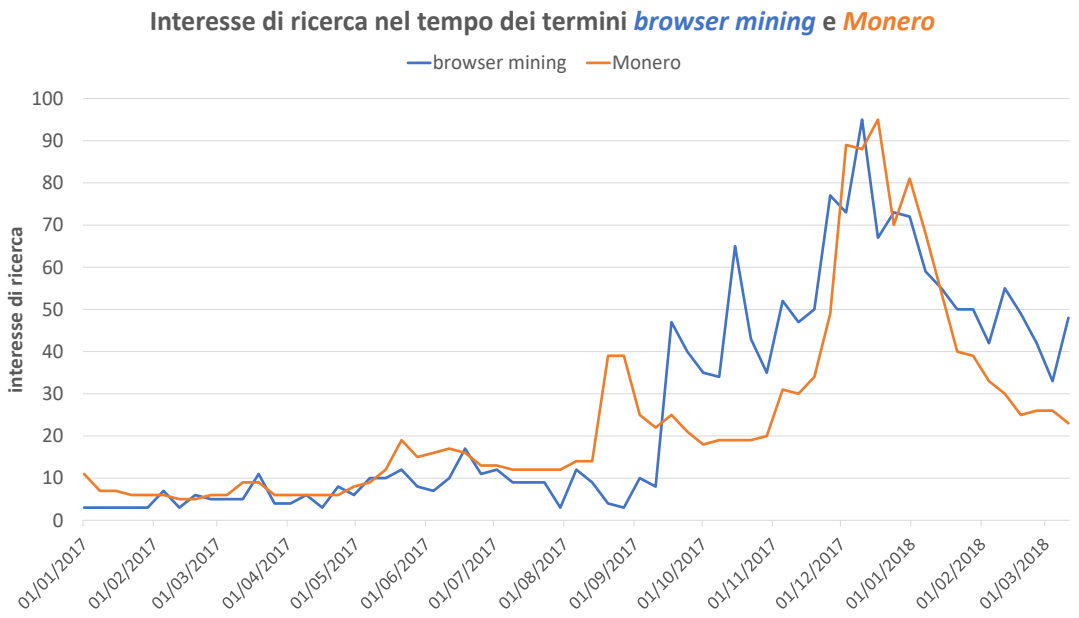

Figura 3.2: Confronto dell'interesse di ricerca, tramite Google, dei termini: *browser mining* e *Monero*. (fonte: trends.google.it)

scritti ad esempio in C, all'interno del browser.

Uno script JavaScript di cryptojacking è solitamente costituito da due componenti [42]:

- uno script JavaScript di orchestrazione (chiamato nel seguito *orchestratore*) che si occupa di analizzare l'hardware a disposizione dell'utente (ad esempio il numero di core) e le caratteristiche del browser (ad esempio il supporto per Wasm). Ad esempio, per conoscere il numero di core disponibili, viene utilizzata la proprietà *hardwareConcurrency* dell'oggetto *Navigator*;
- il codice per il cryptojacking, scritto in linguaggio asm.js o Wasm.

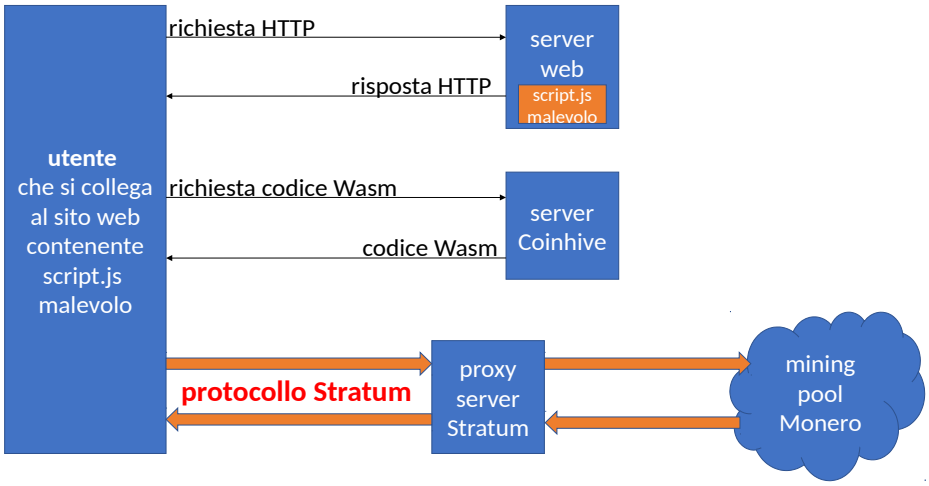

Figura 3.3: Esempio di web-cryptojacking per il mining di Monero tramite API Coinhive.

L'attacco drive-by mining, mostrato in Figura 3.3, si articola nelle seguenti fasi [42]:

1. il sito web malevolo ospita uno script JavaScript che, tramite lo script orchestratore, controlla le risorse hardware e le caratteristiche del browser degli utenti collegati. In base al numero di core disponibili, vengono istanziati il numero appropriato di thread Web Workers;
2. viene scaricato da un sito web esterno, ad esempio il server di Coinhive[15], il codice asm.js o il codice Wasm delle funzioni di hash crittografico utilizzate per il mining della criptovaluta;
3. il browser esegue il codice per il mining della criptovaluta.

Per le comunicazioni tra i miner (gli utenti che visitano il sito) ed il mining pool, viene solitamente utilizzato il protocollo Stratum di cui si è parlato in precedenza (Sezione 3.2). Inoltre, in alcuni casi si osserva l'utilizzo di un proxy server (ad esempio un proxy Stratum) sia per ottimizzare le connessioni verso il mining pool sia per celare la pseudo-botnet[16] di browser e renderla meno individuabile.

Lo script malevolo JavaScript ospitato dai siti web per il cryptojacking può avere diverse origini:

- potrebbe essere inserito dal webmaster del sito web, per compensare la diminuzione degli introiti pubblicitari dovuti all'utilizzo sempre più diffuso degli *ad-blocker*, ovvero plugin per browser in grado di bloccare gli avvisi pubblicitari;
- un'altra possibilità è l'inserimento dello script in seguito ad una violazione del server web oppure dei servizi web da esso utilizzati. Ad esempio, nel gennaio 2018, alcuni analisti hanno scoperto che, tra gli avvisi pubblicitari mostrati da Youtube, alcuni contenevano script Coinhive [43];
- lo script malevolo potrebbe essere nascosto all'interno di plugin per il browser. Ad esempio l'estensione per Chrome *Archive Poster*, è rimasta nel web store di Google per giorni, permettendo il cryptojacking nei confronti di migliaia di utenti che l'avevano aggiunta al proprio browser [44].

3.4.2 Coinhive

Il picco nei grafici mostrati in Figura 3.2, coincide con la nascita, nel Settembre 2017, della società *Coinhive*, che ha sviluppato una API JavaScript per il mining di Monero tramite browser.

Secondo *Eskandari et al.*, Coinhive, nonostante la nascita di diversi competitor, è il leader di mercato nel fornire API per il web cryptojacking. In cambio dell'utilizzo delle sue API, la società richiede il 30% dei proventi ricavati dal mining.

Il Listato 3 mostra un esempio di script JavaScript che utilizza la API di Coinhive.

```
1  <script src="https://coinhive.com/lib/coinhive.min.js">   //orchestratore
2  </script>
3
4  <script>
5          var miner = new CoinHive.Anonymous('SITE_KEY', {throttle: 0.3});
6          miner.start();
7  </script>
```

Listato 3: Esempio di script JavaScript che include API Coinhive per il mining di Monero.

Alcune delle opzioni che è possibile specificare all'interno del costruttore (riga n. 5 del codice mostrato nel Listato 3) sono:

[15] https://coinhive.com/

[16] Le botnet saranno descritte in dettaglio nel Capitolo 4.

1. *SITE_KEY:* la chiave pubblica associata al sito che ospita lo script malevolo, creata quando il webmaster si iscrive al servizio di Coinhive;

2. *throttle:* la frazione di tempo in cui i thread che eseguono il codice per il mining devono rimanere in stato di idle. La documentazione presente sul sito, consiglia di utilizzare il 70% delle risorse di CPU.

Coinhive fornisce anche un servizio di *Captcha*: l'utente, dopo aver cliccato su un apposito widget, deve attendere alcuni secondi durante i quali avviene il cryptojacking di Monero.

Con la diffusione del drive-by mining, è sorto un dibattito sulla legalità dell'attività di tale società: Coinhive, infatti, non richiede il consenso esplicito dell'utente per il mining della criptovaluta e pertanto il suo operato rientra tra le pratiche di cryptojacking. Per tale motivo, la società ha creato il servizio *Authedmine*[17] che differisce da quello descritto in precedenza, soprattutto per la richiesta del consenso dell'utente per utilizzare una parte delle sue risorse hardware in cambio di benefici, quali ad esempio la rimozione parziale o totale degli avvisi pubblicitari. La nascita *Authedmine* mette in luce l'esistenza di alcune problematiche etiche, quali ad esempio la consapevolezza da parte dell'utente di quali siano i risvolti negativi (ad esempio maggiore consumo di energia elettrica e il rallentamento del dispositivo) nel dare il proprio consenso per il mining.

3.5 Cryptojacking tramite malware: MadoMiner

Secondo A. Lioy *et al.* [45], i malware (MALicious softWARE), o software malevoli, sono:

> programmi realizzati per danneggiare i sistemi su cui vengono eseguiti o per sottrarre informazioni sensibili.

Esistono diverse tipologie di malware, talvolta sovrapponibili, tra cui [17]:

- *virus*: è un software malevolo che è in grado di propagarsi all'interno di altri programmi, iniettando una routine malevola. Quando il programma infettato viene eseguito, prima viene eseguita la routine malevola e dopo viene esplicata la funzionalità originaria del programma;

- *cavallo di troia o trojan*: è un malware che viene celato all'interno di programmi, anche chiamati *dropper*, che sembrano non avere fini malevoli;

- *worm*: è un malware che, al contrario dei virus, non è in grado di iniettare codice malevolo in altri programmi, ma sfrutta delle vulnerabilità software (o *exploit*) per replicarsi in altri host connessi alla rete;

- *ransomware*: è un malware che limita l'accesso al sistema informatico che infetta, cifrando il contenuto dei dispositivi di memorizzazione. È possibile che l'attaccante richieda un riscatto per la decifratura, anche tramite criptovaluta come ad esempio è accaduto per il ransomware WannaCry [46];

- *keylogger*: è un software malevolo che è in grado, nei sistemi compromessi, di registrare i dati immessi dagli utenti tramite i dispositivi di input (ad esempio la tastiera).

MadoMiner [47, 48] è un worm, anche con funzionalità di keylogger, in grado di eseguire il mining di XMR. È stato scoperto nel Settembre 2018, dal ricercatore *J. Quinn*.

Questo malware sfrutta le seguenti vulnerabilità:

[17] https://authedmine.com

- CVE-2017-0143[18] e CVE-2017-0146[19]: sono delle vulnerabilità nel protocollo *SMB (Server Message Block)* v1.0 [49], protocollo sviluppato da Microsoft, di condivisione di file all'interno di una rete, solitamente LAN, che opera al di sopra del protocollo TCP/IP. Queste vulnerabilità permettono di eseguire codice remoto sui dispositivi in attesa sulla porta TCP 445, utilizzata da SMB, a seguito dell'invio di una particolare richiesta;
- CVE-2017-0176[20] [50]: è una vulnerabilità presente nella versione del protocollo RDP utilizzata in Windows Server 2003 e Windows XP. *RDP (Remote Desktop Protocol)* [51] è un protocollo proprietario di Microsoft, che permette la comunicazione tra un client RDP e un server RDP in attesa sulla porta TCP 3389. In particolare, per sfruttare questa vulnerabilità, il server RDP deve essere abilitato ad effettuare l'autenticazione tramite Smart Card. Grazie a questo exploit, un attaccante è in grado di installare programmi, modificare o cancellare i dati e creare account con privilegi di amministratore sulla macchina remota.

L'infezione del dispositivo vittima avviene tramite lo strumento malevolo ZombieBoyTools.exe che installa un file *DLL (Dynamic-Link Library)* (x86.dll se il S.O. è a 32 bit, x64.dll se il S.O. è a 64 bit), sfruttando gli exploit menzionati in precedenza. Quando il file DLL viene eseguito, vengono scaricati ed eseguiti in parallelo due eseguibili in formato UPX[21], chiamati *Install.exe* e *Mask.exe*. In Figura 3.4 è riportato lo schema generale di funzionamento di Madominer.

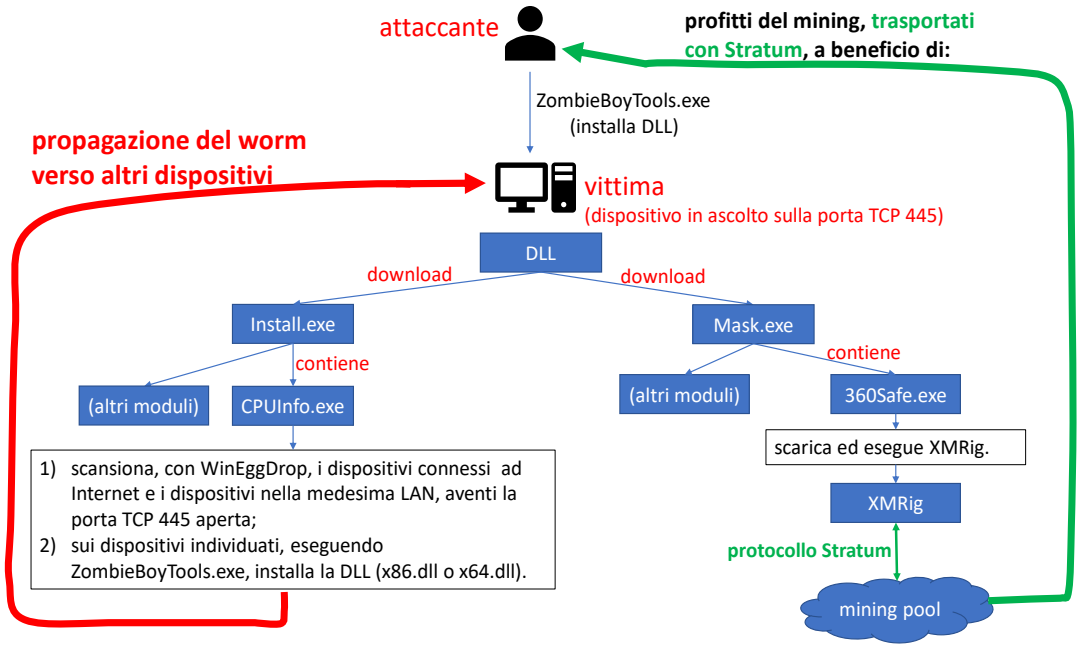

Figura 3.4: Schema generale di funzionamento di MadoMiner.

Il modulo Install.exe è è deputato alla propagazione del worm. Al suo interno vi sono:

- alcuni script batch che hanno le seguenti funzionalità:
  - rendere persistente il worm, aggiungendo alcuni processi allo scheduler del sistema operativo;
  - rendere meno individuabile il malware terminando alcuni suoi processi e cancellando alcuni moduli dopo il loro utilizzo;

[18] https://www.rapid7.com/db/vulnerabilities/msft-cve-2017-0143

[19] https://www.rapid7.com/db/vulnerabilities/msft-cve-2017-0146

[20] https://www.rapid7.com/db/vulnerabilities/msft-cve-2017-0176

[21] https://upx.github.io/

  - supportare l'esecuzione del modulo CPUInfo.exe;

- il dropper *CPUInfo.exe* esegue le seguenti azioni:

  1. scarica i file necessari all'esecuzione di MadoMiner, tra cui ZombieBoyTools.exe, il web scanner *WinEggDrop* e gli strumenti per sfruttare le vulnerabilità citate in precedenza;
  2. tramite il web scanner WinEggDrop, CPUInfo.exe scansiona i dispositivi collegati alla medesima rete locale (LAN) del dispositivo infettato, al fine di individuare quelli in ascolto sulla porta TCP 445 (protocollo SMB attivo); per il medesimo scopo, vengono scansionati gli indirizzi IP pubblici appartenenti allo stesso intervallo di indirizzi della macchina infettata;
  3. per ogni dispositivo individuato, tramite ZombieBoyTools.exe, CPUInfo.exe scarica ed installa il file DLL più opportuno (x86.dll se il S.O. è a 32 bit, x64.dll se il S.O. è a 64 bit);
  4. vengono quindi scaricati i moduli Mask.exe ed Install.exe e la propagazione del worm continua.

Il modulo Mask.exe ha il compito di minare la criptovaluta Monero sui dispositivi infettati. Tali dispositivi costituiscono una botnet (Sezione 4.1).

Si stima che, gli utilizzatori di MadoMiner siano riusciti in questo modo a guadagnare 6015$ al mese, utilizzando il 50% delle risorse computazionali disponibili sui sistemi infettati [47]. Le comunicazioni tra il botmaster, i miner (ovvero i dispositivi infettati) ed il mining pool avviene tramite l'utilizzo del protocollo Stratum (Sezione 3.2), operante al di sopra di TCP senza l'utilizzo di TLS.

Mask.exe, inoltre, scarica ulteriori moduli tra cui *360Safe.exe*. Esso svolge i seguenti compiti:

- installa *NSSM (Non-Sucking Service Manager)*[22], un monitor di processi in grado riavviare un processo qualora termini in modo inaspettato;
- attua delle tecniche anti-VM per rendere più complesso il rilevamento del malware. Una tecnica anti-VM (o anti-Virtual Machine) ha lo scopo di impedire l'esecuzione del malware all'interno di una macchina virtuale. Quest'ultime sono utilizzate dagli esperti di sicurezza per analizzare i malware senza compromettere le macchine su cui viene effettuata l'analisi;
- identifica l'architettura del sistema operativo;
- in base alle caratteristiche del sistema operativo, installa XMRig[23], uno strumento per il mining di XMR, alternativo a XMR-Stak (Sezione 3.3) e che pertanto utilizza il protocollo Stratum (nel caso in esame, non cifrato).

[22] https://nssm.cc/

[23] https://github.com/xmrig/xmrig

Capitolo 4

Botnet

4.1 Introduzione

Una *botnet (roBot Net)*, letteralmente rete di robot, è un insieme di macchine remote compromesse (*bot* o *zombie*) da un attaccante (*botmaster*) sfruttando delle vulnerabilità software e/o utilizzando un malware (un esempio è il worm MadoMiner descritto nella Sezione 3.5). Tipicamente il botmaster utilizza uno o più server chiamati *server master o server di C&C (command-and-control)* [52] per controllare e sincronizzare l'azione dei bot.

Le botnet possono essere utilizzate per svariati scopi:

- *spamming:* inviare grandi volumi di email indesiderate (ad esempio pubblicitarie o a scopo di phishing[1]);
- *cryptojacking:* MadoMiner, ad esempio, crea una botnet allo scopo di minare XMR (vedi Appendice B.4);
- *rubare dati sensibili:* il malware Dridex [53], ad esempio, dopo essere stato scaricato dalla rete sul computer vittima[2], è in grado di rubare dati bancari (ad esempio installando un keylogger) ed utilizzarli per effettuare delle transazioni. Si stima che questo malware sia stato in grado di rubare 40 milioni di dollari solo nel 2015;
- *sferrare attacchi DDoS (Distribuited Denial-of-Service)*: DDoS è un attacco che ha lo scopo di amplificare gli effetti di un attacco DoS tramite l'utilizzo di una botnet. Secondo P. Cichonksi [54] un attacco *DoS (Denial-of-Service)* è:

  > un'azione che impedisce o altera l'utilizzo autorizzato di reti, sistemi o applicazioni esaurendo delle risorse quali la CPU, la memoria, la banda di rete e lo spazio di archiviazione.

  Diversi esempi di attacchi DDoS saranno illustrati nella Sezione 4.3.2.

Un bot può essere un qualsiasi dispositivo connesso alla rete. Una tendenza [55] degli ultimi anni è utilizzare botnet costituite da dispositivi IoT (Internet of Things), ovvero dispositivi costantemente connessi alla rete come ad esempio videocamere, router, dispositivi indossabili (*wearable*) e sensori per la domotica. I vantaggi per gli attaccanti che decidono di creare botnet sfruttando tali dispositivi sono molteplici:

[1] Un attacco di phishing consiste nel tentativo di estorsione di informazioni sensibili (ad esempio il numero di carta di credito dell'utente di un sito web) con mezzi ingannevoli (alcuni esempi saranno descritti nelle sezioni successive).

[2] Lo scaricamento di Dridex su un dispositivo, avviene grazie all'attivazione di una macro, quando l'utente apre un documento MS Word malevolo, ricevuto come allegato di un'email di phishing.

- tali dispositivi presentano solitamente basse misure di sicurezza a causa della scarsa potenza computazionale a loro disposizione: infatti spesso non usano moduli anti-malware oppure ne utilizzano versioni con funzionalità limitate per ridurre la richiesta di risorse;
- tipicamente sono sempre accesi e connessi ad Internet, spesso usano le stesse credenziali di accesso impostate dalla fabbrica produttrice e pertanto rappresentano un bersaglio appetibile per gli attaccanti.

Un esempio di botnet IoT è Mirai [55], utilizzata per la prima volta il 19 settembre 2016, per effettuare un attacco DDoS contro la società di servizi per il cloud OVH e il sito di sicurezza informatica Krebs[3]. Essa è riuscita a generare un volume di traffico di circa 1 Tbps.

4.2 Pybot

Pybot[4] è un software open source, scritto in Python, che non implementa un vero e proprio attacco ma che permette già di ottenere una botnet funzionante. Dal momento che il codice di Pybot è relativamente semplice da comprendere, esso può essere facilmente integrato all'interno di strumenti in grado di implementare attacchi veri e propri.

Pybot è costituito da tre moduli principali:

- *Master.py*: script che emula il funzionamento del nodo master all'interno della botnet;
- *Bot.py*: script che emula il funzionamento di un nodo zombie all'interno della botnet;
- *TargetServer.py*: script che, in esecuzione, emula un server TCP. La sua funzione è quella di fungere da bersaglio dell'attacco e mostrare a video i messaggi testuali inviati dagli attaccanti.

Il flusso di esecuzione di Pybot è il seguente:

1. viene eseguito lo script TargetServer.py, che riceve come argomento da linea di comando la porta su cui deve restare in attesa di nuove connessioni. Il valore di default è 8080;
2. vengono eseguiti uno o più script Bot.py a seconda del numero di bot desiderato. Lo script Bot.py può ricevere i seguenti argomenti da linea di comando:
   - *port* (valore di default 21800): la porta su cui il bot deve restare in attesa di ricevere i comandi dal master;
   - *offset* (valore di default 0 ms): l'offset che viene aggiunto al tempo rilevato al momento dell'esecuzione dello script;
   - *rate* (valore di default 1 ms): la frequenza con la quale il bot invia al target un messaggio testuale (di default è `You're be attacked!!!`) per indicare l'attacco.
3. viene eseguito lo script Master.py, che può ricevere i seguenti argomenti da linea di comando:
   - *target* (valore di default *localhost*): il nome dell'host target (*hostname*) verso cui condurre l'attacco;
   - *port* (valore di default 8080): la porta su cui il target è in attesa di nuove connessioni TCP;
   - *time* (valore di default 10 s): specifica dopo quanti secondi dall'esecuzione dello script deve essere lanciato l'attacco.

Il protocollo di comunicazione tra master, bot e target, rappresentato in Figura 4.1, è simile a quello utilizzato da Telnet [56], e consiste nell'inviare in chiaro su un canale TCP una serie di stringhe di caratteri.

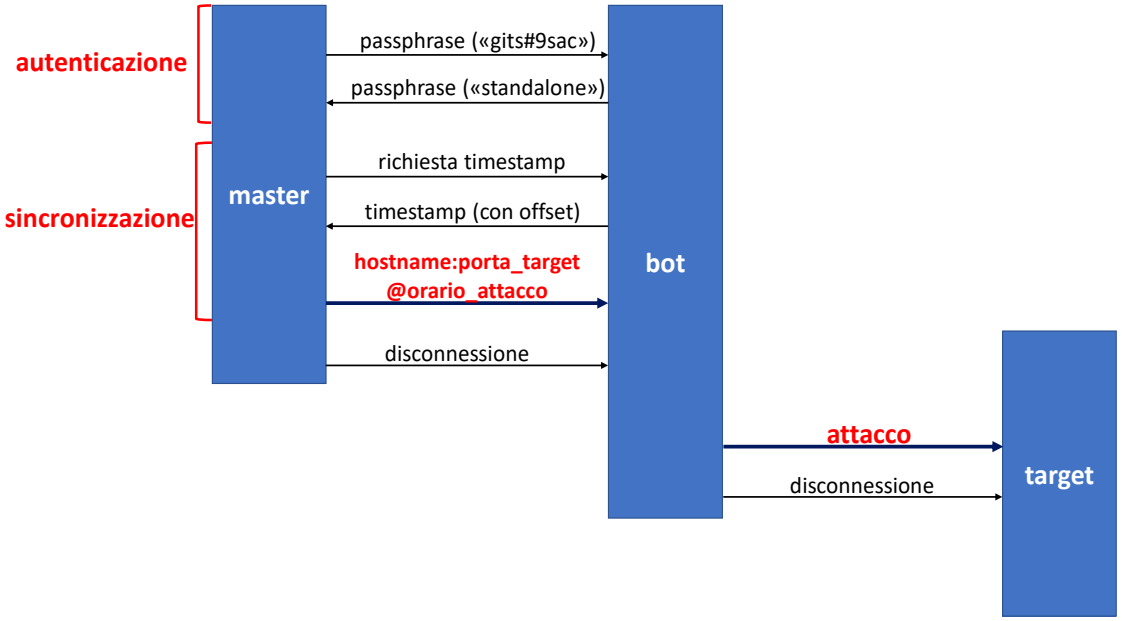

Figura 4.1: Protocollo di comunicazione utilizzato da Pybot.

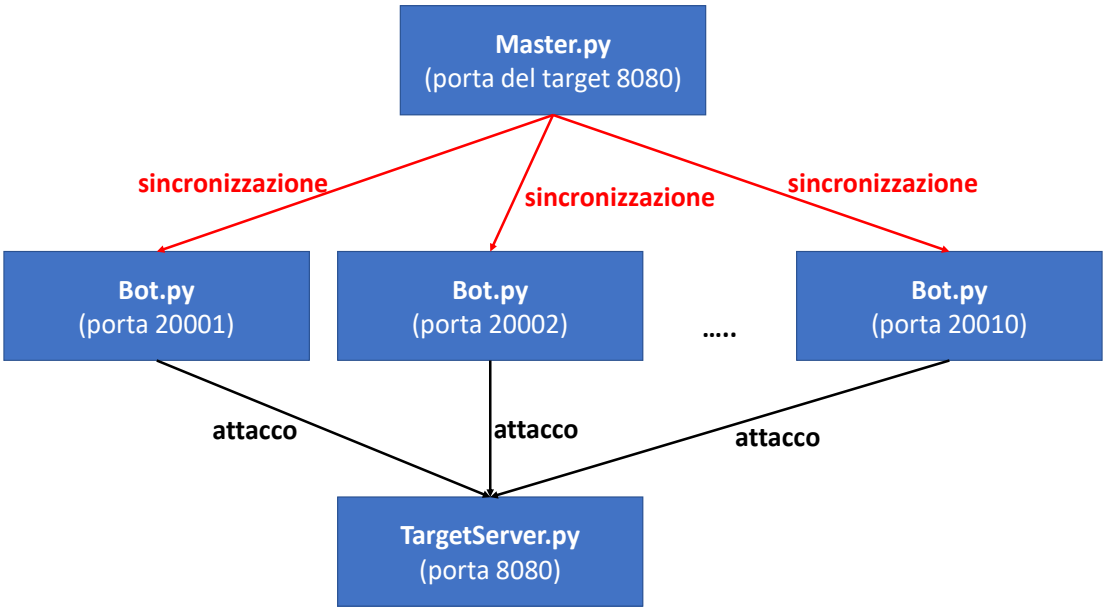

Figura 4.2: Schema generale di attacco tramite Pybot.

L'attacco supportato da Pybot è rappresentato in Figura 4.2, e si articola nelle seguenti fasi:

1. il master legge un file di configurazione in cui è riportata la lista dei bot secondo il formato *hostname:porta*;
2. il master si autentica a ciascun bot, inviando una passphrase (il valore di default è `gits \#9sac`);

[3] https://krebsonsecurity.com/

[4] https://github.com/hacktoolspack/hack-tools/tree/master/pybot

3. se la passphrase ricevuta risulta corretta, ciascun bot si autentica al master inviando la propria passphrase (il valore di default è `standalone`);
4. se la passphrase ricevuta risulta corretta, il master richiede a ciascun bot il timestamp in formato UNIX;
5. ogni bot invia il proprio timestamp, comprensivo del parametro offset ricevuto da linea di comando;
6. il master, per ogni timestamp ricevuto, effettua il seguente calcolo allo scopo di sincronizzare l'attacco dei bot:
$$\text{tempo\_attacco\_bot} = \text{tempo\_attacco} + |\text{tempo\_bot} - \text{tempo\_master}|$$
dove:
   - tempo_attacco_bot: è il tempo di attacco che sarà comunicato al bot;
   - tempo_attacco: è il tempo di attacco ricevuto dal master come parametro da linea di comando;
   - tempo_bot: è il timestamp inviato da ciascun bot;
   - tempo_master: è il tempo misurato dal master;
7. il master invia a ciascun bot le seguenti informazioni: hostname del target, porta del target e tempo di attacco. In seguito si disconnette in modo da non essere rintracciabile;
8. ogni bot rimane in stato di idle fintanto che non arriva il momento di attaccare;
9. durante l'attacco, ogni bot invia al server target un messaggio testuale (`You're being attacked!!!`) con la frequenza specificata tramite il parametro *rate*, per una durata di 30 secondi. In seguito chiude la connessione.

4.3 UFONet

UFONet[5] è uno strumento open source nato nel 2013, scritto tramite i linguaggi Python, Javascript e HTML5/CSSv3. Questo strumento, sfruttando la vulnerabilità *Open Redirect*, è in grado di lanciare attacchi DDoS tramite una botnet.

4.3.1 La vulnerabilità Open Redirect

La vulnerabilità *Open Redirect*, nota anche come CWE (Common Weakness Enumeration) 601[6], è una vulnerabilità software presente in alcune applicazioni web, che permette la redirezione (o redirect) verso siti non fidati. Tale vulnerabilità, ad esempio, è stata riscontrata nelle applicazioni web sviluppate tramite il framework ASP.NET MVC[7] creato da Microsoft (la vulnerabilità[8] era presente sin dalla prima versione del framework ed è stata risolta a partire dalla versione 3) e in quelle sviluppate tramite il framework Express[9] (la vulnerabilità[10] è stata scoperta nella versione 4.8.8 e risolta nella versione 4.9.0).

La redirezione viene utilizzata per svariati scopi legittimi, tra cui[11]:

[5] https://ufonet.03c8.net/

[6] http://cwe.mitre.org/data/definitions/601.html

[7] https://www.asp.net/mvc

[8] https://www.owasp.org/index.php/Unvalidated_Redirects_and_Forwards_Cheat_Sheet

[9] https://expressjs.com/

[10] https://snyk.io/node-js/express

[11] https://developer.mozilla.org/en-US/docs/Web/HTTP/Redirections

- *domain aliasing*: la medesima risorsa risulta raggiungibile da diversi domini che reindirizzano verso l'URL (Uniform Resource Locator) univoco della risorsa;

- a seguito di richieste HTTP che utilizzano metodi non idem-potenti[12]. In tali casi è buona pratica reindirizzare l'utente verso un nuova pagina in modo da evitare la ripetizione involontaria dell'operazione effettuata (ad esempio l'utente potrebbe cliccare inavvertitamente il pulsante di ricaricamento della pagina);

- per reindirizzare le richieste della versione HTTP di un sito verso la propria versione sicura HTTPS.

Un possibile attacco che sfrutta tale vulnerabilità è rappresentato in Figura 4.3. Esso si articola nelle seguenti fasi:

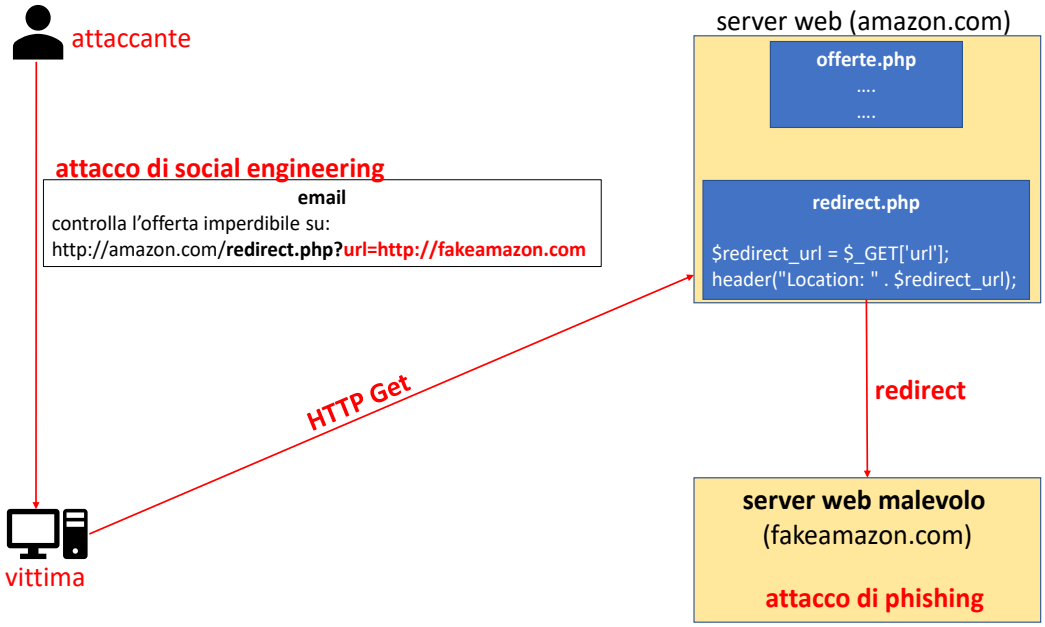

Figura 4.3: Esempio di attacco che sfrutta la vulnerabilità Open Redirect.

1. una applicazione web (es. `amazon.com`) contiene una pagina dinamica (ad esempio `redirect.php`) che estrapola l'URL contenuto in una richiesta HTTP Get e reindirizza (*URL redirection*) l'utente verso la pagina web corrispondente, senza effettuare alcun tipo di controllo (da qui deriva il nome Open Redirect);

2. l'attaccante, tramite attacchi di social engineering (ad esempio un'email che indica un'offerta imperdibile su un sito di commercio elettronico), induce l'utente a visitare la pagina corrispondente all'URL malevolo (ad esempio `http://amazon.com/redirect.php?url=http://fakeamazon.com`). Dal momento che la parte iniziale dell'URL riguarda un sito affidabile, l'utente inesperto non pone molta attenzione all'azione compiuta;

3. l'utente, grazie alla vulnerabilità Open Redirect presente sul sito legittimo di commercio elettronico, viene reindirizzato verso il sito web gestito dall'attaccante;

[12]Un metodo è idem-potente [57] se può essere rieseguito più volte senza che il risultato ottenuto cambi (es. le risorse di un sito web). Ad esempio, il metodo HTTP Get è idem-potente, mentre i metodi HTTP Put, HTTP Post e HTTP Delete non lo sono.

4. su tale sito possono avvenire diversi attacchi di phishing. Una possibilità è che il sito contenga un form in cui sono richiesti i dati dell'utente per portare a termine l'acquisto del prodotto. Una volta compilati tali campi, i dati dell'utente vengono inviati al malintenzionato e l'attacco[13] ha successo.

Esistono alcune contromisure che possono risolvere queste vulnerabilità:

- evitare di utilizzare le redirect;
- se non è possibile evitarne l'utilizzo, applicare la regola aurea valida in generale per la sicurezza delle applicazioni web [17]: qualsiasi dato proveniente dall'esterno deve essere validato lato server. Una possibilità è utilizzare una whitelist contenente i parametri e gli URL considerati validi.

4.3.2 Attacchi DDoS

Lo strumento UFONet permette di effettuare in automatico diversi attacchi DDoS. A tale scopo, è possibile configurare alcuni parametri:

- modificare il contenuto di alcuni header HTTP [57], al fine di celare la provenienza dell'attacco. Alcuni di essi sono:
  - User-agent: esso specifica l'applicazione tramite cui è stata fatta una richiesta HTTP. Di default UFONet attua lo spoofing di tale header, ovvero rimpiazza i dati originali con dati fittizi;
  - X-Forwarded-For: è un header utilizzato per specificare l'indirizzo IP di una richiesta HTTP che è stata fatta tramite uno o più server proxy. Come nel caso precedente, UFONet di default attua lo spoofing di tale header;
  - Referer: è un header che contiene l'URL della risorsa a partire dalla quale si è giunti all'URL corrente. Di default UFONet ne fa lo spoofing;
- *threads*: indica il numero massimo di richieste HTTP che possono essere effettuate in modo concorrente. Modificando tale parametro è possibile aumentare la portata dell'attacco (ad esempio, incrementare il numero di richieste potrebbe saturare la banda del server vittima in tempi più brevi);
- *delay*: indica il ritardo, espresso in secondi, tra due richieste HTTP consecutive. Anche questo parametro, come quello descritto in precedenza, permette di aumentare gli effetti negativi dell'attacco (ad esempio diminuendo il parametro delay, è possibile saturare la banda dell'host vittima più velocemente).

L'attacco DDoS di base supportato da UFONet, mostrato in Figura 4.4, si articola nelle seguenti fasi:

1. la prima fase (opzionale) riguarda la ricerca degli host (ovvero i possibili bot) che espongono la vulnerabilità Open Redirect. Per effettuarla vengono utilizzati, all'interno dei motori di ricerca (Yahoo è il motore di ricerca di default), i *dork*, ovvero delle stringhe di caratteri particolari che permettono di ricercare delle risorse (pagine web, file, eccetera) altrimenti non facilmente visualizzabili. Il Listato 4 mostra un esempio di comando bash per la ricerca di bot, tramite il motore di ricerca Bing, utilizzando il dork `redirect.php?url=` ;

```
$       ./ufonet -s 'redirect.php?url=' --se 'bing'
$       Victim found: http://esempio1.com/vb/redirect.php?url=
$       Victim found: http://esempio2.com/admin/redirect.php?url=
```

Listato 4: Esempio di comando bash per la ricerca di bot, tramite il motore di ricerca Bing, utilizzando il dork `redirect.php?url=` .

[13] Questo particolare attacco di phishing prende il nome di *form grabbing*.

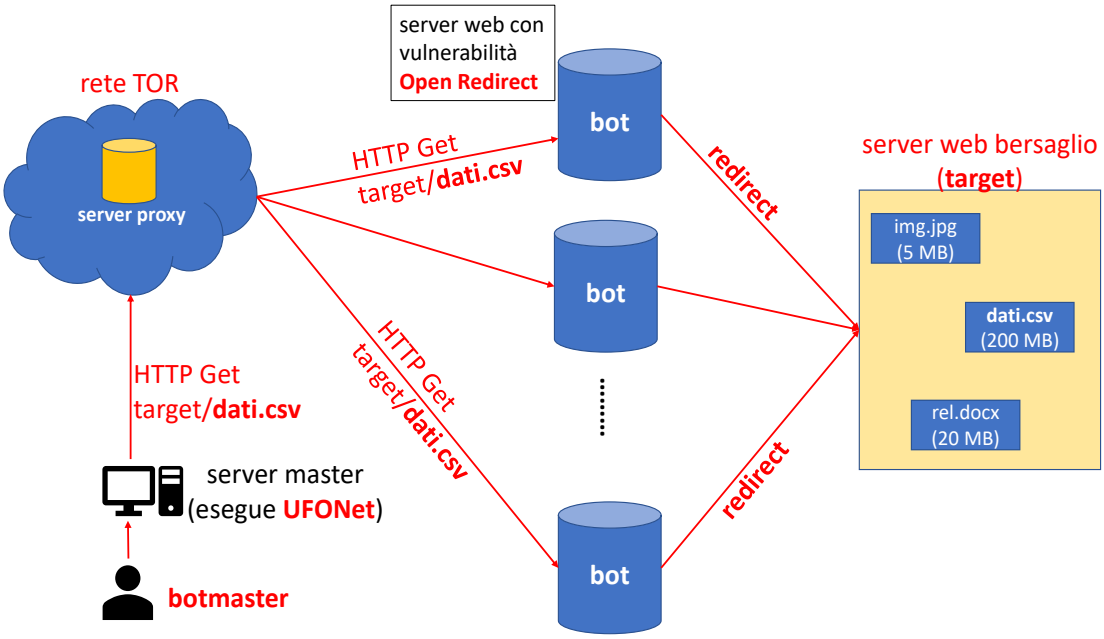

Figura 4.4: Esempio di attacco DDoS lanciato tramite UFONet.

2. in seguito è possibile verificare se gli host trovati presentano effettivamente la vulnerabilità Open Redirect, inviando una richiesta HTTP Head: quest'ultima, a differenza di una HTTP Get, richiede all'host una risposta contenente solo gli header. Essa viene solitamente usata a scopo di debug, per conoscere se l'entità associata ad un URL è accessibile e/o è stata aggiornata. Al termine di questa fase, la botnet è pronta ad essere utilizzata per un attacco DDoS;

3. l'attaccante, tramite il comando `./ufonet -i http://target.com`, può utilizzare un *web crawler*, ovvero un bot che sistematicamente ricerca su Internet tutte le risorse appartenenti al medesimo sito web e ne archivia il contenuto. Solitamente le informazioni derivanti dall'analisi di tali contenuti, sono utilizzate dai motori di ricerca per migliorare i risultati ottenuti dalle ricerche degli utenti. Purtroppo i web crawler possono essere utilizzati anche per fini malevoli: un attaccante, analizzando il contenuto delle pagine memorizzate dal crawler, è in grado di conoscere quali sono le risorse con dimensione più elevata, che pertanto aumentano le probabilità di saturare la banda del sito bersaglio, portando a termine l'attacco;

4. l'attaccante, opzionalmente, può collegarsi ad un server proxy appartenente ad una *darknet* per diminuire le possibilità di essere rintracciato. Una darknet (ad esempio *TOR, The Onion Routing*[14]), è una rete che garantisce l'anonimato nelle comunicazioni;

5. infine l'attacco può essere sferrato. Ad esempio, supponendo che l'attaccante sia riuscito a scoprire che la risorsa di dimensione maggiore presente sul sito bersaglio è `dati.csv`, può condurre l'attacco tramite il comando `./ufonet -a http://target.com -b '/dati.csv'`. Tale comando lancia l'attacco DDoS, comunicando[15] ai bot di scaricare dal server vittima la risorsa voluminosa individuata in precedenza.

Tramite UFONet, sono disponibili altri attacchi DDoS, descritti nelle sezioni successive.

[14] https://www.torproject.org/

[15] A differenza di quanto avvenuto per Pybot, in UFONet non è stato possibile identificare il protocollo utilizzato sul canale di C&C sia perchè il traffico è cifrato tramite TLS sia perchè il codice è di gran lunga più complesso di quello di Pybot.

Attacco di HTTP DB

L'attacco HTTP DB, mostrato in Figura 4.5, consiste nell'inondare la base dati di un sito web con richieste contenenti parametri casuali di ricerca.

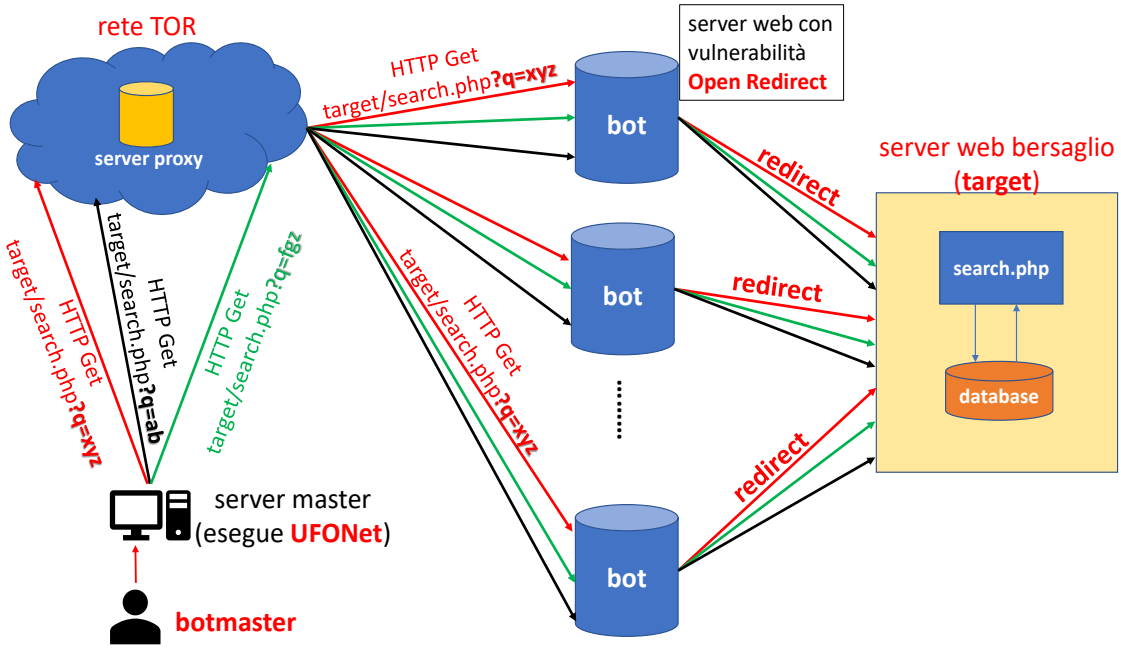

Figura 4.5: Esempio di attacco DDoS di HTTP DB eseguito tramite UFONet.

Digitando da terminale il comando `./ufonet -a http://target.com --db "search.php?q="`, UFONet crea una serie di richieste HTTP Get, contenenti parametri casuali, verso la pagina web dinamica (nell'esempio è `search.php`); tale pagina permette di effettuare delle ricerche all'interno del database del sito web bersaglio. Le richieste, quindi, vengono inviate alla botnet che ne moltiplica il numero, aumentando la probabilità di successo nell'attacco.

Una possibilità per mitigare questo tipo di attacco è limitare il numero di ricerche effettuabili nel database dal medesimo indirizzo IP (ad esempio aggiornando un contatore per ogni richiesta e chiudendo la connessione qualora venga oltrepassata una certa soglia). Ciò consente di ridurre il traffico generato dal medesimo host (se non viene effettuato lo spoofing dell'indirizzo IP), ma non risolve il problema di richieste provenienti da host differenti; inoltre questa soluzione ha lo svantaggio di richiedere memoria aggiuntiva per mantenere lo stato dei contatori delle richieste.

Attacco di TCP-SYN flooding

L'attacco di *TCP-SYN flooding* [17], mostrato in Figura 4.6, consiste nell'iniziare molteplici fasi di TCP three-way handshake senza mai completarle (si parla di connessioni *half-open*, dal momento che la fase di apertura non è ancora completa).

Ogni volta che un host riceve un segmento di SYN per l'apertura di una connessione TCP, viene allocata in memoria una nuova struttura dati, chiamata *TCB (Trasmission Control Block)* [58], in cui sono memorizzate alcune informazioni riguardanti la connessione, come ad esempio l'indirizzo IP locale e remoto, la porta TCP locale e remota e il puntatore al buffer di memoria in cui sono memorizzati i byte ricevuti. Ad esempio in Linux, il TCB è chiamato *sock* ed ha una dimensione di 1300 byte. Il TCB resta in memoria fintanto che non scade il timeout tra la ricezione del segmento di SYN e di quello successivo di SYN-ACK, oppure essendo stato completato il three-way handshake, il TCB rimane in memoria fino a quando la connessione non termina. Se il server riceve un numero molto elevato di richieste di SYN a causa di un attacco di TCP-SYN flooding, alloca di conseguenza un numero elevato di TCB che può saturare lo spazio di memoria a disposizione del server (*overflow*) ed impedire la creazione di nuove connessioni TCP, anche

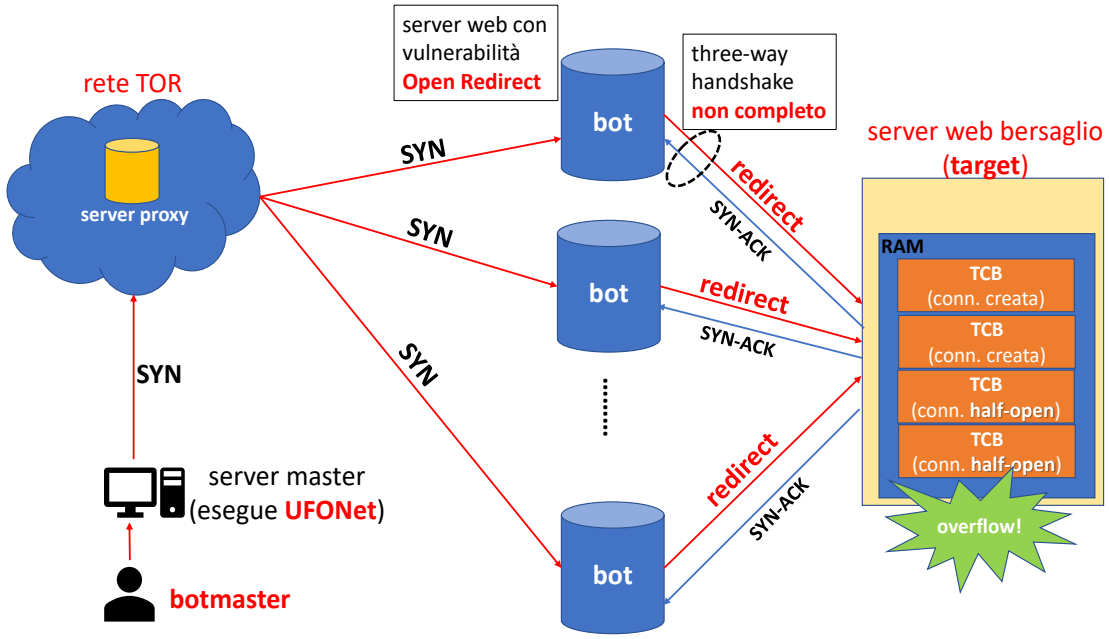

Figura 4.6: Esempio di attacco DDoS di TCP-SYN flooding eseguito tramite UFONet.

se legittime. Per evitare questa situazione, ogni sistema operativo pone un limite al numero di connessioni che è possibile aprire sulla medesima porta TCP. Tale limite può essere modificato dalle applicazioni, tramite il parametro *backlog* [58].

Alcuni metodi per mitigare questo tipo di attacco sono:

- diminuire il timeout per completare il three-way handshake. Questa soluzione può avere anche dei risvolti negativi nel caso di applicazioni client con poche risorse computazionali a disposizione, e pertanto non in grado di inviare il segmento di SYN-ACK entro la scadenza del timeout;
- aumentare il valore del parametro *backlog*. È necessario porre cautela nella modifica di tale parametro, altrimenti potrebbe verificarsi una situazione di overflow di memoria.

Attacco di Smurfing

L'attacco di *Smurfing* [17], mostrato in Figura 4.7, si articola nelle seguenti fasi:

1. il botmaster, tramite UFONet, crea un pacchetto ICMP Echo Request [59] con il campo indirizzo IP sorgente modificato con l'indirizzo IP del bersaglio dell'attacco;
2. il pacchetto ICMP viene inviato in broadcast alla botnet la quale, grazie alla vulnerabilità Open Redirect, farà una redirezione verso gli altri host della rete;
3. la rete costituita dagli host che ricevono la ICMP Echo request, è anche chiamata *riflettore*, poiché moltiplica il numero di attaccanti. Difatti, ogni host di tale rete, risponde con una ICMP Echo reply destinata all'indirizzo IP sorgente della richiesta, ovvero l'indirizzo IP dell'host bersaglio dell'attacco;
4. l'host bersaglio viene quindi inondato da un numero molto elevato di pacchetti ICMP Echo reply che possono saturare la sua banda e causare l'interruzione del servizio offerto.

Una possibile soluzione consiste nel configurare il router di frontiera che connette ad Internet la rete LAN del server, in modo da scartare qualsiasi pacchetto, avente indirizzo IP broadcast,

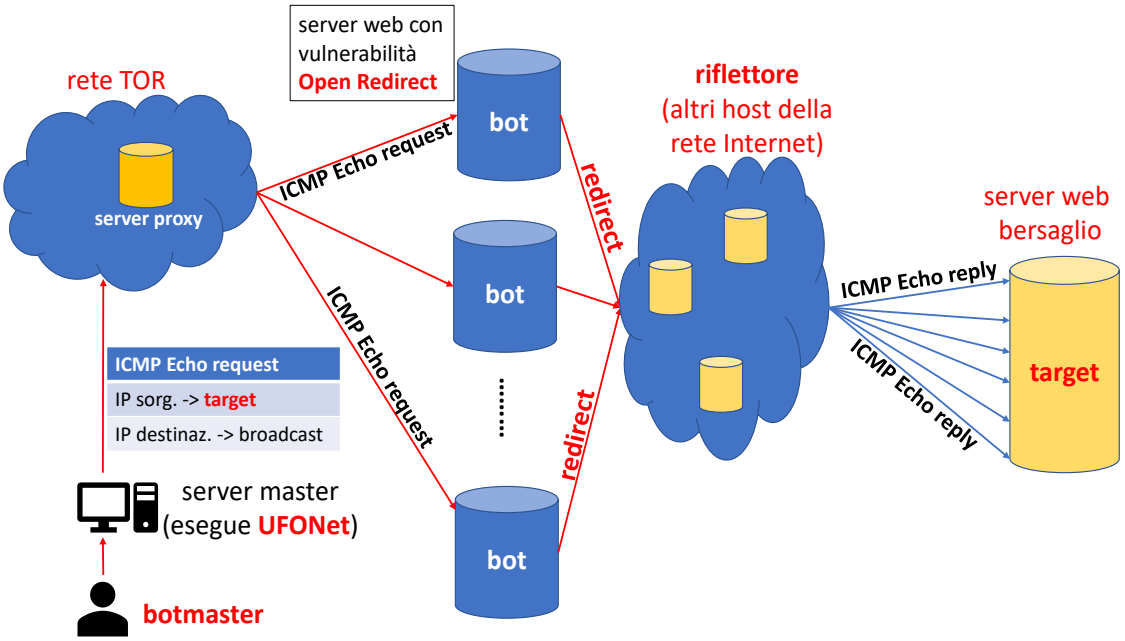

Figura 4.7: Esempio di attacco DDoS di Smurfing eseguito tramite UFONet.

proveniente dall'esterno della LAN. Difatti, il meccanismo del broadcast risulta essenziale all'interno di una rete locale (ad esempio per scoprire i servizi offerti da un host), ma è utilizzato quasi esclusivamente a scopi malevoli sulla rete Internet [60]. Con questa soluzione è possibile evitare attacchi di Smurfing solo nel caso in cui la botnet sia esterna alla LAN del server vittima; in caso contrario, una possibile soluzione consiste nel monitorare la rete locale e cercare di rilevare il traffico malevolo (ad esempio tramite tecniche di ML).

Capitolo 5

Tecniche di classificazione

Negli ultimi decenni si è assistito ad un aumento delle capacità computazionali dei dispositivi e ad un progresso tecnologico nella memorizzazione dei dati.

Oggi siamo in grado di raccogliere notevoli quantità di dati in brevi periodi di tempo: pertanto risulta davvero complesso analizzarli in maniera puntuale.

Per tali ragioni, negli ultimi decenni, la disciplina del Machine Learning (ML) ha acquisito un ruolo sempre più rilevante.

5.1 Classificazione supervisionata e non supervisionata

Il machine learning (ML) [61, 62, 63] è una disciplina che rappresenta il punto d'incontro tra il data mining e l'intelligenza artificiale (AI, Artificial Intelligence).

Il ML è costituito da un insieme di tecniche che hanno lo scopo di creare modelli tramite cui i computer possono apprendere, analizzando un insieme di dati in input. Il termine "apprendimento", in ML, assume il duplice significato di:

- acquisire la capacità di classificare nuovi dati, dopo aver analizzato un insieme di campioni d'esempio. In questa trattazione, l'insieme dei dati di partenza su cui sarà effettuata la successiva classificazione, è chiamato *data set*. Da questo punto di vista, il machine learning può essere considerato una branca del AI: si cerca di trasferire nel mondo dei computer la capacità degli esseri senzienti di imparare dall'esperienza pregressa;
- trovare correlazioni significative tra i dati all'interno di un data set di grandi dimensioni. Questo rappresenta il punto d'incontro tra il ML e il data mining: in particolare il data mining utilizza gli algoritmi di machine lerning per trovare pattern significativi tra i dati.

Le tecniche di classificazione possono essere distinte in due categorie:

- *tecniche di apprendimento supervisionato:* sono le tecniche che prevedono una fase di addestramento sui dati di cui si conosce la classe di appartenenza (da qui il termine "supervisionato"). Tale modello deve essere in grado, con un certo livello di accuratezza, di classificare dati mai osservati in precedenza. Un esempio di questa tecnica sono gli alberi di decisione (Sezione 5.5);
- *tecniche di apprendimento non supervisionato:* sono le tecniche di ML in cui non esiste una fase vera e propria di addestramento. A questa categoria appartengono le tecniche di clustering (ad esempio il k-means) [62, 63].

In questa tesi si tratteranno alcune tecniche di classificazione supervisionata per ottenere *classificatori binari*, ovvero classificatori i cui dati di ingresso appartengono solo a due classi.

Nella procedura di costruzione di un modello, è possibile distinguere tre fasi:

1. *fase di addestramento:* è la fase in cui, analizzando una porzione rilevante del data set chiamata *training set*, il modello apprende dall'esperienza come classificare in futuro osservazioni mai viste;

2. *fase di validazione:* è la fase in cui viene scelto il modello migliore tramite delle tecniche di selezione del modello approfondite nella Sezione 5.4;

3. *fase di test:* è la fase in cui, applicando il modello addestrato su una diversa porzione del data set chiamata *test set*, si valuta la bontà del modello utilizzando alcune statistiche (Sezione 5.2).

5.2 Metriche di valutazione del modello di classificazione

Esistono diverse metriche che possono essere usate per quantificare la bontà di un classificatore [64].

La *matrice di confusione* (Tabella 5.1) è un metodo compatto che permette di illustrare graficamente i risultati di una classificazione.

In questo contesto, un'osservazione è positiva se è classificata come appartenente alla classe a cui si è interessati. In caso contrario, si parla di osservazione negativa. Nel prosieguo della trattazione si supporrà che le classi positive siano quelle relative al traffico malevolo, mentre quelle negative siano quelle relative al traffico non malevolo.

		Classi predette	
Classi reali		veri negativi (TN)	falsi positivi (FP)
		falsi negativi (FN)	veri positivi (TP)

Tabella 5.1: Matrice di confusione per un classificatore binario.

In particolare, i termini che compaiono nella matrice sono:

- *veri positivi (TP, True Positives):* il numero di osservazioni positive correttamente classificate;
- *veri negativi (TN, True Negatives):* il numero di osservazioni negative correttamente classificate;
- *falsi positivi (FP, False Positives):* il numero di osservazioni positive che non sono state correttamente classificate;
- *falsi negativi (FN, False Negatives):* il numero di osservazioni negative che non sono state correttamente classificate.

La metrica più semplice per valutare la bontà di un classificatore è l'accuratezza (o accuracy):

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Equivalentemente, è possibile utilizzare il tasso di errore (o error rate): $error\ rate = 1 - accuracy$.

L'accuracy non è sempre la metrica di valutazione più appropriata. Ad esempio, nel caso in cui il data set contenga un numero esiguo di osservazioni positive rispetto al numero totale di osservazioni (problema anche noto come *class imbalance*), il modello potrebbe classificare correttamente la maggior parte delle osservazioni negative e non riuscire a classificare le poche osservazioni positive; pertanto una misura di accuratezza elevata non è rappresentativa della bontà del classificatore. In tali casi, è preferibile utilizzare le seguenti metriche:

precision	$\frac{TP}{TP+FP}$
recall	$\frac{TP}{TP+FN}$
F-score	$\frac{(\beta^2+1)\cdot TP}{(\beta^2+1)\cdot TP+\beta^2\cdot FN+FP}$
specificity	$\frac{TN}{TN+FP}$
FPR	$\frac{FP}{TN+FP}$
FNR	$\frac{FN}{TP+FN}$
AUC	$\frac{1}{2}\cdot(recall + specificity)$

Tabella 5.2: Statistiche per valutare la bontà di un classificatore.

- *precision*: misura la percentuale di osservazioni etichettate come positive rispetto a quelle che effettivamente lo sono;
- *recall (o sensitivity o True Positive Rate (TPR))*: misura l'efficacia del classificatore nell'identificare le osservazioni positive;
- *F-score*: è una misura che combina recall e precision. β è un numero reale non negativo che rappresenta il *peso* da assegnare alle misure. Solitamente viene scelto $\beta = 1$, ottenendo la media armonica di precision e recall [65];
- *specificity (o True Negative Rate, TNR)*: misura l'efficacia di un classificatore nell'identificare le osservazioni negative;
- *False Positive Rate (FPR)*: è la percentuale di osservazioni positive etichettate come negative;
- *False Negative Rate (FNR)*: è la percentuale di osservazioni negative etichettate come positive;
- *AUC*: misura l'area sottostante ad una curva ROC.

La curva ROC è una rappresentazione che permette di visualizzare graficamente la percentuale di osservazioni positive correttamente classificate (recall o TPR) e quelle classificate in modo errato (FPR). Un esempio di curve ROC appartenenti a due diversi classificatori è mostrato nella Figura 5.1.

In Figura 5.1 è anche rappresentato un classificatore che ha un comportamento *random guess*, ovvero classifica in egual misura ($TPR = FPR$) osservazioni positive in modo corretto ed in modo errato.

Un classificatore ideale ha $TPR = 1$ ed $FPR = 0$, ovvero deve classificare correttamente tutte le osservazioni positive. Nella Figura 5.1 si può notare che, per valori di $FPR < 0.4$, il TPR del $modello_1$ è superiore a quello del $modello_2$, mentre per valori di $FPR > 0.4$ la situazione si rovescia. Pertanto, osservando solo le curve ROC, non è possibile stabilire quale modello sia migliore.

In tali casi, si rivela molto utile la metrica AUC (Area Under Curve) che misura l'area sottostante a ciascuna curva ROC e rappresenta l'abilità di un classificatore nell'evitare classificazione errate. Un classificatore ideale possiede $AUC = 1$. Pertanto un modello è tanto più accurato quanto maggiore è la sua AUC.

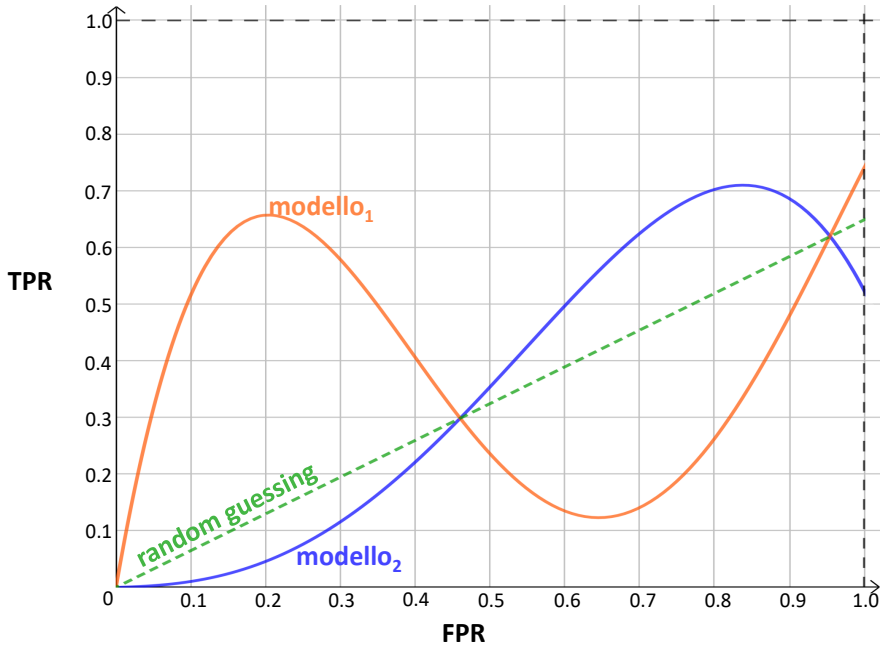

Figura 5.1: Esempio di curve ROC.

5.3 Il problema dell'overfitting e dell'underfitting

Tutte le tecniche di classificazione soffrono del problema dell'overfitting e dell'underfitting [61].

L'*overfitting* si verifica quando il modello risulta eccessivamente specializzato nel classificare le osservazioni del training set: su questa porzione di dati, la classificazione risulta molto accurata, mentre quando il modello viene applicato sul test set, il livello di accuratezza scende drasticamente. Tale fenomeno è particolarmente dannoso in caso di rumore nel training set (ad esempio osservazioni classificate erroneamente) e/o presenza di *outlier*, ovvero osservazioni aventi caratteristiche molto diverse rispetto alle altre. In tali casi potrebbe accadere che i dati rumorosi e/o gli outlier siano inclusi nel modello, compromettendone l'abilità di classificazione.

Con il termine *underfitting* si intende la situazione in cui il modello, essendo stato costruito osservando pochi dati, classifica in modo errato molte osservazioni di test. Per tale motivo è preferibile disporre di data set di ampie dimensioni.

Sia l'overfitting che l'underfitting sono correlati con la complessità del modello: modelli poco complessi possono essere soggetti ad underfitting, mentre modelli più complessi sono maggiormente predisposti al problema dell'overfitting. Ad esempio, negli alberi di decisione (Sezione 5.5), la complessità del modello è legata all'altezza dell'albero e quindi al numero dei nodi: alberi molto profondi sono più soggetti all'overfitting, mentre alberi di altezza ridotta sono più sensibili all'underfitting.

5.4 La selezione del modello

La selezione del modello migliore (o *model selection*) consiste sia nella scelta dell'algoritmo di ML tramite cui ottenere tale modello sia nella scelta dei suoi iperparametri.

5.4.1 Gli iperparametri e le curve di validazione

Gli iperparametri [66] sono parametri di input scelti a-priori prima della costruzione del modello. Ad esempio, nell'algoritmo k-NN (Sezione 5.7), il numero di vicini k rappresenta un iperparametro. Una delle cause dell'overfitting è la scelta poco accurata degli iperparametri. Nonostante

sia possibile applicare un approccio *brute force* per individuare gli iperparametri più opportuni, esistono alcune tecniche che permettono di raggiungere tale scopo con minore spreco di risorse computazionali. Una possibile tecnica è quella delle curve di validazione.

Una curva di validazione è una rappresentazione grafica che permette di confrontare le performance raggiunte dal modello durante la fase di addestramento e durante la fase validazione, al variare di un iperparametro.

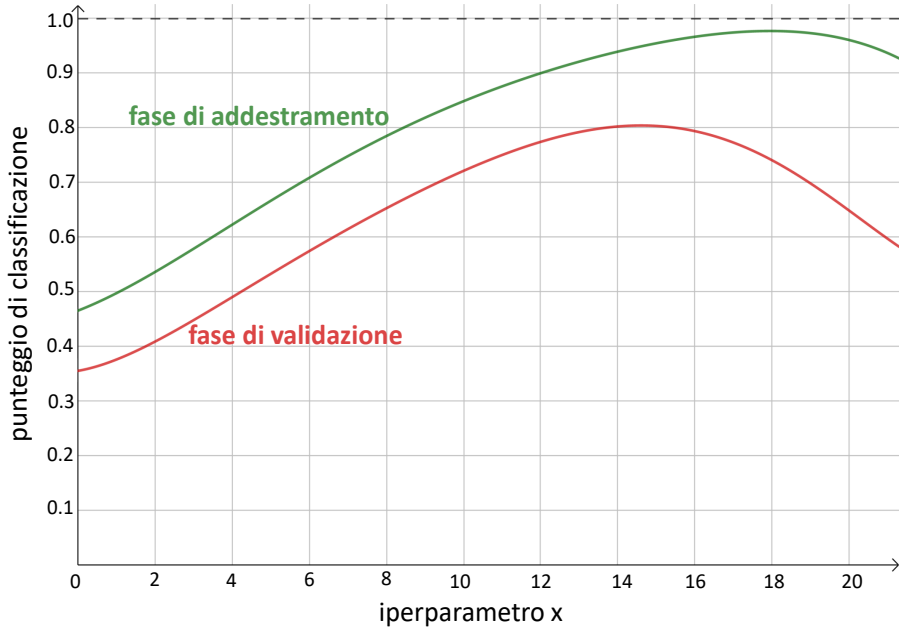

Figura 5.2: Esempio di curve di validazione.

Un esempio è mostrato nella Figura 5.2: la curva ottenuta durante la fase di validazione ottiene punteggi di classificazione crescenti fino al raggiungimento del valore $x = 14$. Da questo valore in poi, le performance del modello durante la fase di validazione decrescono mentre durante la fase di addestramento continuano a crescere. Questo è sintomo che il modello, per valori dell'iperparametro maggiori di 14, entra nella condizione di overfitting. Pertanto, in questo esempio, la scelta ottimale dell'iperparametro è $x = 14$, poiché per tale valore, la curva di validazione raggiunge un punteggio di classificazione elevato ed approssima abbastanza bene la curva di addestramento.

Il punteggio di classificazione presente sull'asse delle ordinate, può essere sostituito da una qualunque delle metriche descritte in precedenza per valutare la bontà di un classificatore.

5.4.2 La tecnica di k-fold cross-validation

La tecnica di *convalida incrociata con k partizioni* (o *k-fold cross-validation*) [67] è una delle tecniche più utilizzate per la selezione del modello perché è relativamente efficiente ed offre risultati stabili dal momento che rappresentano la media dei risultati ottenuti in diverse iterazioni dell'algoritmo (spiegato nel seguito). Questa tecnica cerca di massimizzare il numero di osservazioni usate per costruire il modello. In particolare essa garantisce che tutte le osservazioni siano utilizzate un egual numero di volte per il training set ed esattamente una sola volta per il test set.

L'algoritmo per calcolare la cross-validation con k partizioni (o *fold*) è il seguente:

1. si suddivide il training set iniziale in k partizioni;
2. si utilizzano $k - 1$ partizioni per il training set e la k-esima partizione come test set;
3. si calcola la statistica di interesse del modello quando è applicato al test set;

4. si seleziona, tra le $k-1$ partizioni, una diversa partizione per il test set. La partizione utilizzata in precedenza come test set, entra a fare parte del training set e non potrà essere selezionata nuovamente per essere utilizzata nella fase di test;

5. si iterano i passi precedenti fino a quando tutte le partizioni sono state utilizzate esattamente una volta come test set.

La statistica finale è ottenuta calcolando la media delle statistiche determinate ad ogni passo.

Solitamente viene utilizzato $k = 10$ [67], un valore ottenuto sperimentalmente che permette di ottenere risultati soddisfacenti nella maggior parte dei casi.

5.5 Alberi di decisione

Un albero di decisione [62, 63] è una struttura dati ad albero in cui:

- il nodo radice ed i nodi intermedi contengono gli attributi di decisione (o attributi di *splitting*), ovvero gli attributi che, durante il processo di costruzione dell'albero, sono utilizzati per partizionare il training set;
- i rami sono etichettati con il valore dell'attributo di splitting considerato per la partizione;
- le foglie contengono l'etichetta di classe assegnata alle osservazioni.

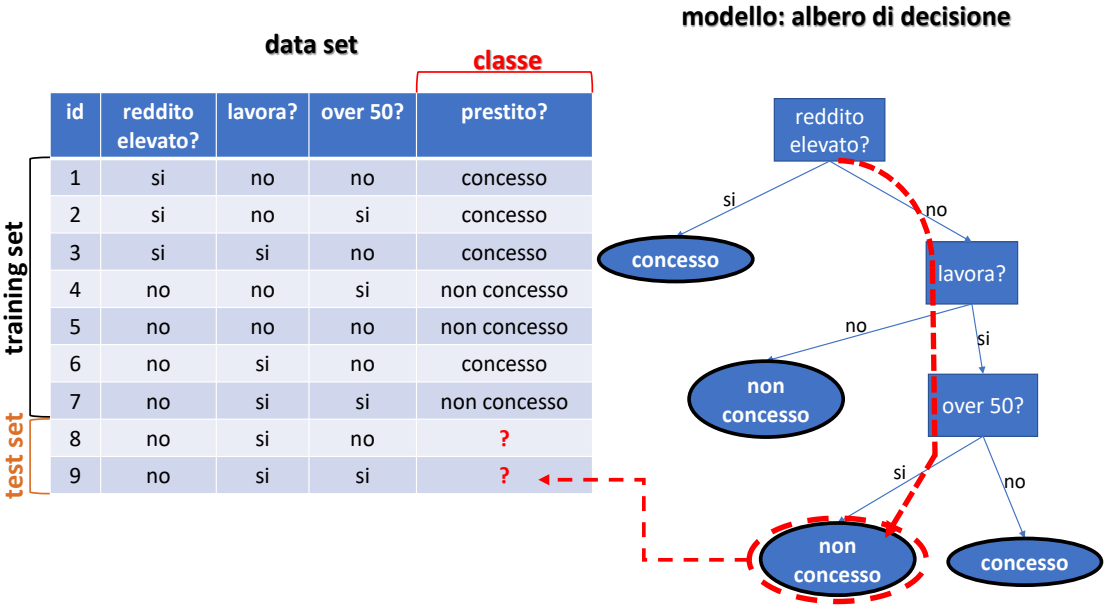

Figura 5.3: Esempio di albero di decisione.

Nella Figura 5.3 è mostrato un esempio di albero di decisione e di come avviene la classificazione di un'osservazione del test set. In particolare, la freccia tratteggiata indica i rami dell'albero da seguire in base al valore degli attributi di decisione fino al raggiungimento di un nodo foglia che rappresenta l'etichetta di classe da assegnare all'osservazione in esame.

La complessità dell'albero di decisione è legata alla sua altezza. Per limitare la complessità del modello ed evitare la condizione di overfitting, è possibile fare il *pruning* dell'albero, ovvero eliminare alcuni dei suoi rami.

Durante il corso degli anni sono stati sviluppati algoritmi differenti per costruire gli alberi di decisione. Nel seguito si farà riferimento al primo algoritmo, noto come algoritmo di Hunt, in onore del suo inventore.

5.5.1 Algoritmo di Hunt

L'algoritmo di Hunt [68] è un algoritmo greedy che consiste nel partizionare ricorsivamente il training set scegliendo ad ogni passo un diverso attributo di splitting. L'attributo scelto ad ogni iterazione è quello che consente di massimizzare l'*indice di purezza*, ovvero massimizzare l'omogeneità della distribuzione delle etichette di classe nelle partizioni ottenute.

L'algoritmo termina se si verifica una delle seguenti condizioni:

- la partizione contiene dati tutti con lo stesso valore di attributo di classe, ovvero ha indice di purezza massimo: l'etichetta di classe sarà il valore di tale attributo;
- il numero di attributi di splitting è terminato: l'etichetta di classe scelta è il valore dell'attributo di classe più frequente.

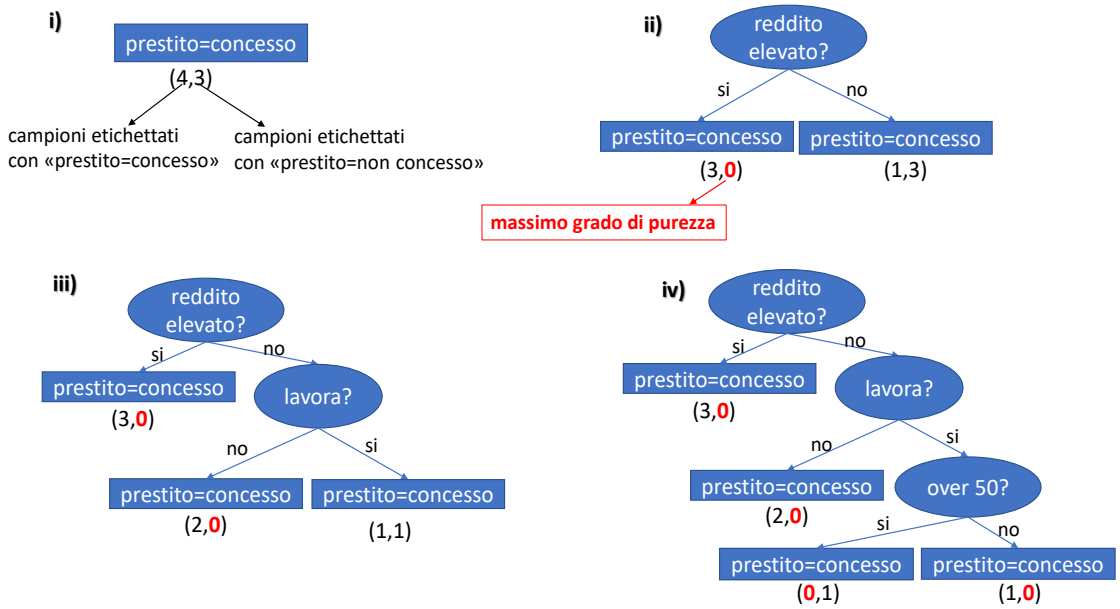

Figura 5.4: Esempio di costruzione di un albero di decisione.

In Figura 5.4 è mostrato un esempio di come opera l'algoritmo di Hunt applicato al training set mostrato nella Figura 5.3. I passi sono i seguenti:

i. all'inizio si ha un'unica partizione coincidente con l'intero training set. In essa quattro campioni sono etichettati con *prestito* = *concesso* e tre campioni sono etichettati con *prestito* = *non concesso*;

ii. selezionando come attributo di decisione *reddito elevato* = *si*, si ottiene una partizione contenente tutti i campioni etichettati con *prestito* = *concesso*, ovvero si ottiene una partizione con indice di purezza massimo. Pertanto la ricorsione, su tale ramo, termina e si ottiene una foglia dell'albero di decisione;

iii. la ricorsione continua sull'altro ramo in cui, al passo precedente, è stata ottenuta una partizione contenente un campione etichettato con *prestito* = *concesso* e tre campioni etichettati con *prestito* = *non concesso*. Scegliendo come attributo di decisione *lavora* = *no*, si ottiene una partizione con indice di purezza massimo; quindi la ricorsione termina su tale ramo e si ottiene un'altra foglia dell'albero di decisione;

iv. la ricorsione continua sull'altro ramo. L'algoritmo, in questo esempio, termina perché sia scegliendo come attributo di decisione *over* 50 = *si* che *over* 50 = *no*, si ottengono due partizioni pure.

Oggigiorno l'algoritmo di Hunt non è più utilizzato nella sua forma originale ma è alla base di diversi algoritmi di ML per la costruzione degli alberi di decisione [62].

5.6 Support Vector Machine (SVM)

Le support vector machine sono delle tecniche di classificazione documentate per la prima volta nel 1992, ad opera degli studiosi Vapnik, Boser e Guyon [69].

SVM cerca di classificare le osservazioni del data set trovando un *iperpiano* (coincidente con una retta nel caso in cui lo spazio degli attributi abbia dimensione due), che riesca a dividere i dati in due regioni distinte. Dal momento che esistono infiniti iperpiani, SVM cerca il *Maximum Marginal Hyperplane (MMH)*, ovvero l'iperpiano in grado di massimizzare l'ampiezza del margine, definito come la minima distanza tra osservazioni appartenenti a due classi diverse. Le osservazioni che definiscono il margine prendono il nome di vettori di supporto (o *support vector*).

5.6.1 SVM lineare

Qualora i dati siano linearmente separabili, ovvero il classificatore sia in grado di trovare un MMH che riesca a separarli perfettamente, si parla di *SVM lineare*. Un esempio è mostrato in Figura 5.5.

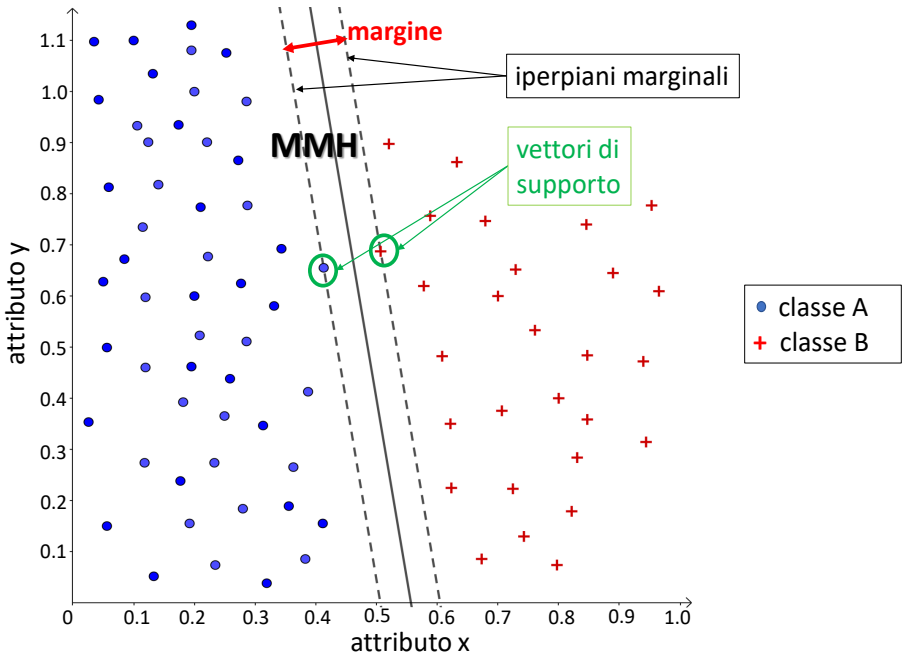

Figura 5.5: Esempio di SVM lineare nel caso di spazio degli attributi 2-D.

L'algoritmo di costruzione del modello SVM appartiene alla categoria degli algoritmi di ottimizzazione: individua i vettori di supporto in grado di garantire il margine più elevato per l'iperpiano MMH. Massimizzare il margine risulta conveniente al fine di ridurre la probabilità di overfitting: avendo a disposizione un margine più elevato, il modello ha più probabilità di classificare correttamente osservazioni di test che, pur appartenendo a classi diverse, risultano abbastanza vicine. La prossimità tra le osservazioni è una situazione che diventa sempre più frequente all'aumentare della dimensionalità, ovvero del numero di attributi delle osservazioni: difatti è stato coniato il termine *maledizione della dimensionalità* [62] per indicare che la complessità dei modelli di machine learning esplode in presenza di elevato numero di attributi.

5.6.2 SVM non lineare

In molti casi le osservazioni non sono linearmente separabili: è necessario creare un classificatore SVM non lineare. Un esempio è mostrato in Figura 5.6.

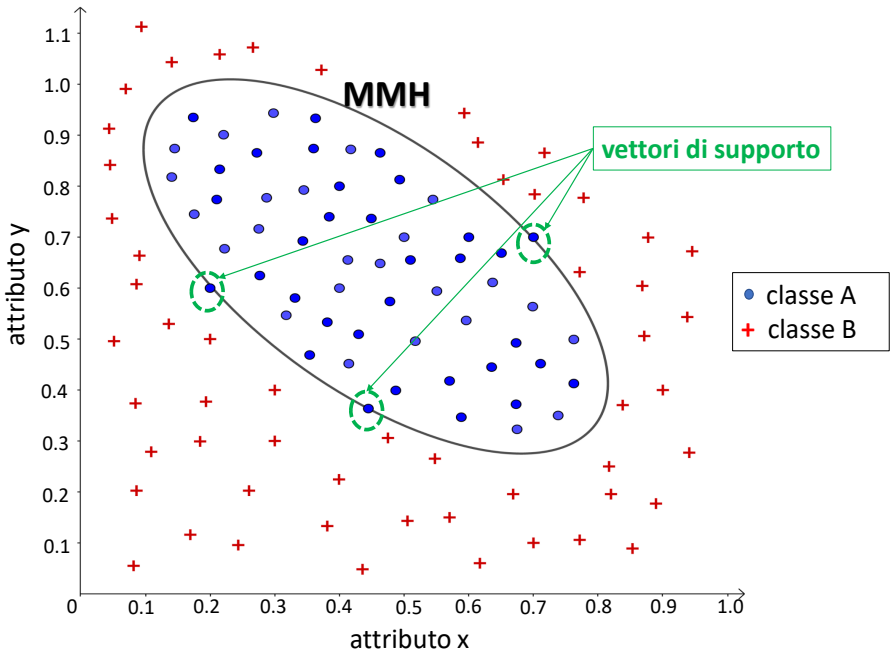

Figura 5.6: Esempio di SVM non lineare nel caso di spazio degli attributi 2-D.

La costruzione del modello avviene in due fasi:

1. si trasforma lo spazio degli attributi originario in uno spazio a dimensionalità più elevata, tramite l'utilizzo di funzioni non lineari;
2. in tale spazio è possibile separare linearmente i dati e quindi costruire un modello SVM lineare. Dal momento che si lavora in uno spazio ad elevata dimensionalità, per evitare che la complessità computazionale dell'algoritmo diventi troppo elevata, si utilizzano delle particolari funzioni chiamate *funzioni kernel*.

Le funzioni kernel sono degli iperparametri delle tecniche SVM, che si rivelano particolarmente utili nel caso di SVM non lineare. Esse permettono di eseguire l'operazione di prodotto scalare nello spazio degli attributi originario nonostante l'avvenuta trasformazione in uno spazio a più elevata dimensionalità. Le principali famiglie sono [70]:

- funzioni kernel polinomiali, espresse tramite la seguente formula:

$$K(x, y) = (\gamma \cdot x^T y + r)^d$$

  dove:

  - x e y sono i vettori degli attributi;
  - γ ed r sono dei parametri liberi;
  - d è il grado della funzione kernel polinomiale.

- funzioni kernel gaussiane o RBF (Radial Basis Function), espresse tramite la seguente formula:

$$K(x, y) = e^{\frac{-|x-y|^2}{2\sigma^2}}$$

  dove:

- x e y sono i vettori degli attributi;
- σ è un parametro libero.

È difficile sapere a-priori quale tipo di funzione kernel utilizzare per massimizzare le performance del classificatore: il modo più appropriato per effettuare tale scelta è procedere per tentativi ed osservare con quale funzione si ottengono i risultati migliori.

5.7 K-Nearest Neighbor (k-NN)

K-Nearest Neighbor (k-NN) è una tecnica di classificazione documentata per la prima volta in un articolo del 1951 [71].

Appartiene alla categoria dei classificatori *lazy*, ovvero pigri: questa tecnica non costruisce il modello a partire dal training set ma lo posticipa fino a quando non viene adoperata per classificare le osservazioni del test set.

K è un iperparametro del modello (l'unico se si considera la variante più semplice dell'algoritmo [72]) che indica il numero di osservazioni del training set "vicine" che il modello deve considerare per classificare una osservazione di test.

Il funzionamento del k-NN è il seguente:

1. l'utente sceglie l'iperparametro k;
2. k-NN calcola le distanze dell'osservazione da classificare con tutte le osservazioni del training set;
3. la tecnica seleziona le k osservazioni più vicine;
4. l'etichetta di classificazione assegnata è quella più frequente tra le osservazioni considerate.

La vicinanza tra le osservazioni può essere misurata in diversi modi. Una possibilità è utilizzare la formula di Minkowski:

$$distanza = (\sum_{k=1}^{n} |p_k - q_k|^r)^{1/r}$$

in cui:

- n è il numero di attributi;
- p_k e q_k sono rispettivamente il k-esimo attributo dell'osservazione p e dell'osservazione q;
- r è un parametro a scelta:
  - $r = 1$ rappresenta la misura di distanza chiamata *City block*;
  - $r = 2$ rappresenta la distanza euclidea;
  - $r \to \infty$ rappresenta la distanza chiamata *suprema*.

È difficile conoscere a-priori quale formula utilizzare per calcolare la distanza: pertanto è opportuno procedere per tentativi ed osservare con quale formula si ottiene il risultato più accurato.

5.8 Le foreste casuali

Le tecniche di combinazione consentono di migliorare l'accuratezza di classificazione combinando tra loro molteplici classificatori, a patto che essi siano debolmente correlati. Quest'ultimo aspetto permette di diminuire la varianza, ovvero la dipendenza del modello dal training set utilizzato e conseguentemente anche l'errore di classificazione. Creare modelli a partire dal medesimo training set che siano ragionevolmente indipendenti non è un problema di facile risoluzione. Pertanto sono nate nel corso del tempo diverse tecniche volte al raggiungimento di tale scopo, tra cui quella delle foreste casuali.

Le foreste casuali (o *random forest*) [73] sono una tecnica di combinazione di alberi di decisione. Per ottenere training set debolmente correlati e quindi aumentare l'accuratezza di classificazione, la foresta può utilizzare diverse tecniche tra cui il *bagging* che consiste nel partizionare il data set di partenza tramite campionamento con rimpiazzamento.

L'algoritmo per la creazione di una foresta di N alberi è il seguente (Figura 5.7):

1. viene utilizzata la tecnica di bagging per creare N training set di partenza;
2. a partire da ogni training set, viene addestrato un albero di decisione. Per ogni nodo dell'albero vengono scelti in modo casuale alcuni attributi di splitting;
3. nel momento in cui bisogna classificare i dati del test set, il valore dell'attributo di classe viene scelto a maggioranza tra i risultati ottenuti dai singoli alberi; ciò diminuisce la probabilità di incorrere nel problema dell'overfitting.

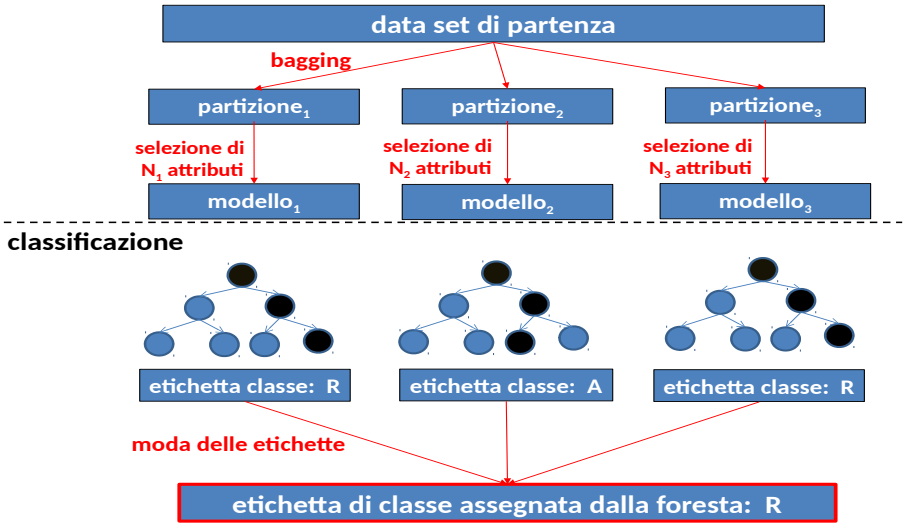

Figura 5.7: Algoritmo di costruzione di una foresta di alberi di decisione.

5.9 Confronto tra classificatori

Non esiste una tecnica di classificazione che risulti migliore delle altre in ogni situazione [72]. Per tale motivo è possibile fare solo delle considerazioni di carattere generale. Inoltre, il ML è una disciplina fortemente empirica: il modo più appropriato per sapere quale tecnica permette di raggiungere le migliori performance di classificazione e quali sono gli iperparametri più adatti per giungere a tale risultato, è quello di procedere per tentativi ed osservare con quale tecnica e con quali iperparametri si ottengono le prestazioni migliori.

K-NN, tra le tecniche descritte, è la più veloce nella fase di addestramento che consiste nella semplice memorizzazione del training set. Inoltre, avendo il solo iperparametro k, la fase di

selezione risulta relativamente semplice rispetto alle altre tecniche. Di contro, k-NN presenta i seguenti svantaggi:

- è la tecnica più lenta nella fase di classificazione poiché necessita di calcolare la distanza dell'osservazione di test da tutte le osservazioni del training set;
- essendo una tecnica lazy, k-NN occupa un notevole spazio in memoria sia nella fase di addestramento che durante la classificazione;
- è poco robusta nei confronti del rumore poiché i dati rumorosi possono falsare le misure di distanza.

Gli alberi di decisione risultano più facilmente interpretabili rispetto alle altre tecniche descritte: seguendo i percorsi dalla radice alle foglie, è abbastanza semplice capire perché una osservazione sia stata etichettata in un certo modo. Inoltre, gli alberi di decisione sono computazionalmente poco costosi da costruire rispetto ad altri modelli. Anche la classificazione di una nuova osservazione avviene rapidamente: la complessità è $O(h)$, dove h è l'altezza dell'albero. Di contro, questa tecnica è in grado di costruire un modello abbastanza accurato solo nel caso in cui lo spazio degli attributi contenga dati sufficientemente spaziati tra loro, tali da essere facilmente partizionabili da rette (anche chiamate *decision boundary*), parallele agli assi x e y (Figura 5.8). La condizione di parallelismo deriva dal fatto che l'algoritmo di costruzione del modello considera, per ogni nuovo nodo, un unico attributo di splitting.

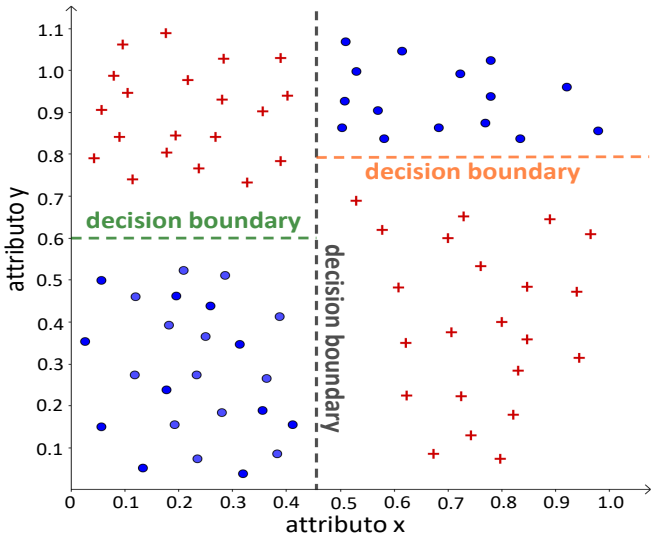

Figura 5.8: Esempio di decision boundary che partizionano correttamente il data set.

SVM permette di costruire il classificatore anche nel caso di attributi non linearmente separabili, al contrario di quanto avviene negli alberi di decisione. É inoltre possibile creare un modello abbastanza accurato pur avendo a disposizione un training set di dimensioni non elevate, poiché nella costruzione del modello vengono considerati solo i vettori di supporto, il cui numero è esiguo rispetto al numero di campioni del training set. Di contro, la tecnica in questione presenta i seguenti svantaggi:

- la fase di addestramento è lenta;
- il risultato della classificazione è scarsamente interpretabile;
- se il training set è troppo rumoroso o contenente molti outlier, è possibile che l'algoritmo SVM selezioni alcuni di questi campioni come vettori di supporto;
- SVM possiede un numero elevato di iperparametri. Per tale motivo, la fase di validazione del modello può risultare complessa.

Le foreste casuali permettono sia la diminuzione del grado di correlazione tra i singoli modelli di base sia una maggiore robustezza nei confronti dell'overfitting. Di contro, essendo costituite da un certo numero di classificatori di base, presentano i seguenti svantaggi:

- richiedono un notevole spazio di memorizzazione;
- sono lente da addestrare;
- i risultati della classificazione non sono facilmente interpretabili;
- il numero di iperparametri è elevato e quindi la scelta dei loro valori può risultare complessa.

Capitolo 6

Implementazione e risultati

Questo capitolo si divide in due parti principali. La prima parte riguarda la classificazione del traffico di cryptojacking (in particolare Stratum, descritto nella Sezione 3.2), mentre la seconda riguarda la classificazione del traffico generato dalle botnet, descritte nel Capitolo 4.

In Appendice D ed in Appendice E sono presenti, rispettivamente, il manuale dell'utente ed il manuale del programmatore con i relativi dettagli.

6.1 Ambiente di esecuzione e software utilizzati

La parte implementativa di questa tesi è stata effettuata su un calcolatore con le seguenti caratteristiche:

- *sistema operativo:* Ubuntu 18.04 LTS x64, versione kernel 4.15;
- *CPU:* Intel Core i5-3570 @ 3.40 GHz;
- *RAM:* 16 GB.

I software utilizzati sono stati:

- Tstat[1] 3.1.1;
- Tshark[2] 2.6.6;
- Matlab[3] R2018b;
- Python[4] 2.7.

6.2 Costruzione del data set

Il procedimento[5] per costruire il data set, mostrato in Figura 6.1, è il seguente:

[1] http://tstat.polito.it

[2] https://www.wireshark.org/docs/man-pages/tshark.html

[3] https://it.mathworks.com/products/matlab.html

[4] https://www.python.org/download/releases/2.7/

[5] Nella Appendice D.2 saranno descritti i passaggi pratici per ottenere tale data set.

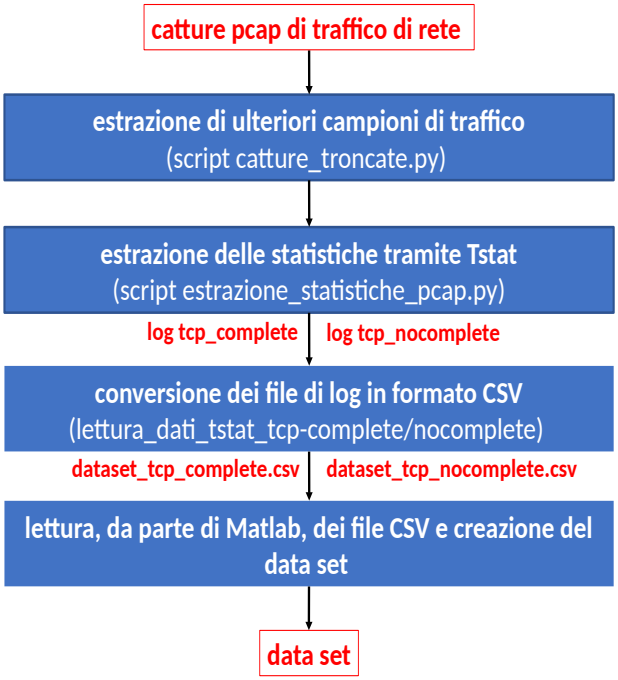

Figura 6.1: Flusso di lavoro seguito per la costruzione del data set.

1. il gruppo di ricerca TORSEC[6] ha fornito alcune catture di traffico legittimo e di traffico potenzialmente malevolo;

2. tramite lo script `catture_troncate.py`, dalle catture di traffico iniziale, sono stati estratti ulteriori campioni di traffico. A questo scopo è stato utilizzato lo strumento Tshark. Tramite un opportuno comando sono stati estratti i primi N secondi di pacchetti da ogni connessione TCP, dove N è un numero compreso tra 0.001 ed 1 secondo. Questo script è stato utilizzato per aumentare il numero di campioni di traffico a disposizione. Inoltre, è stato scelto proprio l'intervallo tra 0.001 ed 1 secondo, in modo che i classificatori avessero a disposizione più dati e dunque maggiore probabilità di individuare il traffico Stratum o il traffico di C&C generato dalle botnet, sin dai primi istanti di osservazione di tale traffico;

3. tramite lo script `estrazione_statistiche_pcap.py`, da ogni cattura di traffico sono state estratte, utilizzando lo strumento Tstat, le 31 statistiche che caratterizzano ogni campione presente nel data set. Nella documentazione Tstat sono utilizzate le seguenti convenzioni, riprese anche nell'ambito di questa tesi:

   - *C2S (Client-to-Server):* è il flusso TCP che dal client (ovvero l'host che per primo ha iniziato il three-way handshake) fluisce verso il server;
   - *S2C (Server-to-Client):* è il flusso TCP che dal server fluisce verso il client.

   Le statistiche utilizzate per l'addestramento dei classificatori sono le seguenti (# indica il numero della misura C2S e S2C all'interno della documentazione di Tstat):

   - *packets* (#3 e #17): numero totale di pacchetti IP della connessione;
   - *RST sent* (#4 e #18): è posto a 0 se non è stato inviato alcun segmento con flag RST impostato;
   - *ACK sent* (#5 e #19): numero di segmenti con flag ACK impostato;
   - *PURE ACK sent* (#6 e #20): numero di segmenti pure ACK, ovvero senza payload;
   - *unique bytes* (#7 e #21): numero di byte inviati all'interno dei payload;

[6] http://security.polito.it/

- *data pkts* (#8 e #22): numero di segmenti inviati, contenenti dati;
- *data bytes* (#9 e #23): numero di byte inviati nel payload, includendo le ritrasmissioni;
- *rexmit pkts* (#10 e #24): numero di segmenti ritrasmessi;
- *rexmit bytes* (#11 e #25): numero di byte ritrasmessi;
- *out seq pkts* (#12 e #26): numero di segmenti fuori sequenza;
- *SYN count* (#13 e #27): numero di segmenti con SYN impostato, incluse le ritrasmissioni;
- *FIN count* (#14 e #28): numero di segmenti con FIN impostato, incluse le ritrasmissioni;
- *completion time* (#31): durata di tempo in ms, intercorsa tra il primo e l'ultimo segmento del flusso;
- *client first payload* (#32): periodo di tempo, espresso in ms, intercorso tra il primo segmento del flusso ed il primo segmento con payload inviato dal client;
- *server first payload* (#33): periodo di tempo, espresso in ms, intercorso tra il primo segmento del flusso ed il primo segmento con payload inviato dal server;
- *client last payload* (#34): periodo di tempo, espresso in ms, intercorso tra il primo segmento del flusso e l'ultimo segmento con payload inviato dal client;
- *server last payload* (#35): periodo di tempo, espresso in ms, intercorso tra il primo segmento del flusso e l'ultimo segmento con payload inviato dal server;
- *client first ACK* (#36): periodo di tempo in ms, intercorso tra il primo segmento del flusso e il primo segmento con flag ACK settato (e flag SYN non impostato) inviato dal client;
- *server first ACK* (#37): periodo in ms, intercorso tra il primo segmento del flusso ed il primo segmento con flag ACK settato (e flag SYN non impostato) inviato dal server.

4. i file di log `tcp_complete` e `tcp_nocomplete` generati da Tstat, sono stati convertiti in file CSV, mediante l'utilizzo degli script `lettura_dati_tstat_tcp-complete.py` e `lettura_dati_tstat_tcp-nocomplete.py`. Questa operazione si è resa necessaria per produrre dei file in formato compatibile con Matlab. Mediante tali script, è stato anche aggiunto al data set l'attributo di classe *Malevolo* con le rispettive etichette:
   (a) etichetta *Y*, ovvero *Yes* (potenzialmente malevolo), per il traffico derivante dalle catture ottenute tramite XMR-Stak, MadoMiner, Coinhive, Pybot ed UFONet;
   (b) etichetta *N*, ovvero *No* (non malevolo), per il traffico derivante dalla navigazione web.
5. tramite lo script `estrazione_statistiche.m`, i file CSV sono stati letti da Matlab e fusi in un'unica struttura dati Matlab, chiamata `table`, che rappresenta il data set. Inoltre, questo script ha calcolato la media su ciascun attributo dei campioni, divisi nel gruppo dei campioni di traffico potenzialmente malevolo ed in quello dei campioni di traffico legittimo.

6.3 Classificazione di Stratum

In questa sezione, in primo luogo sarà descritto il data set utilizzato per la classificazione del traffico Stratum (Sezione 6.3.1). In seguito saranno approfonditi diversi casi:

- nella Sezione 6.3.2 saranno confrontati i classificatori addestrati su una porzione del data set contenente il traffico legittimo ed il traffico Stratum generato tramite XMR-Stak (software descritto nella Sezione 3.3);
- nella Sezione 6.3.3 saranno confrontati i classificatori addestrati su una porzione del data set contenente il traffico legittimo e il traffico malevolo generato da MadoMiner (descritto in precedenza nella Sezione 3.5);

- infine, nella Sezione 6.3.4 saranno confrontati i classificatori addestrati su una porzione del data set contenente il traffico legittimo ed il traffico Stratum, più precisamente websocket, generato mediante gli script di Coinhive (descritto in precedenza nella Sezione 3.4).

Da notare inoltre che, il traffico Stratum, tranne quello generato dal malware MadoMiner che è senza dubbio malevolo, può essere definito malevolo solo se il mining della criptovaluta non avviene volontariamente (l'utente decide di minare criptovaluta), oppure avviene senza il consenso dell'utente (es. script di Coinhive utilizzati in un sito senza il consenso dei visitatori).

6.3.1 Il data set

Il data set è costituito da 148848 campioni, così suddivisi[7]:

- *LG (LeGitimate):* 140343 campioni di traffico legittimo, estratti dalle catture di traffico generato durante la navigazione web con il browser Chrome (versione 48.0.2564 e 68.0.3440), il browser Firefox (versione 42.0 e 62.0) e il browser Edge (versione 42.17134). Tali browser sono stati installati in ambiente Linux e Windows (eccetto Edge disponibile solo per Windows);

- 8505 campioni di traffico potenzialmente malevolo, così suddivisi:

  - *XE (Xmr-Encrypted):* 6011 campioni sono stati estratti dal traffico cifrato generato dal software XMR-Stak. Il traffico in questione, riguarda il protocollo applicativo Stratum, cifrato mediante TLS;
  - *XU (Xmr-Unencrypted):* 1159 campioni sono stati generati dal traffico non cifrato, generato mediante XMR-Stak. Il traffico riguarda nuovamente il protocollo Stratum, ma questa volta trasportato in chiaro mediante TCP;
  - *MM (MadoMiner):* 701 campioni sono stati estratti dal traffico malevolo generato dal malware MadoMiner. Il traffico riguarda il protocollo Stratum trasportato in chiaro da TCP;
  - *CH (CoinHive):* 634 campioni sono stati estratti dal traffico generato durante la visita di siti contenenti gli script di Coinhive. Si tratta di traffico Stratum, in particolare websocket, cifrato mediante TLS.

Nella Tabella 6.1 sono riportate le medie delle statistiche[8] che maggiormente evidenziano le differenze tra il traffico legittimo e quello Stratum.

Dalla Tabella 6.1 si può notare che:

- il numero di byte del traffico legittimo è almeno un ordine di grandezza maggiore del numero di byte di Stratum. Ciò accade perché solitamente tra browser e client avviene uno scambio continuo di dati, anche di grandi dimensioni (ad es. immagini e video), mentre in Stratum, dopo che il miner si è autenticato al mining pool e quest'ultimo gli ha assegnato il job, le comunicazioni diventano più sporadiche (es. solo quando il miner termina il processo di mining contatta il mining pool per ricevere la ricompensa);

- per la stessa ragione spiegata in precedenza, il numero di pacchetti scambiati e di ACK inviati nel traffico legittimo è un ordine di grandezza maggiore di quelli del traffico Stratum;

[7] Per evitare di appesantire la trattazione, si utilizzerà la notazione abbreviata (LG, XE, XU, MM, CH) descritta nel seguito.

[8] Se non diversamente specificato, i valori riportati in tabella sono la media delle statistiche calcolate nella direzione C2S ed in quella S2C.

statistiche di traffico	*legittimo*	*Stratum*
numero pacchetti scambiati	54.00	6.25
numero di segmenti di ACK	53.50	5.74
numero di segmenti di pure ACK	20.46	2.28
numero di byte inviati nel payload	81651.50	2111.37
numero di byte ritrasmessi	255.26	2.19
durata del flusso in ms	4240.30	49370.00
tempo (ms), tra il primo e l'ultimo segmento (*C2S*)	2325.00	34338.00
tempo (ms), tra il primo e l'ultimo segmento (*S2C*)	2347.00	44716.00
tempo (ms), tra il primo segmento di dati e il primo ACK (*C2S*)	75.12	37.48
tempo (ms), tra il primo segmento di dati e il primo ACK (*S2C*)	151.52	61.64

Tabella 6.1: Statistiche medie più rilevanti del data set di Stratum.

- la durata media di un flusso Stratum è un ordine di grandezza maggiore rispetto a quella del traffico legittimo. Ciò accade perché normalmente le conversazioni client-server di traffico legittimo sono di breve durata (es. client contatta il server per dei file e quest'ultimo glieli invia), mentre in Stratum, dopo che il job è stato assegnato, il miner deve affrontare un lungo processo di mining al termine del quale fornisce la proof-of-work del lavoro svolto e riceve la ricompensa, chiudendo in seguito la connessione (se non vuole rimanere in attesa di nuovi job da parte del mining pool).

Nella Tabella 6.2 è riportato come sono stati divisi i campioni nel training set e nel test set, a seconda del traffico potenzialmente malevolo utilizzato per l'addestramento. Laddove viene mostrata una percentuale (ad esempio 70% e 30%) s'intende che, in modo casuale, tale percentuale di campioni è stata estratta dal data set originario per fare parte del training set o del test set.

traffico di addestramento	*training set*	*test set*
XMR-Stak	*n. campioni* = 103261	*n. campioni* = 44252
	70% LG + 70% XE + 70% XU	30% LG + 30% XE + 30% XU
(Sezione 6.3.2)	(98241 + 4208 + 812)	(42102 + 1803 + 347)
MadoMiner	*n. campioni* = 98732	*n. campioni* = 42312
	70% LG + 70% MM	30% LG + 30% MM
(Sezione 6.3.3)	(98241 + 491)	(42102 + 210)
Coinhive	*n. campioni* = 98685	*n. campioni* = 42292
	70% LG + 70% CH	30% LG + 30% CH
(Sezione 6.3.4)	(98241 + 444)	(42102 + 190)

Tabella 6.2: Divisione in training set e test set per i diversi casi di classificazione.

6.3.2 Classificazione del traffico generato tramite XMR-Stak

In questa sezione sono confrontati i classificatori addestrati su una porzione del traffico legittimo derivante dalla navigazione web e su una porzione del traffico Stratum generato da XMR-Stak.

La Tabella 6.3 riporta i risultati ottenuti dai classificatori durante l'addestramento e la validazione tramite la 10-fold cross-validation (indicata in tabella con *C.V.*). Inoltre, per ogni classificatore, sono anche riportati i tempi di classificazione. Per alcuni classificatori, sono stati scelti valori differenti di un iperparametro al fine di osservare la variazione del comportamento dei modelli. I classificatori in questione e gli iperparametri modificati sono stati i seguenti:

- per la foresta casuale è stato variato il numero di alberi;
- per k-NN è stato variato il numero di vicini k.

tecnica	*accuracy (%)*		*AUC (%)*		*F-score (%)*		*tempo (s)*
	train	C.V.	train	C.V.	train	C.V.	test
albero di decisione	99.90	99.76	99.95	99.19	99.00	97.51	0.02
foresta (20 alberi)	99.67	99.57	99.82	99.54	96.74	95.78	0.23
foresta (50 alberi)	99.57	99.58	99.77	99.51	95.78	95.81	0.55
foresta (100 alberi)	99.57	99.56	99.77	99.55	95.80	95.70	1.11
SVM gaussiano	99.81	98.02	99.86	81.61	98.11	75.64	74.27
k-NN (k=5)	99.79	99.67	99.06	98.57	97.86	96.59	23.48
k-NN (k=10)	99.67	99.62	98.44	98.28	96.64	96.08	25.60
k-NN (k=20)	99.57	99.51	98.13	98.00	95.63	95.02	23.63
k-NN (k=50)	99.36	99.30	97.29	97.16	93.48	92.94	26.04
k-NN (k=100)	99.11	99.06	96.75	96.74	91.17	90.71	25.59

Tabella 6.3: Confronto tra i classificatori di XMR-Stak.

Dalla Tabella 6.3 si può osservare come l'albero di decisione sia il classificatore che ha ottenuto i risultati migliori per tutte le statistiche su cui è stato effettuato il confronto. Per tale motivo esso è stato scelto (fase di *model selection*) come classificatore da utilizzare e pertanto sarà descritto nel seguito più in dettaglio. In alternativa, in un contesto di utilizzo diverso (ad esempio un ambiente di produzione), si sarebbe potuto valutare anche l'utilizzo di una foresta casuale, dal momento che, nonostante abbia raggiunto risultati leggermente inferiori all'albero di decisione, è per sua natura più resistente al rumore. Inoltre, è stato scartato dal confronto, perché in overfitting, il classificatore SVM gaussiano. Per quest'ultimo infatti si può notare che, nella fase di validazione si ottengono risultati inferiori rispetto a quelli ottenuti nella fase di addestramento: ad esempio, osservando la metrica F-score, nella validazione si ottiene un valore inferiore del 22.47% rispetto a quello ottenuto nell'addestramento.

Nella Tabella 6.4 sono riportate le statistiche di un albero di decisione calcolate sulla classificazione del training set, tramite la 10-fold cross-validation e su vari test set. Laddove nella tabella compare l'etichetta *test*, s'intende il test set generale (30% LG + 30% XE + 30% XU). Da notare che, per gli altri test effettuati (indicati in tabella con *altri test*) l'unica statistica calcolata è l'accuracy; ciò avviene dal momento che in questi set è presente solo una classe (tutti i campioni sono etichettati o come potenzialmente malevoli o come legittimi), e il calcolo delle altre statistiche perde di significato.

Dalla Tabella 6.4, si può osservare che l'albero di decisione raggiunge ottimi risultati, sempre al di sopra del 96%, sul test set generale. La matrice di confusione in Tabella 6.5 giustifica i

statistiche (%)	*train*	*C.V.*	*test*	*altri test*				
			(generale)	30% LG	30% XU	30% XE	MM	CH
accuracy	99.90	99.76	99.77	99.82	96.25	99.33	0.86	0.63
precision	98.03	96.47	96.46	-	-	-	-	-
recall	100.00	98.57	98.84	-	-	-	-	-
specificity	99.90	99.82	99.82	-	-	-	-	-
AUC	99.95	99.19	99.33	-	-	-	-	-
F-score	99.00	97.51	97.63	-	-	-	-	-

Tabella 6.4: Statistiche sull'albero di decisione, classificatore selezionato per XMR-Stak.

	Non malevolo	XMR-Stak
Non malevolo	$TN = 42024$	$FP = 78$
XMR-Stak	$FN = 25$	$TP = 2125$

Tabella 6.5: Matrice di confusione dell'albero di decisione, classificatore selezionato per XMR-Stak.

risultati ottenuti: il modello riesce a classificare correttamente la maggioranza delle osservazioni positive e di quelle negative. In questo contesto, un'osservazione è positiva se viene classificata come potenzialmente malevola, altrimenti è negativa.

Per quanto riguarda i restanti test set, il classificatore ottiene ottimi risultati nell'individuare il traffico legittimo (*30% LG*) e nell'individuare il traffico contenente il protocollo Stratum (generato da XMR-Stak) sia in chiaro (*30% XU*) che cifrato (*30% XE*). Di contro, il classificatore ottiene pessimi risultati sia nell'individuare il traffico generato da MadoMiner (*MM*) che quello generato da Coinhive (*CH*). Una possibile spiegazione è la seguente: dal momento che non esiste uno standard per il protocollo Stratum, esso può essere implementato in diverse varianti che non sono facilmente correlabili tra loro. Per questa ragione, il classificatore, nonostante sia stato addestrato tramite i campioni di traffico Stratum generato da XMR-Stak, non è in grado di rilevare correttamente nè il traffico Stratum in chiaro generato da MadoMiner nè il traffico Stratum cifrato generato da Coinhive.

Infine, in Figura 6.2 è riportato un grafico che rappresenta la variazione della AUC al variare del numero di pacchetti. Tramite esso si può comprendere quanto velocemente il modello riesca a classificare il traffico. Si può notare che l'AUC cresce all'aumentare del numero di pacchetti, sintomo che il modello migliora le sue capacità di classificazione all'aumentare dei dati a disposizione, fino ad avvicinarsi al 100%, a partire dal dodicesimo pacchetto.

6.3.3 Classificazione del traffico generato da MadoMiner

In questa sezione sono confrontati i classificatori addestrati su una porzione del traffico legittimo derivante dalla navigazione web e su una porzione del traffico malevolo generato dal malware MadoMiner.

In Tabella 6.6 sono confrontati i risultati raggiunti da diversi classificatori durante l'addestramento e la 10-fold cross-validation. Dalla tabella in questione si può osservare che, l'albero di decisione (come accaduto nel caso precedente) è il classificatore che ha raggiunto i risultati migliori per tutte le statistiche su cui è stato effettuato il confronto. Per tale motivo esso è stato scelto per effettuare la classificazione e pertanto sarà descritto nel seguito più in dettaglio. Dal confronto sono stati scartati i seguenti classificatori, poiché in overfitting:

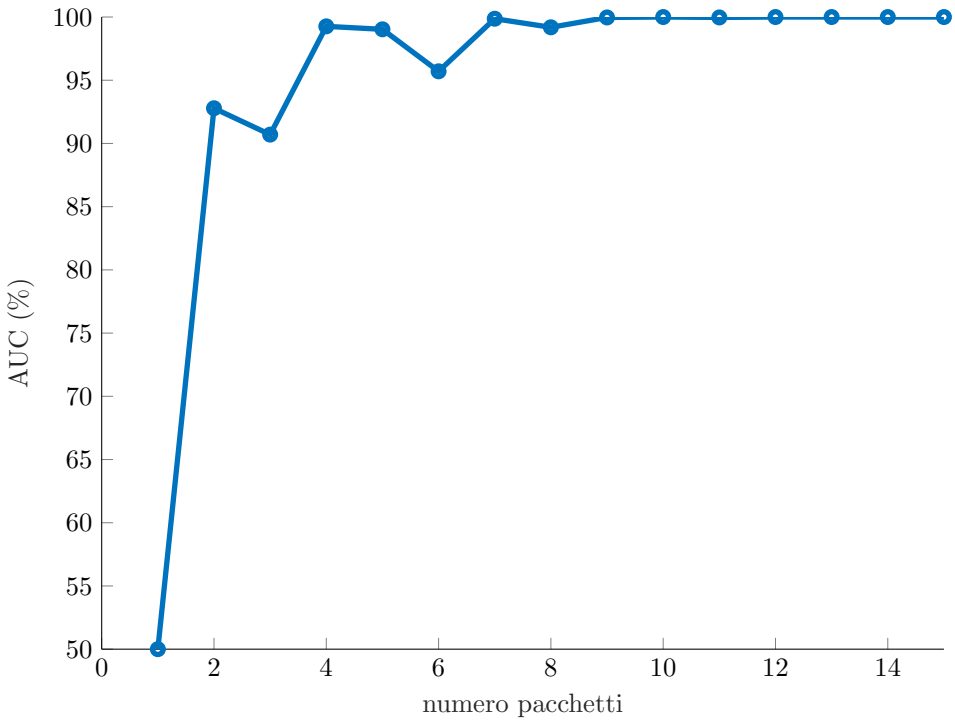

Figura 6.2: Andamento dell'AUC del classificatore di XMR-Stak al variare del numero di pacchetti scambiati.

- le foreste casuali (20 e 50 alberi): si osservi a tal proposito la differenza tra il valore di F-score durante l'addestramento e durante la validazione;
- k-NN con k=100: si osservi nuovamente il diverso valore di F-score durante l'addestramento e la validazione;
- SVM gaussiano: in questo caso l'overfitting è maggiormente accentuato. Difatti l'F-score in fase di validazione è del 48.94% inferiore rispetto a quello ottenuto nell'addestramento.

Nella Tabella 6.7 sono riportati i risultati ottenuti dal classificatore scelto, sia durante l'addestramento e la 10-fold cross-validation, sia durante la classificazione di diversi test set. Per la colonna *test* e le colonne *altri test* valgono le medesime considerazioni fatte nella sezione precedente. Dalla tabella in questione, si può osservare che l'albero di decisione raggiunge ottimi risultati, sempre al di sopra del 94%, sul test set generale. La matrice di confusione in Tabella 6.8 giustifica i risultati ottenuti: il modello riesce a classificare correttamente la maggioranza delle osservazioni positive e di quelle negative. Come nel caso precedente, le osservazioni positive sono quelle di traffico potenzialmente malevolo, mentre quelle negative si riferiscono al traffico legittimo.

Per quanto riguarda i restanti test set, è possibile fare le seguenti osservazioni: i risultati sul test set *30% LG* e sul test set *30% MM* sono ottimi, mentre i risultati su i restanti test set sono pessimi. Questo avvalora l'ipotesi, fatta nella sezione precedente, che le diverse implementazioni del protocollo Stratum influiscono negativamente sulla possibilità di classificare in modo corretto il traffico del protocollo in questione.

Infine, in Figura 6.3 è riportato un grafico che rappresenta la variazione della AUC al variare del numero di pacchetti. Come per il grafico precedente (Figura 6.2), l'AUC cresce fino a raggiungere un valore prossimo al 100%, a partire dall'ottavo pacchetto.

6.3.4 Classificazione del traffico generato da Coinhive

In questa sezione sono confrontati i classificatori addestrati su una porzione del traffico legittimo derivante dalla navigazione web e su una porzione del traffico Stratum generato durante la visita

tecnica	*accuracy (%)*		*AUC (%)*		*F-score (%)*		*tempo (s)*
	train	C.V.	train	C.V.	train	C.V.	test
albero di decisione	99.99	99.96	99.99	98.67	98.79	95.70	0.01
foresta (20 alberi)	99.96	99.76	99.88	98.38	95.70	80.32	0.24
foresta (50 alberi)	99.96	99.74	99.88	98.79	95.89	78.67	0.56
foresta (100 alberi)	99.80	99.75	99.90	98.76	83.43	79.33	1.15
SVM gaussiano	99.93	99.60	99.97	66.19	93.79	44.58	78.96
k-NN (k=5)	99.95	99.93	96.53	95.81	94.81	93.01	25.51
k-NN (k=10)	99.93	99.92	95.20	95.04	93.08	91.76	25.76
k-NN (k=20)	99.91	99.90	93.77	93.82	90.24	90.00	22.38
k-NN (k=50)	99.85	99.81	92.53	89.34	85.13	80.76	25.56
k-NN (k=100)	99.79	99.74	85.10	78.23	76.84	67.81	26.80

Tabella 6.6: Confronto tra i classificatori di MadoMiner.

statistiche (%)	*train*	*C.V.*	*test*	*altri test*				
			(generale)	30% LG	30% MM	XU	XE	CH
accuracy	99.99	99.96	99.96	99.97	98.57	25.80	0.02	0.16
precision	97.61	94.14	94.09	-	-	-	-	-
recall	100.00	97.37	98.57	-	-	-	-	-
specificity	99.99	99.97	99.97	-	-	-	-	-
AUC	99.99	98.67	99.27	-	-	-	-	-
F-score	98.79	95.70	96.28	-	-	-	-	-

Tabella 6.7: Statistiche sull'albero di decisione, classificatore selezionato per MadoMiner.

di siti web contenenti gli script di Coinhive.

In Tabella 6.9, sono confrontati i risultati raggiunti da i diversi classificatori durante l'addestramento e la 10-fold cross validation. Dalla tabella in questione si può osservare che, l'albero di decisione, come è avvenuto nei casi precedenti, è il classificatore che ha ottenuto i risultati migliori per tutte le statistiche su cui è stato effettuato il confronto. Per questo merita di essere descritto più in dettaglio nel seguito. Dal confronto sono stati scartati i seguenti classificatori, perchè in underfitting:

- le foreste casuali: si osservi a tal proposito il basso valore di F-score raggiunto sia nella fase di addestramento che in quella di validazione;
- k-NN con k uguale a 50 e 100: anche in questi casi, specialmente con $k = 100$, vengono raggiunti bassi valori di F-score sia in fase di addestramento che di validazione.

Inoltre, per quanto riguarda la situazione di overfitting, è stato scartato il classificatore SVM gaussiano. Difatti i valori di AUC ed F-score raggiunti in fase di validazione si discostano molto da quelli raggiunti in fase di addestramento.

	Non malevolo	MadoMiner
Non malevolo	$TN = 42089$	$FP = 13$
MadoMiner	$FN = 3$	$TP = 207$

Tabella 6.8: Matrice di confusione dell'albero di decisione, classificatore selezionato per MadoMiner.

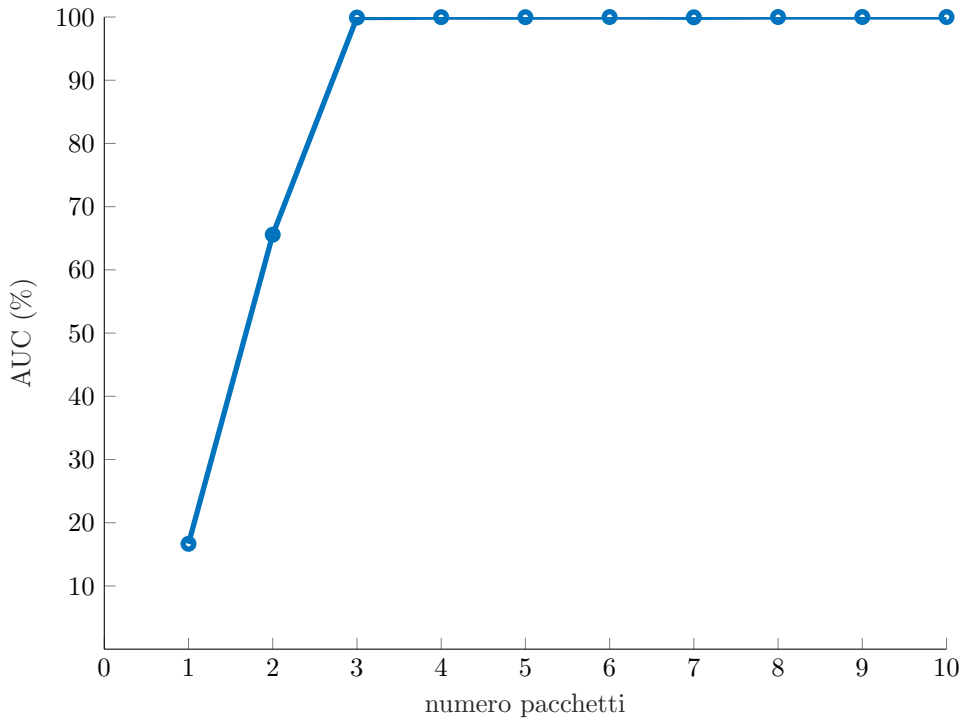

Figura 6.3: Andamento di AUC del classificatore di MadoMiner al variare del numero di pacchetti scambiati.

Nella Tabella 6.10 sono riportati i risultati ottenuti sia durante l'addestramento e la 10-fold cross-validation, sia durante la classificazione dei diversi test set.

Dalla Tabella 6.10, si può osservare che l'albero di decisione raggiunge sul test set generale un'ottima accuracy, specificity, AUC e recall (quest'ultima è di poco inferiore alle precedenti, ma comunque elevata). Di contro, la precision risulta particolarmente bassa rispetto alle altre statistiche, perché il numero di osservazioni false positive (FP) è confrontabile con quelle vere positive (TP), come è possibile anche osservare dalla matrice di confusione in Tabella 6.11. Di conseguenza anche la F-score[9] risulta inferiore alla media delle altre statistiche.

Per quanto riguarda i restanti test set, i risultati sul test set *30% LG* e sul test set *30% CH* sono abbastanza buoni, ovvero il classificatore riesce ad individuare correttamente la maggior parte dei campioni di traffico legittimo ed una buona parte dei campioni di traffico generato da Coinhive. Di contro, i risultati ottenuti su i restanti test set sono pessimi per la stessa ragione ipotizzata sia nel caso trattato nella Sezione 6.3.2 che in quello trattato nella Sezione 6.3.3, ovvero la mancanza di uno standard per il protocollo Stratum.

Infine, in Figura 6.4 è riportato il grafico che rappresenta la variazione dell'AUC al variare del numero di pacchetti. Si può notare come la statistica in questione cresca all'aumentare dei

[9]Si ricorda che la metrica F-score è la media tra precision e recall.

tecnica	*accuracy (%)*		*AUC (%)*		*F-score (%)*		*tempo (s)*
	train	C.V.	train	C.V.	train	C.V.	test
albero di decisione	99.90	99.75	99.95	91.04	89.70	74.46	0.01
foresta (20 alberi)	98.37	98.51	99.18	93.33	35.50	34.95	0.24
foresta (50 alberi)	98.21	98.46	99.10	93.49	33.50	34.36	0.57
foresta (100 alberi)	98.39	98.48	99.19	94.09	35.86	34.79	1.14
SVM gaussiano	99.72	99.31	99.86	55.23	76.16	11.82	80.13
k-NN (k=5)	99.81	99.74	85.78	82.76	77.28	69.28	25.80
k-NN (k=10)	99.77	99.73	81.50	79.98	70.71	65.94	25.39
k-NN (k=20)	99.72	99.68	78.22	76.62	64.11	59.82	25.75
k-NN (k=50)	99.64	99.63	69.89	66.24	50.00	43.45	25.85
k-NN (k=100)	99.58	99.54	55.51	51.36	18.99	4.94	28.08

Tabella 6.9: Confronto tra classificatori di Coinhive.

statistiche (%)	*train*	*C.V.*	*test*	*altri test*				
			(generale)	30% LG	30% CH	XU	XE	MM
accuracy	99.90	99.75	99.74	99.81	84.74	2.33	0.13	0.14
precision	81.32	68.21	66.53	-	-	-	-	-
recall	100.00	82.25	84.74	-	-	-	-	-
specificity	99.90	99.83	99.81	-	-	-	-	-
AUC	99.95	91.04	92.27	-	-	-	-	-
F-score	89.70	74.46	74.54	-	-	-	-	-

Tabella 6.10: Statistiche sull'albero di decisione, classificatore selezionato per Coinhive.

pacchetti, fino a raggiungere valori prossimi al 100%.

6.4 Classificazione del traffico generato dalle botnet

In questa sezione, in primo luogo sarà descritto il data set utilizzato per la classificazione del traffico di C&C generato dalle botnet (Sezione 6.4.1). In seguito saranno approfonditi diversi casi:

- nella Sezione 6.4.2 saranno confrontati i classificatori addestrati su una porzione del data set contenente il traffico legittimo e il traffico di C&C generato dalla botnet Pybot;
- nella Sezione 6.4.3 saranno confrontati i classificatori addestrati su una porzione del data set contenente il traffico legittimo e il traffico di C&C generato dalla botnet UFONet.

	Non malevolo	Coinhive
Non malevolo	$TN = 42021$	$FP = 81$
Coinhive	$FN = 29$	$TP = 161$

Tabella 6.11: Matrice di confusione dell'albero di decisione, classificatore selezionato per Coinhive.

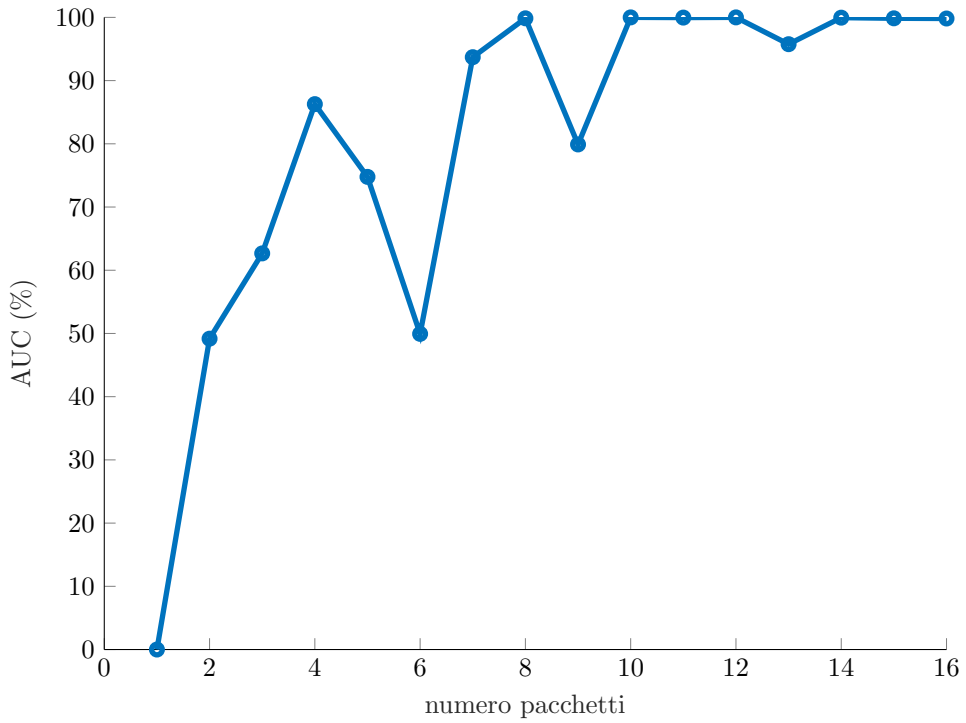

Figura 6.4: Andamento di AUC del classificatore di Coinhive al variare del numero di pacchetti scambiati.

6.4.1 I data set

I data set di Pybot ed UFONet sono costituiti rispettivamente da traffico legittimo (il data set *LG* descritto nella Sezione 6.3.1) e dal seguente traffico malevolo:

- *PY (PYbot):* 3197 campioni di traffico di C&C (descritto nel Capitolo 4), generato dalla botnet Pybot;
- *UF (UFonet):* 4820 campioni di traffico di C&C, generato dalla botnet UFONet.

Nella Tabella 6.12 sono riportate le medie delle statistiche[10] che maggiormente evidenziano le differenze tra il traffico legittimo e quello malevolo.

Dalla Tabella 6.12 si può notare che:

- il numero di byte inviati nel traffico legittimo è circa due ordini grandezza maggiore del numero di byte del traffico di C&C. Ciò accade perché, solitamente tra il browser e il server avviene uno scambio continuo di dati, anche di grandi dimensioni (ad es. immagini e video),

[10]Se non diversamente specificato, i valori riportati in tabella sono la media delle statistiche calcolate nella direzione C2S ed in quella S2C.

statistiche di traffico	*LG*	*PY*	*UF*
numero pacchetti scambiati	54.01	4.59	5.63
numero di segmenti di ACK	53.51	4.09	5.11
numero di segmenti di pure ACK	20.46	1.53	1.90
numero di byte inviati nel payload	40826.00	21.99	951.30
numero di byte ritrasmessi	255.26	0.11	4.74
durata del flusso in ms	4240.30	1406.70	235.47
tempo (ms), tra il primo e l'ultimo segmento (*C2S*)	2325.20	40.13	196.48
tempo (ms), tra il primo e l'ultimo segmento (*S2C*)	2347.10	40.14	208.22
tempo (ms), tra primo segmento di dati e primo ACK (*C2S*)	2347.10	29.47	12.59
tempo (ms), tra primo segmento di dati e primo ACK (*S2C*)	75.12	18.60	85.50

Tabella 6.12: Statistiche medie più rilevanti dei data set delle botnet.

mentre nelle botnet, dopo che l'attaccante ha comunicato le istruzioni ai bot, solitamente la connessione viene chiusa per evitare che l'autore dell'attacco venga rintracciato;

- per la stessa ragione spiegata in precedenza, il numero di pacchetti scambiati e di ACK inviati nel traffico legittimo è un ordine di grandezza maggiore di quelli del traffico di C&C delle botnet;
- la stessa spiegazione può essere data anche nel caso della durata di un flusso di C&C, in media un ordine di grandezza inferiore rispetto a quella del traffico legittimo.

Nella Tabella 6.13 è riportato come sono stati divisi i campioni nel training set e nel test set, a seconda del traffico malevolo utilizzato per l'addestramento. Laddove viene mostrata una percentuale (ad esempio 70% e 30%) s'intende che, in modo casuale, tale percentuale di campioni è stata estratta dal data set originario per fare parte del training set o del test set.

traffico di addestramento	*training set*	*test set*
Pybot	*n. campioni* = 100479	*n. campioni* = 43061
	70% LG + 70% PY	30% LG + 30% PY
(Sezione 6.4.2)	(98241 + 2238)	(42102 + 959)
UFONet	*n. campioni* = 101615	*n. campioni* = 43548
	70% LG + 70% UF	30% LG + 30% UF
(Sezione 6.4.3)	(98241 + 3374)	(42102 + 1446)

Tabella 6.13: Divisione in training set e test set per i diversi casi di classificazione.

6.4.2 Classificazione del traffico generato da Pybot

In questa sezione sono confrontati i classificatori addestrati su una porzione del traffico legittimo derivante dalla navigazione web e su una porzione del traffico malevolo generato da Pybot.

La Tabella 6.14 riporta i risultati ottenuti dai classificatori durante l'addestramento e la validazione tramite la 10-fold cross-validation (indicata in tabella con *C.V.*). Dalla tabella in questione si può osservare che l'albero di decisione raggiunge l'AUC più elevata, mentre k-NN ($k = 5$ e $k = 10$) ottengono un risultato migliore nella statistica F-score. La scelta del classificatore migliore ricade comunque sull'albero di decisione, poiché ha un tempo di classificazione di due ordini di grandezza inferiore rispetto a k-NN. Dal confronto è stato scartato il classificare SVM gaussiano poiché in overfitting. A tal proposito si osservi che il valore di F-score raggiunto in fase di validazione, è circa il 20% inferiore rispetto a quello raggiunto in fase di addestramento.

tecnica	*accuracy (%)*		*AUC (%)*		*F-score (%)*		*tempo (s)*
	train	C.V.	train	C.V.	train	C.V.	test
albero di decisione	99.95	99.87	99.97	99.17	98.85	97.17	0.01
foresta (20 alberi)	99.78	99.74	99.41	99.36	95.29	94.36	0.20
foresta (50 alberi)	99.78	99.75	99.41	99.39	95.27	94.72	0.50
foresta (100 alberi)	99.78	99.74	99.50	99.39	95.26	94.33	0.91
SVM gaussiano	99.66	98.91	99.83	82.47	92.94	72.58	69.85
k-NN (k=5)	99.95	99.95	98.99	98.97	98.90	98.78	22.97
k-NN (k=10)	99.95	99.94	99.06	98.95	98.96	98.68	23.40
k-NN (k=20)	99.93	99.93	98.52	98.50	98.37	98.30	21.59
k-NN (k=50)	99.92	99.92	98.48	98.46	98.19	98.08	23.45
k-NN (k=100)	99.86	99.84	98.31	98.25	96.76	96.48	23.53

Tabella 6.14: Confronto tra i classificatori di Pybot.

Nella Tabella 6.15 sono riportati i risultati ottenuti dal classificatore scelto, sia durante l'addestramento e la 10-fold cross-validation, sia durante la classificazione dei diversi test set. In particolare, per giustificare i risultati ottenuti in fase di test, nella Tabella 6.15 è riportata la matrice di confusione. Le statistiche di classificazione così elevate, sono giustificate dal fatto che, al contrario di quanto normalmente avviene nel traffico web, Pybot invia pochi dati e per brevi periodi di tempo; è possibile affermare ciò poiché lo strumento in questione trasmette del testo mediante TCP non cifrato.

statistiche (%)	*train*	*C.V.*	*test*	*altri test*	
			(generale)	30% LG	30% PY
accuracy	99.95	99.87	99.87	99.90	98.54
precision	97.73	95.97	95.55	-	-
recall	100	98.44	98.54	-	-
specificity	99.95	99.91	99.90	-	-
AUC	99.97	99.17	99.22	-	-
F-score	98.85	97.17	97.02	-	-

Tabella 6.15: Statistiche sull'albero di decisione, classificatore selezionato per Pybot.

Infine, in Figura 6.5 è riportato il grafico che rappresenta la variazione della AUC al variare del numero di pacchetti. Come per i grafici precedenti, si può osservare la crescita del valore di AUC

	Non malevolo	Pybot
Non malevolo	$TN = 42058$	$FP = 44$
Pybot	$FN = 14$	$TP = 945$

Tabella 6.16: Matrice di confusione dell'albero di decisione, classificatore selezionato per Pybot.

all'aumentare del numero di pacchetti, sintomo del miglioramento dell'abilità di classificazione.

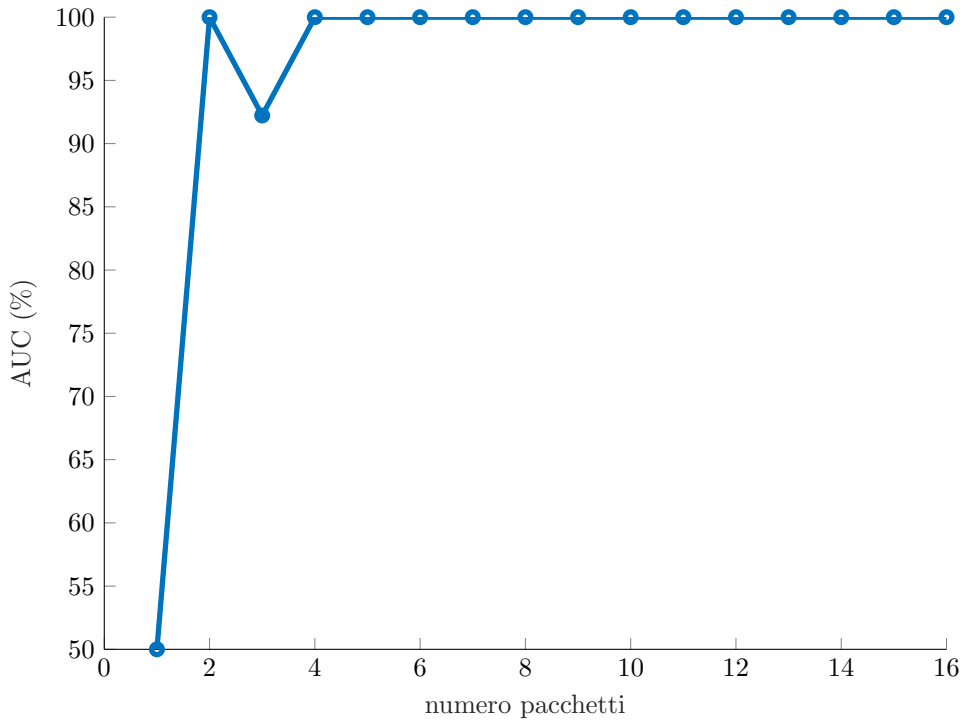

Figura 6.5: Andamento di AUC del classificatore di Pybot al variare del numero di pacchetti scambiati.

6.4.3 Classificazione del traffico generato da UFONet

In questa sezione sono confrontati i classificatori addestrati su una porzione del traffico legittimo derivante dalla navigazione web e su una porzione del traffico malevolo generato da UFONet.

La Tabella 6.17 riporta i risultati ottenuti dai classificatori durante l'addestramento e la validazione tramite la 10-fold cross-validation (indicata in tabella con *C.V.*). Dalla tabella in questione si può osservare che l'albero di decisione, foreste casuali e k-NN ottengono risultati prossimi tra loro. Dovendo selezionare un classificatore, la scelta ricade sull'albero di decisione, dal momento che è il più veloce nella classificazione ed inoltre richiede meno spazio in memoria.

Nella Tabella 6.18 sono riportati i risultati ottenuti dal classificatore scelto, sia durante l'addestramento e la 10-fold cross-validation, sia durante la classificazione dei diversi test set. Inoltre, per giustificare i risultati ottenuti in fase di test, nella Tabella 6.19 è riportata la matrice di confusione. Anche in questo caso, come è accaduto per Pybot, il classificatore riesce ad individuare correttamente la maggior parte dei campioni di traffico cifrato prodotti da UFONet. La ragione potrebbe risiedere nel fatto che tale strumento invia pochi dati e per brevi periodi di tempo, a differenza di quanto solitamente accade nel traffico web.

Infine, in Figura 6.6 è riportato il grafico che rappresenta la variazione della AUC al variare del numero di pacchetti. Come per i grafici precedenti, si può osservare la crescita del valore di

tecnica	*accuracy (%)*		*AUC (%)*		*F-score (%)*		*tempo (s)*
	train	C.V.	train	C.V.	train	C.V.	test
albero di decisione	99.96	99.88	99.98	99.43	99.32	98.20	0.02
foresta (20 alberi)	99.77	99.78	99.83	99.78	96.69	96.72	0.21
foresta (50 alberi)	99.78	99.78	99.83	99.79	96.73	96.73	0.50
foresta (100 alberi)	99.77	99.77	99.81	99.79	96.60	96.56	1.00
SVM gaussiano	99.77	97.31	99.88	62.63	96.58	38.60	84.21
k-NN (k=5)	99.94	99.91	99.37	99.19	99.06	98.59	23.18
k-NN (k=10)	99.90	99.87	99.03	98.93	98.48	97.97	23.43
k-NN (k=20)	99.87	99.85	99.07	98.90	97.99	97.76	22.14
k-NN (k=50)	99.82	99.82	98.85	98.87	97.29	97.24	23.31
k-NN (k=100)	99.77	99.76	98.81	98.78	96.62	96.46	24.00

Tabella 6.17: Confronto tra i classificatori di UFONet.

statistiche (%)	*train*	*C.V.*	*test*	*altri test*	
			(generale)	30% LG	30% UF
accuracy	99.96	99.88	99.90	99.91	99.24
precision	98.66	97.45	97.69	-	-
recall	100.00	98.96	99.24	-	-
specificity	99.95	99.91	99.92	-	-
AUC	99.98	99.43	99.58	-	-
F-score	99.32	98.20	98.46	-	-

Tabella 6.18: Statistiche sull'albero di decisione, classificatore selezionato per UFONet.

AUC fino al raggiungimento in modo stabile di valori prossimi al 100%, a partire dall'undicesimo pacchetto.

	Non malevolo	UFONet
Non malevolo	$TN = 42068$	$FP = 34$
UFONet	$FN = 11$	$TP = 1435$

Tabella 6.19: Matrice di confusione dell'albero di decisione, classificatore selezionato per UFONet.

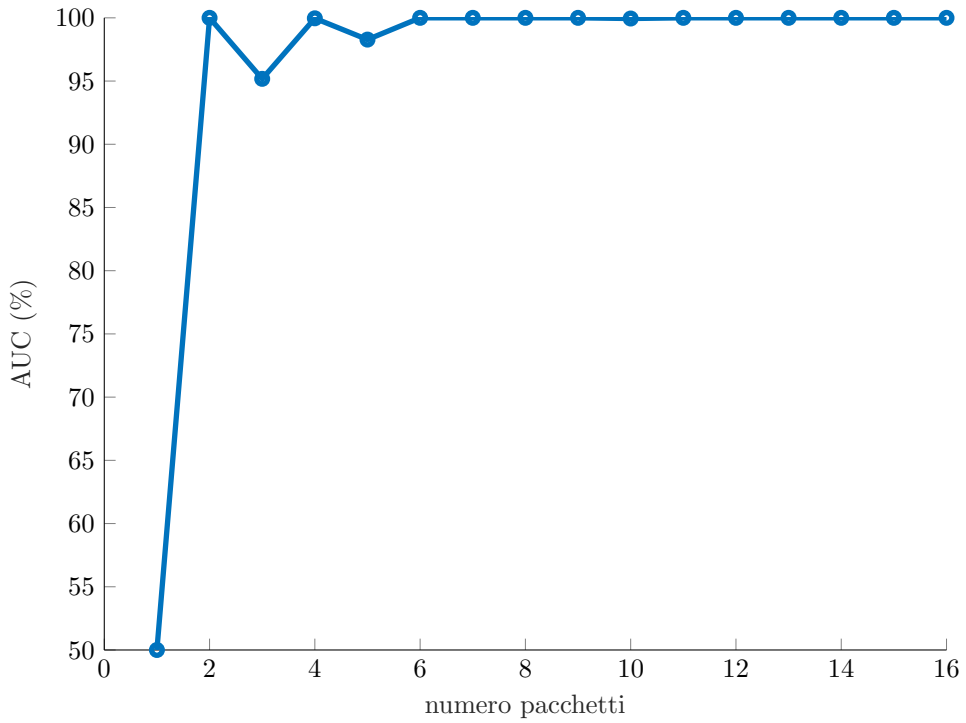

Figura 6.6: Andamento di AUC del classificatore UFONet al variare del numero di pacchetti scambiati.

Capitolo 7

Lavori collegati

In questo capitolo verrà descritto lo stato dell'arte degli argomenti su cui verte questo lavoro di tesi, ovvero l'applicazione di tecniche di machine learning per i seguenti scopi:

- per l'individuazione di traffico di cryptojacking (Sezione 7.1);
- per l'individuazione di traffico generato da botnet (Sezione 7.2);
- altre applicazioni nell'ambito della sicurezza informatica (Sezione 7.3).

7.1 Lavori collegati riguardanti il cryptojacking

L'esplosione del fenomeno delle criptovalute e la nascita di monete digitali in grado di essere minate su hardware comune hanno permesso, in questi ultimi anni, la diffusione del cryptojacking (descritto nel Capitolo 3). Per tale motivo, diversi lavori di ricerca sono stati condotti in tale ambito. In questa sezione saranno analizzati i lavori ritenuti più interessanti riguardanti il rilevamento, tramite tecniche di ML, del traffico di cryptojacking.

Iqbal *et al.* [74] hanno sviluppato un ad-blocker chiamato *AdGraph*, che utilizza la tecnica delle foreste casuali per individuare la pubblicità on-line indesiderata e/o utilizzata per scopi illeciti (ad esempio cryptojacking) e bloccarla. Come descritto nella Sezione 3.4.1, la pubblicità on-line è un potenziale vettore di attacco di cryptojacking. Il funzionamento di AdGraph può essere riassunto nelle seguenti fasi:

1. AdGraph, utilizzando una versione modificata del browser Chromium[1], estrae durante il caricamento della pagina web, informazioni su HTML (ad esempio gli elementi che costituiscono l'albero *DOM*[2][3] della pagina web) e Javascript (ad esempio le interazioni tra gli script Javascript e gli elementi HTML);
2. le informazioni estratte sono utilizzate per caratterizzare i nodi di una struttura dati a grafo (da qui deriva il nome *AdGraph*). Gli archi del grafo mettono in evidenza le relazioni esistenti tra i diversi nodi (descritti in seguito), come ad esempio le relazioni padre-figlio tra gli elementi HTML e quali script Javascript interagiscono con un determinato elemento HMTL. Nel grafo, sono presenti i seguenti tipi di nodi:

[1] https://www.chromium.org/

[2] https://www.w3schools.com/js/js_htmldom.asp

[3] Il DOM (Document Object Model) è un modello ad albero che rappresenta gli elementi HTML come oggetti, definendone i metodi per accedervi, le proprietà e gli eventi a cui possono essere soggetti.

- nodi HTML, ovvero caratterizzati da informazioni estratte dal codice HTML. Ad esempio i nodi *HTML image* e *HTML style* si riferiscono rispettivamente a porzioni di codice HTML contenenti il tag `<img>` e il tag `<style>`;
- nodi Javascript, ovvero caratterizzati dalle interazioni, registrate durante il caricamento della pagina web, tra gli elementi HTML dell'albero DOM e gli script Javascript (ad esempio l'accesso, da parte di uno script Javascript, ad un elemento HTML tramite il metodo `getElementById`);
- nodi HTTP URL, ovvero caratterizzati dagli URL estratti dagli script Javascript e dagli elementi HTML.

3. a partire dal grafo, sono estratti i seguenti attributi che caratterizzano ciascun campione del data set:
   - attributi che riguardano i nodi del grafo come per esempio il numero di archi entranti ed uscenti da un nodo ed il numero di nodi raggiungibili da un altro nodo;
   - attributi che riguardano le connessioni interne al grafo come per esempio la distanza massima di un nodo dagli altri e la media degli archi uscenti o entranti nei nodi vicini ad un determinato nodo;
   - attributi URL, ovvero riguardanti le proprietà degli URL associati ad un determinato nodo;
   - attributi relativi a parole chiave contenute negli URL (ad esempio "advertise", "banner", eccetera).
4. una parte dei campioni del data set è stata utilizzata per addestrare la foresta casuale ed in seguito validarla tramite la 10-fold cross-validation. La foresta casuale, quindi, è utilizzata per classificare i nodi malevoli, ovvero contenenti pubblicità indesiderata e/o a scopo di cryptojacking, ed i nodi legittimi.

AdGraph è riuscito ad ottenere ottimi risultati nella classificazione, raggiungendo un'accuracy del 97.7%, una precision del 83.0%, una recall del 81.9% ed un'AUC pari al 98.0%.

Al contrario di quanto è avvenuto in questo lavoro di tesi, AdGraph utilizza la DPI per estrarre le informazioni per l'addestramento del classificatore. Questa pratica, come descritto nel Capitolo 1, nella maggior parte delle situazioni, rappresenta una violazione della privacy ed inoltre non è applicabile al traffico cifrato.

Rodriguez *et al.* [75] hanno creato un framework per rilevare attività di cryptojacking. Il framework, a tale scopo, analizza i seguenti aspetti:

- le API (ad esempio creazione dei Web Workers, creazione del codice WebAssembly, operazioni di lettura e scrittura tramite WebSocket) invocate durante il caricamento di una pagina web. Le API monitorate, rientrano tra quelle utilizzate dagli script Javascript di cryptojacking, come descritto nella Sezione 3.4.1. Per registrare le API invocate, è stato utilizzato lo strumento Chrome DevTools Protocol[4] presente all'interno di Chromium;
- le risorse hardware (GPU, CPU, rete e memoria) utilizzate durante la visita della pagina web. Al fine di evitare che il computo delle risorse hardware utilizzate dal browser fosse falsato dalle risorse utilizzate da altri processi, sono stati utilizzati dei container[5] Docker[6]. I campioni estratti hanno costituito il data set delle risorse utilizzate.

[4] https://chromedevtools.github.io/devtools-protocol/

[5] Un container è un ambiente di esecuzione virtuale, simile ad una VM (Virtual Machine). La differenza principale consiste nel fatto che una VM virtualizza l'hardware e richiede maggiori risorse computazionali rispetto ad un container, ma garantisce il massimo grado di isolamento dell'ambiente virtualizzato dall'host, durante l'esecuzione dell'applicazione. Al contrario, il container offre una virtualizzazione a livello di S.O, richiedendo meno risorse computazionali ma al contempo offrendo anche un minore isolamento.

[6] https://www.docker.com/

La tecnica di ML utilizzata è stata SVM con kernel lineare; inoltre il classificatore è stato validato tramite la 4-fold cross-validation. I risultati ottenuti sono stati i seguenti:

- per la classificazione del cryptojacking basata sulle API è stata ottenuta una recall del 95.83%, una precision del 80.31% ed un F-score del 82.38%;
- per la classificazione del cryptojacking basata sul computo delle risorse utilizzate, è stata ottenuta una recall del 93.89%, una precision del 13.32% ed un F-score del 23.31%.

L'approccio utilizzato nel lavoro di ricerca in questione risulta particolarmente complesso: monitorare le API invocate durante il caricamento di una pagina web e calcolare le risorse hardware utilizzando dei container risulta più esoso di risorse computazionali e quindi anche meno scalabile rispetto all'estrazione di statistiche di rete, approccio seguito in questa tesi. Inoltre, l'approccio utilizzato da Rodriguez *et al.* richiede l'installazione di software (es. i container) sul client, mentre i classificatori ottenuti in questa tesi possono essere utilizzati su qualsiasi nodo di rete e non richiedono l'installazione di software aggiuntivo.

M. Saad *et al.* [76] hanno utilizzato delle tecniche di analisi statica[7] degli script Javascript di cryptojacking (tra cui quelli di Coinhive approfonditi nella Sezione 3.4.2), per costruire un data set di campioni, analizzato mediante una tecnica di apprendimento non supervisionato di clustering, chiamata Fuzzy C-Means (FCM) [78]. I campioni del data set generato sono stati classificati dal modello in tre cluster:

- cluster di campioni appartenenti a script legittimi;
- cluster di campioni appartenenti a script Javascript malevoli, ma non di cryptojacking;
- cluster di campioni appartenenti a script di web-based cryptojacking.

L'accuratezza raggiunta è stata del 96.4%.

Il risultato raggiunto è inferiore al 99.74%, accuracy raggiunta dal classificatore di traffico generato da script di Coinhive, nell'implementazione di questa tesi.

Carlin *et al.* [79] hanno descritto un procedimento che utilizza l'analisi dinamica per costruire il data set da utilizzare nell'addestramento di un classificatore di traffico di cryptojacking. Il procedimento per costruire il data set è stato il seguente:

1. sono stati interpretati, tramite il browser Firefox[8], i file HTML di pagine web contenenti gli script di cryptojacking. Il browser è stato collegato a OllyDbg[9], un debugger per Windows, in grado di analizzare e memorizzare il codice assembler eseguito sul dispositivo, in un file chiamato traccia;
2. da ogni traccia è stato calcolato quante volte compariva il medesimo opcode[10]. Gli opcode considerati, sono quelli presenti nell'architettura Intel x86/x64.

Il data set, integrato con i campioni ottenuti da pagine web legittime, è stato utilizzato per addestrare una foresta casuale, validata tramite 10-fold cross-validation. Il classificatore è riuscito a raggiungere in media una precision, una recall ed un F-score pari al 99%.

L'approccio utilizzato nel lavoro di ricerca in questione risulta meno scalabile rispetto a quello usato in questa tesi, poiché di solito l'analisi dinamica è un processo più complesso ed esoso di risorse computazionali rispetto a quello di estrazione di statistiche dal traffico di rete.

[7] Una tecnica di analisi statica, al contrario di una tecnica di analisi dinamica, cerca di esaminare un malware senza eseguirlo [77], comparandone la firma. La *firma* di un malware (o *malware signature*) è una sequenza di byte che identifica univocamente il malware.

[8] https://www.mozilla.org/en-US/firefox

[9] http://www.ollydbg.de/

[10] È la porzione di istruzione in linguaggio macchina che identifica univocamente l'operazione eseguita dall'istruzione stessa (ad esempio l'opcode ADD identifica l'operazione aritmetica di somma).

7.2 Lavori collegati riguardanti le botnet

Negli ultimi anni si è assistito alla diffusione di botnet P2P (Peer-to-Peer) [52], ovvero botnet in cui non esistono nodi specifici con funzionalità di master, ma in cui ciascun bot può diventare un master. Il vantaggio principale di questa categoria di botnet è che il server master non costituisce più un *single point-of-failure*, poiché può essere rimpiazzato da un altro nodo della botnet. Pertanto, nel prosieguo del sezione, saranno analizzati alcuni articoli riguardanti il rilevamento tramite tecniche di ML, sia di botnet centralizzate, ovvero botnet in cui esiste un nodo master predefinito con cui i bot comunicano (ad esempio le botnet Pybot ed UFONet descritte nel Capitolo 4), sia di botnet P2P.

Nell'articolo scritto da S. Saad *et al.* [80], è stato documentato l'utilizzo di alcune tecniche di classificazione supervisionata, tra cui SVM con kernel lineare e k-NN, per individuare il traffico di C&C generato da botnet P2P. Gli autori dello studio, estraendo 17 statistiche dai flussi TCP di traffico di rete (ad esempio l'indirizzo IP sorgente/destinazione e la porta sorgente/destinazione di un flusso, la dimensione media dei pacchetti per flusso, il numero di pacchetti per flusso), hanno costruito dei modelli per classificare il traffico in una delle seguenti tre classi:

- traffico P2P generato da una botnet;
- traffico P2P legittimo;
- traffico non P2P (ad esempio HTTP, SMTP ecc..).

Per la validazione dei modelli, è stata utilizzata la tecnica della 10-fold cross-validation, mentre per valutare la bontà dei classificatori, è stata utilizzata la metrica recall e la metrica error rate; inoltre, nella comparazione tra classificatori, è stato considerato anche il tempo impiegato per l'addestramento e per la classificazione. È emerso che il classificatore SVM con kernel lineare, pur essendo il più lento sia nella fase di addestramento che in quella di classificazione, raggiunge le performance migliori, ottenendo nella classificazione del traffico di C&C delle botnet, l'error rate minore (inferiore al 7%) e la recall maggiore (prossima al 98%).

In questo lavoro di ricerca, le performance raggiunte dal classificatore sono state inferiori rispetto a quelle raggiunte nell'implementazione di questa tesi. In quest'ultimo caso, infatti, la recall minima raggiunta nella classificazione del traffico di C&C delle botnet è stata del 98.54% mentre l'error rate è stato sempre inferiore al 1%.

Haddadi *et. al* [81] hanno utilizzato le tecniche supervisionate di ML degli alberi di decisione e di naive Bayes per classificare il traffico di C&C generato dalle botnet, di tipo centralizzato, che utilizzano il protocollo HTTP per la comunicazione tra master e bot. Tali botnet sono state create dai seguenti malware:

- *Zeus*[11]: un malware, scoperto nel 2007, che è stato in grado di infettare milioni di dispositivi. Questo malware è in grado di svolgere diverse azioni malevoli tra cui l'estorsione di dati sensibili, ad esempio tramite un keylogger (definito nella Sezione 3.5), utilizzati per eseguire frodi finanziarie;
- *Citadel*[12]: un malware, che è stato in grado, a partire dal 2011, di infettare 11 milioni di computer e rubare, tramite un keylogger, milioni di dollari.

Il flusso di lavoro, seguito dagli autori dello studio, è stato il seguente:

1. dalle catture di traffico effettuate durante il collegamento verso server web appartenenti sia a domini legittimi che a domini malevoli, sono stati estratti i flussi TCP, utilizzando lo strumento NetFlow[13] sviluppato da Cisco;

[11] https://www.kaspersky.it/resource-center/threats/zeus-trojan-malware

[12] https://www.cyber.nj.gov/threat-profiles/trojan-variants/citadel

[13] https://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html

2. per ogni flusso sono state calcolate 23 statistiche tra cui la durata, il numero di pacchetti, il throughput e il numero di byte per pacchetto;

3. a partire dai dati ottenuti, sono stati addestrati i classificatori alberi di decisione e naive Bayes.

Per la validazione dei modelli, è stata utilizzata la 10-fold cross-validation. Gli alberi di decisione sono stati i classificatori più performanti, raggiungendo una recall del 88% per il rilevamento del traffico di C&C di Citadel e del 86% per il rilevamento del traffico di C&C di Zeus. Inoltre, filtrando il traffico HTTP dalle catture di traffico iniziali, le performance dei classificatori, in alcuni casi, sono migliorate: infatti gli alberi di decisione hanno ottenuto un miglioramento della recall del 9% per il rilevamento di Citadel, mentre la recall per il rilevamento di Zeus è rimasta invariata.

I risultati raggiunti dal classificatore di questo lavoro di ricerca sono inferiori a quelli raggiunti dall'albero decisione, classificatore di traffico selezionato nella parte implementativa di questa tesi, riguardante le botnet. In quest'ultimo caso, infatti, la recall minima registrata è stata del 98.54%.

Stevanovic *et al.* [82] hanno confrontato otto diverse tecniche di machine learning supervisionato, tra cui SVM, alberi di decisione e foreste casuali, per il rilevamento di traffico di C&C generato da botnet. In particolare le catture di traffico malevolo, utilizzate per l'addestramento dei classificatori, hanno riguardato le botnet P2P Storm e Waledac:

- la botnet Storm[14], scoperta nel 2007, si è diffusa grazie all'invio di spam che induce al download del malware omonimo, responsabile dell'aggiunta dei dispositivi infettati alla botnet. Secondo i ricercatori di sicurezza [83], la fonte principale di guadagno dei creatori di Storm, è stata affittare la botnet a terzi per scopi illegali (ad esempio spamming e attacchi DDoS);

- la botnet Waledac[15], creata tramite l'omonimo worm, è stata impiegata, secondo il sito KrebsonSecurity[16], per inviare all'incirca 1.5 miliardi di email di spam al giorno prima che fosse chiusa da Microsoft nel 2010.

Il flusso di lavoro seguito durante la ricerca, è stato il seguente:

1. a partire da alcune catture di traffico malevolo e traffico legittimo, sono state estratte 39 statistiche (ad esempio numero totale di pacchetti, numero totale di byte e numero medio di byte per pacchetto) riguardanti i flussi TCP e UDP;

2. le misure estratte sono state utilizzate per ottenere il data set; il 66% di tale data set è stato utilizzato per l'addestramento degli otto classificatori.

Dal confronto è emerso che la foresta casuale è il classificatore più performante, ottenendo una precision di 96%, una recall di 97% ed un F-score di 95%. Anche gli alberi di decisione hanno raggiunto prestazioni elevate, di poco inferiori a quelle delle foreste casuali.

Le performance raggiunte dal classificatore in esame, pur elevate, sono inferiori a quelle raggiunte dall'albero di decisione, il classificatore selezionato in questo lavoro di tesi. Difatti in questa tesi, per il rilevamento del traffico di C&C generato da botnet, in media, è stata raggiunta una precision del 96.62%, una recall del 98.89% ed un F-score del 97.74%.

[14] https://www.symantec.com/security-center/writeup/2001-060615-1534-99

[15] https://www.symantec.com/security-center/writeup/2008-122308-1429-99

[16] https://krebsonsecurity.com/tag/waledac/

7.3 Lavori collegati riguardanti altri aspetti della sicurezza informatica

Le tecniche di ML sono ampiamente impiegate all'interno degli *IDS (Intrusion Detection System)* [84, 85]. Un IDS, è un sistema che, secondo la definizione riportata nel RFC 2828 [86], offre:

> un servizio di sicurezza che monitora ed analizza gli eventi allo scopo di rilevare in tempo reale e segnalare tentativi di accedere alle risorse di sistema in modo non autorizzato.

Pertanto un IDS può rilevare diverse minacce tra cui la compromissione del dispositivo causata da un malware (worm, virus, eccetera), il tentativo di utilizzo del dispositivo all'interno di una botnet, attacchi di spamming, attacchi di scanning (un esempio di scanning è quello effettuato da UFONet (Sezione 4.3) per ricercare gli host che espongono la vulnerabilità open-redirect) e diverse altre. Un IDS può impiegare alcune tecniche di funzionamento, tra cui:

- tecniche di rilevamento basate sulla firma: esse rilevano attacchi noti in base alla loro *firma* (una sequenza di byte che identifica univocamente il malware). Lo svantaggio principale è che tali tecniche non sono in grado di scoprire attacchi *zero-day*, ovvero attacchi che sfruttano vulnerabilità software appena scoperte. Un esempio di HIDS[17] con tali caratteristiche è un software anti-virus: esso rileva una minaccia se la sua firma è contenuta nella propria base dati di firme di virus (*virus signature*), che pertanto deve essere frequentemente aggiornata;
- tecniche di rilevamento basate su anomalie: esse identificano gli attacchi in base ad anomalie, come ad esempio il traffico di rete inconsueto. Tali tecniche sono in grado di rilevare attacchi zero-day, ma hanno il difetto di generare un elevato tasso di falsi allarmi *(FAR, False Alarm Rate)*: ad esempio, se un utente autorizzato non ricorda la password e compie numerosi tentativi per accedere ad un sistema, potrebbe generare involontariamente un falso allarme e contribuire all'aumento del FAR. Nel seguito saranno descritti due esempi di questa tecnica.

Bilge *et al.* [87] hanno implementato un IDS chiamato EXPOSURE, che si basa sul rilevamento delle anomalie riguardanti il traffico DNS. Il sistema è costituito da cinque moduli:

- modulo che gestisce una base dati in cui sono memorizzati i domini già classificati come legittimi oppure come malevoli;
- modulo che si occupa di registrare il traffico DNS prodotto dalla rete protetta dall'IDS;
- modulo che estrae dal traffico catturato, gli attributi dei campioni del data set. Gli attributi in questione possono essere distinti nelle seguenti categorie:
  - attributi temporali: misurano l'istante in cui sono state effettuate le richieste DNS. È probabile che domini utilizzati a fini malevoli siano oggetto di un numero elevato di richieste DNS per brevi periodi di tempo (ad esempio il dominio di un server di command-and-control di una botnet);
  - attributi riguardanti le risposte alle richieste DNS, come ad esempio il numero di indirizzi IP associati al medesimo dominio e l'ubicazione di tali indirizzi IP;
  - attributi riguardanti il campo TTL[18] del record DNS. Il campo TTL può essere sfruttato, ad esempio, per recuperare più velocemente le funzionalità di una botnet in caso

[17] Un HIDS (Host-based IDS) [17] è un IDS specializzato nell'offrire il servizio di *Intrusion Detection (ID)* all'interno di un host. Al contrario, un NIDS (Network-based IDS) è un IDS specializzato nel monitorare la rete per identificare attività sospette.

[18] Il campo TTL (Time-To-Live) di un record DNS specifica il periodo di validità del record all'interno della cache DNS, che è locale al dispositivo. Allo scadere di tale periodo, l'host deve effettuare una richiesta DNS esterna, verso la gerarchia dei server DNS (*Name Server*).

di chiusura dei server di C&C. Difatti, in questa circostanza, i bot non potrebbero più ricevere i comandi per compiere un'azione malevola (ad esempio attaccare un nuovo bersaglio). Ma, se esistono ulteriori server di C&C, avendo impostato un basso valore di TTL nei record memorizzati nella cache DNS dei bot, tali record scadranno in tempi più brevi. Pertanto, grazie alla richiesta inviata verso la gerarchia DNS, i record DNS in questione saranno aggiornati con quelli contenenti i nuovi indirizzi IP dei server di C&C da contattare;

- attributi riguardanti il nome del dominio: spesso i siti malevoli presentano nomi di dominio più lunghi e complessi rispetto a siti legittimi poiché in molti casi tali domini sono creati tramite un *DGA (Domain Generating Algorithm)*[19].

- modulo che si occupa sia di addestrare il classificatore tramite la tecnica degli alberi di decisione sia di validarlo tramite la 10-fold cross-validation;

- modulo che utilizza il classificatore addestrato per suddividere il traffico DNS in traffico legittimo e illegittimo.

EXPOSURE è riuscito ad ottenere ottimi risultati di classificazione, misurati tramite la recall (il valore ottenuto è stato 98.5%) e AUC (il valore ottenuto è stato 98.7%).

Kumar *et al.* [88] hanno sviluppato il framework *MLSPD (Machine Learning Based Spam and Phishing Detection)* per classificare spam e URL utilizzati per il phishing. Il principio di funzionamento è il seguente:

1. ogni email in arrivo viene inviata al modulo di rilevamento di spam che estrae alcuni dati riguardanti lo stile di scrittura dell'email, tramite lo strumento IBM Watson Tone Analyzer[20]. I dati estratti sono utilizzati per addestrare dei classificatori basati su alcune tecniche supervisionate di ML, tra cui le foreste casuali;

2. se l'email contiene un URL, esso viene inviato al modulo di rilevamento di phishing. Tale modulo estrae alcune caratteristiche dell'URL (ad esempio lunghezza e presenza di caratteri speciali) che costituiranno gli attributi del training set.

Tra i diversi classificatori addestrati, le foreste casuali hanno raggiunto i migliori risultati. Essi sono i seguenti:

- per il rilevamento di spam, è stata raggiunta una precision del 89.2%, un F-score del 90%, un'accuracy del 89.2% ed un AUC del 96.8%;

- per il rilevamento di phishing, è stata raggiunta una precision del 98.3%, un F-score del 97.5%, un'accuracy del 97.7% ed un AUC del 99.7%.

Un altro aspetto della sicurezza informatica in cui l'utilizzo delle tecniche di ML sta acquisendo un ruolo sempre più rilevante, è quella della classificazione di traffico cifrato.

Anderson *et al.* [89] hanno comparato le performance di diverse tecniche supervisionate di machine learning, tra cui SVM, alberi di decisione e foreste casuali per la classificazione di traffico cifrato tramite TLS. Lo scopo di questa ricerca è quello di confrontare il comportamento dei classificatori al variare della percentuale di rumore all'interno del data set e al variare del traffico di rete nel tempo. Inoltre gli studiosi hanno voluto evidenziare come l'utilizzo di statistiche derivanti dall'analisi di TLS possa migliorare la bontà dei classificatori. Il procedimento seguito è stato il seguente:

[19] Un DGA è un modulo software spesso integrato nei malware, per creare e registrare una serie di domini alternativi tramite cui gli host infettati possono comunicare con il server dell'attaccante. In questo modo, qualora il server di C&C venga scoperto, quest'ultimo può essere facilmente sostituito da un server di C&C di un altro dominio.

[20] https://www.ibm.com/watson/services/tone-analyzer/

1. è stato raccolto il traffico di una rete aziendale costituita da due sotto-reti dislocate in aree geografiche diverse, ed il traffico generato da un software di analisi di malware tramite virtual machine (*malware sandbox*);
2. dal traffico sono state estratte 22 statistiche tra cui il minimo, il massimo, la media e la deviazione standard della dimensione dei pacchetti e del tempo di arrivo relativo tra due pacchetti successivi. Inoltre, sono state aggiunte delle statistiche riguardanti le cipher-suite e le estensioni TLS osservate nel protocollo di TLS handshake;
3. il training set ottenuto è stato utilizzato per l'addestramento dei classificatori. Inoltre, i classificatori sono stati validati tramite la 10-fold cross-validation;
4. infine, la fase di test dei classificatori, è stata effettuata su un traffico di rete diverso da quello utilizzato nella fase di addestramento.

I risultati ottenuti hanno mostrato come l'utilizzo di attributi estratti dal traffico TLS migliori nettamente le performance dei classificatori. Riguardo alla degradazione delle performance dei classificatori al variare della percentuale di rumore nel data set, il modello SVM è stato quello che ha fatto registrare la diminuzione più accentuata delle prestazioni. Al contrario, gli alberi di decisione e le foreste casuali si sono rivelati tra i classificatori più performanti e resistenti sia alla variazione del traffico nel tempo sia alla variazione della percentuale di rumore del data set. I risultati ottenuti sono stati i seguenti:

- gli alberi di decisione hanno raggiunto un'accuratezza del 97.02% sul data set ottenuto dal traffico della rete aziendale e del 83.33% sul data set ottenuto tramite la malware sandbox;
- le foreste casuali hanno raggiunto un'accuratezza del 99.99% sul data set ottenuto dal traffico della rete aziendale e del 76.79% sul data set ottenuto tramite la malware sandbox.

La ricerca in questione, in modo simile a quanto fatto per questa tesi, utilizza le statistiche estratte dal traffico di rete per addestrare i classificatori e rilevare traffico malevolo. Ma essa si è limitata a classificare traffico cifrato, al contrario di quanto è stato fatto in questa tesi. Nonostante oggi il traffico cifrato sia in maggioranza, la percentuale di traffico in chiaro non è ancora trascurabile (si attesta intorno al 28% [3]).

Capitolo 8

Conclusioni

In questo lavoro di tesi è stato affrontato il problema di rilevare il traffico Stratum sia cifrato che in chiaro, generato dal software XMR-Stak, dal malware MadoMiner e dagli script di Coinhive, e di rilevare il traffico di C&C generato dalle botnet Pybot ed UFONet.

A questo scopo, sono state utilizzate le seguenti tecniche supervisionate di machine learning:

- albero di decisione;
- SVM con kernel non lineare, in particolare gaussiano;
- k-NN, variandone l'iperparametro k;
- foresta casuale, variandone l'iperparametro riguardante il numero di classificatori di base.

In tutti i casi considerati, l'albero di decisione è stato il classificatore che ha ottenuto i risultati migliori durante l'addestramento e la validazione.

I dati utilizzati per l'addestramento, validazione e test dei classificatori, sono costituiti dalle statistiche TCP di traffico rete estratte tramite lo strumento Tstat. Non sono state utilizzate statistiche legate alla DPI, dal momento che questa pratica in molti casi comporta la violazione del diritto alla privacy, ed inoltre risulta inefficace nel caso di traffico cifrato.

La Tabella 8.1 e la Tabella 8.2 mostrano i risultati ottenuti nella classificazione, tramite alberi di decisione, dei test set contenenti rispettivamente i campioni di traffico Stratum ed i campioni di traffico generato da botnet. Qualche criticità è emersa nella classificazione del traffico Stratum generato dagli script di Coinhive: in questo caso precision, recall ed F-score hanno valori inferiori rispetto a quelli ottenuti negli altri casi presi in esame.

statistiche (%)	*XMR-Stak*	*MadoMiner*	*Coinhive*
accuracy	99.77	99.96	99.74
precision	96.46	94.09	66.53
recall	98.84	98.57	84.74
specificity	99.82	99.97	99.81
AUC	99.33	99.27	92.27
F-score	97.63	96.28	74.54

Tabella 8.1: Risultati di classificazione del traffico Stratum tramite albero di decisione.

Una possibile estensione di questo lavoro di tesi potrebbe consistere nell'utilizzare la tecnica supervisionata di ML delle reti neurali per identificare sia il traffico di cryptojacking che quello

statistiche (%)	*Pybot*	*UFONet*
accuracy	99.87	99.90
precision	95.55	97.69
recall	98.54	99.24
specificity	99.90	99.92
AUC	99.22	99.58
F-score	97.02	98.46

Tabella 8.2: Risultati di classificazione del traffico generato da botnet tramite albero di decisione.

generato dalle botnet. Difatti, Auld *et. al* [90] hanno ottenuto risultati promettenti nell'applicare tale tecnica di ML alla classificazione di traffico di rete. Questa tecnica di machine learning permette anche di utilizzare le GPU che, per loro natura, sfruttano il calcolo parallelo. L'uso delle GPU, pertanto, permetterebbe di classificare in parallelo dozzine o centinaia di flussi TCP, rendendo l'analisi più vicina alla realtà. Inoltre, questo approccio permetterebbe di capire come i modelli si comportano nella classificazione di un numero elevato di dati.

Un altro sviluppo futuro potrebbe consistere nell'ampliare il set di dati di traffico legittimo includendo altre tipologie di traffico, oltre quello web. Sarebbe interessante, ad esempio, osservare come si comportano i classificatori, inserendo traffico di posta, traffico SSH e traffico generato da applicazioni di largo utilizzo (ad esempio Skype).

Dal momento che, come descritto nel Capitolo 4, una tendenza recente è utilizzare botnet di dispositivi IoT, sarebbe interessante cercare di classificare, tramite le tecniche di ML approfondite in questa tesi, il traffico di C&C generato da questa tipologia di botnet, come ad esempio il traffico di C&C di Mirai.

A novembre 2018, i ricercatori di McAfee hanno scoperto un nuovo malware di cryptojacking, chiamato WebCobra[1]. Esso installa, sui dispositivi infettati, un software in grado di minare criptomonete basate su tecnologia CryptoNight (Sezione B.4). Un possibile sviluppo futuro potrebbe consistere nell'utilizzo dei modelli addestrati su Stratum per classificare il traffico generato da questo malware.

[1] https://securingtomorrow.mcafee.com/other-blogs/mcafee-labs/webcobra-malware/

Appendice A

Statistiche aggiuntive Tstat

Le seguenti statistiche sono disponibili solo per i flussi TCP e vengono salvate all'interno dei file `log_tcp_complete`.

Per una migliore comprensione di tali statistiche, si prega di fare riferimento alla Sezione 2.1.

Le statistiche (disponibili sia nella direzione *C2S* che nella direzione *S2C*) sono le seguenti:

- *average RTT:* RTT medio, calcolato tramite l'algoritmo di Karn;
- *RTT min:* RTT minimo osservato durante la connessione;
- *RTT max:* RTT massimo osservato durante la connessione;
- *stdev RTT:* deviazione standard del RTT;
- *RTT count:* numero di campioni RTT validi osservati;
- *TTL min:* TTL (Time-To-Live) minimo osservato nei pacchetti della connessione;
- *TTL max:* TTL (Time-To-live) massimo osservato nei pacchetti della connessione;
- *window scale option:* assume valore 1, se uno dei due attori coinvolti nella connessione ha inviato un segmento con l'opzione WSCALE;
- *window scale factor:* rappresenta il valore del *scale factor* che è stato negoziato;
- *timestamp option:* assume valore 1 se è stato inviato un segmento con l'opzione Timestamp impostata;
- *SACK request:* assume valore 1 se è stato osservato un segmento contenente l'opzione SACK-Permitted;
- *SACK sent:* numero di segmenti con opzione SACK impostata;
- *MSS:* massima dimensione in byte del segmento che può essere utilizzata nella connessione;
- *maximum segment size:* massima dimensione del segmento che è stata osservata;
- *minimum segment size:* minima dimensione del segmento che è stata osservata;
- *win max:* massimo valore di *rwnd (receive window)*, comprensivo del window scale factor, osservato durante connessione;
- *win min:* minimo valore di rwnd, comprensivo di window scale factor, osservato durante la connessione;
- *win zero:* numero di segmenti con rwnd pari a 0;
- *cwin max:* stima della *cwnd (congestion window)* massima osservata;

- *cwin min:* stima della cwnd minima osservata;
- *initial cwin:* valore di cwnd iniziale;
- *rtx RTO:* numero di segmenti ritrasmessi a causa della scadenza del RTO;
- *rtx FR:* numero di segmenti ritrasmessi tramite il Fast Retransmit;
- *reordering:* numero di packet-reordering che sono stati osservati durante la connessione;
- *flow control:* numero di segmenti ritrasmessi al fine di aumentare o diminuire la rwnd del mittente;
- *unnecessary rtx RTO:* numero di trasmissioni inutili dopo la scadenza del RTO;
- *unnecessary rtx FR:* numero di trasmissioni inutili dopo un fast retransmit;
- *!= SYN sequence number:* settato ad 1 se i segmenti di SYN ritrasmessi durante la connessione hanno un sequence number iniziale differente.

Appendice B

La blockchain, il processo di mining e la criptovaluta XMR

B.1 La blockchain

La blockchain [35] è stata descritta per la prima volta nel 1990 ad opera di Haber e Stornetta [91], ma ha trovato ampio utilizzo solo con la nascita e diffusione del bitcoin. Oggigiorno la blockchain è ampiamente utilizzata anche in molti altri ambiti tra cui identità digitale, tasse, servizi notarili, attività di voto, memorizzazione di dati e molti altri [35].

B.1.1 Struttura dati

La blockchain [32, 92, 93] è costituita da un insieme di blocchi inseriti in ordine cronologico e tra loro concatenati. Un blocco è una struttura dati composta da una intestazione e da un payload contenente le transazioni, come mostrato in Figura B.1.

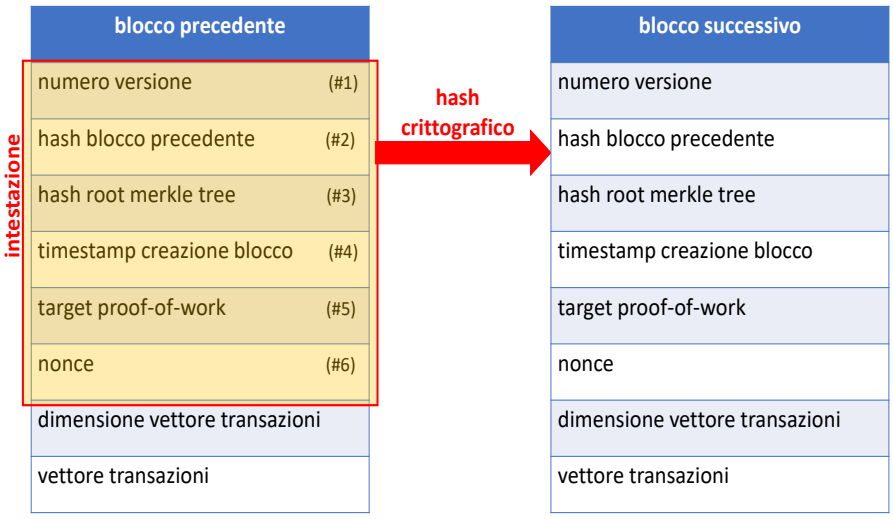

Figura B.1: Riferimento tra blocchi all'interno della blockchain.

Ogni blocco è identificato dall'hash ottenuto tramite l'applicazione sull'intestazione, di un algoritmo di hashing crittografico (ad esempio, nel caso del bitcoin, il double-SHA-256, ovvero l'algoritmo SHA-256 [94] applicato due volte). Pertanto l'identificativo di un blocco è calcolato nel seguente modo (il simbolo || indica la concatenazione):

$$id = hash(campo\#1 \;||\; campo\#2 \;||\; campo\#3 \;||\; campo\#4 \;||\; campo\#5 \;||\; campo\#6)$$

Tale identificatore non è contenuto nel blocco corrente, bensì nell'intestazione del blocco cronologicamente successivo, come mostrato in Figura B.1.

I restanti campi dell'header sono:

- *timestamp:* indica, in formato UNIX, l'istante di creazione del singolo blocco. L'ordine cronologico dei blocchi inseriti nella catena, è assicurato dal valore di questo campo;
- *root del merkle tree:* il merkle tree [95] è un albero binario che è costruito, a partire dalle foglie, applicando ricorsivamente l'algoritmo di hashing crittografico sulle transazioni incluse nel blocco, come mostrato nella Figura B.2. Questa struttura dati permette di verificare, con complessità logaritmica, se una transazione è stata inclusa o meno all'interno di un blocco;
- *target:* rappresenta il livello di difficoltà del puzzle crittografico (o hash puzzle) [35] da risolvere per il mining del blocco corrente. Il significato di questo campo, insieme a quello di nonce, saranno approfonditi nella Sezione B.3.

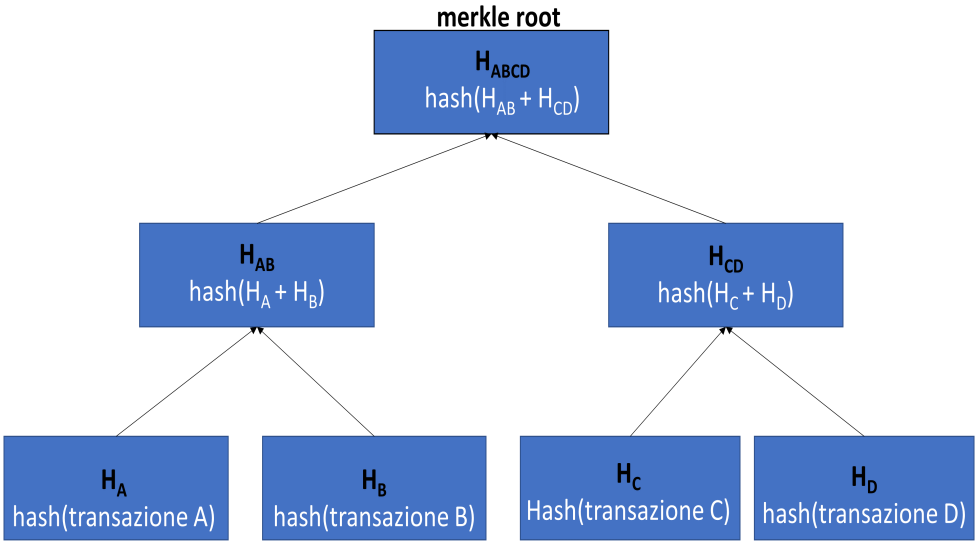

Figura B.2: Esempio di merkle tree.

B.1.2 Proprietà

É possibile notare come la blockchain applichi diffusamente gli algoritmi di hashing crittografico. Questo risulta fondamentale per garantire l'integrità dei dati contenuti al suo interno.

Spesso si utilizza il termine immutabilità della blockchain per indicare che, dopo l'aggiunta di un blocco alla catena, è computazionalmente molto oneroso cercare di modificarlo: un attaccante, per modificare con successo un blocco, dovrebbe risolvere l'hash puzzle (Sezione B.3) di tutti i blocchi precedentemente inseriti. Inoltre, l'immutabilità è dinamica poiché si rafforza ogni volta che un nuovo blocco viene aggiunto: modificare un blocco aggiunto meno recentemente è computazionalmente più oneroso di modificare un blocco inserito più di recente.

B.2 Struttura delle transazioni e processo di validazione

Le transazioni [32, 92, 93] permettono di trasferire il possesso della criptovaluta. Per validare tali transazioni, viene utilizzata la crittografia asimmetrica con algoritmo di firma *EdDSA (Edwards-curve DSA)* [96] nel caso della criptomoneta XMR (Sezione B.4).

Il portafoglio di una criptovaluta (o *wallet*) è un software oppure un dispositivo hardware, a cui è associato un indirizzo derivato dalla chiave pubblica del proprietario. Il wallet ha il compito di verificare quali output di transazioni precedenti (Figura B.3) sono destinate all'indirizzo del proprietario e permettere la spesa della criptovaluta creando nuove transazioni in favore di terzi.

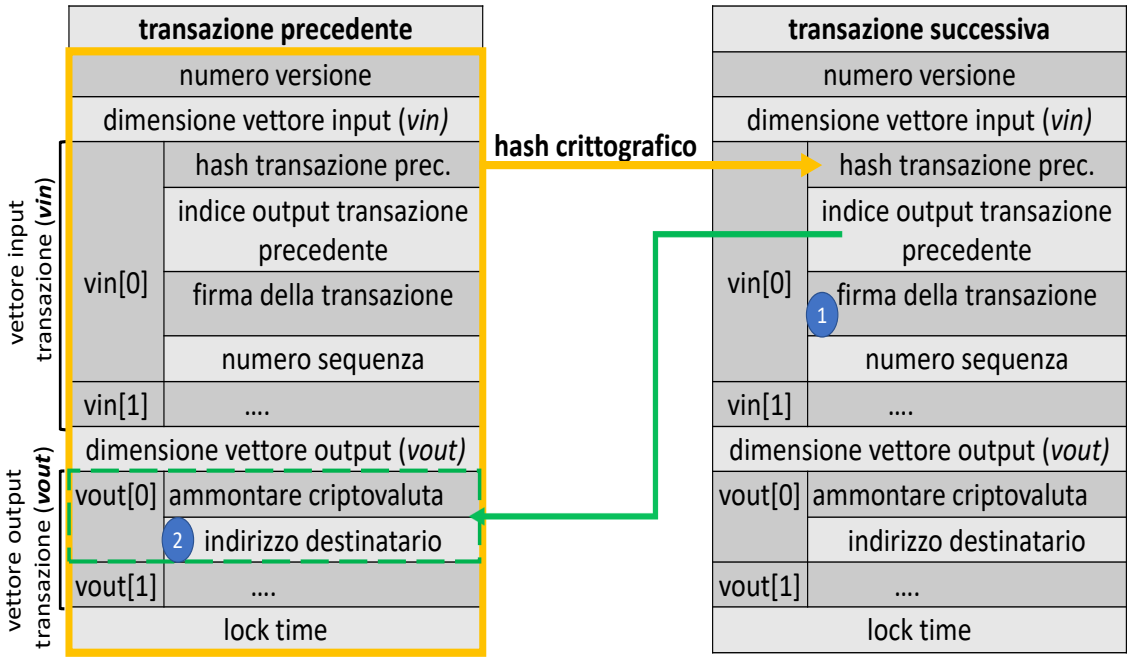

Figura B.3: Struttura e verifica di una transazione.

Ogni transazione, come mostrato in Figura B.3, è costituita principalmente dai seguenti campi:

- uno o più *input* (indicati in Figura B.3 con *vin*), ovvero campi all'interno della struttura dati di una transazione, che si riferiscono a somme di criptovaluta derivanti da transazioni precedenti;
- uno o più *output* (indicati in Figura B.3 con *vout*), ovvero campi all'interno della struttura dati di una transazione, che contengono l'*indirizzo* del *portafoglio* del destinatario e la somma di criptovaluta trasferita;
- l'hash ottenuto applicando un algoritmo di hashing crittografico sui campi di una transazione precedente;
- un indice che permette di identificare lo specifico output della transazione precedente.

La validazione di una transazione avviene nel modo seguente (l'elenco corrisponde ai numeri 1 e 2 in Figura B.3):

1. colui che desidera spendere una certa somma di criptovaluta, crea una nuova transazione (indicata in Figura B.3 come *transazione successiva*) in cui vi è un campo input che si riferisce ad un output di una transazione precedente ed applica la propria firma alla transazione;
2. la transazione, insieme alla firma apportata, viene inviata in broadcast ai nodi della rete deputata allo scambio della moneta digitale. Ogni nodo, durante il processo di mining (Sezione B.3), verifica la firma con la chiave pubblica derivata dal campo *indirizzo destinatario* presente all'interno dell'output della transazione precedente.

B.2.1 La transazione coinbase

All'interno del protocollo Stratum (descritto nella Sezione 3.2), la transazione coinbase viene divisa nelle seguenti quattro regioni:[1]

- *coinbase1*: la prima porzione della transazione coinbase (Figura B.4). Essa di solito comprende i campi iniziali della transazione, incluso una porzione dell'input;
- *extranonce1* (approfondito nella Sezione 3.2.2);
- *extranonce2* (approfondito nella Sezione 3.2.2);
- *coinbase2*: si estende dalla fine del campo extranonce2 fino al termine della transazione.

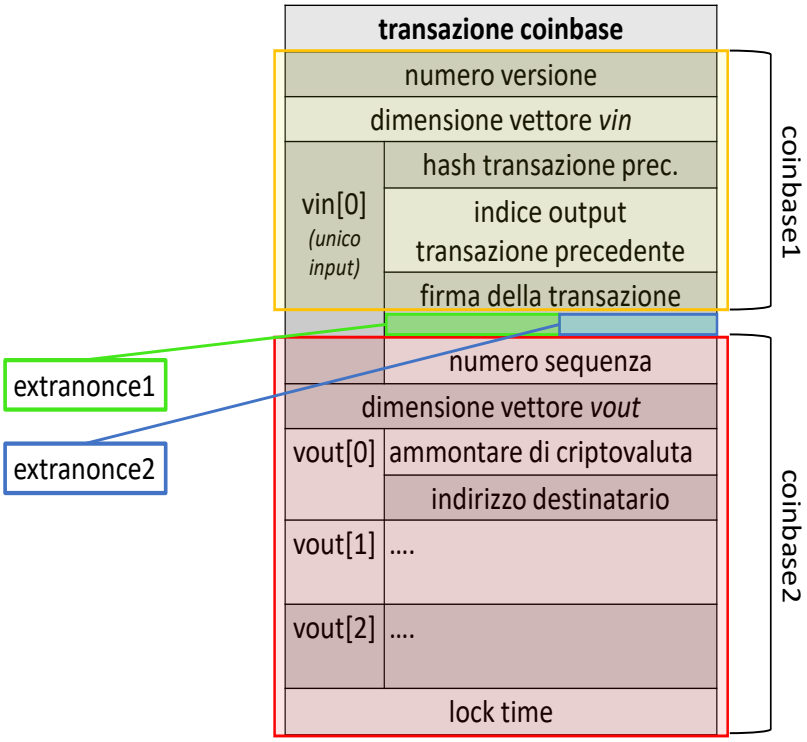

Figura B.4: Struttura della transazione coinbase.

Al di fuori del contesto di utilizzo del protocollo Stratum, i campi extranonce1 ed extranonce2, vengono solitamente considerati come un unico campo di lunghezza variabile, chiamato *extranonce*. Tale campo viene utilizzato insieme al *nonce* (campo #6 in Figura B.1) per risolvere l'hash puzzle (Sezione B.3).

B.3 Il processo di mining

La procedura di mining [32, 92, 93], eseguita da ogni miner, può essere riassunta nelle seguenti fasi:

1. il miner seleziona il blocco aggiunto più di recente alla catena ed appartenente al percorso più lungo. La blockchain, infatti, può contenere delle biforcazioni causate dagli inevitabili

[1] Dal momento che il protocollo Stratum non è stato standardizzato, i campi coinbase1, coinbase2, extranonce1 ed extranonce2, hanno dimensioni variabili e comprendono campi della transazione coinbase differenti a seconda del mining pool. Per tali ragioni è possibile darne solo una descrizione qualitativa.

ritardi di propagazione dei nuovi blocchi minati all'interno della rete. Solitamente queste situazioni vengono risolte in breve tempo: la biforcazione con numero inferiore di blocchi viene eliminata e le transazioni in essa contenute vengono considerate come nuove transazioni da validare ed aggiungere ai futuri blocchi del ramo principale della blockchain;

2. il miner calcola l'hash del blocco aggiunto più di recente alla blockchain e lo inserisce nell'intestazione del nuovo blocco da minare. In questo modo viene creato il riferimento tra l'ultimo blocco inserito nella catena ed il nuovo blocco;

3. il miner calcola il merkle tree sulle transazioni validate ed inserisce il valore della radice del merkle tree all'interno dell'intestazione del nuovo blocco;

4. ogni miner partecipa alla competizione per risolvere l'hash puzzle.

B.3.1 Algoritmo Proof-of-Work

L'hash puzzle o algoritmo Proof-of-Work consiste nell'applicare ripetutamente un algoritmo di hashing crittografico sull'header del blocco da minare, fintanto che l'hash prodotto non risulti inferiore o uguale al target presente nel campo *target proof-of-work* (campo #5 in Figura B.1):

$$hash(\ campo\#1\ ||\ campo\#2\ ||\ campo\#3\ ||\ campo\#4\ ||\ campo\#5\ ||\ campo\#6\ )\ \leq\ target$$

Ad ogni iterazione si modificano 2 parametri:

1. extranonce della transazione coinbase. Ogni volta che viene modificata la transazione coinbase, varia di conseguenza anche il valore della radice del merkle tree, inserito nell'header del nuovo blocco;

2. nonce dell'header del blocco.

Il primo miner che riesce a risolvere l'hash puzzle, inserisce il nonce all'interno dell'header ed invia in broadcast il blocco appena minato. Tale nonce rappresenta la proof-of-work, ovvero la prova del lavoro svolto. Se gli altri nodi concordano sul risultato trovato, il miner viene ricompensato con:

- l'ammontare di moneta virtuale contenuta nella transazione coinbase;
- la somma di criptovaluta, derivante dalle tasse pagate dai creatori delle transazioni contenute nel blocco appena minato, calcolate nel modo seguente:

$$\text{tasse\_totali} = somma(\text{input\_transazioni}) - somma(\text{output\_transazioni})$$

B.3.2 L'hashing race

Gli ultimi anni sono stati caratterizzati da un inasprimento dell'hashing race dovuto all'esplosione del fenomeno delle criptovalute. Questo è testimoniato anche da un incremento pressoché esponenziale dell'*hash rate* come mostrato in Figura B.5. L'hash rate è la misura del numero di operazioni di hashing al secondo ($1\ \frac{MH}{s}$ equivale a 10^9 operazioni di hashing al secondo).

La necessità di disporre di ingenti risorse computazionali per poter vincere la competizione per il mining di un nuovo blocco, ha causato, oltre che la nascita dei mining pool, anche un grande investimento, da parte dei singoli miner, per aumentare le risorse computazionali dell'hardware utilizzato per il mining, anche chiamato *rig*. In un primo momento, si è assistito all'utilizzo di software che sfruttava le GPU poiché risultano efficienti nel calcolo parallelo. In seguito, si è passati all'utilizzo di hardware dedicato ASIC per le operazioni di hashing [36].

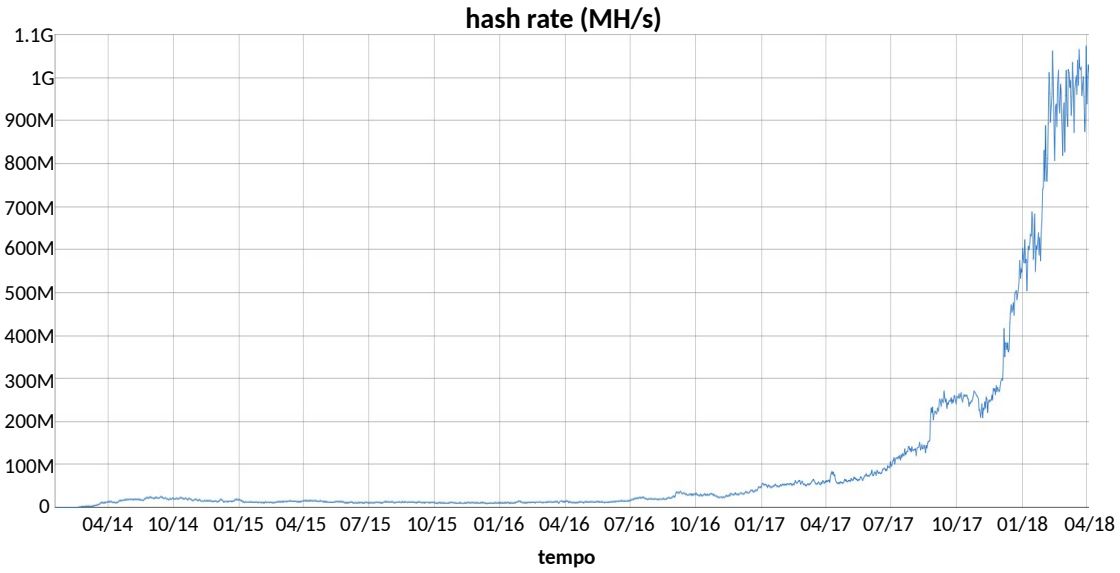

Figura B.5: Crescita dell'hashing power di XMR (fonte: bitinfocharts.com).

B.4 Monero (XMR)

Il funzionamento della criptomoneta Monero si basa sui meccanismi descritti nei paragrafi precedenti, a cui si aggiungono alcune caratteristiche peculiari della tecnologia *CryptoNote*. Tale tecnologia [97, 40] ha l'obiettivo di garantire l'anonimato nelle transazioni delle criptovalute e di permettere una hashing race più equa, rendendo poco vantaggioso l'utilizzo di hardware dedicato rispetto a quello general-purpose. Grazie alla sua difficile tracciabilità, XMR è molto utilizzata sia da utenti particolarmente attenti allo loro privacy che da criminali. Difatti, secondo il report 2018 della società Chainalysis [98], XMR è tra le monete più utilizzate all'interno dei mercati illeciti del dark web ed il numero di transazioni di questa moneta è aumentato del 230%, solo nel 2017.

CryptoNote utilizza la crittografia asimmetrica a curve ellittiche (ECC) [99], basata su operazioni aritmetiche tra punti appartenenti a curve ellittiche. In tale tipologia di crittografia, la chiave pubblica *(A)* è un punto su una particolare curva ellittica, ottenuto moltiplicando la chiave privata *(a)*, ovvero un valore casuale scelto all'interno di un determinato intervallo, per un altro punto della curva chiamato *punto generatore* G:

$$A = a \cdot G$$

Nel contesto di CryptoNote, si utilizzano le seguenti convenzioni:

- *chiave privata CN (CryptoNote)*: è una coppia di chiavi private *(a, b)*;
- *chiave pubblica CN (CryptoNote)*: è la coppia di chiavi pubbliche *(A, B)*, corrispondenti alla chiave privata CN.

Al fine di garantire l'anonimato, CryptoNote utilizza la *one-time key* e la *ring signature*, mentre impiega l'algoritmo di hash crittografico *CryptoNight* per permettere il mining su hardware comune. Di seguito verrà esposto il loro principio di funzionamento.

B.4.1 One-time key

Come già descritto nei paragrafi precedenti, ogni indirizzo è associato ad un'unica chiave pubblica. Ma, poiché la blockchain è memorizzata in chiaro, è possibile osservare se transazioni differenti

sono destinate al medesimo indirizzo. Al fine di garantire la privacy, CryptoNote permette di utilizzare, come destinazione delle transazioni, le one-time key, ovvero indirizzi generati al volo, a partire dalla chiave pubblica del destinatario e da un numero casuale scelto dal mittente. In tale modo, transazioni differenti a beneficio del medesimo destinatario non possono essere correlate tra loro, pur comparendo in chiaro all'interno della blockchain.

Il principio di funzionamento della one-time key è il seguente:

1. il mittente della transazione sceglie un numero casuale r;
2. il mittente, a partire da r e dalla chiave *(A, B)* (chiave pubblica CN del destinatario), calcola una one-time key nel seguente modo:

$$P = hash(r \cdot A) \cdot G + B$$

3. il mittente utilizza P come indirizzo destinazione della transazione ed inoltre inserisce nell'header della transazione il valore R, ottenuto nel seguente modo:

$$R = r \cdot G$$

4. il portafoglio del destinatario, ad intervalli regolari, effettua una scansione della blockchain per verificare quali transazioni siano dirette al suo proprietario. Per ogni transazione trovata, estrae il parametro R ed utilizzando la chiave privata *(a, b)* effettua il seguente calcolo:

$$P' = hash(R \cdot a) \cdot G + b \cdot G$$

Se la condizione $P' = P$ è verificata, allora la transazione esaminata è diretta al proprietario del portafoglio.

B.4.2 Ring signature

É una tipologia di firma inventata nel 1991 da D. Chaum and E. van Heyst [100] ed in seguito migliorata da altri studiosi [101]. Tramite essa, non è possibile risalire con esattezza all'identità della persona che ha firmato un messaggio. In particolare, la chiave privata è unica ed è utilizzata per la firma, ma quest'ultima può essere verificata da un gruppo di chiavi pubbliche, chiamato *ring*. Pertanto è possibile affermare con certezza che la firma sia stata creata a partire dalla chiave privata di uno dei membri del gruppo, ma non è possibile risalire all'identità di tale membro.

Se applicata al contesto delle criptovalute, la ring signature inserita nell'input di una transazione può essere verificata tramite una qualsiasi delle chiavi del ring inserite nell'output della transazione precedente (Figura B.3). In tale modo, se il ring è costituito da numerosi membri, la probabilità di tracciare la transazione, ovvero di scoprirne il mittente, è ridotta.

B.4.3 Algoritmo CryptoNight

È una tipologia di algoritmo proof-of-work, utilizzata per rendere meno conveniente l'utilizzo di hardware dedicato per il mining. Difatti, come descritto nella Sezione B.3, il processo di mining richiede l'applicazione ripetuta di funzioni di hash crittografico che solitamente sono *CPU-bound*, ovvero sfruttano in maniera massiccia le risorse computazionali.

L'algoritmo CryptoNight è basato su una funzione di hash crittografico *memory-bound* dal momento che utilizza molteplici operazioni di I/O verso aree di memoria per calcolare il risultato. Pertanto le CPU general-purpose che dispongono di memoria ad accesso veloce, come ad esempio la memoria cache L3, sono favorite rispetto alle GPU o all'hardware ASIC.

Appendice C

Struttura dei messaggi scambiati nel protocollo Stratum

In questa sezione sono mostrati i dettagli della struttura, in formato JSON, dei tipici messaggi scambiati nel protocollo Stratum. In particolare:

- nel Listato 5 sono rappresentati i messaggi di sottoscrizione;
- nel Listato 6 sono rappresentati i messaggi per l'autorizzazione;
- nel Listato 7 sono rappresentati i messaggi per la notifica di un nuovo job;
- nel Listato 8 sono rappresentati i messaggi per ottenere la ricompensa per il lavoro svolto.

```
/*******************Richiesta SOTTOSCRIZIONE************************/
{
"id": 1,                                        //id
"method": "mining.subscribe",                   //metodo
"params": []                                    //parametri
}


/*******************Risposta SOTTOSCRIZIONE*************************/
{
"id":1,                                         //id
"result":                                       //risultato
[
["mining.notify","...."],                   //id connessione
"08000002",                                 //extranonce1
4                                           //dimensione extranonce2
],
"error":null                                    //errore
}
```

Listato 5: Messaggi di sottoscrizione usati nel protocollo Stratum.

```
/******************Richiesta AUTORIZZAZIONE*************************/
{
"id": 1,                                        //id
"method": "mining.authorize",                   //metodo
"params":                                       //parametri
[
"username",
"password"

]
}


/******************Risposta AUTORIZZAZIONE*************************/
{
"id":2,                                         //id
"result": true,                                 //risultato
"error":null                                    //errore
}
```

Listato 6: Messaggi di autorizzazione usati nel protocollo Stratum.

```
/******************Notifica DIFFICOLTA'*****************************/
{
"id": null,                                     //id
"method": "mining.set_difficulty",              //metodo
"params": [                                     //parametri
"difficoltà"
]
}


/******************Notifica NUOVO JOB*******************************/
{
"id": null,                             //id
"method": "mining.notify",              //metodo
"params":                               //parametri
[
"bf",                                   //identificativo del job
"4d16b6f85",                            //hash blocco precedente
"2f503253482f04b8864e500",              //parte iniziale transazione coinbase
"072f739903327048ed988a0",              //parte finale transazione coinbase
["...", "...", "..."],                  //lista degli hash dei nodi merkle tree
"00000002",                             //versione del blocco
"1c2ac4af",                             //target
"504e86b9",                             //timestamp creazione blocco
false                                   //opzione clean-job
]
}
```

Listato 7: Messaggi di notifica usati nel protocollo Stratum.

```
/******************Richiesta RICOMPENSA*****************************/
{
"id": 3,                                    //id
"method": "mining.submit",                  //metodo
"params":                                   //parametri
[
"username",                                //id del miner
"job",                                     //id del job
0000000001,                                //extranonce2
"504e86ed",                                //timestamp
"b2957c02"                                 //nonce

]
}

/******************Risposta RICOMPENSA*****************************/
{
"id":3,                                     //id
"result": true,                             //risultato
"error":null                                //errore
}
```

Listato 8: Messaggi per ottenere la ricompensa usati nel protocollo Stratum.

Appendice D

Manuale utente

D.1 Installazione

In questa sezione sarà descritto quali software sono necessari per eseguire gli script scritti nella parte implementativa di questa tesi.

È richiesto, inoltre, l'utilizzo del sistema operativo Linux.

D.1.1 Tstat 3.1.1

Per installare Tstat, procedere nel modo seguente:

1. scaricare l'archivio `tstat-3.1.1.tar.gz` dal sito[1] di Tstat;
2. scompattare l'archivio e seguire le istruzioni indicate nel file `README`.

D.1.2 Tshark 2.6.6

È possibile installare Tshark, tramite il seguente comando:

```
$ apt-get install tshark
```

D.1.3 Python 2.7

È possibile installare Python, tramite il seguente comando:

```
$ apt-get install python2.7
```

D.1.4 Matlab R2018b (64-bit)

Per installare Matlab, procedere nel modo seguente:

1. scaricare l'archivio `matlab_R2018b_glnxa64.zip` dal sito[2] di Matlab;
2. scompattare l'archivio e seguire le istruzioni contenute nel file `install_guide.pdf`.

---

[1] http://tstat.polito.it/software.php.

[2] https://it.mathworks.com/downloads/web_downloads

D.1.5 Script Python e Matlab

Gli script, creati per l'implementazione della tesi, sono contenuti in due cartelle:

- `script_python` contiene quattro script Python;
- `script_matlab` contiene 23 script Matlab.

Per eseguirli è sufficiente copiare tali cartelle in una posizione arbitraria del file system.

D.2 Procedimento per costruire il data set

In questa sezione sarà spiegato come ottenere, a partire da una generica cartella `catture` contenente i file pcap da analizzare, i file CSV dei campioni di traffico. Questi file possono essere letti da Matlab per ottenere il set dei dati su cui lavorare.

Il procedimento è il seguente:

1. eseguire lo script python `catture_troncate.py` per creare ulteriori file pcap contenenti altri campioni. Tali file pcap sono ottenuti considerando solo i primi N secondi di pacchetti di ogni flusso. Di default (parametro di input = 0), N è compreso tra 0.001 ed 1 secondo e viene incrementato ogni volta di 0.001. Il comando da digitare per eseguire lo script è il seguente:

   ```
   python catture_troncate.py catture/ (0/1)
   ```

   dove:

   - `catture`: è la cartella delle catture originali;
   - (0/1): specificando 0, sono utilizzati i parametri di default. Mentre specificando 1, lo script richiederà tre ulteriori parametri: valore di partenza di N (`start`), valore di arrivo di N (`stop`) e valore di incremento di N (`step`). Essi dovranno essere immessi, tramite una stringa con la seguente sintassi: `start;stop;step` ;

2. eseguire lo script python `estrazione_statistiche.py` per lanciare il programma Tstat, su ogni file pcap contenuto nella cartella `catture`. Il comando da digitare è il seguente:

   ```
   python estrazione_statistiche.py catture/
   ```

   Questo script riceve come parametro di input la cartella delle catture pcap ottenute al passo precedente, e produce come output, nella medesima cartella, i risultati derivanti dall'utilizzo di Tstat su ogni singolo file pcap. Inoltre, le catture pcap vengono copiate nella sotto-cartella `catture/catture_originali`. Tstat produce, per ogni file pcap, una cartella contenente alcuni file di log, tra cui i file `log_tcp_complete` e `log_tcp_nocomplete`, utilizzati nelle fasi seguenti del procedimento;

3. eseguire gli script `lettura_dati_tstat_tcp-complete.py` e `lettura_dati_tstat_tcp -nocomplete.py` per ottenere i file CSV da fare analizzare a Matlab. Il primo script legge i file `log_tcp_complete`, il secondo legge i file `log_tcp_nocomplete`. I comandi da digitare sono:

   ```
   python lettura_dati_tstat_tcp-complete.py catture/ opzione
   python lettura_datit_stat_tcp-nocomplete.py catture/ opzione
   ```

   Questi script ricevono come parametro la cartella contenente i risultati dell'analisi di Tstat, e una stringa che sarà utilizzata per identificare i file da cui estrarre i dati da inserire nel file CSV. Inoltre, la stringa comparirà nel nome del file CSV (si veda l'esempio proposto di seguito). Questo script aggiunge anche la colonna `malevolo`, con il valore:

- `'Y'` (ovvero potenzialmente malevolo), se come secondo parametro viene specificata una stringa diversa da `'good'`;
- `'N'` (ovvero non malevolo), se come secondo parametro viene specificata la stringa `'good'`;

Ad esempio, specificando come opzione `coinhive`, saranno prodotti i seguenti file CSV:

- `dataset_tcp_complete_coinhive.csv` se viene eseguito lo script `lettura_dati_tstat_tcp-complete.py`;
- `dataset_tcp_nocomplete_coinhive.csv` se viene eseguito lo script `lettura_dati_tstat_tcp-nocomplete.py`;

In entrambi i casi, l'attributo `malevolo` assumerà per ciascun campione il valore `'Y'`.

Le stringhe da specificare come secondo parametro di input, a seconda del traffico considerato, sono:

- `coinhive` da specificare in caso di catture Coinhive;
- `xmrstak-encrypted` da specificare in caso di catture XMR-Stak cifrate;
- `xmrstak-unencrypted` da specificare in caso di catture XMR-Stak in chiaro;
- `madominer` da specificare in caso di catture di MadoMiner;
- `good` da specificare in caso di catture di traffico legittimo;
- `pyBot_Bot_cc` da specificare in caso di catture di traffico di C&C di Pybot;
- `ufonet_Bot_cc` da specificare in caso di catture di traffico di C&C di UFOnet;
- `new` da specificare in caso di nuove catture di traffico, scelte dall'utente.

In quest'ultimo caso, lo script richiederà di inserire una stringa, che deve anche comparire nel nome dei file pcap da analizzare oppure `q` per terminare immediatamente l'esecuzione. Per quanto detto prima, se la stringa è diversa da `'good'`, tutti i campioni di tale data set saranno etichettati come potenzialmente malevoli;

4. infine, dopo aver avviato Matlab, è necessario eseguire lo script `estrazione_statistiche.m` per leggere i file CSV ed unirli in una tabella (`table`). Lo script ricerca questi file all'interno della cartella in cui è collocato. I data set generati sono salvati in file `mat`. Inoltre, questo script calcola la media su ciascun attributo dei campioni del data set, divisi nel gruppo dei campioni di traffico potenzialmente malevolo ed in quello dei campioni di traffico legittimo. I risultati di quest'ultima analisi sono salvati in un file CSV.

   All'inizio dello script, è richiesto l'inserimento da tastiera di una delle seguenti opzioni:

   - `1`: opzione da scegliere nel caso di data set contenente traffico legittimo e traffico Stratum. In questo caso sono creati i seguenti file mat:
     - `dataset_BUONO.mat` che rappresenta il data set di traffico legittimo;
     - `dataset_XMR_encrypted.mat` che rappresenta il data set di traffico cifrato generato da XMR-Stak;
     - `dataset_XMR_unencrypted.mat` che rappresenta il data set del traffico in chiaro generato da XMR-Stak;
     - `dataset_MADOMINER.mat` che rappresenta il data set del traffico generato da MadoMiner;
     - `dataset_COINHIVE.mat` che rappresenta il data set del traffico generato da Coinhive.

     Inoltre, viene creato il file `dataset_stratum.csv`, contenente le medie degli attributi dei campioni;
   - `2`: opzione da scegliere nel caso di data set contenente traffico legittimo e traffico di Pybot. In questo caso sono creati i seguenti file mat:
     - `dataset_BUONO.mat` che rappresenta il data set di traffico legittimo;

  - `dataset_pybot_cc.mat` che rappresenta il data set di traffico di C&C generato da Pybot.

  Inoltre, viene creato il file `dataset_pybot.csv`, contenente le medie degli attributi dei campioni;

- `3`: opzione da scegliere nel caso di data set contenente traffico legittimo e traffico di UFONet. In questo caso sono creati i seguenti file mat:
  - `dataset_BUONO.mat` che rappresenta il data set di traffico legittimo;
  - `dataset_ufonet_cc.mat` che rappresenta il data set di traffico di C&C generato da UFONet.

  Inoltre, viene creato il file `dataset_ufonet.csv`, contenente le medie degli attributi dei campioni.
- `4`: tramite questa opzione è possibile ottenere un data set a partire da file CSV contenenti del traffico a scelta dell'utente. A questo scopo, sarà richiesto l'inserimento del nome di un file CSV ottenuto a partire dai file `log_tcp_complete` e il nome di un file CSV ottenuto a partire dai file `log_tcp_nocomplete`.

  Verranno inoltre generati i seguenti file:
  - `dataset_utente.mat` che rappresenta il data set;
  - `dataset_utente.csv` che è il file CSV con le medie degli attributi del data set.

D.3 Procedimento per utilizzare i classificatori

In questa sezione sarà descritto come addestrare, validare (tramite la 10-fold cross-validation) e testare i classificatori descritti nella tesi. Lo script Matlab da considerare per la classificazione, è `classificatori.m`. All'inizio dell'esecuzione dello script, sarà richiesto l'inserimento da tastiera di una delle seguenti opzioni:

- `1`: opzione da scegliere per classificare il traffico sia cifrato che in chiaro di XMR-Stak;
- `2`: opzione da scegliere per classificare il traffico generato da MadoMiner;
- `3`: opzione da scegliere per classificare il traffico generato da Coinhive;
- `4`: opzione da scegliere per classificare il traffico generato da Pybot;
- `5`: opzione da scegliere per classificare il traffico generato da UFONet;
- `6`: opzione da scegliere per classificare il traffico scelto dall'utente e salvato in precedenza nel file `dataset_utente.mat`.

Dopo la scelta di questa opzione, è richiesto di specificare gli iperparametri da utilizzare per costruire i modelli:

- numero degli alberi della foresta casuale;
- funzione kernel da utilizzare per il modello SVM;
- numero di vicini k per il modello k-NN.

Infine sarà richiesto come suddividere in modo casuale il data set. In particolare, sarà richiesta la percentuale di osservazioni da utilizzare nel test set. Ad esempio, per questa tesi, è stata scelta la seguente proporzione: 70% dei campioni nel training set e il restante 30% nel test set. Per ottenere tale risultato, il valore da specificare è 0.30.

Durante l'esecuzione dello script in questione, per ogni classificatore, saranno mostrate a video le seguenti informazioni:

- il tempo impiegato per l'addestramento;
- le statistiche di accuratezza ottenute durante la fase di addestramento;
- le statistiche di accuratezza ottenute durante la 10-fold cross-validation;
- il tempo impiegato per il test;
- le statistiche di accuratezza ottenute dal classificatore per ciascun caso di test;
- la matrice di confusione per ciascun caso di test.

In base al classificatore scelto come migliore, è possibile ottenere il grafico sull'andamento dell'AUC al variare del numero dei pacchetti, utilizzando lo script `grafico_stats.m`. Durante la sua esecuzione, lo script richiederà l'inserimento dei seguenti parametri:

- il traffico classificato (XMR-Stak, Pybot, eccetera), ovvero una delle sei opzioni specificate all'inizio dello script `classificatori.m`;
- il modello che è stato selezionato:
  - `tree`: se il modello scelto è l'albero di decisione;
  - `rf`: se il modello scelto è la foresta casuale;
  - `knn`: se il modello scelto è k-NN;
  - `svm`: se il modello scelto è SVM.

Appendice E

Manuale del programmatore

In questo capitolo sarà descritto il codice utilizzato nell'implementazione della tesi, e come eventualmente modificarlo.

E.1 Descrizione degli script

Gli script Python sono i seguenti:

- `catture_troncate.py`: viene utilizzato per creare ulteriori file pcap, considerando solo i primi N secondi di pacchetti di ogni flusso;
- `estrazione_statistiche_pcap.py`: viene utilizzato per lanciare il programma Tstat, su ogni file pcap contenuto nella cartella passata come parametro di input;
- `lettura_dati_tstat_tcp-complete.py`: viene utilizzato per generare un file CSV a partire dai dati contenuti all'interno dei file `log_tcp_complete` ottenuti tramite Tstat. Inoltre, lo script in questione ha il compito di aggiungere l'etichetta di classe `malevolo`;
- `lettura_dati_tstat_tcp-nocomplete.py`: ha lo stesso compito dello script precedente, solo che i dati utilizzati per il file CSV provengono dai file `log_tcp_nocomplete`.

Gli script Matlab principali sono i seguenti:

- `estrazione_statistiche.m`: questo script si occupa di leggere i file CSV ottenuti tramite gli script Python precedenti, fonderli in una `table` ed estrarre la media dagli attributi dei campioni, dopo averli raggruppati in campioni legittimi e potenzialmente malevoli;
- `classificatori.m`: questo script consente di addestrare, cross-validare e testare i diversi classificatori descritti nella tesi, permettendo anche di variare alcuni iperparametri.

I restanti script[1] Matlab sono:

- `lettura_dataset:m`: questo script si occupa di fondere i file CSV letti, in una unica `table`, mantenendo solo gli attributi comuni;
- `crea_train_testset.m`: questo script si occupa di dividere in modo casuale, tramite la funzione di libreria `cvpartition`, i vari data set in training set e test set;

[1] Per evitare di appesantire la trattazione, poiché il codice degli script utilizzati per addestrare, validare e testare i diversi classificatori è simile (cambiano solo alcune funzioni di libreria), si descriveranno solo gli script relativi all'albero di decisione.

- `grafico_stats.m`: tale script si occupa di disegnare il grafico sull'andamento dell'AUC al variare del numero di pacchetti;
- `train_decision_tree.m` e sue varianti[2]: si occupa di addestrare il classificatore, calcolare e stampare le statistiche ottenute nell'addestramento;
- `accuracy_stats.m`: si occupa del calcolo delle sei statistiche, descritte in questa tesi, per valutare la bontà di un classificatore;
- `print_stats.m`: si occupa di mostrare a video le statistiche calcolate;
- `CV_decision_tree.m` e sue varianti[3]: questo script si occupa di calcolare la 10-fold cross-validation tramite la funzione di libreria `crossval`. Quest'ultima necessita di un puntatore alla funzione `tree_cv`;
- `tree_cv.m` e sue varianti[4]: questo script addestra i 10 modelli della cross-validation e ne calcola le statistiche;
- `test_decision_tree.m` e sue varianti[5]: questo script si occupa di calcolare e stampare a video le statistiche di accuratezza raggiunte nei vari casi di test. Inoltre, esso mostra la matrice di confusione ottenuta in ciascun caso.

E.2 Personalizzazione dei classificatori

Per aggiungere un nuovo iperparametro è necessario:

1. definire un nuova variabile globale nello script `classificatori.m` per contenere il valore dell'iperparametro;
2. abilitarne l'utilizzo nella funzione di libreria utilizzata per addestrare il classificatore, invocata all'interno dello script `train_decision_tree.m` e sue varianti.

Per aggiungere un nuovo classificatore (chiamato nel seguito `classifier`), è sufficiente definire le seguenti funzioni:

- `train_classifier`;
- `CV_classifier` e la funzione da essa invocata `classifier_cv`;
- `test_classifier`.

Tali funzioni dovranno essere invocate dallo script `classificatori.m`. Il loro contenuto sarà simile a quello descritto per le funzioni utilizzate per l'albero di decisione e per gli altri classificatori trattati in questa tesi.

E.3 Aggiunta di una nuova statistica

Per aggiungere una nuova statistica è necessario modificare lo script `accuracy_stats.m`, aggiungendo il codice necessario per il suo calcolo ed includendo il risultato all'interno della `struct ris` (valore di ritorno della funzione `accuracy_stats`). Per mostrare il risultato ottenuto, è necessario aggiungere il codice per la stampa a video della statistica, all'interno dello script `print_stats.m`.

[2]Le varianti sono: `train_random_forest.m`, `train_SVM.m` e `train_KNN.m`.

[3]Le varianti sono: `CV_random_forest.m`, `CV_SVM.m` e `CV_KNN.m`.

[4]Le varianti sono: `forest_cv.m`, `svm_cv.m` e `knn_cv.m`.

[5]Le varianti sono: `test_random_forest.m`, `test_SVM.m` e `test_KNN.m`.

Bibliografia

[1] Cisco, "Cisco Visual Networking Index: Forecast and Trends, 2017-2022." https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html, visitato il 22/02/2019

[2] A. Daly, "The legality of deep packet inspection", International Journal of Communications, No. 14 (2011), giugno 2010. https://ssrn.com/abstract=1628024

[3] J. Maddison, "Encrypted traffic reaches a new threshold", novembre 2018, https://www.networkcomputing.com/network-security/encrypted-traffic-reaches-new-threshold, visitato il 05/02/2019

[4] M. Frankel, "How many cryptocurrencies are there?", marzo 2018, https://www.fool.com/investing/2018/03/16/how-many-cryptocurrencies-are-there.aspx, visitato il 10/02/2019

[5] Verizon Enterprise, "2018 Data Breach Investigations Report." https://www.verizonenterprise.com/verizon-insights-lab/dbir/

[6] K. R. Fall and R. W. Stevens, "TCP/IP illustrated, vol. 1", Addison-Wesley, 2012, ISBN: 978-0-321-33631-6

[7] J. F. Kurose and K. W. Ross, "Computer networking: A top-down approach", Pearson, 2013, ISBN: 978-0-13-285620-1

[8] H. Zimmermann, "OSI reference model: The ISO model of architecture for open systems interconnection", IEEE Transactions on communications, vol. 28, no. 4, 1980, pp. 425–432, DOI 10.1109/TCOM.1980.1094702

[9] J. Postel, "Transmission Control Protocol (TCP)", STD 7, RFC Editor, settembre 1981. http://www.rfc-editor.org/rfc/rfc793.txt

[10] V. Paxson, M. Allman, J. Chu, and M. Sargent, "Computing TCP's retransmission timer", RFC 6298, RFC Editor, giugno 2011. http://www.rfc-editor.org/rfc/rfc6298.txt

[11] P. Karn and C. Partridge, "Improving Round-Trip Time (RTT) estimates in reliable transport protocols", ACM SIGCOMM Computer Communication review, 1987, pp. 2–7, DOI 10.1145/55483.55484. Stowe (USA), 11-13 agosto

[12] V. Jacobson, B. Braden, and D. Borman, "TCP extensions for high performance", RFC 1323, RFC Editor, maggio 1992. http://www.rfc-editor.org/rfc/rfc1323.txt

[13] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP selective acknowledgment options", RFC 2018, RFC Editor, ottobre 1996. https://tools.ietf.org/html/rfc2018

[14] M. Allman, V. Paxson, and E. Blanton, "TCP congestion control", RFC 5681, RFC Editor, settembre 2009. http://www.rfc-editor.org/rfc/rfc5681.txt

[15] E. Rescorla, "HTTP over TLS", RFC 2818, RFC Editor, maggio 2000. http://www.rfc-editor.org/rfc/rfc2818.txt

[16] T. Dierks and C. Allen, "The TLS protocol version 1.0", RFC 2246, RFC Editor, gennaio 1999. http://www.rfc-editor.org/rfc/rfc2246.txt

[17] W. Stallings, "Cryptography and network security: principles and practice", Pearson, 2014, ISBN: 978-0-13-335469-0

[18] FIPS Pub 197, "Advanced Encryption Standard (AES)", National Institute of Standards and Technology (NIST), novembre 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

[19] H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-hashing for message authentication", RFC 2104, RFC Editor, febbraio 1997. http://www.rfc-editor.org/rfc/rfc2104.txt

[20] W. Diffie and M. Hellman, "New directions in cryptography", IEEE transactions on Information Theory, vol. 22, no. 6, 1976, pp. 644–654, DOI 10.1109/TIT.1976.1055638

[21] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk, "Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile", RFC 5280, RFC Editor, maggio 2008. http://www.rfc-editor.org/rfc/rfc5280.txt

[22] R. L. Rivest, A. Shamir, and L. M. Adleman, "Cryptographic communications system and method", settembre 1983, US Patent 4,405,829, https://patents.google.com/patent/US4405829A/en

[23] D. Wagner and B. Schneier, "Analysis of the SSL 3.0 protocol", The Second USENIX Workshop on Electronic Commerce Proceedings, 1996, pp. 29–40. https://dl.acm.org/citation.cfm?id=1267171; Oakland (USA), 18-21 novembre

[24] F. Risso and L. Degioanni, "An architecture for high performance network analysis", Computers and Communications, 2001. Proceedings. Sixth IEEE Symposium on, 2001, pp. 686–693, DOI 10.1109/ISCC.2001.935450. Hammamet (Tunisia), 5 luglio

[25] S. L. Garfinkel and M. Shick, "Passive TCP reconstruction and forensic analysis with Tcpflow", 2013, https://calhoun.nps.edu/handle/10945/36026

[26] R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Nielsen, L. Masinter, P. J. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1", RFC 2616, RFC Editor, giugno 1999. http://www.rfc-editor.org/rfc/rfc2616.txt

[27] J. Klensin, "Simple Mail Transfer Protocol (SMTP)", RFC 5321, RFC Editor, ottobre 2008. http://www.rfc-editor.org/rfc/rfc5321.txt

[28] T. Bray, "The JavaScript Object Notation (JSON) data interchange format", STD 90, RFC Editor, dicembre 2017. https://www.rfc-editor.org/rfc/rfc8259.txt

[29] P. Mockapetris, "Domain names: Concepts and facilities", RFC 882, RFC Editor, novembre 1983. http://www.rfc-editor.org/rfc/rfc882.txt

[30] Y. Shafranovich, "Common format and MIME type for Comma-Separated Values (CSV) files", RFC 4180, RFC Editor, ottobre 2005. http://www.rfc-editor.org/rfc/rfc4180.txt

[31] A. Finamore, M. Mellia, M. Meo, M. M. Munafo, Politecnico Di Torino, and D. Rossi, "Experiences of Internet traffic monitoring with Tstat", IEEE Network, vol. 25, maggio 2011, pp. 8–14, DOI 10.1109/mnet.2011.5772055

[32] A. Antonopoulos, "Mastering bitcoin", O'Reilly, 2017, ISBN: 978-1-491-95438-6

[33] A. S. Tanenbaum and M. van Steen, "Distributed systems: principles and paradigms", luglio 2011, ISBN: 978-0273760597

[34] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system", 2008. https://bitcoin.org/bitcoin.pdf

[35] D. Drescher, "Blockchain basics", Apress, 2017, ISBN: 978-1-4842-2603-2

[36] S. Eskandari, A. Leoutsarakos, T. Mursch, and J. Clark, "A first look at browser-based cryptojacking", 2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), 2018, DOI 10.1109/eurospw.2018.00014. Londra (UK), 23-27 aprile

[37] M. Palatinus, "Stratum mining protocol - ASIC ready", settembre 2012, https://mining.bitcoin.cz/stratum-mining, visitato il 22/10/2018

[38] R. Recabarren and B. Carbunar, "Hardening Stratum, the Bitcoin Pool Mining Protocol", Proceedings on Privacy Enhancing Technologies, vol. 2017, no. 3, 2017, pp. 57 – 74, DOI 10.1515/popets-2017-0028

[39] I. Fette and A. Melnikov, "The WebSocket protocol", RFC 6455, RFC Editor, dicembre 2011. http://www.rfc-editor.org/rfc/rfc6455.txt

[40] M. Musch, C. Wressnegger, M. Johns, and K. Rieck, "Web-based cryptojacking in the wild", arXiv preprint, 2018, DOI arXiv:1808.09474

[41] D. Y. Huang, H. Dharmdasani, S. Meiklejohn, V. Dave, C. Grier, D. McCoy, S. Savage, N. Weaver, A. C. Snoeren, and K. Levchenko, "Botcoin: Monetizing stolen cycles", NDSS, 2014. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.434.9481&rep=rep1&type=pdf; San Diego (USA), 23-26 febbraio

[42] R. K. Konoth, E. Vineti, V. Moonsamy, M. Lindorfer, C. Kruegel, H. Bos, and G. Vigna, "Minesweeper: An in-depth look into drive-by cryptocurrency mining and its defense", Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, 2018, pp. 1714–1730, DOI 10.1145/3243734.3243858. Toronto (Canada), 15-19 ottobre

[43] C. Liu and J. C. Chen, "Malvertising campaign abuses Google's doubleclick to deliver cryptocurrency miners", gennaio 2018, https://blog.trendmicro.com/trendlabs-security-intelligence/malvertising-campaign-abuses-googles-doubleclick/, visitato il 29/10/2018

[44] M. Schiaffino, "Un miner nascosto in un'estensione di Chrome con 100.000 utenti", gennaio 2018, https://www.securityinfo.it/2018/01/02/un-miner-nascosto-unestensione-chrome-100-000-utenti/, visitato il 5/11/2018

[45] M. Mezzalama, A. Lioy, and H. Metwalley, "Anatomia del malware", Mondo Digitale, no. 47, 2013, p. 2. http://www.polito.it.pgp.net/doc/public/torsec_malware_2013.pdf

[46] T. Micro, "Massive WannaCry/Wcry ransomware attack hits various countries", maggio 2017, https://blog.trendmicro.com/trendlabs-security-intelligence/massive-wannacrywcry-ransomware-attack-hits-various-countries/, visitato il 10/01/2019

[47] J. Quinn, "Madominer part 1 - install", settembre 2018, https://www.alienvault.com/blogs/labs-research/madominer-part-1-install, visitato il 15/01/2019

[48] J. Quinn, "Madominer part 2 - mask", ottobre 2018, https://www.alienvault.com/blogs/labs-research/madominer-part-2-mask, visitato il 15/01/2019

[49] Microsoft, "Microsoft SMB protocol and CIFS protocol overview", maggio 2018, https://docs.microsoft.com/it-it/windows/desktop/FileIO/microsoft-smb-protocol-and-cifs-protocol-overview, visitato il 20/12/2018

[50] D. Yin, "Deep analysis of Esteemaudit", maggio 2017, https://www.fortinet.com/blog/threat-research/deep-analysis-of-esteemaudit.html, visitato il 05/12/2018

[51] Microsoft, "Descrizione del protocollo RDP (Remote Desktop Protocol)", aprile 2018, https://support.microsoft.com/it-it/help/186607/understanding-the-remote-desktop-protocol-rdp, visitato il 21/12/2018

[52] G. Vormayr, T. Zseby, and J. Fabini, "Botnet communication patterns", IEEE communications surveys and tutorials, vol. 19, no. 4, 2017, pp. 2768–2796, DOI 10.1109/COMST.2017.2749442

[53] N. Slepogin, "Dridex: A history of evolution", maggio 2017, https://securelist.com/dridex-a-history-of-evolution/78531/, visitato il 12/01/2019

[54] P. Cichonski, T. Millar, T. Grance, and K. Scarfone, "Computer security incident handling guide", NIST Special Publication, vol. 800, no. 61, 2012, pp. 1–147. http://infohost.nmt.edu/~sfs/Regs/sp800-61.pdf

[55] Radware, "A quick history of IoT botnets", marzo 2018, https://blog.radware.com/uncategorized/2018/03/history-of-iot-botnets/, visitato il 30/01/2019

[56] J. Postel and J. Reynolds, "Telnet protocol specification", STD 8, RFC Editor, maggio 1983. http://www.rfc-editor.org/rfc/rfc854.txt

[57] R. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and content", RFC 7231, RFC Editor, giugno 2014. http://www.rfc-editor.org/rfc/rfc7231.txt

[58] W. Eddy, "TCP SYN flooding attacks and common mitigations", RFC 4987, RFC Editor, agosto 2007. https://tools.ietf.org/html/rfc4987

[59] J. Postel, "Internet Control Message Protocol", STD 5, RFC Editor, settembre 1981. http://www.rfc-editor.org/rfc/rfc792.txt

[60] D. Senie, "Changing the default for directed broadcasts in routers", BCP 34, RFC Editor, agosto 1999. https://tools.ietf.org/html/rfc2644

[61] S. Shalev-Shwartz and S. Ben-David, "Understanding Machine Learning: From theory to algorithms", Cambridge University Press, maggio 2014, ISBN: 978-1107057135

[62] P.-N. Tan, M. Steinbach, and V. Kumar, "Introduction to data mining", Pearson, maggio 2005, ISBN: 0-321-42052-7

[63] J. Han, M. Kamber, and J. Pei, "Data mining: Concepts and techniques", Morgan Kaufmann Publishers, luglio 2011, ISBN: 978-0123814791

[64] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks", Information Processing & Management, vol. 45, no. 4, 2009, pp. 427–437, DOI 10.1016/j.ipm.2009.03.002

[65] Y. Sasaki, "The truth of the F-measure", Teach Tutor mater, vol. 1, no. 5, 2007, pp. 1–5. https://www.cs.odu.edu/~mukka/cs795sum09dm/Lecturenotes/Day3/F-measure-YS-26Oct07.pdf

[66] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of Machine Learning algorithms", Advances in Neural Information Processing Systems 25 (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 2951–2959, Curran Associates, Inc., 2012. http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf

[67] M. Kuhn and K. Johnson, "Applied predictive modeling", Springer, 2013, ISBN: 978-1461468486

[68] J. W. Hunt and T. G. Szymanski, "A fast algorithm for computing longest common subsequences", Commun. ACM, vol. 20, maggio 1977, pp. 350–353, DOI 10.1145/359581.359603

[69] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers", Proceedings of the fifth annual workshop on Computational learning theory, 1992, pp. 144–152, DOI 10.1145/130385.130401. Pennsylvania (USA), 27-29 luglio

[70] Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard, and C.-J. Lin, "Training and testing low-degree polynomial data mappings via linear SVM", Journal of Machine Learning Research, vol. 11, no. aprile, 2010, pp. 1471–1490. http://www.jmlr.org/papers/v11/chang10a.html

[71] E. Fix and J. L. Hodges Jr, "Discriminatory analysis-nonparametric discrimination: consistency properties", tech. rep., Berkeley University, 1951. https://apps.dtic.mil/dtic/tr/fulltext/u2/a800276.pdf

[72] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques", Emerging artificial intelligence applications in computer engineering, vol. 160, 2007, pp. 3–24. https://dl.acm.org/citation.cfm?id=1566773

[73] L. Breiman, "Random forests", Machine learning, vol. 45, no. 1, 2001, pp. 5–32, DOI 10.1023/A:1010933404324

[74] U. Iqbal, Z. Shafiq, P. Snyder, S. Zhu, Z. Qian, and B. Livshits, "AdGraph: A machine learning approach to automatic and effective adblocking", arXiv preprint, 2018, DOI arXiv:1805.09155

[75] J. D. P. Rodriguez and J. Posegga, "RAPID: Resource and api-based detection against in-browser miners", Proceedings of the 34th Annual Computer Security Applications Conference, 2018, pp. 313–326, DOI 10.1145/3274694.3274735. San Juan (USA), 03-07 dicembre

[76] M. Saad, A. Khormali, and A. Mohaisen, "End-to-end analysis of in-browser cryptojacking", Computing Research Repository (CoRR), vol. abs/1809.02152, 2018. https://arxiv.org/abs/1809.02152

[77] M. Sikorski and A. Honig, "Practical malware analysis", No starch press, 2012, ISBN: 978-1-59327-290-6

[78] J. C. Bezdek, R. Ehrlich, and W. Full, "FCM: The Fuzzy C-Means clustering algorithm", Computers & Geosciences, vol. 10, no. 2-3, 1984, pp. 191–203, DOI 10.1016/0098-3004(84)90020-7

[79] D. Carlin, P. O'Kane, S. Sezer, and J. Burgess, "Detecting cryptomining using dynamic analysis", 2018 16th Annual Conference on Privacy, Security and Trust (PST), 2018, pp. 1–6, DOI 10.1109/PST.2018.8514167. Belfast (UK), 28-30 agosto

[80] S. Saad, I. Traore, A. Ghorbani, B. Sayed, D. Zhao, W. Lu, J. Felix, and P. Hakimian, "Detecting P2P botnets through network behavior analysis and machine learning", 2011 Ninth Annual International Conference on Privacy, Security and Trust, 2011, pp. 174–180, DOI 10.1109/PST.2011.5971980. Montreal (Canada), 19-21 luglio

[81] F. Haddadi, J. Morgan, E. Gomes Filho, and A. N. Zincir-Heywood, "Botnet behaviour analysis using IP flows: with http filters using classifiers", Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on, 2014, pp. 7–12, DOI 10.1109/WAINA.2014.19. Victoria (Canada), 13-16 maggio

[82] M. Stevanovic and J. M. Pedersen, "An efficient flow-based botnet detection using supervised machine learning", Computing, Networking and Communications (ICNC), 2014 International Conference on, 2014, pp. 797–801, DOI 10.1109/ICCNC.2014.6785439. Honolulu (USA), 3-6 febbraio

[83] C. Garretson, "Storm: the largest botnet in the world?." https://www.networkworld.com/article/2286172/lan-wan/storm--the-largest-botnet-in-the-world-.html, visitato il 08/01/2019

[84] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection", IEEE Communications Surveys & Tutorials, vol. 18, no. 2, 2016, pp. 1153–1176, DOI 10.1109/COMST.2015.2494502

[85] D. Sumeet and D. Xian, "Data mining and machine learning in cybersecurity", CRC Press, 2011, ISBN: 978-1-4398-3943-0

[86] R. Shirey, "Internet security glossary", RFC 2828, RFC Editor, maggio 2000. https://www.ietf.org/rfc/rfc2828.txt

[87] L. Bilge, S. Sen, D. Balzarotti, E. Kirda, and C. Kruegel, "Exposure: A passive DNS analysis service to detect and report malicious domains", ACM Transactions on Information and System Security (TISSEC), vol. 16, no. 4, 2014, p. 14, DOI 10.1145/2584679

[88] S. Kumar, A. Faizan, A. Viinikainen, and T. Hamalainen, "MLSPD-Machine Learning Based Spam and Phishing Detection", International Conference on Computational Social Networks, 2018, pp. 510–522, DOI 10.1007/978-3-030-04648-4_43. Shanghai (Cina), 18-20 dicembre

[89] B. Anderson and D. McGrew, "Machine learning for encrypted malware traffic classification: accounting for noisy labels and non-stationarity", Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2017, pp. 1723–1732, DOI 10.1145/3097983.3098163. Halifax (Canada), 13-17 agosto

[90] T. Auld, A. W. Moore, and S. F. Gull, "Bayesian neural networks for internet traffic classification", IEEE Transactions on neural networks, vol. 18, no. 1, 2007, pp. 223–239, DOI 10.1109/TNN.2006.883010

[91] S. Haber and W. S. Stornetta, "How to time-stamp a digital document", Conference on the Theory and Application of Cryptography, 1990, pp. 437–455, DOI 10.1007/BF00196791. Santa Barbara (USA), 11-15 agosto

[92] K. Okupski, "Bitcoin developer reference", tech. rep., Technische Universiteit Eindhoven, The Netherlands, 2016. https://lopp.net/pdf/Bitcoin_Developer_Reference.pdf

[93] C. Clark, "Bitcoin internals: A technical guide to bitcoin", novembre 2013. Kindle store eBook

[94] FIPS Pub 180-2, "Secure Hash Standard", National Institute of Standards and Technology (NIST), agosto 2002. https://csrc.nist.gov/publications/detail/fips/180/2/archive/2004-02-25

[95] R. C. Merkle, "A digital signature based on a conventional encryption function", Conference on the theory and application of cryptographic techniques, 1987, pp. 369–378, DOI 10.1007/3-540-48184-2_32. Santa Barbara (USA), 16-20 agosto

[96] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures", Journal of Cryptographic Engineering, vol. 2, no. 2, 2012, pp. 77–89, DOI 10.1007/s13389-012-0027-1

[97] N. Van Saberhagen, "Cryptonote v 2.0", 2013, http://www.secg.org/SEC2-Ver-1.0.pdf

[98] Chainalysis, "The changing nature of cryptocrime." https://blog.chainalysis.com/reports/report-the-changing-nature-of-cryptocrime; visitato il 15/11/2018

[99] S. Blake-Wilson, "SEC1: Elliptic curve cryptography", Certicom Research, Working Draft Ver. 0.4, Toronto, Canada, 1999. http://www.secg.org/sec1-v2.pdf

[100] D. Chaum and E. Van Heyst, "Group signatures", Workshop on the Theory and Application of of Cryptographic Techniques, 1991, pp. 257–265. https://chaum.com/publications/Group_Signatures.pdf, Brighton (UK), 8-11 aprile

[101] R. L. Rivest, A. Shamir, and Y. Tauman, "How to leak a secret", Advances in Cryptology — ASIACRYPT 2001, 2001, pp. 552–565, DOI 10.1007/3-540-45682-1_32. Gold Coast (Australia), 9-13 dicembre