

UNIVERSITÀ DEGLI STUDI DI FIRENZE

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea Magistrale in Informatica



Progetto per il corso di  
ANALISI QUANTITATIVA DEI  
SISTEMI

**GRUPPO 3**

Bernini Riccardo matr. 5435313

Papini Tommaso matr. 5537529

A.A. 2012/2013



# Indice

<b>Testo del progetto</b>	<b>1</b>
<b>Introduzione a Snort</b>	<b>3</b>
<b>1 Pianificazione delle attività</b>	<b>5</b>
1.1 Sfasamento dei clock . . . . .	6
<b>2 Instrumentazione del sistema</b>	<b>11</b>
2.1 Installazione e configurazione di Snort . . . . .	11
2.2 Tool di fault injection . . . . .	12
2.2.1 NMap . . . . .	13
2.2.2 Ping . . . . .	14
2.2.3 Metasploit . . . . .	14
2.2.4 Traceroute . . . . .	15
2.3 Tool di data integration e analisi . . . . .	16
<b>3 Analisi temporale</b>	<b>19</b>
3.1 Nmap . . . . .	19
3.1.1 Nmap, -A vs Default . . . . .	19
3.1.2 Nmap, --osscan-guess vs Default . . . . .	20
3.1.3 Nmap, --min-rate 100/300/500 vs Default . . . . .	21
3.1.4 Nmap, -T 2/3/4/5 vs Default . . . . .	22
3.1.5 Nmap, --version-all/light vs Default . . . . .	24
3.2 Ping . . . . .	25
3.3 Metasploit . . . . .	26
3.4 Traceroute . . . . .	27
3.4.1 Traceroute, -N 20/30/40 vs Default . . . . .	27
3.4.2 Traceroute, -T vs Default . . . . .	27
3.4.3 Traceroute -I vs Default . . . . .	28
3.5 Protocolli . . . . .	30

3.6 Alert generati . . . . .	32
<b>Conclusioni</b>	<b>37</b>
<b>A Codice</b>	<b>39</b>

# Testo del progetto

## Esame di Analisi Quantitativa dei Sistemi – Progetto

Appello del 26 Luglio 2013

Progetto per il gruppo n. 3 - Analisi Sperimentale

Riccardo Bernini - matricola 5435313

Tommaso Papini - matricola 5537529

**Obiettivo del progetto è caratterizzare il comportamento temporale del tool Snort. La consegna del progetto dovrà essere:**

- una relazione che descrive la metodologia di valutazione sperimentale adottata e discute i risultati
- un archivio contente: file di log, eventuale codice sviluppato, dump di un eventuale database, eventuali regole di SNORT sviluppate.

Riportiamo di seguito dettagli del progetto, al fine di chiarirne l'obiettivo. Il progetto richiede di affrontare un esercizio di valutazione sperimentale, con applicazione della metodologia descritta a lezione, volto a caratterizzare Snort e la sua capacità di attack detection da un punto di vista temporale. Riportiamo gli elementi principali del progetto (si noti bene che tali elementi non sono necessariamente gli unici o imprescindibili, e si lascia la possibilità di effettuare variazioni, specialmente sul livello di dettaglio):

- Creare il sistema di misura e configurare opportunamente il tool SNORT per agire come Intrusion Detection System. Presumibilmente, il sistema risultante sarà composto da due nodi connessi:
  - Nodo A svolge il ruolo di “nodo da proteggere”, su cui è installato Snort
  - Nodo B svolge il ruolo di sistema attaccante. Notare che non si pongono comunque vincoli per l'utilizzo di differenti configurazioni.

- Definire un faultload da generare ed esercitare sul sistema. Non si pongono vincoli sulla complessità del faultload o sulla selezione degli strumenti per la sua generazione. Può essere necessario costituire delle regole Snort adeguate per identificare per il faultload definito.
  - Esempi di faultload che possono essere realizzati sono: iniettare un elevato numero di ping da parte di un singolo nodo, oppure effettuare un port scan, effettuare un SQL injection.
  - A titolo di esempio, menzioniamo alcuni strumenti che possono essere utilizzati per la generazione del faultload: ping, nmap, traceroute, MetaSploit, sqlsus.
- Introdurre nel sistema opportuni meccanismi di monitoring (sonde) per rilevare le misure di tempo richieste. Possibili misure da rilevare sono: i) l'istante di tempo in cui si inietta l'attacco sul nodo A; ii) l'istante di tempo in cui l'attacco è individuato da parte di Snort; iii) l'istante di tempo in cui l'attacco è generato dal nodo B; v) il ritardo di trasmissione tra i due nodi A e B; iv) l'istante di tempo in cui l'attacco si verifica sul sistema, nel caso in cui Snort non sia in grado di rilevarlo; v) etc.
  - Si noti che gli istanti di tempo menzionati sopra permettono di calcolare intervalli quali: i) il tempo di detection dell'attacco (dall'inizio dell'attacco sul nodo A alla sua rilevazione da parte di Snort); ii) la durata dell'attacco (dalla generazione dell'attacco su B alla sua rilevazione da parte di Snort, o al suo completamento); iii) etc.
- Eseguire gli esperimenti per il faultload selezionato, raccogliendo i dati.
- Presentare i risultati raccolti, cercando di motivarli opportunamente. È importante concentrarsi sulle misure temporali raccolte. Discutere in particolare i tempi di rilevazione (dall'iniezione dell'attacco alla sua rilevazione da parte di Snort), osservare la loro varianza ed identificare eventuali distribuzioni.

**Suggerimento 1.** Per raccogliere le misure temporali, si possono utilizzare varie soluzioni: leggere i dati contenuti nel log di snort, utilizzare strumenti per il monitoring della rete, oppure creare apposite procedure in un qualsiasi linguaggio.

**Suggerimento 2.** Si può esercitare il sistema utilizzando diverse configurazioni di Snort per cercare di studiare la perturbazione introdotta da Snort sul sistema. Ad esempio, si può osservare come variano le misure quando il numero di regole applicate da Snort sono ridotte al minimo indispensabile, oppure quando parallelamente al faultload si utilizzano differenti workload.

Per chiarimenti, contattare Andrea Ceccarelli [andrea.ceccarelli@unifi.it](mailto:andrea.ceccarelli@unifi.it) ed il docente in CC.

# Introduzione a Snort

Snort è un software Open Source, che realizza un "semplice" sistema per il rilevamento di intrusioni in rete (NIDS) basato sulla libreria libpcap (<http://www-nrg.ee.lbl.gov/>). Il termine "semplice" non deve essere frainteso: si tratta infatti di un software molto efficace, con capacità di eseguire in tempo reale sia analisi del traffico che packet logging (utile per debuggare il traffico della rete) su IP networks. Esso può compiere analisi dei protocolli, può individuare una varietà di attacchi e probes, così come buffer overflow, port scanning, attacchi CGI e probes SMB. Guardando i file di log si riesce a determinare a quale tipo di attacco si è stati soggetti. Usa un semplice linguaggio di descrizione di regole flessibili e potenti che consente di descrivere che tipo di traffico dovrebbe essere catturato o ignorato e di individuare traffico ostile o semplicemente sospetto sulla rete.

Snort può lavorare in diversi modi:

- **Packet Sniffer:** è capace di ispezionare il carico dei pacchetti sulla rete, decodificando il livello di applicazione di un pacchetto e catalogando il traffico basato su un certo contenuto di dati. Può anche filtrare il traffico attraverso BPF (comandi Berkley Packet Filter), i quali lo rendono flessibile nel catalogare specifici tipi di dati basati sul suo insieme di regole.
- **Packet Logger:** può anche effettuare il log dei pacchetti su linea di comando indirizzati ad una specifica locazione, in un syslog, ed invia alert a video. Uno dei migliori vantaggi è che esso effettua il log in formato leggibile decodificato in una directory basata su IP sorgenti. Esegue anche il log di pacchetti in formato binario su un singolo file.
- **Intrusion Detection:** può essere usato come IDS su reti dove sono richieste alte prestazioni. Precisamente può essere posizionato tra il firewall, che controlla una sottorete, e la linea esterna non sicura, analizzando in questo modo sia il traffico diretto al firewall, che il traffico nella sottorete controllata. Ha un piccolo sistema di firme ed è disegnato per essere un tool veloce di alerting per gli amministratori quando sono individuate attività sospette.

Nei capitoli successivi andremo ad analizzare più in dettaglio il comportamento temporale di Snort, configurato per funzionare come IDS.



## Pianificazione delle attività

Il progetto consiste nel valutare da un punto di vista temporale i risultati degli esperimenti eseguiti con Snort. Per fare questo abbiamo utilizzato due macchine con OS Ubuntu 12.04 su cui, seguendo i passi descritti nel capitolo successivo, abbiamo installato e opportunamente configurato Snort come IDS. Il sistema alla base delle analisi quindi si compone di una prima macchina che avrà funzione di sistema da difendere (nodo A) ed una seconda con la funzione di attaccante (nodo B) collegate ad una rete wifi locale. La necessità di eseguire Snort su entrambe le componenti del sistema è legata alla necessità di dover tracciare i tempi d'invio, d'iniezione e d'individuazione delle operazioni di fault injection. In pratica, nel nodo A Snort provvederà sia a loggare tutto il traffico diretto verso se stesso dal nodo B, sia a generare gli alert dovuti agli attacchi individuati, mentre nel nodo B provvederà a loggare tutti i pacchetti diretti da B ad A, così da tracciare i tempi d'invio degli attacchi. I codici relativi alle regole Snort che sono state scritte per permettere ai due nodi di fare sia da Packet Sniffer che da Packet Logger possono essere trovati nell'Appendice A, Codici A.2 e A.3.

Il sistema di misura si compone quindi dei due file di log generati rispettivamente sui due nodi che, assieme all'output dei software usati per generare la fault injection, permettono di registrare tutta la storia temporale degli attacchi al fine di verificare l'azione delle regole di Snort.

Purtroppo, in un primo momento di analisi del problema e di sperimentazione, ritenevamo sufficiente loggare soltanto i pacchetti in entrata ed in uscita, rispettivamente, nel nodo A e dal nodo B. Soltanto in fase di analisi ci siamo resi conto che, senza il flusso di dati in entrambe le direzioni ed in entrambi i nodi, alcune misure (come ad esempio la durata totale dell'attacco) non potevano essere calcolate correttamente. Per ragioni di tempistiche, abbiamo deciso di mantenere gli esperimenti e i relativi log così com'erano, precisando comunque che alcune misure, per questo motivo, sono state approssimate e consci del fatto che un'applicazione più rigida della metodologia di testing avrebbe richiesto la ripetizione di tutti gli esperimenti con le nuove specifiche.

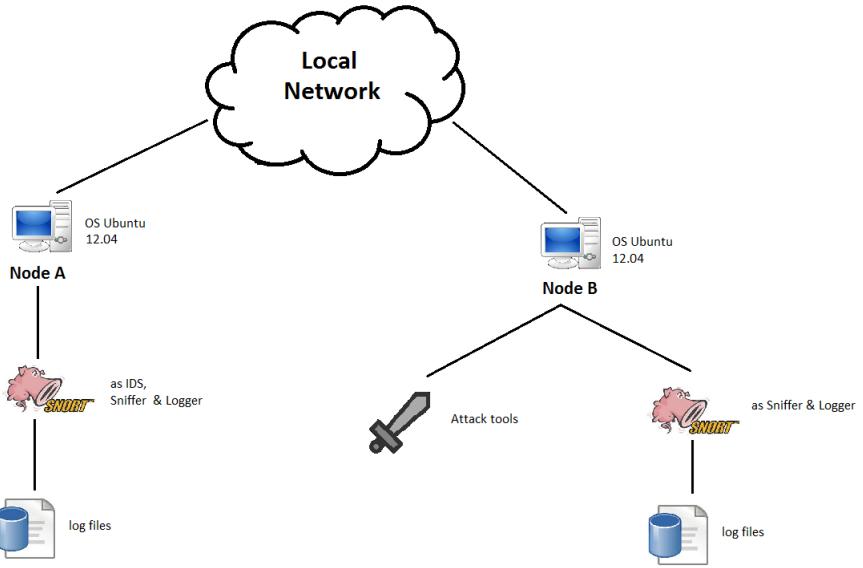


Figura 1.1: Schema della configurazione utilizzata nei vari esperimenti.

## 1.1 Sfasamento dei clock

Il “semplice” sistema di misura brevemente descritto in precedenza deve però tenere conto del problema dello sfasamento dei clock sulle due macchine, è infatti errato assumere che entrambe abbiano la stessa visione del tempo che è appunto governata dal clock interno a ciascuna macchina. Inoltre, anche se in un certo istante il clock delle due macchine segnasse lo stesso valore, non è detto che questo valga sempre: è infatti possibile che i due clock si allontanino o si avvicinino durante il tempo, rendendo impossibile fare qualsiasi stima di natura statica.

Per fare sì che i tempi registrati nel nodo A siano consistenti a quelli registrati nel nodo B è necessario mettere in piedi un processo di sincronizzazione. Un protocollo utile per, parzialmente, risolvere il problema è il Network Time Protocol, in sigla NTP, un protocollo per la sincronizzazione degli orologi di computer all'interno di una rete a commutazione di pacchetto.

Come prima cosa quindi è necessario scaricare ed installare su entrambe le macchine il pacchetto *ntp* che mette a disposizione varie funzionalità per la sincronizzazione.

Un primo approccio può fare uso del comando **ntpd**, il quale mette in esecuzione un processo *demone* che in *background* provvede a sincronizzare l'orologio di sistema con quello di un server opportunamente specificato nel file (*/etc/ntp.conf*). Questo approccio ha però due problemi: richiede alcune ore per arrivare a registrare una buona approssimazione del tempo del server e continuando la sua esecuzione in *background* e in modo asincrono rispetto all'altra macchina che compone il sistema può andare a modificare la visione del tempo di una macchina durante l'esecuzione di un'esperimento rischiando

di produrre inconsistenze tra i dati raccolti. Per fornire evidenza di questa situazione riportiamo qui di seguito due frammenti di log relativi al nodo A e al nodo B durante un esperimento di fault injection (alcuni dettagli non rilevanti sono stati omessi per una maggiore chiarezza di lettura):

**Nodo A:**

TIMESTAMP	MSG	PROTOCOL	FROM_IP	TO_IP
07/09-17:38:37.271491	ICMP PING BSDtype	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:37.271491	ICMP PING *NIX	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:37.271491	PACKET RECEIVED	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:37.271491	ICMP PING	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:37.271544	ICMP Echo Reply	ICMP	192.168.137.180	192.168.137.170
07/09-17:38:38.194984	ICMP PING BSDtype	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:38.194984	ICMP PING *NIX	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:38.194984	PACKET RECEIVED	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:38.194984	ICMP PING	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:38.195038	ICMP Echo Reply	ICMP	192.168.137.180	192.168.137.170
07/09-17:38:39.116375	ICMP PING BSDtype	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:39.116375	ICMP PING *NIX	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:39.116375	PACKET RECEIVED	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:39.116375	ICMP PING	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:39.116406	ICMP Echo Reply	ICMP	192.168.137.180	192.168.137.170
07/09-17:38:40.140671	ICMP PING BSDtype	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:40.140671	ICMP PING *NIX	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:40.140671	PACKET RECEIVED	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:40.140671	ICMP PING	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:40.140727	ICMP Echo Reply	ICMP	192.168.137.180	192.168.137.170
07/09-17:38:47.513307	PACKET RECEIVED	TCP	192.168.137.170	192.168.137.180

**Nodo B:**

TIMESTAMP	MSG	PROTOCOL	FROM_IP	TO_IP
07/09-17:38:37.128073	ICMP PING BSDtype	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:37.128073	ICMP PING *NIX	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:37.128073	PACKET SENT	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:37.128073	ICMP PING	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:37.320961	ICMP Echo Reply	ICMP	192.168.137.180	192.168.137.170
07/09-17:38:38.129178	ICMP PING BSDtype	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:38.129178	ICMP PING *NIX	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:38.129178	PACKET SENT	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:38.129178	ICMP PING	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:38.243095	ICMP Echo Reply	ICMP	192.168.137.180	192.168.137.170
07/09-17:38:39.130354	ICMP PING BSDtype	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:39.130354	ICMP PING *NIX	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:39.130354	PACKET SENT	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:39.130354	ICMP PING	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:39.163513	ICMP Echo Reply	ICMP	192.168.137.180	192.168.137.170
07/09-17:38:40.131860	ICMP PING BSDtype	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:40.131860	ICMP PING *NIX	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:40.131860	PACKET SENT	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:40.131860	ICMP PING	ICMP	192.168.137.170	192.168.137.180
07/09-17:38:40.190195	ICMP Echo Reply	ICMP	192.168.137.180	192.168.137.170
07/09-17:38:47.564978	PACKET SENT	UDP	192.168.137.170	192.168.137.1

Osservando i tempi d'invio ed i tempi di ricezione dei messaggi si può facilmente notare come per i primi pacchetti i tempi di ricezione, come ci si può aspettare, siano maggiori dei tempi d'invio mentre questo non vale dal quinto pacchetto in poi dove i tempi di ricezione risultano improvvisamente minori dei tempi d'invio. Questa inconsistenza può essere frutto di una sincronizzazione dell'orologio di una delle due macchine per mezzo del demone generato con il comando *ntpd* o anche uno sfasamento dei clock. La possibilità d'occorrenza di questa situazione rende quindi impossibile dare confidenza sui risultati ottenuti a causa della variabilità della visione temporale, di una o di entrambe le macchine, durante l'esecuzione di un esperimento.

Questo problema può essere risolto utilizzando altri due comandi. Sarà infatti necessario disattiva il servizio *ntp* eliminando il problema della sincronizzazione degli orologi asincrona tra le due macchine, utilizzando il comando *service ntp stop*. Successivamente dopo l'esecuzione di ogni esperimento sarà necessario utilizzare il comando *ntpdate*, il quale permette di sincronizzare l'orologio e ricevere l'offset di aggiustamento applicato. Utilizzando quest'offset che corrisponde ad una stima dell'errore temporale commesso da ciascuna macchina, durante l'esecuzione dell'esperimento, possiamo cercare di "correggere" le misure registrate. Ad esempio se l'offset restituito da *ntpdate* sul nodo A è +2ms mentre l'offset restituito sul nodo B è -4ms, si può cercare di correggere le misure registrate sul nodo A e sul nodo B rispettivamente sommando e sottraendo tali offset.

Questa correzione naturalmente non elimina totalmente l'errore sistematico commesso, in quanto gli offset sono stime dell'errore, ma si riesce comunque a limitare l'errore di misurazione. Quindi il sistema di misura introdotto avrà come riferimento l'orologio (il clock) del server utilizzato con il comando *ntpdate*, la procedura di misura consisterà nell'effettuare il *timestamp* di ogni pacchetto inviato/ricevuto con una risoluzione di  $10^{-6}s$  ed un'intrusività trascurabile per la risoluzione apprezzata. Ad ognuno di questi tempi sarà poi sommato/sottratto l'offset tra il tempo macchina e il tempo del server riscontrato durante l'esperimento.



# Capitolo 2

## Instrumentazione del sistema

### 2.1 Installazione e configurazione di Snort

Il progetto consiste nel valutare il comportamento, dal punto di vista temporale, di Snort come IDS. Andremo ad installare Snort in due macchine che utilizzano come OS Ubuntu 12.04.

Per installare e configurare Snort come IDS è necessario seguire i seguenti passi:

- **Scaricare librerie necessarie a Snort:** dobbiamo cioè verificare se nel nostro sistema sono già installate librerie come: libpcap0.8, libpcre3, libdnet-1.12, daq-1.1.1, flex, libpcap-ruby, libtool.
- **Scaricare la versione aggiornata di Snort** dal sito <http://www.snort.org/snort-downloads> e procediamo con l'installazione vera e propria del software.
- **Avere il prompt**, ed eseguire in successione i comandi:
  - `sudo tar zxvf snort-2.9.4.6.tar.gz`
  - `cd snort-2.9.4.6`
  - `sudo ./configure --prefix=/usr/local/snort --enable-sourcefire`
  - `sudo make`
  - `sudo make install`
  - `sudo mkdir /var/log/snort`
  - `sudo chown snort:snort /var/log/snort`
- **Scaricare il file delle regole:** per fare questo eseguiamo i comandi:
  - `sudo tar zxvf snortrules-snapshot-2946.tar.gz -C /usr/local/snort`
  - `sudo mkdir /usr/local/snort/lib/snort_dynamicrules`

- `sudo cp /usr/local/snort/snort_rules/precompiled/Ubuntu-10-4/i386/2.9.4.6/* /usr/local/snort/lib/snort_dynamicrules`
- `sudo touch /usr/local/snort/rules/white_list.rules`
- `sudo touch /usr/local/snort/rules/black_list.rules`
- `sudo ldconfig`

- **Configurare Snort:** dobbiamo modificare il file di configurazione `snort.conf` eseguendo il comando `sudo gedit /usr/local/snort/etc/snort.conf` e modificare le righe:

- `var WHITE_LIST_PATH ..//rules` e `var BLACK_LIST_PATH ..//rules` rispettivamente in `var WHITE_LIST_PATH /usr/local/snort/rules` e `var BLACK_LIST_PATH /usr/local/snort/rules`
- `dynamicpreprocessor directory /usr/local/lib/snort_dynamicpreprocessor/`,  
`dynamicengine /usr/local/lib/snort_dynamicengine/libsf_engine.so` e  
`dynamicdetection directory /usr/local/lib/snort_dynamicrules` rispettivamente in  
`dynamicpreprocessor directory /usr/local/snort/lib/snort_dynamicpreprocessor/`,  
`dynamicengine /usr/local/snort/lib/snort_dynamicengine/libsf_engine.so` e  
`dynamicdetection directory /usr/local/snort/lib/snort_dynamicrules`

ed aggiungere nella sezione relativa ai file di output generati, la riga `output alert_csv: alert.csv default` per far sì che il report generato sia già in formato csv (ci tornerà utile in fase di analisi).

- **Lanciare Snort come IDS:** per lanciare Snort come IDS è necessario eseguire il comando snort come root e specificare:

- la rete che s'intende monitorare, per mezzo dell'opzione `-i` seguita dal nome della rete (ad esempio `wifi0` per la rete wifi);
- la richiesta di riportare anche su terminale i pacchetti che trafficano nella rete, usando l'opzione `-dev`;
- il percorso in cui Snort andrà a salvare il report degli alert rilevati, usando l'opzione `-l` seguita appunto dal percorso;
- il percorso in cui Snort può trovare il proprio file di configurazione, usando l'opzione `-c` seguita dal percorso.

Snort è stato così installato e ben configurato su entrambe le macchine che andranno a comporre il sistema su cui faremo l'analisi temporale richiesta.

## 2.2 Tool di fault injection

In questa sezione descriviamo le caratteristiche e le opzioni dei vari tool utilizzati per gli esperimenti di fault injection, questi tool dovranno essere installati sulla macchina che svolgerà il ruolo di attaccante (nodo B).

### 2.2.1 NMap

Nmap ("Network Mapper") è uno strumento open source per la network exploration e l'auditing. È stato progettato per scansionare rapidamente reti di grandi dimensioni, ma è indicato anche per l'utilizzo verso singoli host. Nmap usa pacchetti IP raw (grezzi, non formattati) in varie modalità per determinare quali host sono disponibili su una rete, che servizi (nome dell'applicazione e versione) vengono offerti da questi host, che sistema operativo (e che versione del sistema operativo) è in esecuzione, che tipo di firewall e packet filters sono usati, e molte altre caratteristiche. L'output di Nmap è uno scan di un elenco di obiettivi, con informazioni supplementari per ognuno a seconda delle opzioni usate. Tra queste informazioni è vitale la "tabella delle porte interessanti". Questa tabella elenca il numero della porta e il protocollo, il nome del servizio, e il suo stato attuale. La tabella delle porte può anche includere dettagli quali le versioni dei software disponibili se è stata usata l'opzione appropriata.

Installare *nmap* è semplicissimo: sarà sufficiente digitare il comando *sudo apt-get install nmap* sul terminale.

Negli esperimenti abbiamo scelto di valutare il comportamento di Snort soggetto ad attacchi generati con *nmap* e le seguenti opzioni:

- *-osscan\_guess* (*Guess OS detection results*), con questa opzione attiva se nmap non è in grado di rilevare una corrispondenza esatta dell'OS, propone come possibilità gli OS più vicini alla rilevazione.  
La corrispondenza però deve essere molto simile perché Nmap lo faccia per default. Entrambe queste opzioni (equivalenti) fanno sì che Nmap proceda con il riconoscimento dell'OS in modo più aggressivo.
- *-version\_light* (*Attiva la modalità Light*), questa modalità rende il Version Scanning drasticamente più veloce, riducendone però la capacità di identificare accuratamente i servizi.
- *-version\_all* (*Prova ogni singolo pacchetto-sonda*), questa opzione assicura che ogni singolo pacchetto-sonda venga utilizzato su ogni singola porta.
- *-T* (*Imposta un template di temporizzazione*), tramite questa opzione nmap offre un approccio per il testing più semplice, lo fa mediante sei "timing templates", ovvero opzioni pre-impostate per regolare l'aggressività della scansione. Esse si specificano mediante l'opzione -T seguita dal numero del template corrispondente o dal suo nome. Essi sono: paranoico (0), furtivo (1), educato (2), normale (3), aggressivo (4) e folle (5). I primi due vengono usati per evitare i sistemi anti-intrusione (IDS). La modalità "gentile" rallenta la scansione in modo da usare meno banda e risorse sulla macchina bersaglio. La modalità "normale" è di default (e pertanto l'opzione -T3 non modifica nulla). La modalità "aggressiva" incrementa la velocità assumendo che si è su una rete veloce ed affidabile. Infine la modalità "folle" dà per scontato che si è su una rete estremamente veloce ed affidabile o che si vuole sacrificare l'accuratezza in nome della velocità.

- **-A** (*Opzioni di scan aggressive*), questa opzione abilita opzioni addizionali avanzate e aggressive.

- **-sn**, esegue una scansione ping ed esclude la scansione sulle porte.

Questo comando è stato testato, tra i vari esperimenti condotti in questo lavoro, ma non è stato rilevato in alcun modo da Snort, né sotto forma di alert né come pacchetti ricevuti. Quindi questo particolare attacco è stato escluso nella fase di analisi.

### 2.2.2 Ping

Ping verifica le connessioni a computer remoti. Invia pacchetti di echo ICMP Internet Control Message Protocol a un computer e attende i pacchetti di risposta. Ping attende di default fino a 1 secondo per ogni pacchetto inviato e viene stampato il numero di pacchetti trasmessi e ricevuti sulla console. L'attesa per ogni pacchetto inviato può essere specificata utilizzando l'opzione **-i**. Il comando *ping* è già disponibile all'interno della distribuzione Ubuntu utilizzata.

Per i nostri esperimenti andremo ad utilizzare l'opzione **-i** del comando, tale opzione permette di esplicitare l'intervallo di tempo da attendere tra l'invio di ogni pacchetto. Il valore di default è un secondo, o non attendere in modalità *flood*. Solo un super-utente può impostare l'intervallo a valori inferiori di 0,2 secondi.

Per verificare la variazione di funzionamento di Snort a seguito di pacchetti inviati più velocemente abbiamo creato una nuova regola:

```
alert icmp any any -> 192.168.137.180 any (msg:"TOO MUCH PING"; threshold: type both, track by_src, count 10, seconds 3; classtype:attempted-dos; sid:51000; rev:2;)
```

Tale regola ci permette di discriminare il comportamento tenuto da Snort al variare del valore per l'opzione **-i**.

### 2.2.3 Metasploit

Metasploit Project è un progetto open source (rilasciato con licenza BSD) dedicato alla sicurezza informatica il quale fornisce informazioni sulle vulnerabilità della rete e ne semplifica le operazioni di penetration testing e ne aiuta nello sviluppo di sistemi di rilevamento di intrusioni. Assieme al progetto Metasploit troviamo anche Metasploit Framework uno strumento dedicato principalmente allo sviluppo e l'esecuzione di exploit ai danni di una macchina remota. Sono messe a disposizione varie interfacce ma nel seguito andremo ad utilizzare l'interfaccia web msfweb, utile per usare il framework con tutta la comodità di un'interfaccia web. Metasploit Project viene rilasciato con un semplice installar il quale ci permetterà d'installare l'applicazione in qualsiasi distribuzione Linux. Per installare Metasploit Project su OS Ubuntu basta digitare:

- `wget http://downloads.metasploit.com/data/releases/metasploit-latest-linux-installer.run`

- `chmod +x metasploit-latest-linux-installer.run`
- `sudo ./metasploit-latest-linux-installer.run`

a questo punto si aprirà un wizard con il quale procederemo all'installazione di Metasploit e in cui ci verrà chiesto d'inserire la porta SSL dedicata al servizio e successivamente la creazione dei certificati SSL.

Al termine del Wizard d'installazione potremo accedere alla Web Ui di Metasploit la qual ci confermerà la corretta installazione. Potremo a questo punto lanciare l'interfaccia web di Metasploit dove sarà richiesto di creare ed attivare un account.

Per i nostri esperimenti si è scelto di utilizzare *metasploit* come *port scan*, sarà necessario digatere l'indirizzo ip su cui eseguire la scansione e successivamente specificare le varie opzioni avanzate per la scansione. Le scansioni che effettueremo sono essenzialmente due, quella di default senza specificare alcuna opzione avanzata e una seconda in cui abiliteremo la *Scan SNMP community strings Information*: quest'opzione lancia un task in background per analizzare i device SNMP che corrispondono ad una serie di stringhe (essendo un lavoro generalmente lungo, viene eseguito in parallelo con le altre scansioni base).

#### 2.2.4 Traceroute

*Traceroute* cerca di tracciare il percorso seguito dai pacchetti IP per un qualche host internet. Cerca i passaggi intermedi, lanciando pacchetti sonda con un piccolo TTL, e quindi resta in ascolto di un *ICMP reply* da un router intermedio. *Traceroute* inizia la rilevazione con un TTL di uno, e lo incrementa fino a quando viene ricevuto un *ICMP port unreachable reply*. Questo significa che la sonda o ha raggiunto l'host, o ha raggiunto il TTL massimo.

Per rendere disponibile questo tool, se non già presente tra le librerie dell'OS, è necessario eseguire il comando *apt-get install traceroute* tramite il quale sarà installato e quindi reso disponibile.

Negli esperimenti andremo ad attivare alternativamente le seguenti opzioni:

- *-I (icmp)*, abilita il comportamento ad ora usuale per traceroute, utilizzando pacchetti ICMP ECHO per le sonde.
- *-T (tcp)*, abilita un nuovo comportamento per traceroute destinato ad aggirare i firewall.

Utilizza una porta di destinazione costante (di default è 80, http).

Questo metodo utilizza la ben nota "*half-open technique*", che impedisce alle applicazioni sull'host di destinazione di far vedere le nostre sonde a tutti.

Includendo l'opzione *-T* all'attacco con traceroute, Snort rileva (sul nodo A) una serie di pacchetti in arrivo ma non genera alcun tipo di alert (o perché l'attacco viene considerato innocuo, o perché Snort non ha regole sufficienti a rilevarlo). Nella fase di analisi, quindi, sono state escluse le misurazioni sui tempi di detection che riguardano quest'esperimento.

- *-N (squeries)*, Specifica il numero di pacchetti sonda inviati simultaneamente. L'invio di più sonde contemporaneamente può accelerare notevolmente traceroute (il valore predefinito è 16). Si noti che alcuni router e gli host possono utilizzare *ICMP rate throttling* in una tale situazione specificando un numero troppo grande si può avere la perdita di alcune risposte.

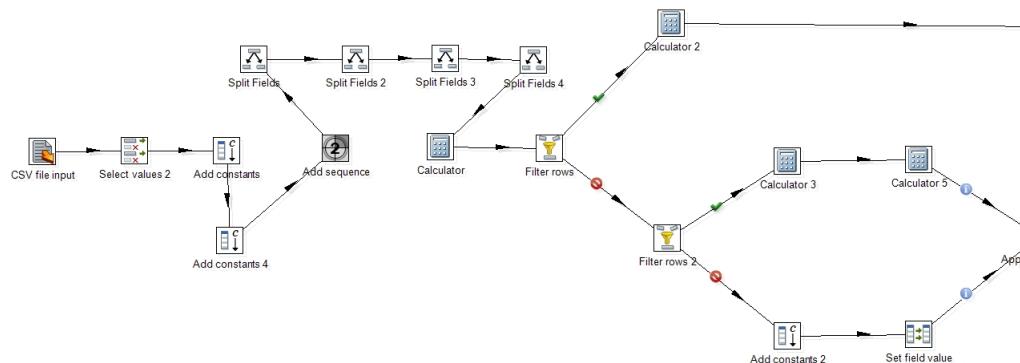
## 2.3 Tool di data integration e analisi

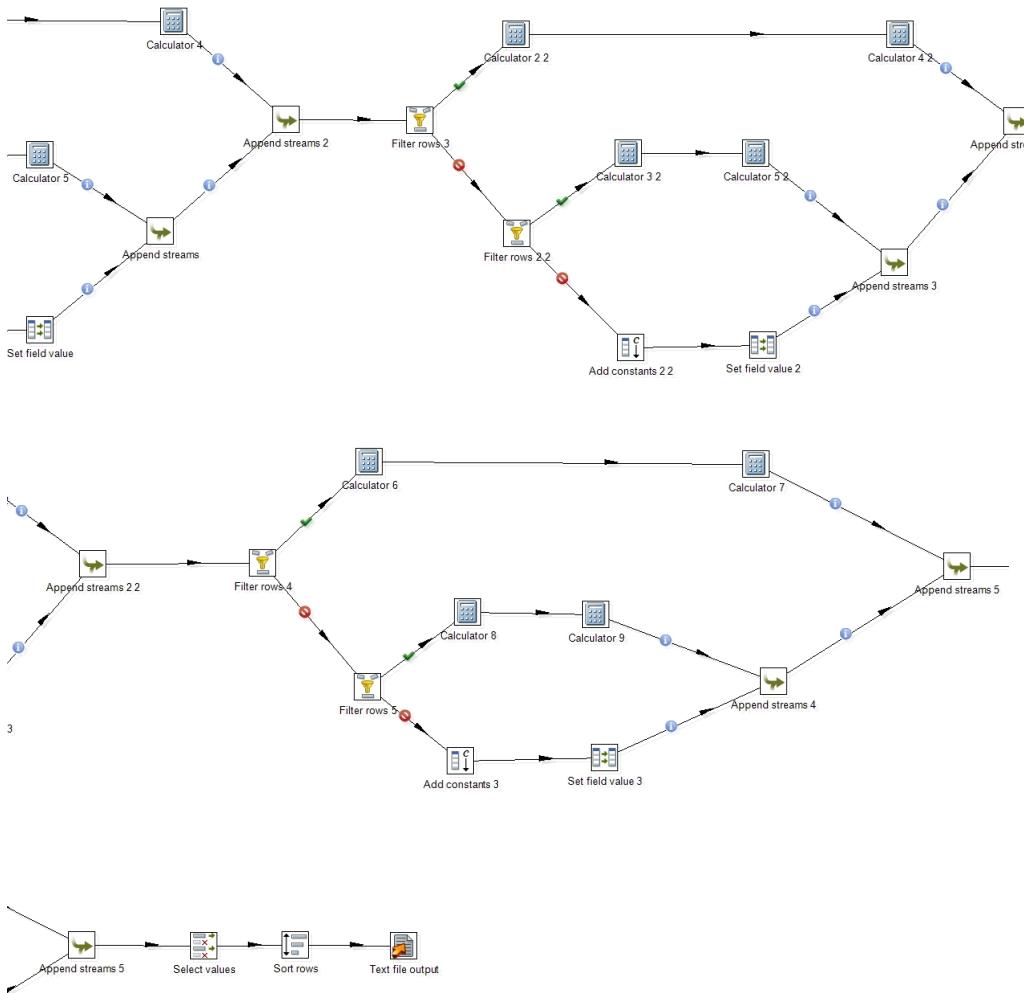
In questa ultima sezione andiamo a descrivere il tool che utilizzeremo per integrare e manipolare i dati raccolti al fine di generare successivamente un sistema OLAP su cui effettuare tutte le analisi.

Il framework utilizzato è *Pentaho*, precisamente il tool *Kettle* per la *Data Integration* ed il motore MOLAP *Mondrian* per analisi di tipo OLAP.

Il framework messo a disposizione da Pentaho permette di sviluppare soluzioni complete per la *Business Intelligence* ma noi ci limiteremo ad utilizzare i tool sopra citati, che sono ampiamente sufficienti per soddisfare le nostre richieste.

Utilizzando il tool *Kettle*, è possibile integrare e manipolare i file *.csv* generati in fase sperimentale, così da renderli in una forma più consona alla memorizzazione all'interno di un database, ad esempio sarà possibile dividere il campo relativo al timestamp nelle sue sottocomponenti, così da rendere più semplice la correzione del tempo indicata dall'offset di sfasamento degli orologi rilevato tramite il comando *ntpdate* oppure ancora permette di aggiungere nuovi campi come ad esempio “EXP\_ID” che andrà a identificare univocamente i pacchetti relativi ad un determinato esperimento. Vediamo qui la trasformazione (per comodità divisa in quattro immagini) che permetterà di integrare tra loro i dati relativi ai vari esperimenti eseguiti:





Il file generato da Kettle (file di log *logCompleto.csv*), ancora in formato *.csv*, consisterà nell'elenco di tutti i pacchetti loggati a cui sono associate tutte le informazioni registrate durante gli esperimenti. Ovviamente tale file è spesso e volentieri ridondante, proprio come chiede la logica OLAP.

A questo punto utilizzando il tool *Mondrian* è possibile definire uno *star-schema*, che avrà la struttura indicata in Figura 2.1.

Questa struttura ci permetterà di modellare il fatto "*Logged Packet*". L'analisi, infatti, è volta ad analizzare come Snort reagisce e varia le proprie prestazioni al variare sia di tool di fault injection che, pur rimanendo all'interno dello stesso tool, delle opzioni attive. Prima di caricare nel database tutte le informazioni contenute nel file *.csv*, per velocizzare il processo di analisi utilizzeremo un semplice programma *Java* (Codice A.1, Appendice A), il quale, analizzando il file generato da *Kettle*, permetterà di calcolare le seguenti tempistiche necessarie per l'analisi:

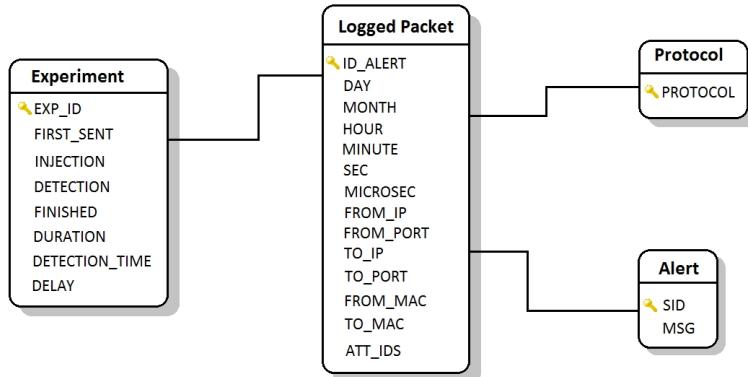


Figura 2.1: *Schema a stella* costruito sul file di log finale.

- L’istante di tempo in cui l’attacco è generato dal nodo B.
- L’istante di tempo in cui l’attacco è iniettato sul nodo A.
- L’istante di tempo in cui l’attacco è individuato da parte di Snort.
- Il ritardo di trasmissione introdotto dalla rete.
- Il tempo di *detection*, cioè il tempo impiegato da Snort per rilevare l’attacco.
- La durata dell’attacco.

Questi tempi, per una più semplice manipolazione, saranno “assoluti” cioè si riferiranno ad uno stesso giorno di uno stesso mese e l’orario sarà espresso in microsecondi. Ogni esperimento avrà ovviamente proprie tempistiche.

Arrivati a questo punto tutti i dati raccolti e le quantità necessarie per l’analisi saranno nella forma più adatta per la memorizzazione in un database (file di log *logFinale.csv*). Procederemo con l’analisi per mezzo di funzioni OLAP rese disponibili anch’esse nel framework *Pentaho*, precisamente usando *Mondrian*. Sarà così semplice fornire grafici o report di ogni tipo incrociando a piacimento i dati raccolti per i vari esperimenti. Riposteremo le analisi effettuate nei seguenti capitoli. Occorre però osservare che il numero di comparazioni effettuabili sono molteplici e tutte facilmente ottenibili utilizzando *Pentaho*, sfruttando la ridondanza creata per mezzo di *kettle* nel file *.csv* che svolge la funzione di repository dei dati sperimentali.

# Capitolo 3

## Analisi temporale

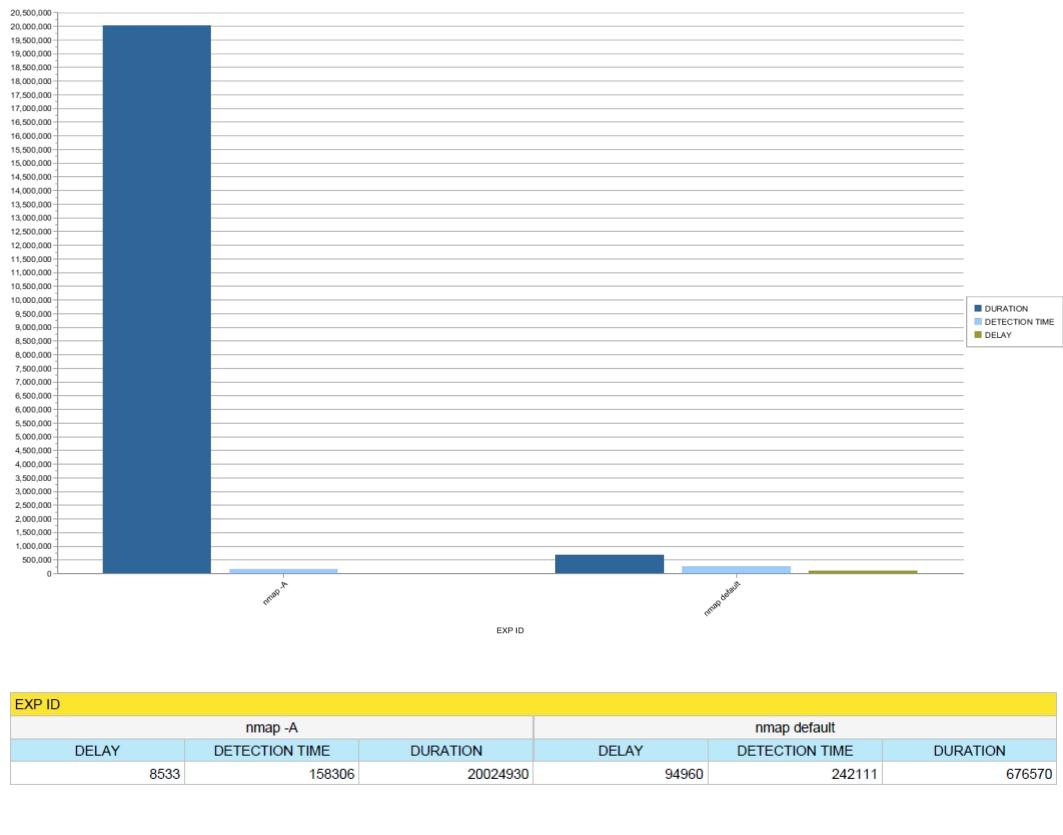
### 3.1 Nmap

In questo capitolo vengono presentati i risultati della comparazione tra le varie istanze di esecuzione fatte con *nmap*, selezionando alternativamente le opzioni precedentemente descritte. L'analisi effettuata è indirizzata sulla durata dell'attacco, il tempo impiegato da Snort per rilevarlo ed il ritardo introdotto dalla rete.

#### 3.1.1 Nmap, -A vs Default

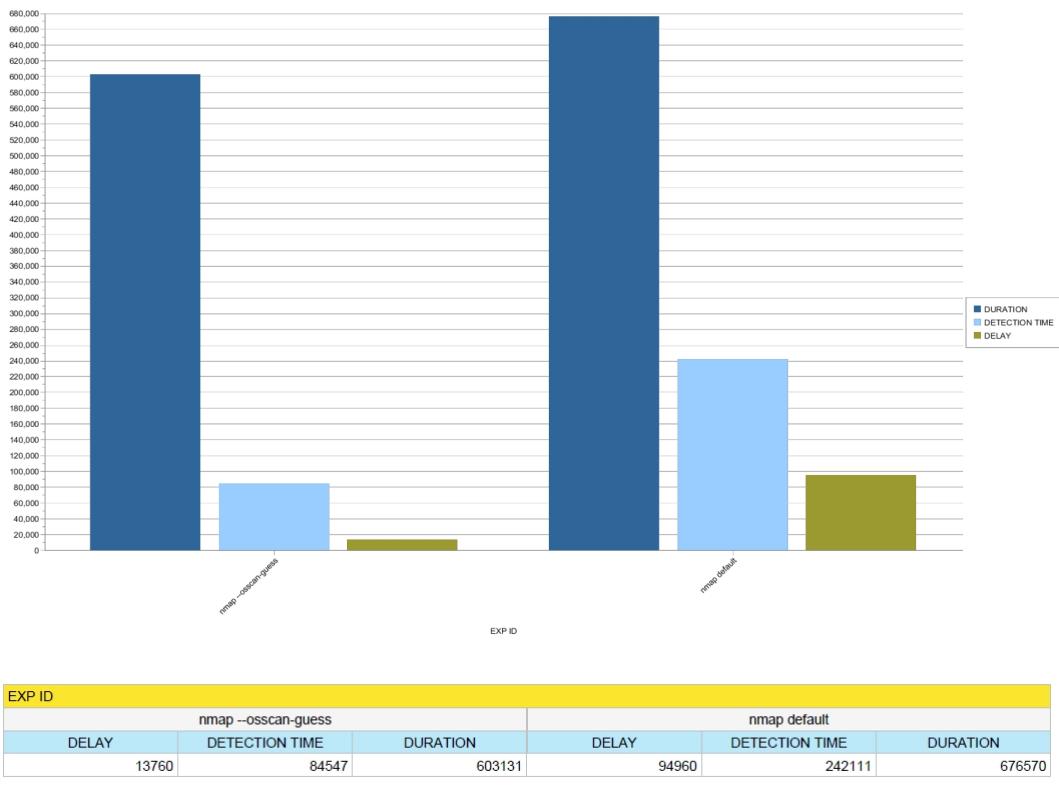
A parte il tempo di rilevamento molto simile, che risulta essere di circa un decimo di secondo più veloce per *nmap -A*, si nota come l'opzione *-A*, rispetto alla versione standard di *nmap*, faccia uso di molte più operazioni per la scansione delle porte del sistema attaccato, puntando ad effettuare una scansione più esaustiva del normale, come mostrato dai valori di durata dell'attacco.

Più intuitivo e chiarificante risulta essere il seguente grafico:



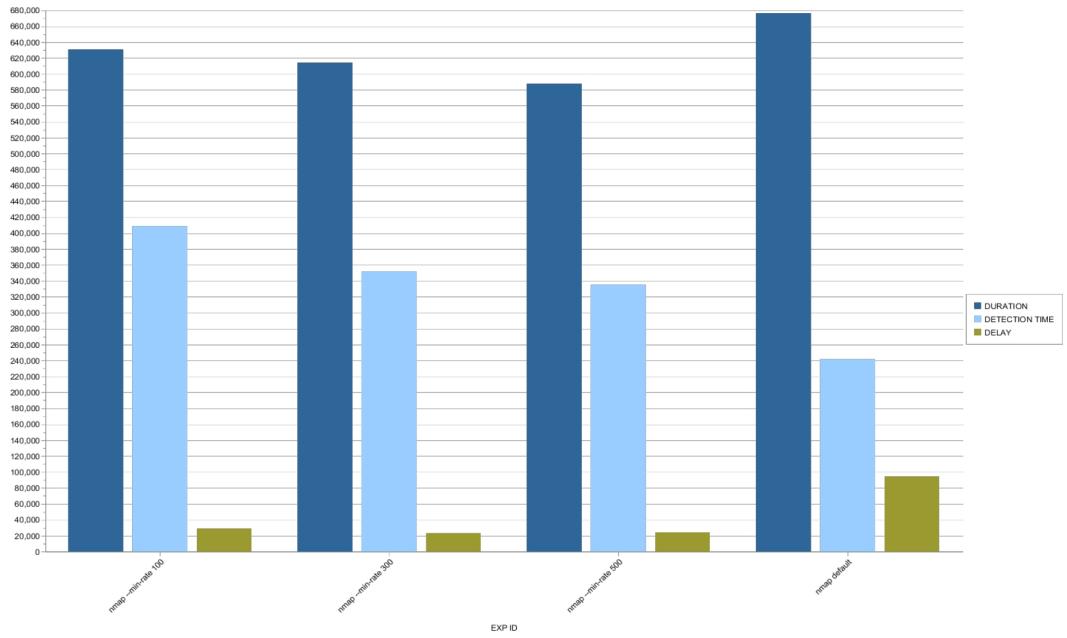
### 3.1.2 Nmap, –osscan-guess vs Default

Come descritto dal manuale di nmap, utilizzando l'opzione –osscan-guess si esegue un attacco più aggressivo, destinato, quindi, a terminare prima. Snort, come si può vedere dai risultati mostrati dai grafici, riesce a rilevarlo in un tempo più breve rispetto ad una scansione più cauta che quindi passa in principio più inosservata.



### 3.1.3 Nmap, `-min-rate 100/300/500` vs Default

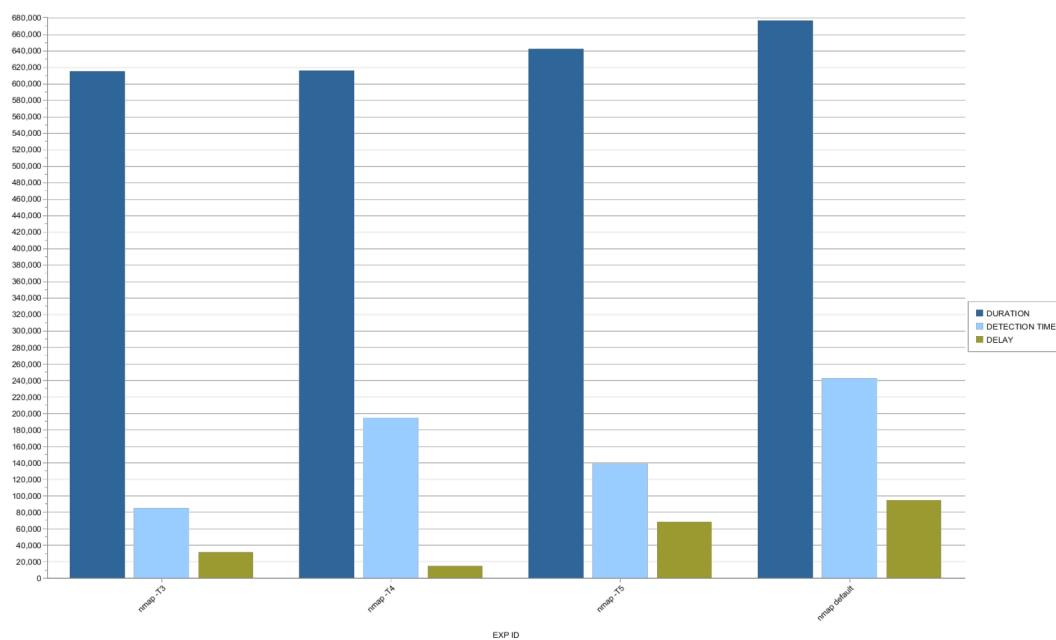
Dalla comparazione dei grafici ottenuti dall'utilizzo di diversi valori (100, 300 e 500 pacchetti/secondo) per l'opzione `-min-rate` del comando nmap possiamo dedurre, intuitivamente, che forzare l'attacco ad inviare più pacchetti al secondo rende l'attacco più veloce e più facilmente rilevabile, dal momento che il numero/tipo di pacchetti necessari a Snort per rilevare l'attacco arrivano con frequenza maggiore. Comparando, inoltre, questi risultati con la versione standard (senza opzioni) di nmap, possiamo ipotizzare che quest'ultima invii i pacchetti necessari ad un rate maggiore di 500 pacchetti/secondo (si vede facilmente seguendo la curva dei tempi di rilevamento). Infine, comparando anche la durata totale dell'attacco, possiamo dedurre che la versione default di nmap esegue più operazioni (o operazioni più complesse) rispetto alla versione che utilizza l'opzione `-min-rate`.



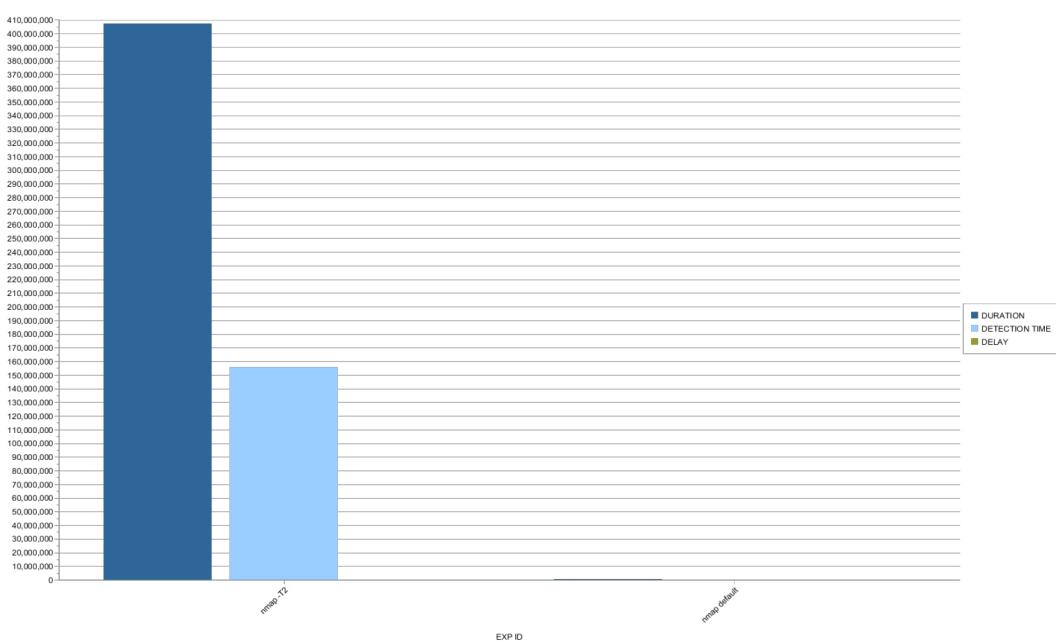
EXP ID	nmap -min-rate 100			nmap -min-rate 300			nmap -min-rate 500			nmap default		
	DELAY	DETECTION TIME	DURATION	DELAY	DETECTION TIME	DURATION	DELAY	DETECTION TIME	DURATION	DELAY	DETECTION TIME	DURATION
29361	29361	409407	631098	23371	353021	614533	24657	335177	588254	94960	242111	676570

### 3.1.4 Nmap, -T 2/3/4/5 vs Default

Notiamo innanzitutto come le tempistiche di durata e di rilevamento per l'attacco "educato" di nmap (per comodità visualizzato in un grafico a parte) siano fuori scala rispetto a tutti gli altri timing templates. Questo tipo di attacco è infatti stato progettato per agire lentamente, cercando di ridurre al minimo ogni possibile rilevamento da parte di un ipotetico IDS. Per quanto riguarda gli altri templates, possiamo dedurre che l'attacco "aggressivo" (-T4) è probabilmente il miglior punto d'incontro tra velocità d'attacco e furtività, permettendo una rapida scansione in grado di aggirare per breve tempo un generico IDS.



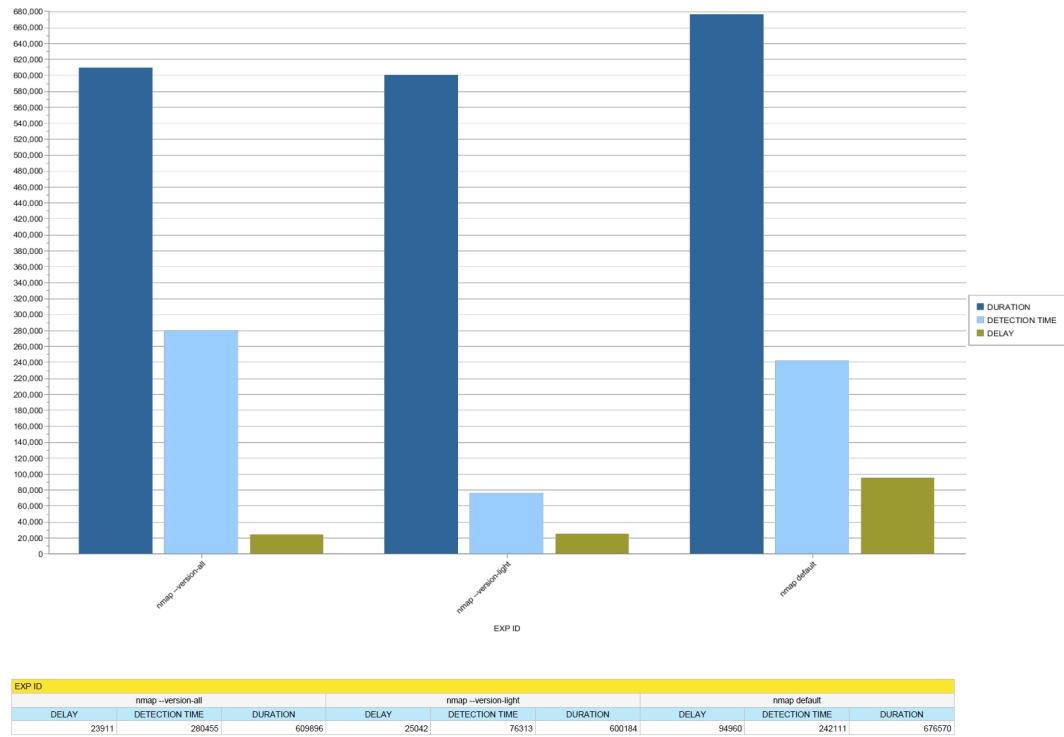
EXP ID											
nmap -T3			nmap -T4			nmap -T5			nmap default		
DELAY	DETECTION TIME	DURATION	DELAY	DETECTION TIME	DURATION	DELAY	DETECTION TIME	DURATION	DELAY	DETECTION TIME	DURATION
31038	84078	615144	14226	194498	615515	67920	133685	642421	94960	242111	676570



EXP ID					
nmap -T2			nmap default		
DELAY	DETECTION TIME	DURATION	DELAY	DETECTION TIME	DURATION
154891	155640544	407595636	94960	242111	676570

### 3.1.5 Nmap, *-version-all/light* vs Default

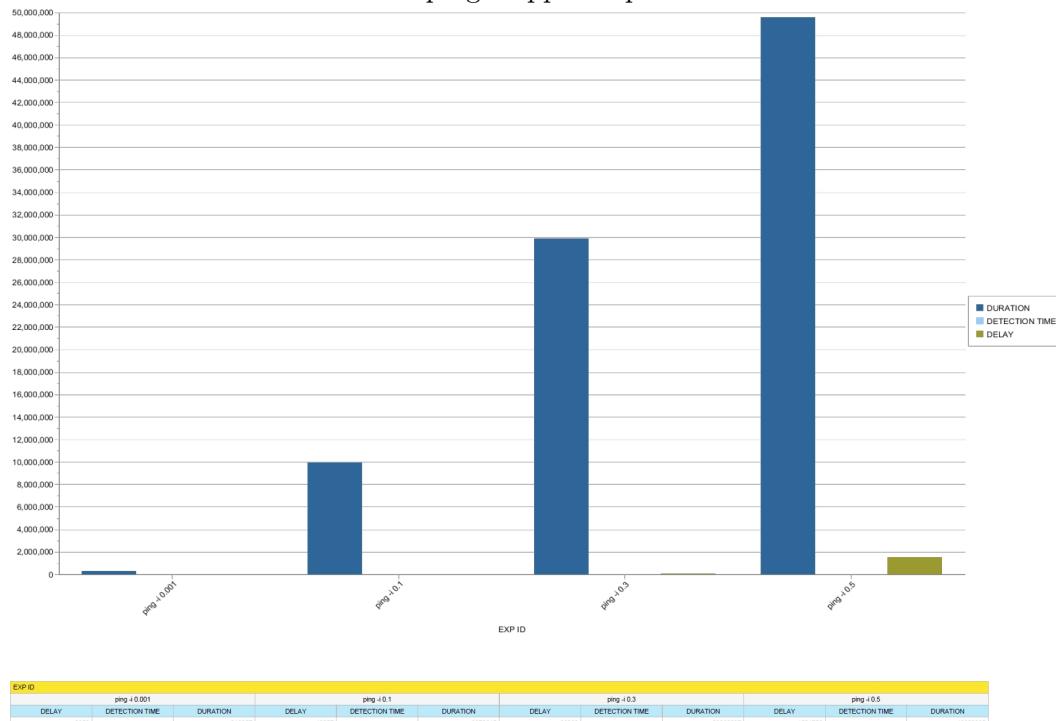
Osservando il grafico seguente, ed in particolare le colonne relative al tempo di detection dell'attacco, si vede subito come la versione leggera (*-version-light*) sia più velocemente riconoscibile da un IDS come Snort rispetto alla sua controparte esaustiva (*-version-all*). Confrontando questi risultati con la versione standard di nmap possiamo inoltre supporre che quest'ultima sia formata da un tradeoff delle versioni light e all, in quanto il suo tempo di rilevazione si trova più o meno tra i due sopra citati.



## 3.2 Ping

Per quanto riguarda l'utilizzo del comando *ping* l'opzione utilizzata è la *-i* con i valori 0.001/0.1/0.3/0.5.

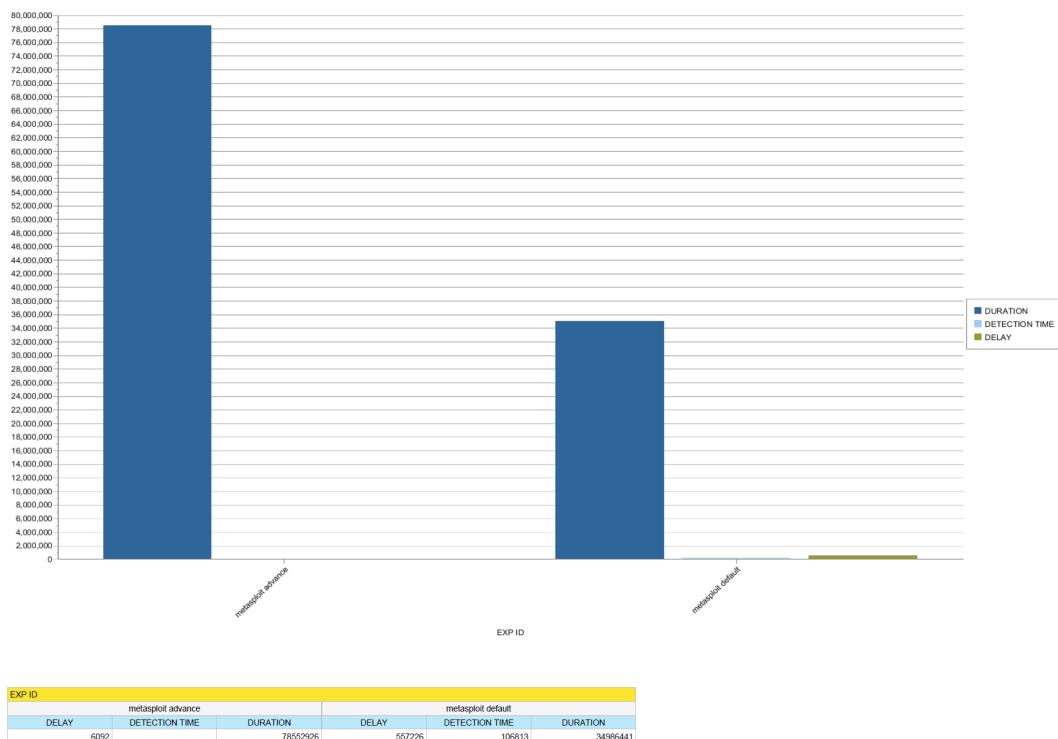
Come già accennato in precedenza, Snort rileva immediatamente un qualsiasi attacco basato su pacchetti ICMP, come ad esempio gli attacchi che utilizzano il comando ping (ping of death, denial of service, ...). Questo si vede facilmente dai grafici che mostrano come in tutti i casi il tempo di rilevamento da parte di Snort di un attacco ping sia immediato. Per quanto riguarda invece la durata complessiva dell'attacco si ha che, come ci si può aspettare, a parità di pacchetti inviati (100, in questi esperimenti) la durata dell'attacco aumenta al diminuire del rate di invio: inviando ad un rate di 0.001 pacchetti/secondo l'attacco termina in meno di mezzo secondo, mentre inviando ad una frequenza di 0.5 pacchetti/secondo la durata aumenta fino a quasi 50 secondi complessivi. Inoltre dall'analisi dei log è possibile osservare che per un rate di invio pare a 0.5 pacchetti/secondo non viene generato alcun alert del tipo "TOO MUCH PING", verificando che l'introduzione della nostra regola permette di stabilire una soglia oltre la quale si può ritenere dannosa la sottomissione di ping troppo frequenti.



### 3.3 Metasploit

Metasploit è stato utilizzato come port scan nella sua versione di default e successivamente abilitando l'opzione *Scan SNMP*.

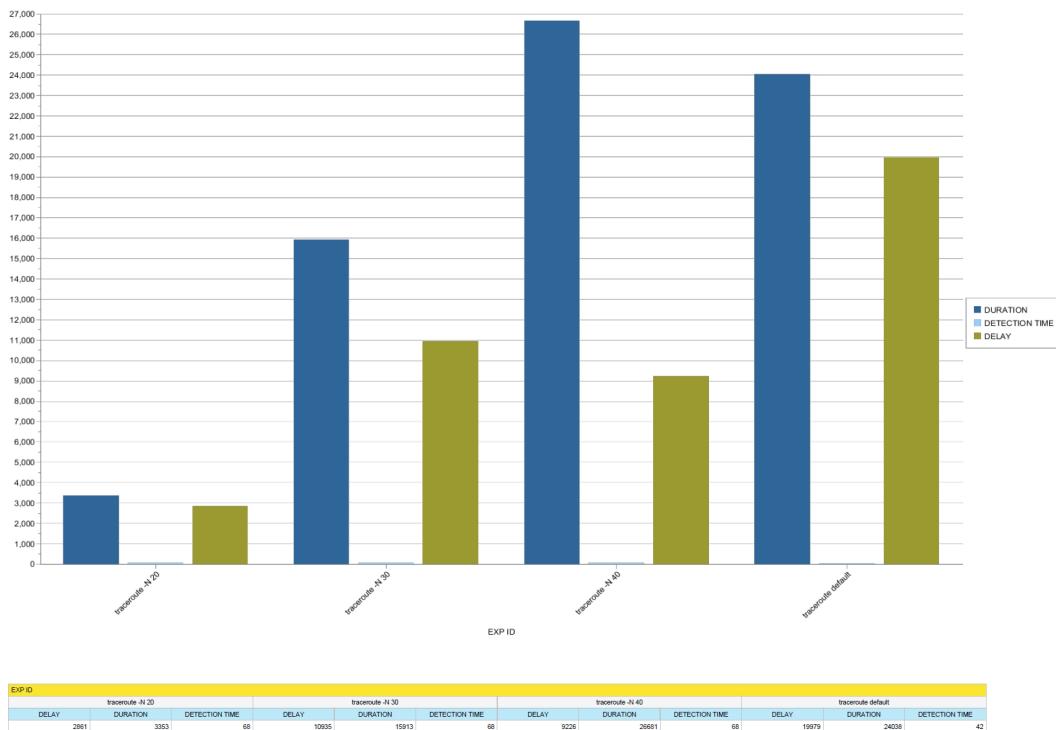
Dai grafici si nota come la versione standard dell'attacco di Metasploit sia più corta rispetto alla versione completa, probabilmente a causa delle operazioni in più che quest'ultima compie. Inoltre possiamo vedere che il tempo di detection dell'attacco standard è più alto rispetto alla versione avanzata, che invece viene rilevata subito da Snort, forse a causa delle tecniche di intrusione più aggressive introdotte da quest'ultima.



## 3.4 Traceroute

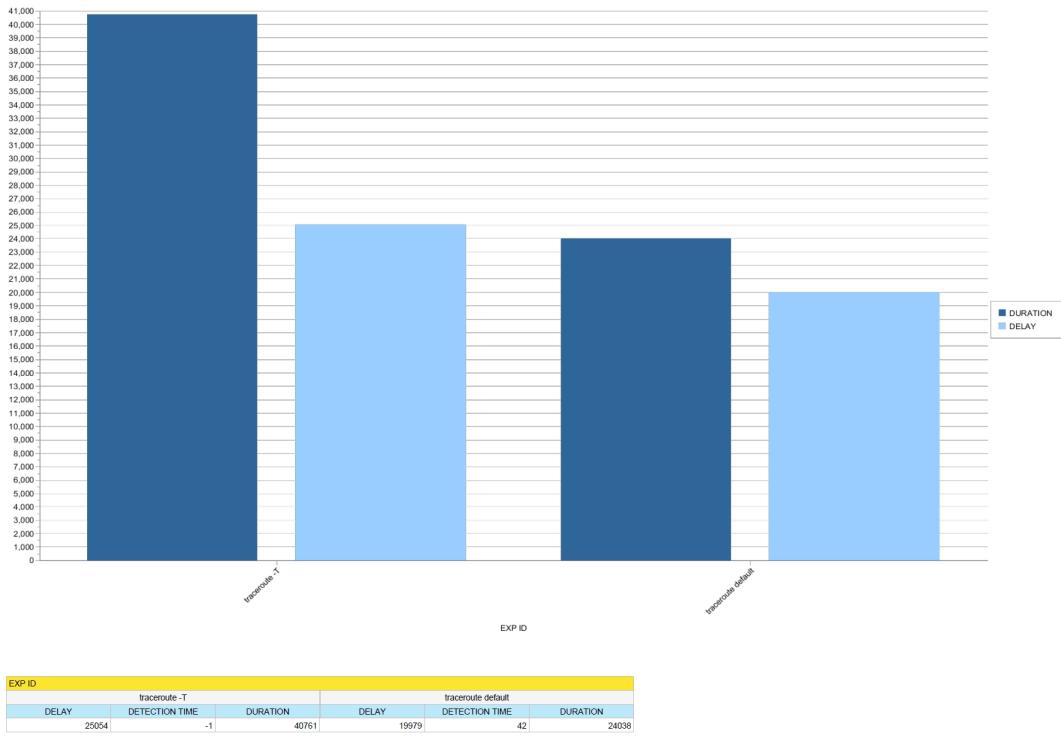
### 3.4.1 Traceroute, -N 20/30/40 vs Default

Osservando i valori della detection si vede come questi siano esattamente identici (68 microsecondi) per i tre esperimenti con l'opzione -N, mentre sia leggermente più bassa (42 microsecondi) per quanto riguarda l'attacco traceroute di default. Rispetto alla durata dell'attacco, come ci si aspetta questa aumenta all'aumentare del numero di pacchetti-sonda inviati dal comando, mentre rimane circa a metà strada tra -N 30 e -N 40 per quanto riguarda la versione standard di traceroute. Quindi, dal momento che il manuale afferma che il valore di default per i pacchetti-sonda è 16, dai valori di durata della versione standard possiamo supporre che quest'ultima esegua operazioni in più, o più complesse, rispetto al comando che utilizza l'opzione -N.



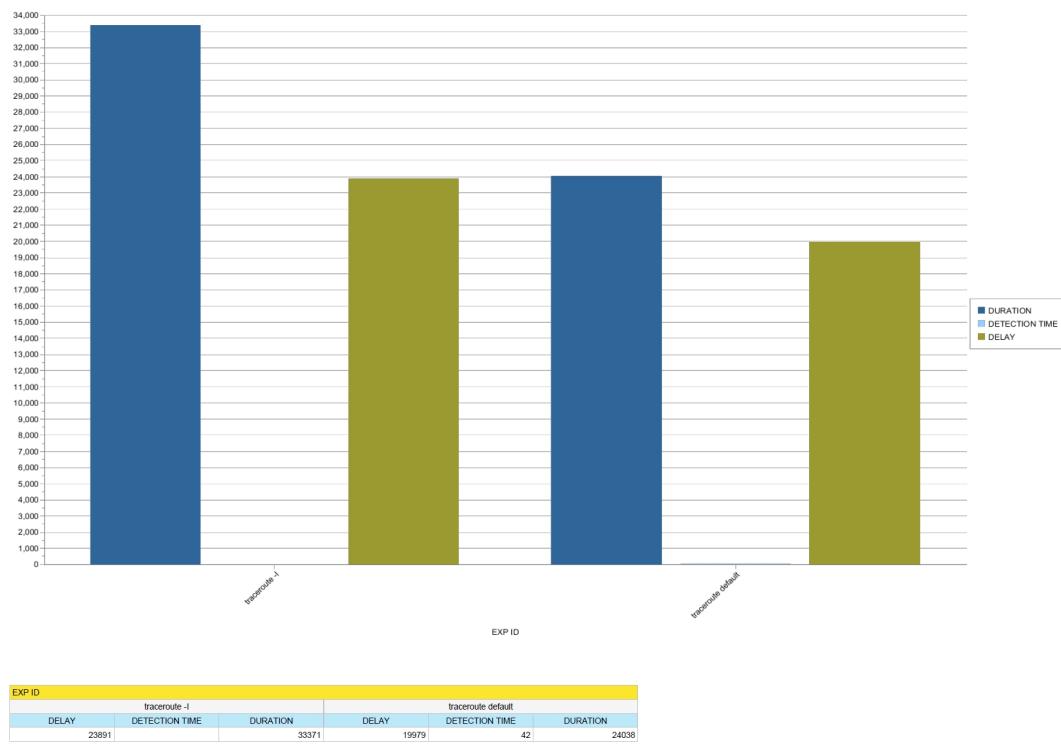
### 3.4.2 Traceroute, -T vs Default

In questi grafici, per comodità, sono stati omessi i valori di detection in quanto, come già accennato, il comando traceroute con opzione -T non genera alcun alert da parte di Snort. Si è comunque deciso di includere gli altri valori in quanto, dalla parte del sistema attaccato, vengono comunque ricevuti e loggati molti pacchetti che evidenziano, ad esempio, la maggior complessità dell'attacco traceroute -T, che impiega più tempo a terminare la scansione rispetto alla versione standard di traceroute.



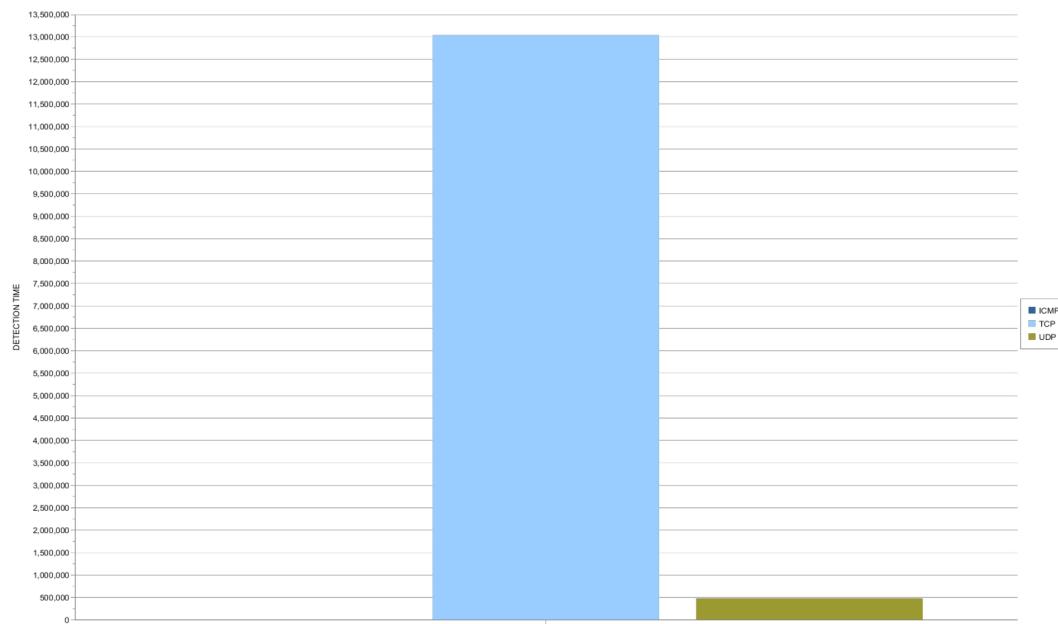
### 3.4.3 Traceroute -I vs Default

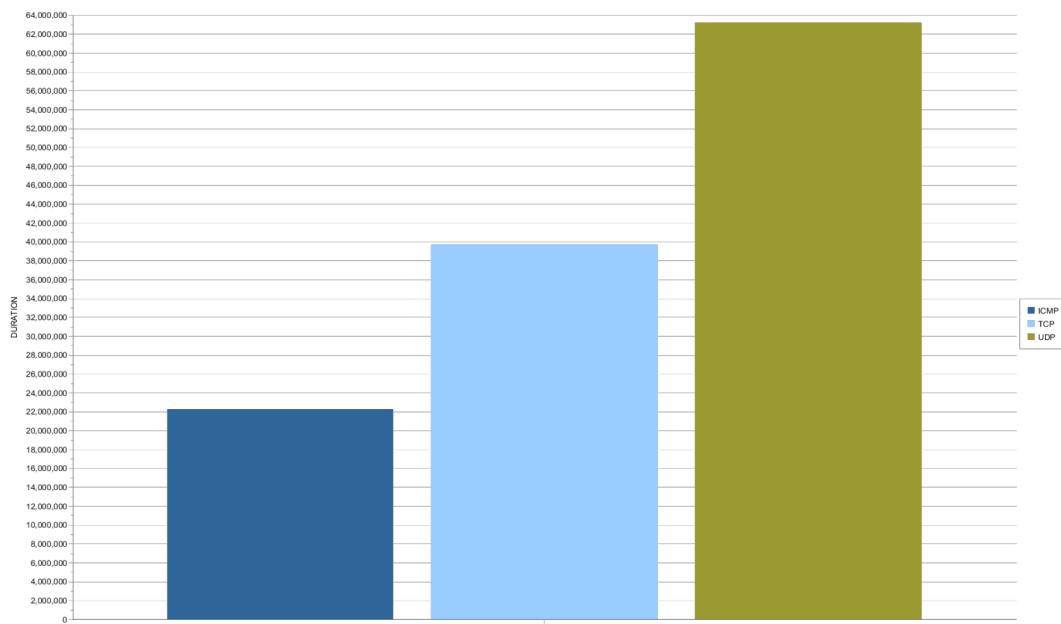
Come ci potevamo attendere, indicando a traceroute di eseguire un attacco tramite pacchetti ICMP ECHO si avrà che quest'attacco verrà identificato immediatamente (ovvero all'arrivo del primo pacchetto) da parte di Snort sul sistema attaccato. Infatti sappiamo che qualsiasi ping ricevuto da Snort verrà immediatamente loggato come alert. A parte questo, notiamo come (analogamente a quanto visto per l'opzione -T) la durata complessiva dell'attacco sia maggiore includendo l'opzione -I, che probabilmente aggiunge complessità all'attacco base di traceroute.



### 3.5 Protocolli

Analizzando le tempistiche di detection per i tre protocolli principali utilizzati in questi esperimenti (ICMP, TCP ed UDP) si nota innanzitutto come i tempi di rilevamento per attacchi basati su ICMP (ad es. ping) sia pressoché nulla, rendendo inefficace un qualsiasi tipo di attacco. Per quanto riguarda invece TCP e UDP notiamo un netto distacco, a favore di TCP, secondo i tempi di detection: è possibile, dato che UDP è un protocollo non affidabile che punta tutto sulla velocità d'invio, che gli attacchi basati su UDP siano progettati in modo da agire velocemente, velocizzando quindi anche il processo di rilevazione da parte degli IDS in ascolto, mentre un attacco basato su TCP potrebbe agire più lentamente ed in modo più silenzioso. Un'altra possibile spiegazione è che, dato che il protocollo TCP è affidabile per natura, molte analisi su questi pacchetti vengono già fatte dalla stessa infrastruttura di rete, quindi è possibile che un IDS come Snort decida di dare meno peso (con meno regole o regole meno stringenti) a questo tipo di pacchetti, preferendo invece concentrarsi su pacchetti meno controllati come quelli UDP o ICMP. Osservando invece la durata media degli attacchi raggruppata per protocollo, vediamo come gli attacchi ICMP siano mediamente i più veloci (pensiamo ad attacchi come ping o nmap -I), subito seguiti da attacchi TCP e da attacchi UDP, che evidentemente fanno, in media, più operazioni (o operazioni più complesse) rispetto agli attacchi basati su ICMP o TCP.





PROTOCOL					
ICMP		TCP		UDP	
DETECTION TIME	DURATION	DETECTION TIME	DURATION	DETECTION TIME	DURATION
3813	22294369	13044848	39689263	480079	63235207

### 3.6 Alert generati

Esponiamo per una più chiara interpretazione del grafico a seguito di che genere di eventi gli alert vengono generati:

- **ICMP PING undefined code**, questo alert viene generato quando un utente esterno effettua un ping su un server interno usando una richiesta di tipo echo ICMP. Generalmente un attaccante utilizza scansioni di questo tipo per ottenere informazioni sul network o inviare un elevato numero di ping nel tentativo di causare un flood della rete per creare un DoS.
- **ICMP Echo Reply undefined code**, questo alert viene generato quando un host genera una risposta di tipo echo ICMP, (con un codice ICMP non valido o non definito). È evidente che questo messaggio è speculare rispetto al precedente: l'host di destinazione invia un messaggio di risposta all'host richiedente indicando quindi di essere alive.
- **SNMP AgentX/tcp request**, questo alert viene generato quando si tenta di attaccare un dispositivo utilizzando SNMP v1. L'impatto varia in base all'implementazione, da un tentativo di Denial of Service (DoS) all'esecuzione di codice. SNMP (Simple Network Management Protocol) è un protocollo largamente adottato per reti IP, utile ai
- ni di manutenzione e monitoring.
- **SNMP request tcp**, questo alert viene generato quando si invia una richiesta per determinare se un dispositivo stia usando l'SNMP; l'attaccante invia un pacchetto (solitamente diretto alla porta tcp o udp 161), e se ciò avviene con successo viene generata una risposta, in base alla quale potrà scegliere di inviare eventuali ulteriori attacchi all'SNMP daemon.
- **SHELLCODE x86 inc ebx NOOP**, questo alert viene generato quando è probabile sia stato effettuato un tentativo di eseguire del codice su di un host nella rete protetta proveniente da una sorgente esterna a tale rete.
- **SCAN nmap XMAS**, questo alert viene generato quando è stata individuata una scansione da parte della piattaforma XMAS di Nmap. Generalmente avviene quando un attaccante effettua una scansione con Nmap al fine di determinare quali siano le porte aperte, esattamente ciò che abbiamo effettuato. Ciò significa che SNORT ha individuato la nostra scansione con Nmap.
- **ICMP PING**, questo alert viene generato quando viene effettuata una richiesta ICMP echo di tipo generico per determinare se l'host è attivo. Analogamente all'ICMP PING undefined code.
- **ICMP Echo Reply**, analogo all'ICMP PING Echo Reply undefined code.

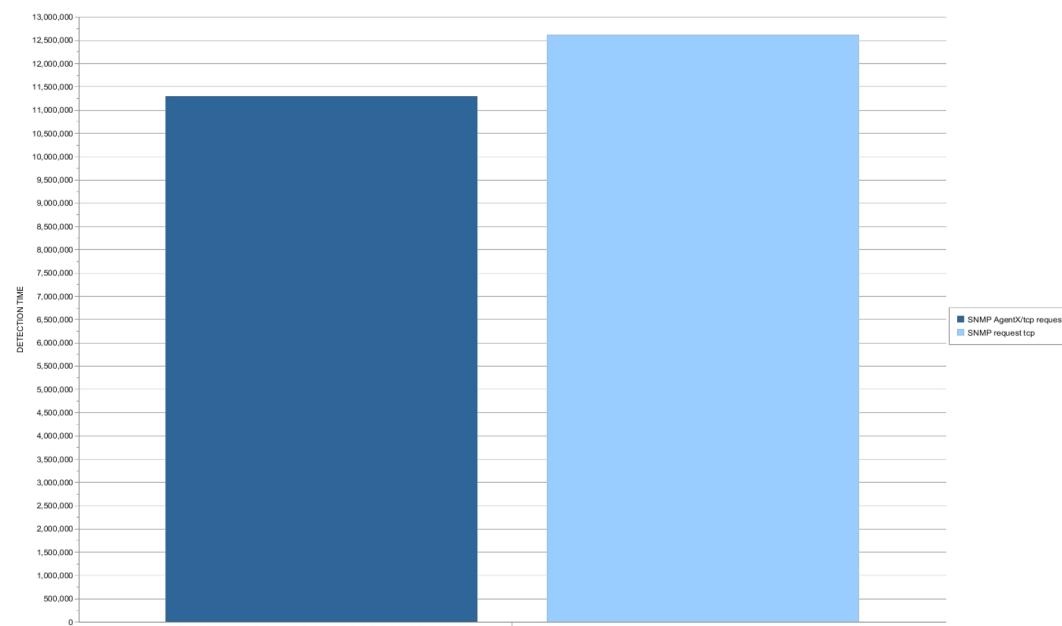
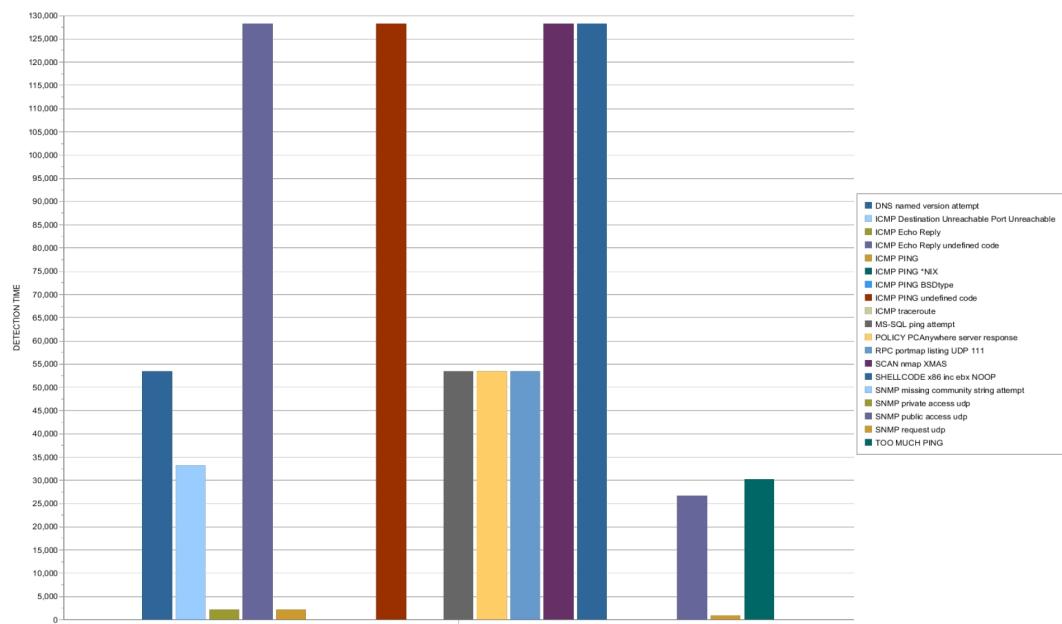
- **ICMP Destination Unreachable Port Unreachable**, questo alert può indicare che qualcuno ha tentato di connettersi ad una porta o ad un sistema non disponibile e che probabilmente la sorgente di un determinato pacchetto sta effettuando una scansione o un'altra attività maligna.
- **ICMP PING \*NIX**, questo alert indica che è stato effettuato un ping originato da un host su cui è in esecuzione Unix.
- **SNMP public access udp**, questo alert viene generato quando si effettua una connessione di tipo SNMP tramite UDP utilizzando la community 'public'; molte implementazioni di SNMP sono preconfigurate con communities 'public' e 'private' e se non vengono disabilitate un attaccante può tentare di ottenere informazioni sul dispositivo.
- **SNMP request udp**, è un alert analogo all'SNMP request tcp, generato quando si invia una richiesta per determinare se un dispositivo stia usando l'SNMP; in questo caso la porta sulla quale l'attaccante invia un pacchetto è di tipo udp.
- **MS-SQL ping attempt**, questo alert viene generato quando si identifica un tentativo di SQL ping nel traffico di rete. SNORT segnala che può essere stato utilizzato Nessus per accertarsi dell'esistenza di un database SQL sull'host e può preludere ad un attacco verso tale servizio o indicare che altri tool siano in uso per determinare lo stato dei server SQL.
- **POLICY PCAnywhere server response**, questo alert viene generato quando il traffico di rete indica l'utilizzo di un'applicazione o di un servizio potenzialmente in grado di violare una politica di corporate security. Una violazione della politica di corporate security può manifestare un serio rischio per le attività della compagnia.
- **RPC portmap listing UDP 111**, il servizio di mappatura delle porte registra tutti i servizi RPC (Remote Procedure Call) sugli host di tipo UNIX. Quando un evento generato quando è possibile che sia stato effettuato un tentativo di scoprire quali servizi RPC siano offerti e su quali porte siano in ascolto.
- **DNS named version attempt**, questo alert viene generato quando viene effettuato un tentativo di scoprire la versione di BIND (Berkeley Internet Name Domain, il più usato specialmente su sistemi Unix) specifica che il server DNS sta eseguendo. Così facendo un potenziale attaccante può scoprire quali server siano potenzialmente vulnerabili a degli exploit associati ad una versione di BIND.
- **ICMP traceroute**, questo alert viene generato quando è probabile sia stato identificato Windows traceroute, comando generalmente utilizzato da un attaccante che voglia determinare gli host ed i router attivi su una rete in preparazione di un attacco.
- **SNMP missing community string attempt**, questo alert viene generato quando le comunicazioni SNMP non contengono un nome di community. Un nome di

community SNMP è il processo di autenticazione che un host, che esegue SNMP, utilizza per concedere l'accesso. Fornendo una community vuota e un attaccante può tentare di ottenere l'accesso a funzionalità SNMP per un dispositivo che non è correttamente ben configurato.

- **ICMP PING BSDtype**, questo alert indica che è stata inviata dalla rete una richiesta di ping. Questo genere di richieste solitamente vengono utilizzate per determinare se un host è "responsive", ma sono anche utilizzabili per mappare la rete.
- **TOO MUCH PING**, questo alert viene generato ogni qualvolta che il numero di ping sottomessi al secondo sono più di 10. Permette di rilevare una sorta di DoS improntato su un eccessivo invio di ping verso l'host monitorato.
- **SNMP private access udp**, questo alert viene generato quando viene stabilita una connessione SNMP su UDP utilizzando la community di default. SNMP (Simple Network Management Protocol) utilizza le community e gli indirizzi IP per autenticare la comunicazione tra il client SNMP e il demone SNMP. Molte implementazioni di SNMP sono preconfigurate con community "private" e "public". Se queste non sono disattivate, l'attaccante può raccogliere una grande quantità di informazioni sul dispositivo che esegue il demone SNMP.

Analizzando i tempi di detection raggruppati e mediati per tipo di alert generato, vediamo come ci siano due tipi di alert, "SNMP request tcp" e "SNMP AgentX/tcp request" (per comodità visualizzati a parte), che registrano dei tempi di rilevazione molto più alti della media: gli attacchi che generano questi alert (metasploit ed nmap) sono tra i più difficili da rilevare tra quelli esaminati in questi esperimenti. Osservando i valori di detection relativi agli altri alert, possiamo vedere come in generale gli alert di pacchetti ICMP siano quelli più velocemente rilevati (alcuni di essi hanno addirittura tempo di rilevazione nullo rispetto all'arrivo del primo pacchetto), mentre altri, come ad esempio "SCAN nmap XMAS", richiedano più tempo per essere rilevati da Snort.

Inoltre analizzando il tempo di detection per ogni singolo alert generato si può vedere che attacchi più specifici del semplice ping sul protocollo ICMP hanno un tempo di detection più alto. Come ad esempio *ICMP PING undefined code* e *ICMP Echo Reply undefined code*.



RPC portmap listing UDP 111	SCAN nmap XMAS	SHELLCODE x86 inc ebx NOOP	SNMP AgentXtcp request	SNMP missing community string attempt	SNMP private access udp	SNMP public access udp	SNMP request tcp	SNMP request udp	TOO MUCH PING
DETECTION TIME	DETECTION TIME	DETECTION TIME	DETECTION TIME	DETECTION TIME	DETECTION TIME	DETECTION TIME	DETECTION TIME	DETECTION TIME	DETECTION TIME
53407	128335	128335	11293125			26703	12618845	890	30264

MSG										
DNS named version attempt	ICMP Destination Unreachable Port Unreachable	ICMP Echo Reply	ICMP Echo Reply undefined code	ICMP PING	ICMP PING *NIX	ICMP PING BSDtype	ICMP PING undefined code	ICMP traceroute	MS-SQL ping attempt	POLICY PCAnywhere server response
DETECTION TIME	DETECTION TIME	DETECTION TIME	DETECTION TIME	DETECTION TIME	DETECTION TIME	DETECTION TIME	DETECTION TIME	DETECTION TIME	DETECTION TIME	DETECTION TIME
53407	33150	2131	128335	2131			128335		53407	53407

# Conclusioni

Concludiamo adesso con alcune considerazioni sui tools, gli esperimenti e le analisi viste in questo lavoro.

Il tool principale di questo lavoro, Snort, si è rivelato essere molto utile e molto potente: non soltanto permette una rapida esecuzione tramite una serie di semplici comandi, ma da anche la possibilità all'utente di poter personalizzare molti dei suoi aspetti salienti. Per questo progetto, ad esempio, è stata scritta da zero una sola regola Snort (TOO MUCH PING) in quanto l'obiettivo principale era l'analisi temporale dei risultati e non la detection di più attacchi possibile, ma viste le potenzialità offerte non è assurdo ammettere che questo tool possa essere benissimo utilizzato in qualsiasi ambito che abbia a che fare con una rete di sistemi.

Dall'analisi temporale si è visto che le tempistiche di dection, al variare della tipologia dell'attacco, sono comunque ampiamente sotto il secondo e le tipologie di attacco non rilevate riscontrate sono state essenzialmente due traceroute e nmap con l'opzione -sn. Questo osservando inoltre che Snort è stato utilizzato con la totalità delle regole di dection può dare una buona confidenza sulla sua efficienza. Inoltre è supportato da una vasta community che costantemente segnala bug, aiuta lo sviluppo del software e, soprattutto, condivide con il mondo le proprie regole Snort, fornendo all'utente finale una maggior copertura e possibilità di scelta per quanto riguarda la customizzazione di Snort sul proprio sistema.

L'altro tool, utilizzato nella fase di analisi, è lo strumento di Business Intelligence Pentaho Enterprise Edition: questo tool di BI ci ha permesso, in modo molto veloce, efficiente ed elegante, di organizzare e modificare i dati raccolti dagli esperimenti, tramite il tool Kettle di Data Integration, e successivamente di poter effettuare analisi OLAP su questi dati, tramite lo strumento Mondrian di Business Analytics.

La parte relativa agli esperimenti ci ha permesso di poter esaminare più da vicino alcuni comandi, come nmap o traceroute, o veri e propri tool di penetration testing come Metasploit. Certo gli esperimenti eseguiti non sono particolarmente interessanti o complessi ma nemmeno vogliono esserlo: dal momento che gli obiettivi principali di questo lavoro

sono l'analisi dei dati ottenuti dagli esperimenti e la corretta applicazione di una valida metodologia di testing, piuttosto che gli esperimenti stessi, si è scelto di eseguire esperimenti semplici ed efficaci, per poterci poi concentrare sulla fase di analisi.

Infine l'analisi temporale dei risultati sperimentalni ci ha permesso innanzitutto di osservare da diversi punti di vista i vari tool o protocolli coinvolti negli esperimenti, dandoci la possibilità di trarre deduzioni o avanzare ipotesi sull'effettivo comportamento di questi. Inoltre è stato personalmente gratificante poter sfruttare le conoscenze acquisite durante il corso di Data Warehousing, sia per quanto riguarda tutta la teoria sottostante all'analisi multidimensionale OLAP, sia riguardo all'utilizzo di strumenti come Pentaho, che ci hanno agevolato non poco tutta la fase di analisi dei file di log.

## Codice

Codice A.1: Codice Java utilizzato per arricchire il log CSV con una serie di misure temporali derivate.

```
1 import java.io.FileReader;
2 import java.io.FileWriter;
3 import java.io.IOException;
4 import java.util.List;
5
6 import au.com.bytecode.opencsv.CSVReader;
7 import au.com.bytecode.opencsv.CSVWriter;
8
9 public class ProcessCSV {
10
11     public static long getAbsolute(String[] row) {
12         long absolute;
13
14         absolute = ((Long.parseLong(row[4]) * 60 + Long.
15                     parseLong(row[5])) * 60 + Long.
16                     parseLong(row[6])) * 1000000 +
17                     Long.parseLong(row[7]));
18
19         return absolute;
20     }
21
22     public static long getFirstMsg(String exp_id, String msg
23                                     ,
24                                     List<String[]> list) {
25         long min = Long.MAX_VALUE;
26         long min_temp;
```

```

25         for (String [] row : list) {
26             if (row[1].equals(exp_id) && row[9].
27                 equals(msg)) {
28                 min_temp = getAbsolute(row);
29                 if (min_temp < min)
30                     min = min_temp;
31             }
32         }
33     }
34
35     public static long getFirstAlert(String exp_id, List<
36         String[]> list) {
37         long min = Long.MAX_VALUE;
38         long min_temp;
39         for (String [] row : list) {
40             if (row[1].equals(exp_id) && !row[9].
41                 equals("PACKET_SENT")
42                     && !row[9].equals("PACKET_RECEIVED")
43                     && row[17].equals("IDS"))
44             ) {
45                 min_temp = getAbsolute(row);
46                 if (min_temp < min)
47                     min = min_temp;
48             }
49         }
50     }
51
52     public static long getLastMsg(String exp_id, String msg,
53         List<String[]> list) {
54         long max = 0;
55         long max_temp;
56         for (String [] row : list) {
57             if (row[1].equals(exp_id) && row[9].
58                 equals(msg)) {
59                 max_temp = getAbsolute(row);
60                 if (max_temp > max)
61                     max = max_temp;
62             }
63         }
64     }
65 }
```

```
61
62     }
63
64     public static void writeExperiment(String exp_id, List<
65         String[]> list,
66         CSVWriter writer) {
67
68         String msg = "PACKET_SENT";
69         long first_sent = getFirstMsg(exp_id, msg, list);
70
71         msg = "PACKET RECEIVED";
72         long injection = getFirstMsg(exp_id, msg, list);
73
74         long detection = getFirstAlert(exp_id, list);
75
76         long finished = getLastMsg(exp_id, msg, list);
77
78         long duration = finished - first_sent;
79
80         long detection_time;
81         if (detection == Long.MAX_VALUE)
82             detection_time = -1;
83         else
84             detection_time = detection - injection;
85
86         long delay = (injection - first_sent);
87
88         for (String[] row : list) {
89             if (row[1].equals(exp_id)) {
90                 String[] new_row = { row[0], row[1], row[2], row[3], row[4],
91                     row[5], row[6], row[7], row[8], row[9], row[10],
92                     row[11], row[12], row[13], row[14], row[15],
93                     row[16], row[17], String.valueOf(
94                         row[18]) };
95
96             }
97         }
98     }
99 }
```

```

91                                     first_sent) ,
String.valueOf(
injection),
String.
valueOf(
detection),
String.valueOf(
finished),
String.
valueOf(
duration),
String.valueOf(
detection_time
), String.
valueOf(
delay) };

writer.writeNext(new_row);

95
}

97 }

99 public static void main(String [] args) throws
IOException {

101     CSVReader csv = new CSVReader(new FileReader(
        "C:\\\\Users\\\\Tommy\\\\Uni\\\\Progetti
        & Esercizi\\\\AQS\\\\log.csv"),
        ';' );
103

105     List<String[]> list = csv.readAll();

107     csv.close();

109     CSVWriter writer = new CSVWriter(new FileWriter(
        "C:\\\\Users\\\\Tommy\\\\Uni\\\\Progetti
        & Esercizi\\\\AQS\\\\log2.csv")
        ,
        ',');
111

113     String[] header = { "ID_ALERT" , "EXP_ID" , "DAY" ,
        "MONTH" , "HOUR" ,
        "MINUTE" , "SEC" , "MICROSEC" , "
        SID" , "MSG" , "PROTOCOL" ,

```

```

115          "FROM_IP" , "FROM_PORT" , "TO_IP" ,
116          "TO_PORT" , "FROM_MAC" ,
117          "TO_MAC" , "ATT_IDS" , "FIRST_SENT"
118          " , "INJECTION" , "DETECTION" ,
119          "FINISHED" , "DURATION" , "
120          DETECTION_TIME" , "DELAY" } ;
writer . writeNext ( header ) ;

121
122 String exp_id ;
123
124 exp_id = " metasploit_advance" ;
writeExperiment ( exp_id , list , writer ) ;

125 exp_id = " metasploit_default" ;
writeExperiment ( exp_id , list , writer ) ;

126 exp_id = " nmap_default" ;
writeExperiment ( exp_id , list , writer ) ;

127 exp_id = " nmap_A" ;
writeExperiment ( exp_id , list , writer ) ;

128 exp_id = " nmap_min_rate_100" ;
writeExperiment ( exp_id , list , writer ) ;

129 exp_id = " nmap_min_rate_300" ;
writeExperiment ( exp_id , list , writer ) ;

130 exp_id = " nmap_min_rate_500" ;
writeExperiment ( exp_id , list , writer ) ;

131 exp_id = " nmap_osscan_guess" ;
writeExperiment ( exp_id , list , writer ) ;

132 exp_id = " nmap_T2" ;
writeExperiment ( exp_id , list , writer ) ;

133 exp_id = " nmap_T3" ;
writeExperiment ( exp_id , list , writer ) ;

134 exp_id = " nmap_T4" ;
writeExperiment ( exp_id , list , writer ) ;

```

```

155     exp_id = "nmap_-T5";
156     writeExperiment(exp_id, list, writer);
157
158     exp_id = "nmap_-version-all";
159     writeExperiment(exp_id, list, writer);
160
161     exp_id = "nmap_-version-light";
162     writeExperiment(exp_id, list, writer);
163
164     exp_id = "ping_-i_0.001";
165     writeExperiment(exp_id, list, writer);
166
167     exp_id = "ping_-i_0.1";
168     writeExperiment(exp_id, list, writer);
169
170     exp_id = "ping_-i_0.3";
171     writeExperiment(exp_id, list, writer);
172
173     exp_id = "ping_-i_0.5";
174     writeExperiment(exp_id, list, writer);
175
176     exp_id = "traceroute_-I";
177     writeExperiment(exp_id, list, writer);
178
179     exp_id = "traceroute_-N_20";
180     writeExperiment(exp_id, list, writer);
181
182     exp_id = "traceroute_-N_30";
183     writeExperiment(exp_id, list, writer);
184
185     exp_id = "traceroute_-N_40";
186     writeExperiment(exp_id, list, writer);
187
188     exp_id = "traceroute_default";
189     writeExperiment(exp_id, list, writer);
190
191     exp_id = "traceroute_-T";
192     writeExperiment(exp_id, list, writer);
193
194     writer.close();
195 }
```

Codice A.2: Regola di Snort utilizzata lato attaccante per loggare i pacchetti in uscita.

```
1 alert 192.168.137.170 any -> any any (msg: "PACKET_SENT"; sid :50002;)
```

Codice A.3: Regole di Snort utilizzate lato IDS per loggare i pacchetti in entrata e rilevare un eccessivo numero di ping.

```
1 alert 192.168.137.170 any -> 192.168.137.180 any (msg: "PACKET RECEIVED"; sid:51004;)  
alert icmp any any -> 192.168.137.180 any (msg: "TOO MUCH PING"; threshold: typeboth , track by_src , count 10, seconds 3; classtype:attempted-dos; sid:51000;)
```