

Università degli Studi di Camerino

SCUOLA DI SCIENZE E TECNOLOGIE

Corso di Laurea in Informatica (Classe L-31)



**Suricata:
caso di studio di
Intrusion Detection and Prevention System**

Laureando

Mirco Pazzaglia
Matricola 081190

Relatore

Prof. Fausto Marcantoni

A.A. 2011/2012

*«Learn the rules so you know
how to break them properly.»*

Dalai Lama

*«The quieter you become,
the more you can hear.»*

Ram Dass

Ringraziamenti

«Ringrazio tutti e tutto,
niente e nessuno»

I miei più sentiti ringraziamenti vanno a tutti coloro che hanno fatto e fanno parte della mia vita.

Ringrazio la mia famiglia per il supporto e l'aiuto che mi ha fornito, e che da sempre mi segue in ogni mio passo verso il futuro. Ringrazio Noemi, mia sorella, che da sempre crede in me e mi sprona a dare il meglio dandomi la fiducia di cui ho bisogno.

Non posso mancare di ringraziare **tutti** i miei amici, da coloro che mi seguono dalla mia infanzia a quelli che si sono aggiunti nel corso degli eventi della mia vita. Compagni d'ogni momento, dal più triste al più felice, hanno tutti contribuito ad arricchire il mio essere e sono loro infinitamente grato per le esperienze che abbiamo condiviso.

Un ringraziamento anche ai miei colleghi di studio con i quali c'è sempre stato aiuto e supporto reciproco. Nello specifico vorrei ringraziare i miei due, oltre che amici, *colleghi Moschettieri* Giacomo e Giacomo con i quali ho condiviso i momenti migliori di questo percorso universitario, siano stati di studio, di svago o di brainstorming.

Ringrazio il corpo docenti del corso di laurea in Informatica dell'Università di Camerino e nello specifico il Prof. Fausto Marcantoni, il quale si è sempre dimostrato disponibile e pronto a condividere le sue conoscenze senza mai mancare della giusta dose di spirito e di allegria.

Ringrazio anche me stesso perché, dopotutto, se sono giunto fin qui è anche merito mio.

Indice

1	Introduzione.....	9
1.1	Panorama sulle tecnologie IDS/IPS.....	9
1.2	OISF	18
1.3	Suricata.....	19
1.4	Comparativa con altri software IDPS.....	20
1.4.1	Comparativa con Snort.....	21
2	Installazione ed implementazione.....	26
2.1	Le versioni di Suricata.....	27
2.2	GNU/Linux.....	28
2.2.1	Installazione da pacchetto.....	28
2.2.2	Compilazione sorgenti.....	30
2.2.3	Verifica dell'installazione.....	34
2.3	Microsoft Windows.....	34
2.3.1	Installazione tramite Windows Installer.....	35
2.3.2	Compilazione sorgenti.....	36
2.3.3	Verifica dell'installazione.....	39
2.3.4	Differenze dalla versione Linux.....	39
2.4	Aggiornamento di Suricata.....	40
2.5	Opzioni da riga di comando.....	41
2.6	File di configurazione suricata.yaml.....	43
2.7	Le regole.....	51
2.7.1	Live Rules Swap.....	55
2.7.2	Emerging Threats.....	57
2.7.3	Vulnerability Research Team	58
2.7.4	Rule Manager.....	58
2.8	Distribuzioni IDS/IPS pronte all'uso.....	59
2.8.1	Smooth-Sec.....	59
2.8.2	Security Onion.....	61
3	Report, segnalazioni e tools.....	63
3.1	File di log.....	64
3.2	Unified2 e Barnyard2.....	67
3.3	Prelude e Prewikka.....	69

3.4 Squert, Snorby e Sguil.....	75
3.4.1 Sguil.....	75
3.4.2 Squert.....	76
3.4.3 Snorby.....	78
3.5 Rules Editor.....	80
4 Caso di studio.....	84
4.1 La topologia della rete.....	84
4.1.1 Host Vittima.....	84
4.1.2 Gateway con Suricata IDPS.....	85
4.1.3 Host Attaccante.....	87
4.1.4 Test di connettività.....	88
4.2 Suricata come Intrusion Detection System.....	91
4.3 Suricata come Intrusion Prevention System.....	95
4.3.1 Modalità in-line ed IPTables.....	96
4.3.2 Regole per il drop dei pacchetti.....	98
4.3.3 Test di prevenzione.....	98
4.4 Esempio aggiuntivo: Suricata come proxy.....	100
5 Funzionalità.....	103
5.1 PF_RING.....	104
5.2 NVIDIA CUDA.....	105
5.3 LUA.....	108
Conclusioni.....	112
Appendice.....	114
A- Parametri di compilazione.....	114
B- Lista di parametri per l'avvio di Suricata.....	120

1 Introduzione

In questo documento verrà trattato Suricata, quale software open source di Intrusion Detection and Prevention System di prossima generazione in sviluppo presso la fondazione OISF. Verrà spiegato di cosa si tratta e di come funziona; saranno presentati strumenti ed interfacce di gestione disponibili per questo software. Sarà poi presentato un caso d'uso del software cercando di dimostrarne le capacità e l'impiego in termini di sicurezza delle reti, siano esse aziendali o di privati. Saranno trattati, per quanto possibile, le sue caratteristiche, le sue capacità e le sue funzionalità ma verranno messi in mostra anche i suoi limiti e le sue problematiche.

Il presente documento non vuole essere una guida che tratta nello specifico ogni dettaglio del software, poiché sarebbe impensabile per programma ancora in corso di sviluppo e che avanza di versione a ritmi sostenuti. Inoltre, benché Suricata sia un software multiplatforma, alcune funzionalità sono esclusiva del mondo GNU/Linux, o funzionano al meglio in esso, pertanto, diversi passaggi a seguire saranno trattati solo per questo ambiente.

1.1 Panorama sulle tecnologie IDS/IPS

L'acronimo IDS sta per **Intrusion Detection System** (sistema di individuazione di intrusione) e si riferisce ad una componente sia solo software che hardware con software embedded dedicato, atta ad analizzare il traffico in transito da e verso una specifica rete entro la quale viene installata. Lo scopo di disporre un IDS in una rete (usualmente LAN) è quello di monitorare il traffico al fine di rilevare eventuali attività sospette e/o malevoli nei confronti di qualunque host (sia esso client o server) presente all'interno della rete. In base al tipo di IDS ed alla sua disposizione, esso può essere in grado di monitorare il traffico di rete da e verso Internet ed il traffico di dati interno alla

rete stessa. Tali sistemi sono spesso in grado di generare *alert* e *file di log* sulla base della loro attività di analisi o, avendo a disposizione database relazionali, salvare le informazioni sul traffico di rete repute di interesse dall'amministratore di rete.

L'acronimo IPS sta per **Intrusion Prevention System** (sistema di prevenzione di intrusione); come l'IDS, un sistema di prevenzione è una componente software o hardware disposta all'interno di una rete. Tuttavia, lo scopo di un IPS non è quello di tenere sotto controllo il flusso dei dati da e verso la rete in cui esso è collocato e di tenerne uno storico, ma quello di prevenire tentativi di attacco o movimenti sulla rete potenzialmente pericolosi per l'incolumità degli host che ne fanno parte. Questo è genericamente possibile per un IPS quando disposto in modalità *in-line*. Le comuni azioni di prevenzione prese da un IPS possono essere il *drop* dei pacchetti o della sessione incriminata, il *reset* della sessione o il *blocco* e l'aggiunta ad una *black-list* dell'host che muove l'attacco verso la rete.

IDS ed IPS sono tecnologie complementari nell'ambito della sicurezza delle reti e sono in grado di lavorare in sinergia. Entrambe sono accomunate dall'analisi del traffico di rete ed entrambe entrano in funzione (di alert o di prevention) sulla base del *matching* fra determinate regole fornite dall'amministratore di rete ed i pacchetti in transito. Proprio per questi motivi, diversi software implementano sia la funzione di *detection* che di *prevention* in un unico *engine*, dando così origine a prodotti ibridi noti con l'acronimo di IDPS o **Intrusion Detection and Prevention System**. Questi sistemi agiscono ai livelli 2, 3, 4 dello stack

TCP/IP. Questi sistemi spesso aggiungono alla componente sicurezza anche alcune funzionalità come la correzione di errori CRC (Cyclic Redundancy Check), la deframmentazio-

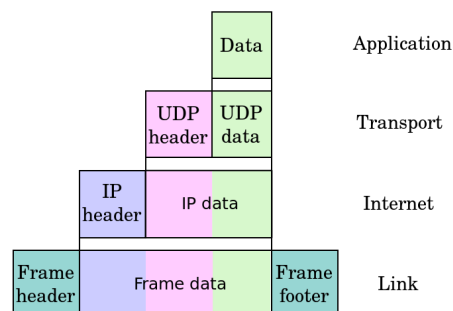


Figura 1.1: Stack TCP/IP

ne dei pacchetti, o il riordinare in sequenza i pacchetti TCP.[1]

Esistono quattro tipologie diverse di tecnologie IDPS[2]:

- **Network-Based IDPS:** monitorano il traffico, con particolare riguardo al livello applicativo e di rete, di specifici segmenti o dispositivi di rete. Tipicamente viene installato ai limiti topologici della rete per esempio in prossimità dei firewall che antecedono i gateway per altre reti o per Internet, o agli estremi di *zone demilitarizzate (DMZ)*. Questi *sensori* hanno spesso accesso al traffico di rete per mezzo di *monitor ports*¹ presenti nei switch o predisponendo delle *network tap*² nei cavi d'interesse.

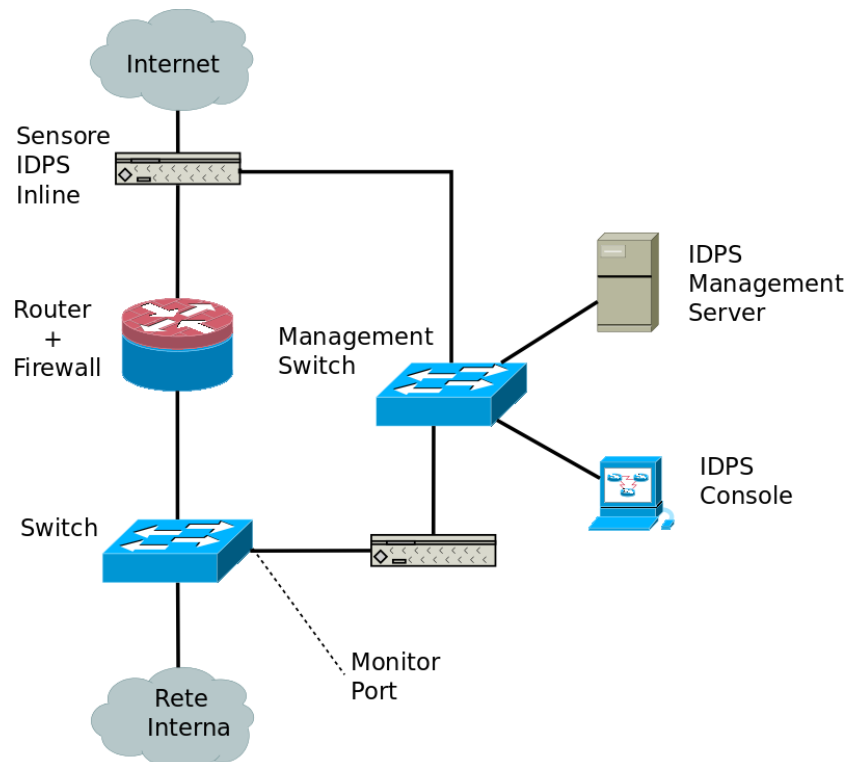


Figura 1.2: Esempio di network-based IDPS

-
- 1 Si tratta di una porta dedicata presente in switch professionali sulla quale vengono copiati e trasmessi tutti i pacchetti in transito su quello switch.
 - 2 Si tratta di un dispositivo hardware che si interpone in modo trasparente, fra due endpoint di un collegamento via cavo ed aggiunge una porta per l'ascolto in copia del traffico in transito.

- **Wireless IDPS:** monitorano il traffico wireless ed esclusivamente ciò che riguarda i protocolli di rete wireless. Non agiscono a livello applicativo o di rete.

Wireless Clients

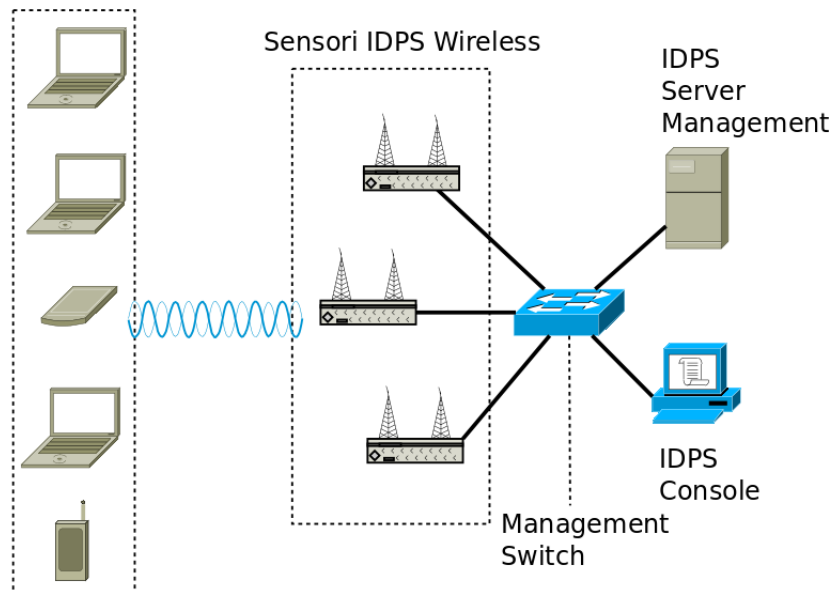


Figura 1.3: Esempio di wireless IDPS

- **Network Behaviour Analysis (NBA) IDPS:** esaminano il traffico per identificare minacce che generano traffico non comune per una rete come nel caso di tentati *attacchi DDoS* (Distributed Denial of Service), in presenza di malware o violazioni di policy. Vengono spesso impiegati per monitorare il traffico interno alle stesse reti, o nel caso in cui si voglia rendere disponibile l'accesso alla propria rete da parte di terzi.

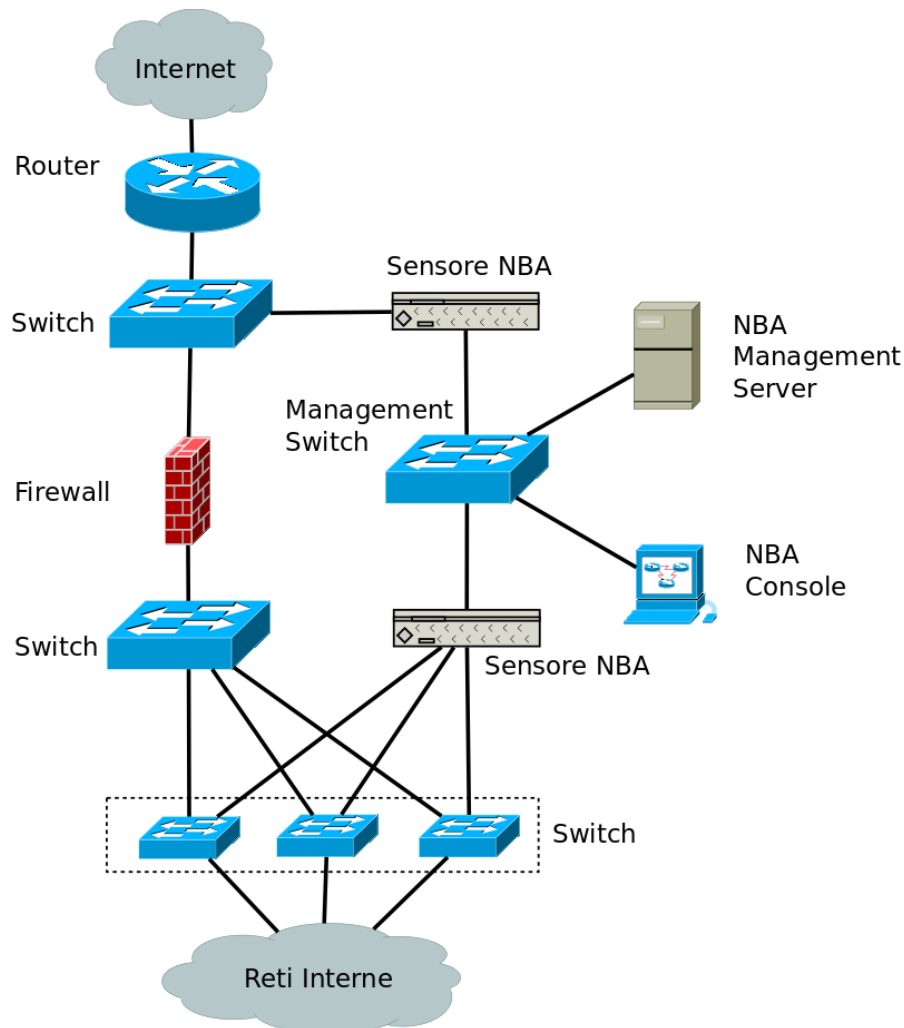


Figura 1.4: Esempio di NBA IDPS

- **Host-Based IDPS:** monitorano tutto ciò che accade all'interno del singolo host su cui vengono installati. L'host su cui vengono installati è in genere un server accessibile dall'esterno della rete o un client di accesso pubblico. I dati monitorati variano dai processi in esecuzione sulla macchina, agli accessi ai file, ai log di sistema, etc.

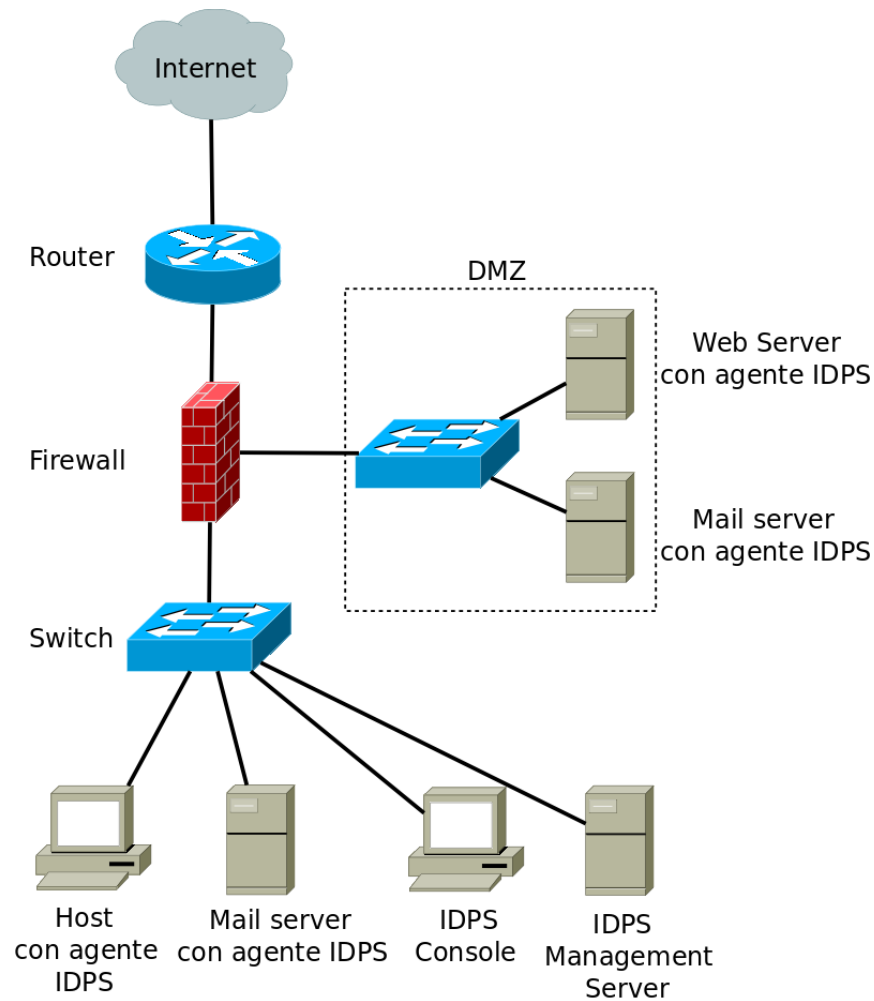


Figura 1.5: Esempio di host-based IDPS

Il contesto generale di utilizzo di questi sistemi di intrusione e prevenzione è quello aziendale. Il possesso ed il trattamento di dati sensibili da parte di aziende e organizzazioni, impone loro che vengano adoperati opportuni ed efficaci sistemi di sicurezza nelle loro reti interne onde evitare violazioni delle policy di sicurezza aziendali. Le tecnologie IDPS si vanno dunque ad affiancare (e mai a sostituire) ad altre tecnologie di sicurezza informatica quali *firewall* ed *antivirus*. Ad ogni modo, non sono rari casi di utilizzo di IDPS anche in ambito domestico, laddove si senta la necessità di aumentare le difese della propria rete o semplicemente si voglia sperimentare questo tipo di tecnologie.

La maggior parte degli IDPS utilizza diverse tecnologie di detection, eventualmente combinate, per fornire un miglior grado di accuratezza. Tre sono le principali tecnologie:

- **Signature-based:** effettuano una comparazione fra le *firme* (o signatures) note, gestite tramite apposite *rules* dall'amministratore di rete, ed i pacchetti in transito. Sono molto efficaci nell'individuare minacce note che hanno pattern di attacco statici ma sono quasi inutili nei confronti di minacce non note. L'analisi basata sulle firme non è inoltre in grado di tenere sotto controllo lo *stato* delle comunicazioni nel flusso di rete.
- **Statistical anomaly-based detection:** avendo a disposizione dati statistici riguardo i flussi di rete reputati normali, avvertono situazioni fuori dalla norma sulla base di quanto determinati parametri deviano dai propri valori standard. Questo metodo di identificazione di intrusione necessita di una fase preliminare di studio della rete per stabilire quali sono i valori normali. Se in questa fase di "taratura" sono però già in corso attacchi o situazioni anomale, questi produrranno una definizione di stato normale di rete errata, portando in futuro a casi di *falsi positivi* o a ritenere innocue situazioni pericolose.
- **Stateful protocol analysis:** analizzano il flusso di rete comparandolo con appositi profili che contengono azioni normalmente reputate non nocive nel contesto di specifici protocolli *stateful*. Così si può ad esempio controllare che un utente che ha accesso ad un server FTP, ma che non ha ancora ottenuto privilegi tramite autenticazione, non tenti di usare comandi non pertinenti al suo corrente stato di utente anonimo; nessuna segnalazione avverrebbe nel caso in cui invece tali comandi venissero eseguiti una volta guadagnato i diritti previa autenticazione. Questo tipo di analisi è molto sofisticata ma è al contempo molto diffi-

cile produrre profili per ogni protocollo, che comprendano tutti gli scenari possibili d'uso, leciti e non leciti. Altro contro è l'elevato costo computazionale che consegue dall'analisi degli stati.

Le soluzioni IDPS sono usualmente composte da più elementi. Indispensabili sono i **sensori** o **agenti** (nel caso di tecnologie host-based) ovvero la macchina su cui i software IDS/IPS sono in esecuzione. Ci può poi essere un server che centralizzi l'operato di più sensori. Questo viene chiamato **management server** e può essere in grado di effettuare un'analisi più consapevole di ciò che sta succedendo nella rete, potendo disporre dei dati di tutti i sensori. C'è poi un **database server** d'appoggio per lo storage dei log e degli alert dei sensori. Infine può essere di comodo una **console** per la gestione dei server e dei sensori da parte dell'amministratore di rete.

A seguire una figura che, alla luce di quanto detto finora, rappresenta una possibile implementazione di un sistema IDPS in una rete complessa.

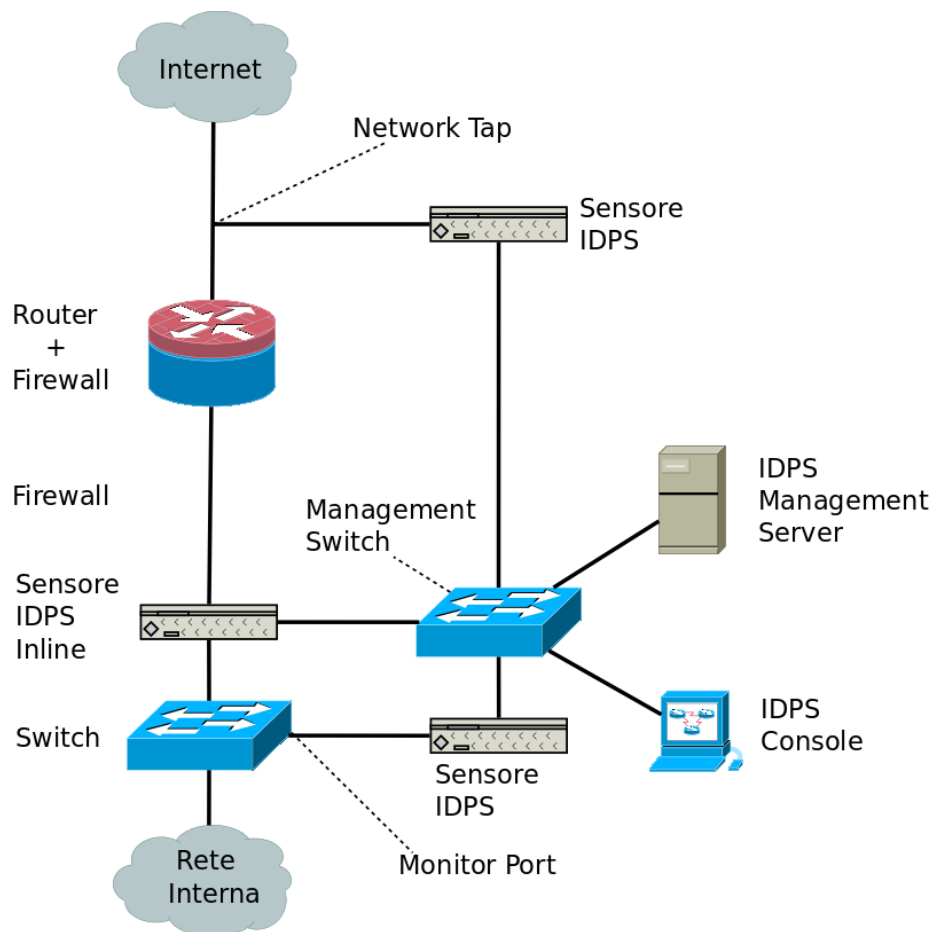


Figura 1.6: Esempio di rete con IDPS

1.2 OISF



La **Open Information Security Foundation** (OISF) è una fondazione non-profit il cui scopo primario è quello di creare un engine IDPS open source di prossima generazione. Nel core team si annoverano nomi quali Matt Jonkman (fondatore di Emerging Threats) e Victor Julien (creatore del firewall *Vuurmuur*) mentre fra i contributori vi è Luca Deri (leader del progetto *NTOP*). Diversi membri hanno inoltre alle loro spalle esperienza nel campo della sicurezza informatica e nello sviluppo di *Snort*, famoso software IDPS. Intrinseca nella natura open source del progetto di OISF è la partecipazione della community sia tramite *mailing list*³ sia partecipando attivamente allo sviluppo. I finanziamenti per il lavoro di OISF vengono direttamente da *US Department of Homeland Security (DHS)* e da compagnie private che formano l'*OISF Consortium*. La fondazione è raggiungibile, al giorno d'oggi, dal sito www.openinfosecfoundation.org ed è possibile entrarne in contatto come segue:

*117 North 5'th Street
Suite 318
Lafayette, Indiana 47901*

oisf-team@openinfosecfoundation.org

765-429-0398

³ <http://lists.openinfosecfoundation.org/mailman/listinfo>

1.3 Suricata



Figura 1.7: Logo di Suricata

Il progetto in sviluppo presso la OISF ed argomento principe di questa tesi è il software *Suricata*, un engine IDPS di carattere principalmente network-based. Il progetto, in accordo con la direzione della OISF, è open source e la data di rilascio nella sua prima versione è stata nel corso di dicembre 2009. In data 7 dicembre 2012 il software ha raggiunto la versione 1.3.5 nel ramo di produzione *stable* e la versione 1.4rc1 nel ramo di produzione *preview*. Il software è scritto in linguaggio C ed ha la prerogativa di essere multiplatforma. È possibile infatti compilare ed eseguire Suricata in ambiente Microsoft Windows, Linux, Mac OS, Unix e FreeBSD. Suricata viene rilasciato sotto licenza GPLv2⁴, eccezion fatta per le compagnie che fanno parte dell'OISF Consortium, alle quali viene concesso in licenza non GPL.

Benché il software non abbia ancora raggiunto la sua completezza, non manca di integrare tecnologie e funzionalità degne di un next-gen IDPS engine. Fra queste funzionalità si hanno ad esempio, il supporto al multithreading, l'uso di accelerazione grafica per l'analisi tramite la tecnologia nVidia CUDA® per il GPGPU, o il trigger di script in linguaggio LUA, output in formato Unified2 o Prelude o l'uso di Flow variables che permettono la crea-

⁴ I termini di licenza GPLv2 sono disponibili al seguente indirizzo

<http://www.gnu.org/licenses/gpl-2.0.html>

zione di regole sofisticate o ancora l'uso di HTP per il parsing di dati HTTP.

1.4 Comparativa con altri software IDPS

Suricata non è l'unico software di Intrusion Detection and Prevention System in circolazione al momento. Il mercato mette a disposizione diversi progetti gratuiti o a pagamento. Fra i vari software IDPS commerciali vi sono, ad esempio, i prodotti CISCO⁵, IBM⁶ o Juniper⁷. Queste soluzioni spesso hanno costi elevati, ma forniscono assistenza e aggiornamenti con estrema efficienza e celerità.

Il panorama dei software free invece è governato da software il cui sviluppo dipende non da enti privati ma da comunità finanziate dalla presenza di fondi (come la OISF) o meno in cui la velocità di sviluppo e l'aggiornamento delle regole di sicurezza procedono più lentamente. In questo scenario i protagonisti sono Snort e Suricata in ambito network-based ed OSSEC specializzato invece nell'host-based analysis. Altri software che svolgono funzioni di IDS e/o IPS sono per esempio Sagan⁸, Bro⁹, AIDE¹⁰, o OpenIDS¹¹.

5 http://www.cisco.com/en/US/products/ps5729/Products_Sub_Category_Home.html

6 http://www-947.ibm.com/support/entry/portal/overview//software/tivoli/proventia_network_intrusion_prevention_system

7 <http://www.juniper.net/us/en/products-services/security/idp-series/>

8 <http://sagan.quadrantsec.com/>

9 <http://www.bro-ids.org/>

10 <http://aide.sourceforge.net/>

11 <http://openids.org/>

1.4.1 Comparativa con Snort



Snort è un software di Intrusion Detection and Prevention System network-based che nasce per mano di Matt Roesch nel 1998. Da allora il progetto ha continuato ad evolversi ed è giunto alla versione 2.9.4 nel corso di novembre 2012. Scritto in C e rilasciato sotto licenza GPL. In un certo senso Snort è il precursore di Suricata, per il quale ha gettato le basi ed ha permesso la creazione dell'environment di SIEM (Security Information and Event Management) e software di reportistica o management tutt'oggi validi nell'uso con Suricata. Il motore IDPS di Suricata è inoltre snort-rule compliant, il che significa che supporta l'uso di regole scritte e formattate così come quelle per Snort, rendendo così facile la transizione da un sistema all'altro; la gestione di un unico repository di regole risulta sicuramente più facile.

Ad oggi ci sono diverse peculiarità in Suricata che lo rendono un software di spicco e promettente per il futuro. A seguire, una tabella riassuntiva di paragone fra Suricata e Snort[3][4].

Parametro	Suricata	Snort
Modalità IPS	Opzionale tramite <code>enable-nfqueue</code>	Usando il parametro <code>-q</code> o l'applicativo <code>Snort_inline</code>
Regole	<ul style="list-style-type: none"> • VRT::Snort rules • EmergingThreats rules 	<ul style="list-style-type: none"> • VRT::Snort rules • SO rules • EmergingThreats rules
Thread	Multi-threaded	Single-threaded
Installazione	Perlopiù manuale	Tramite pacchetto
Log degli eventi	File, Unified2 per Barnyard2, database	
Documentazione	Poche risorse	Molte risorse
Analisi offline (pcap)	Sì	Sì
File di configurazione	File YAML	File .conf
Linguaggio di programmazione	C	C
Sistema Operativo	Multipiattaforma	Multipiattaforma
Supporto IPv6	Sì	Sì
Sviluppatore	OISF	Sourcefire, Inc.
In sviluppo da	1998	2009
Versione stabile corrente	1.3.5	2.9.4

Sébastien Damaye ha effettuato una serie di test sul campo per confrontare al meglio i due software. I dati che seguono sono solo perlopiù indicativi e

vanno presi con cautela:

Test	Priorità	# di test	Punteggio Suricata	Punteggio Snort
Test rules	3	8	6	8
Bad Traffic (non RFC compliant)	2	4	1	1
Frammentazione pacchetti	2	2	1	3
Login falliti consecutivamente	3	1	1	0
Tecniche di evasione	2	15	21	29
Malware & viruse	3	14	9	7
Shellcodes	3	11	12	7
Denial of Service (DoS)	3	3	3	3
Attacchi Client-side	3	257	127	157
Performance	3	0	2	1
Modalità Inline / Prevention	2	0	1	1
TOTALE (Somma)		315	184	217
TOTALE (Somma pesata)			528	617

Un'ulteriore analisi comparativa delle performance[5] vede Snort avere un peso minore sulla CPU in modalità single-thread ma una scalabilità inferiore nel momento in cui si passa ad architetture multicore o multicpu e genericamente una minore precisione nell'analisi rispetto a Suricata.

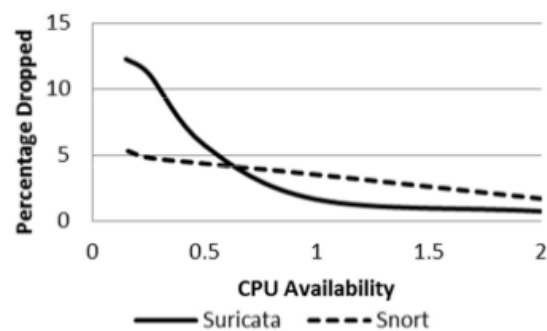


Figura 1.8: Perdita di pacchetti a 3.2 MBps

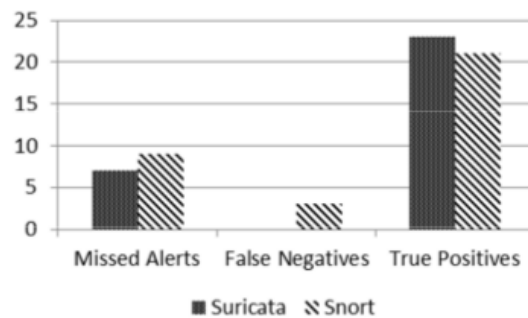


Figura 1.9: Precisione nell'individuazione di attacchi

Dalla figura 1.8 può evincere che Suricata ha un drop-rate minore dei pacchetti all'aumentare dei core presenti nella macchina in cui viene installato; al crescere dei core, Suricata riesce a scalare e a sfruttare a pieno l'architettura, a differenza di Snort per il quale l'uso di architetture multicore giova meno. Riuscire ad analizzare costantemente il flusso dei dati implica anche una maggiore precisione dell'analisi del traffico di rete. Infatti la figura 1.9 mostra come Suricata, nel caso di test, abbia meno falsi negativi e più riscontri positivi.

Con il grafico in figura 1.10 si vuole ribadire l'ottimizzazione di Snort per macchine a singolo core. Suricata infatti ha picchi di utilizzo di CPU non in-differenti nei confronti del concorrente.

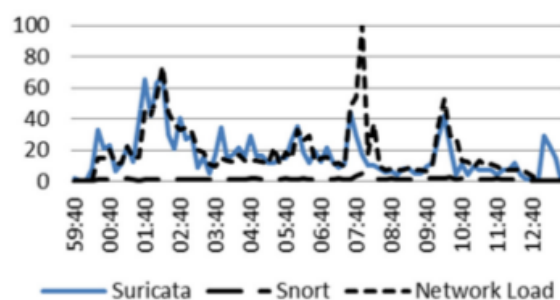


Figura 1.10: Snort e Suricata in architettura single-threaded

Diverso è lo scenario che si dipinge nel momento in cui si passa a macchine multicore come nella figura 1.11 e nella figura 1.12. Nel confronto dei due gra-

fici risulta chiaro che in ambiente multicore, Suricata goda di una migliore scalabilità.

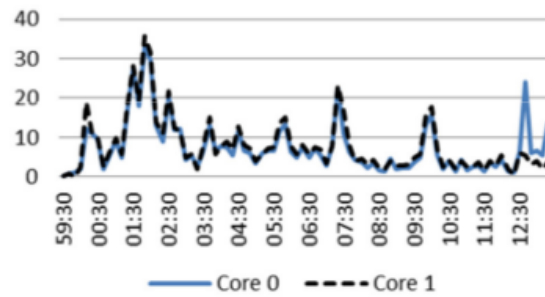


Figura 1.11: Carico di Suricata sulla CPU in uno scenario dual core

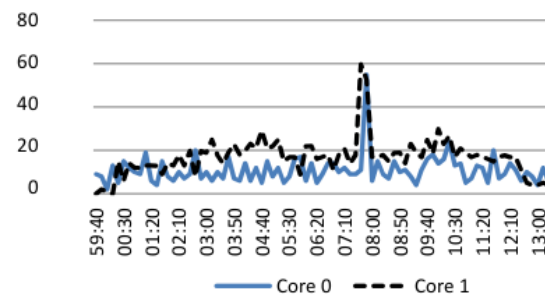


Figura 1.12: Carico di Snort sulla CPU in uno scenario dual core

2 Installazione ed implementazione

In questo capitolo verranno affrontate nello specifico le procedure di installazione e configurazione di Suricata IDPS. Per via della natura open e multi-piattaforma del progetto si può disporre di diverse versioni del software. Oltre a dover scegliere la versione opportuna per le proprie esigenze, si può dover ricorrere alla ricompilazione del codice del programma per poter abilitare o integrare eventuali funzionalità desiderate.

Bisogna prestare poi attenzione ai settaggi nel file di configurazione di Suricata. Nel file *suricata.yaml* vengono definiti vari parametri di funzionamento per il software ed i percorsi di sistema per caricare le regole o salvare i file di log, oltre che per specificare determinati comportamenti.

Le regole sono anch'esse di vitale importanza per il funzionamento di Suricata. I file *.rules* contengono le definizioni che vengono caricate dall'engine all'avvio del programma, e consentono a Suricata di analizzare il traffico di rete e prendere decisioni per i singoli pacchetti, scaturendo alert, salvando file di log e, se disposto in *modalità in-line* effettuare il drop. Verrà spiegato il funzionamento delle regole e come scriverne di proprie. Successivamente verranno presentati dei sistemi automatici di gestione ed aggiornamento delle regole, ed i repository da dove prelevare set di regole già pronte all'uso.

Infine saranno discussi degli ambienti SIEM preconfigurati con funzionalità di IDS/IPS svolti da Suricata. Questi sono immagini disco liberamente scaricabili dalla rete, pronti ad essere usati sia in modalità live-CD che installati su apposite macchine, rendendo meno faticosa l'installazione e la configurazione di sensori IDPS.

2.1 Le versioni di Suricata

Suricata è disponibile al download dal sito della OISF¹². I rami di sviluppo principali sono due, **stable** e **preview** e la pagina di download viene costantemente aggiornata al rilascio di ogni nuova versione, lasciando comunque reperibili le versioni outdated, nel caso si abbia la necessità di fare rollback da qualche nuova versione che abbia bug o che risulti instabile sulla propria macchina.

La versione di *preview* contiene di solito le più recenti funzionalità introdotte nel codice, le novità e le nuove feature che necessitano di essere testate dagli sviluppatori e dai membri della community che lo desiderano. Con tale versione bisogna però essere coscienti di far parte del processo di testing, poiché a tutti gli effetti il software in uso è ad uno stadio di sviluppo di *beta*.

La versione *stable*, invece, è quella che dispone di codice testato a lungo nel corso di precedenti fasi beta. Non contiene le ultime novità, ma è sicuramente più affidabile poiché la maggior parte dei difetti sono stati corretti nel processo di sviluppo. Negli ambiti in cui la sicurezza domina sul diletto di voler testare le ultime novità, questa è sicuramente la versione di cui si ha bisogno.

Suricata è rilasciato per i sistemi Microsoft Windows come eseguibile auto-installante (compilato a 32-bit) che facilita di molto l'installazione, lasciando comunque all'utente che ne fa uso, l'obbligo di accomodare il file di configurazione secondo le proprie esigenze.

Per i sistemi GNU/Linux è possibile invece che il pacchetto di installazione sia presente nei repository di default della distribuzione, o è altresì possibile rinvenire pacchetti di installazione nel web. Spesso, comunque, questi pacchetti o quelli contenuti nei repository sono versioni molto vecchie del software e quindi, sconsigliabili da utilizzare.

La fondazione OISF mette a disposizione di tutti i sorgenti del loro soft-

¹² <http://www.openinfosecfoundation.org/index.php/download-suricata>

ware così da permetterne la compilazione per il proprio sistema (sia esso Linux, Unix, Windows, Mac OS o FreeBSD). Usando i pacchetti di installazione precedentemente discussi viene meno una possibilità offerta invece dalla compilazione manuale del sorgente di Suricata: la possibilità di abilitare a piacimento determinate funzionalità. Le diverse specifiche di compilazione di Suricata verranno discusse in seguito.

In ultimo, a disposizione degli utenti, viene messo il *repository git* da cui è possibile prelevare il codice aggiornato alle ultimissime modifiche da parte del team OISF e della community.

2.2 GNU/Linux

Lo sviluppo di Suricata nasce e prosegue in ambiente GNU/Linux. Nel lavoro svolto e riassunto in questa tesi il software è stato utilizzato senza problemi con le versioni del kernel 3.2 e 3.6, ma il software dovrebbe poter essere eseguito senza particolari problemi a partire dalla versione 2.6 del kernel GNU/Linux. Le distribuzioni utilizzate sono state Linux Mint 13 e Linux Mint Debian Edition, rispettivamente in versione 64bit e 32bit.

2.2.1 Installazione da pacchetto

Per le distribuzioni basate su kernel GNU/Linux è possibile che il pacchetto di installazione di Suricata sia presente nei repository di default, come nel caso di Ubuntu e Linux Mint.

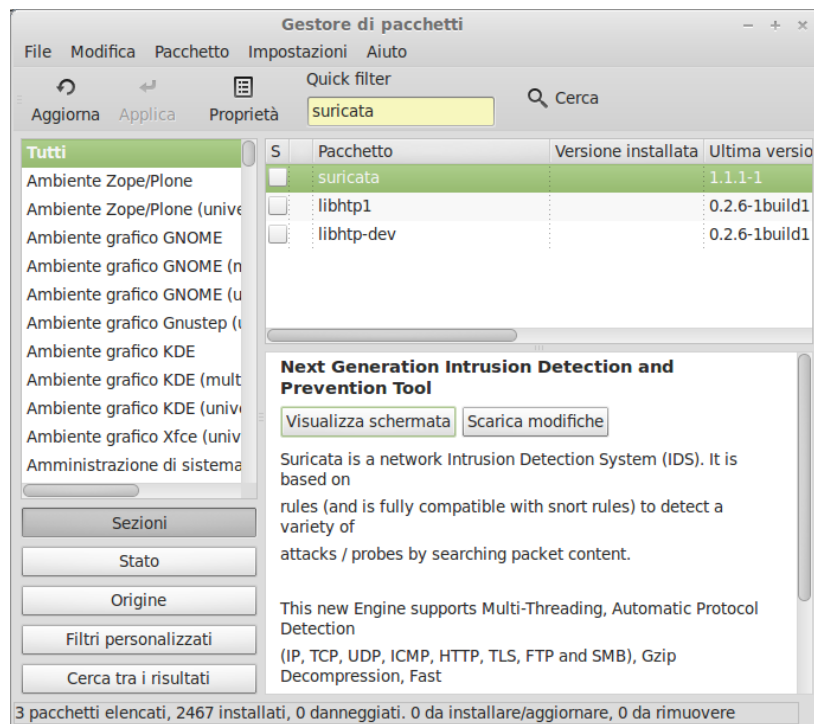


Figura 2.1

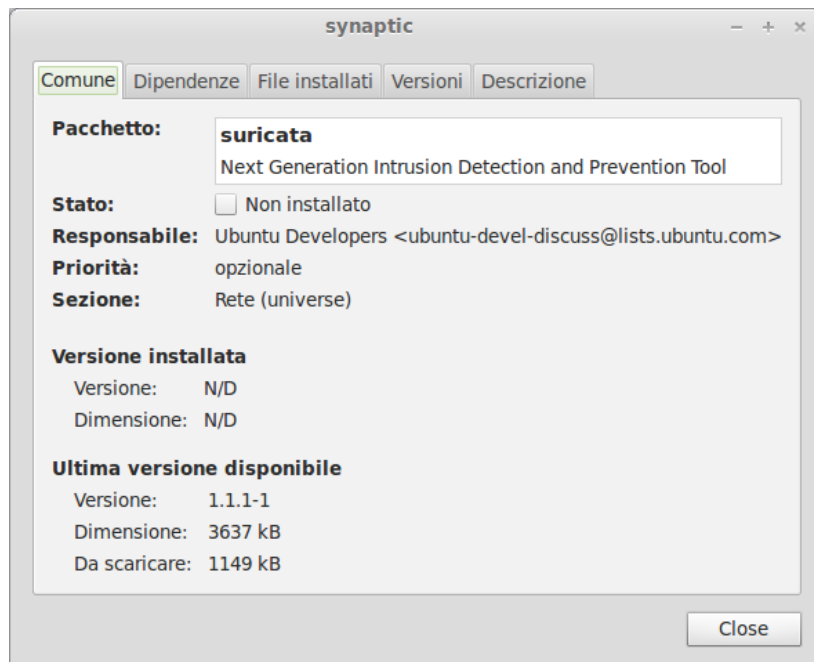


Figura 2.2

In tal caso sarà sufficiente selezionare il software dal proprio gestore di pacchetti tramite interfaccia grafica se si sta adoperando un desktop environment

oppure tramite shell, è possibile impartire il comando

```
# apt-get install suricata
```

qualora il proprio gestore di pacchetti sia *apt*¹³, come nelle distribuzioni basate su Debian.

Il gestore di pacchetti si occuperà di scaricare Suricata e tutte le *dipendenze* necessarie al suo funzionamento, lasciando all'utente il solo onere di modificare il file di configurazione e di occuparsi delle regole. Installare Suricata tramite un gestore di pacchetti può dare il vantaggio di poter installare, in maniera relativamente semplice, eventuali aggiornamenti del programma, qualora questi venissero inseriti nei repository. In tal caso, infatti, in ambiente Debian e derivati, basterebbe eseguire da shell:

```
# apt-get update && apt-get upgrade
```

In alternativa all'uso dei package manager è possibile trovare i pacchetti di installazione per la propria distribuzione direttamente sul web¹⁴.

2.2.2 Compilazione sorgenti

Come scritto precedentemente, l'installazione di Suricata da pacchetto risulta abbastanza facile. Il rovescio della medaglia di tale operazione è, ad ogni modo, il non poter disporre sempre dell'ultima versione rilasciata da OISF (sia essa del ramo Stable che Preview). In ambito di sicurezza informatica, specie quando ci si interessa a software che non hanno ancora raggiunto una piena maturità e il cui sviluppo procede a ritmi sostenuti quale è il caso di Suricata, è **molto** importante tenere il proprio software aggiornato.

13 Il comando da shell varia a seconda del proprio gestore di pacchetti. Per esempio, nel caso di *yum*, sarebbe stato *yum install suricata*.

14 Per esempio al seguente indirizzo http://pkgs.org/fedora-17/fedora-updates-x86_64/suricata-1.3.2-1.fc17.x86_64.rpm.html è possibile scaricare l'rpm di Suricata 1.3.2 .

Per avere sempre l'ultima versione disponibile di Suricata è necessario scaricarlo dalla pagina di download del sito ufficiale di OISF:

<http://www.openinfosecfoundation.org/index.php/download-suricata>

Una volta scaricato il file d'archivio per la versione desiderata, si può procedere con l'estrazione dei file sorgenti. L'estrazione può essere portata a conclusione tramite qualunque programma con interfaccia grafica o da terminale:

```
$ tar -xzvf suricata-1.3.4.tar.gz
```

Nel caso in cui invece si voglia utilizzare il codice aggiornato alle ultime modifiche si può ricorrere al download diretto dal repository git di OISF. Per scaricare il codice dal repository GIT è necessario (in ambiente GNU/Linux) eseguire il seguente comando una volta posizionatisi nella directory opportuna:

```
$ git clone https://github.com/inliniac/suricata.git
```

Ottenendo in output qualcosa di simile:

```
m1rcu2@m1rcu2-laptop ~/desktop/surigit $ git clone
https://github.com/inliniac/suricata.git
Cloning into 'suricata'...
remote: Counting objects: 26568, done.
remote: Compressing objects: 100% (4790/4790), done.
remote: Total 26568 (delta 21882), reused 26397 (delta 21736)
Receiving objects: 100% (26568/26568), 7.59 MiB | 55 KiB/s,
done.
Resolving deltas: 100% (21882/21882), done.
```

È bene ricordare che l'uso della versione beta o git di Suricata non è sempre consigliabile per via dei rischi di bug o instabilità che queste versioni di testing portano con loro.

Una volta che si hanno a disposizione i sorgenti di Suricata, è tempo di passare alla fase di compilazione, cioè generare il file eseguibile del programma per l'architettura della propria macchina.

Suricata non è un software completamente a se stante; infatti, nel suo svi-

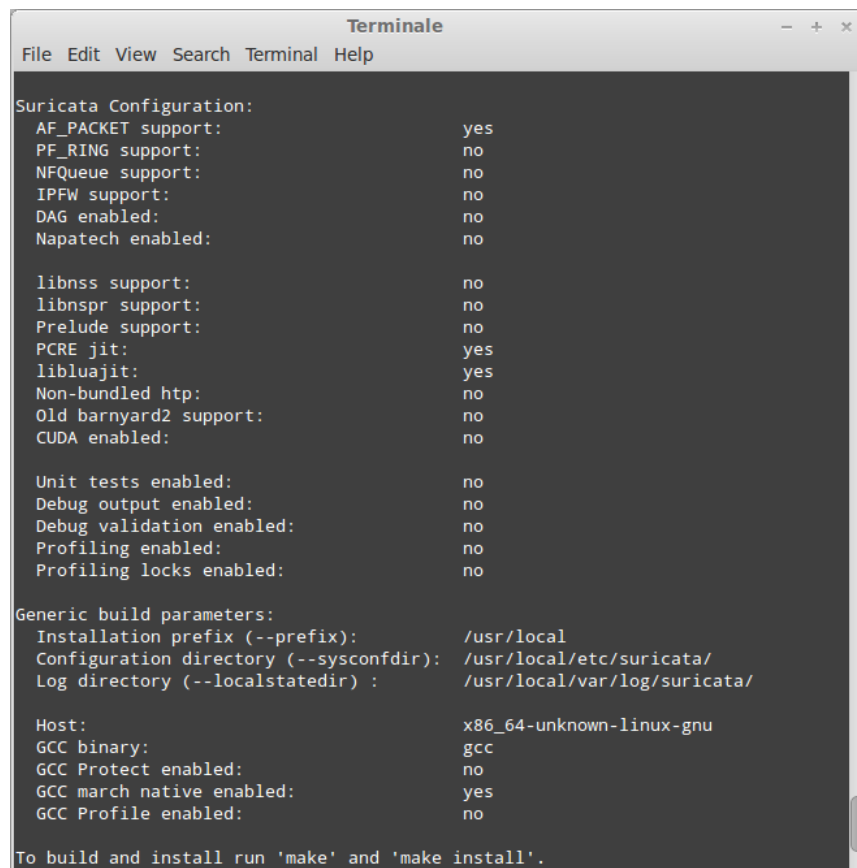
luppo sono state utilizzate diverse librerie prodotte da terzi nel panorama delle community open source. Questo significa che Suricata ha bisogno di soddisfare diverse dipendenze per poter procedere con una corretta compilazione, e quindi, laddove non presenti, i pacchetti mancanti dovranno essere installati nel proprio sistema. Segue una lista esaustiva delle dipendenze ad oggi richieste da Suricata:

- libc6
- libgcc1
- libcap-ng0
- libpcap0.8
- libmagic1
- libhttp1
- libnet1
- libnetfilter-queue1
- libnetlink0
- libpcre3
- libprelude2
- libyaml-0-2

Per installare le dipendenze si può fare ricorso al package manager della propria distribuzione o scaricare manualmente i pacchetti dal web.

Una volta certi di avere tutte le dipendenze installate (in caso negativo, la procedura di configure avviserà di cosa manca per poter procedere con la compilazione) si può procedere con la costruzione del file di make e la successiva compilazione del codice. Da terminale, una volta cambiata directory di lavoro alla root directory del codice di Suricata, impartire i seguenti comandi:

```
$ ./autogen.sh
$ ./configure --prefix=/usr --sysconfdir=/etc --
localstatedir=/var
```

```

Terminale
File Edit View Search Terminal Help

Suricata Configuration:
AF_PACKET support:          yes
PF_RING support:           no
NFQueue support:           no
IPFW support:               no
DAG enabled:                no
Napatech enabled:          no

libnss support:             no
libnspr support:           no
Prelude support:           no
PCRE jit:                   yes
libluajit:                  yes
Non-bundled htp:           no
Old barnyard2 support:     no
CUDA enabled:               no

Unit tests enabled:         no
Debug output enabled:       no
Debug validation enabled:   no
Profiling enabled:          no
Profiling locks enabled:    no

Generic build parameters:
Installation prefix (--prefix): /usr/local
Configuration directory (--sysconfdir): /usr/local/etc/suricata/
Log directory (--localstatedir) : /usr/local/var/log/suricata/

Host:                       x86_64-unknown-linux-gnu
GCC binary:                  gcc
GCC Protect enabled:         no
GCC march native enabled:    yes
GCC Profile enabled:         no

To build and install run 'make' and 'make install'.

```

Figura 2.3: Resoconto delle funzionalità abilitate dal `./configure`

La fase di configurazione del file di make è una fase molto delicata, infatti è in questo momento che si decide quali funzionalità si vorranno integrare in Suricata, come per esempio la modalità in-line, l'uso della tecnologia CUDA e così via. Il comando scritto precedentemente produrrà il binario di Suricata con le sue funzioni basilari, posizionando rispettivamente i file binari in `/usr/suricata` i file di configurazione e le regole in `/etc/suricata` e i file di log in `/var/log/suricata`. Tuttavia, per un'analisi più approfondita dei parametri di configurazione, si rimanda alle sezioni ed i capitoli a venire e all'appendice Parametri di compilazione.

Completata la fase di configurazione del file di make, si può procedere all'effettiva compilazione del codice:

```
$ make
```

In ultimo si deve procedere all'installazione dei file nelle rispettive directory.

```
# make install-full
```

Usando il parametro *install-full* piuttosto che *install*, make si prenderà carico di creare le directory d'uso di Suricata, di inizializzare il file di configurazione *suricata.yaml* e di scaricare il set di regole da *Emerging Threats*.

2.2.3 Verifica dell'installazione

Per accertarsi della corretta installazione di Suricata è possibile eseguire il comando:

```
$ suricata -V
```

se tutto è andato per il verso giusto, l'output sarà la versione di Suricata installata:

```
$ suricata -V
This is Suricata version 1.4dev (rev 02874a1)
```

2.3 Microsoft Windows

Anche in ambiente Microsoft Windows l'installazione di Suricata può essere svolta sia tramite pacchetto auto-installante sia da sorgente. Come si vedrà, l'operazione di compilazione risulta più complessa di quella svolta in ambiente GNU/Linux. Ad ogni modo, una volta che il programma sarà correttamente installato e configurato, svolgerà il suo lavoro esattamente come nella sua versione GNU/Linux.

Quale che sia il metodo di installazione scelto, si dovrà installare la libreria *Winpcap*. Suricata si avvale di questa libreria per ambienti Windows per gestire la cattura dei pacchetti in transito da far processare al suo motore di detection e prevention. Il pacchetto è disponibile gratuitamente al seguente indirizzo: <http://www.winpcap.org/install/default.htm> .

Come per GNU/Linux, le versioni che si hanno a disposizione sono tre:

- *Suricata Stable*, per una versione funzionante ma priva delle ultime modifiche ed aggiunte. Per l'ambiente Windows è disponibile sia sotto forma di Windows Installer che di sorgente da compilare.

- *Suricata Preview*, per una versione aggiornata alle ultime modifiche funzionanti ma ancora in fase di test e quindi potenzialmente instabile. Per l'ambiente Windows è disponibile sia sotto forma di Windows Installer che di sorgente da compilare.
- Da repository *git*, per una versione *bleeding-edge*, ossia aggiornata alle ultimissime modifiche o aggiunte; in quanto tale ha un più alto rischio di instabilità. È disponibile solo sotto forma di sorgente da compilare.

2.3.1 Installazione tramite Windows Installer

Installare Suricata tramite Windows Installer è una procedura abbastanza semplice. Poiché ogni release del team OISF è composta oltre che dai sorgenti anche da un file *Windows Installer .msi* per win32¹⁵, si userà quest'ultimo.

Scaricare il file .msi della versione Stable o Preview dal seguente indirizzo:

<http://www.openinfosecfoundation.org/index.php/download-suricata>

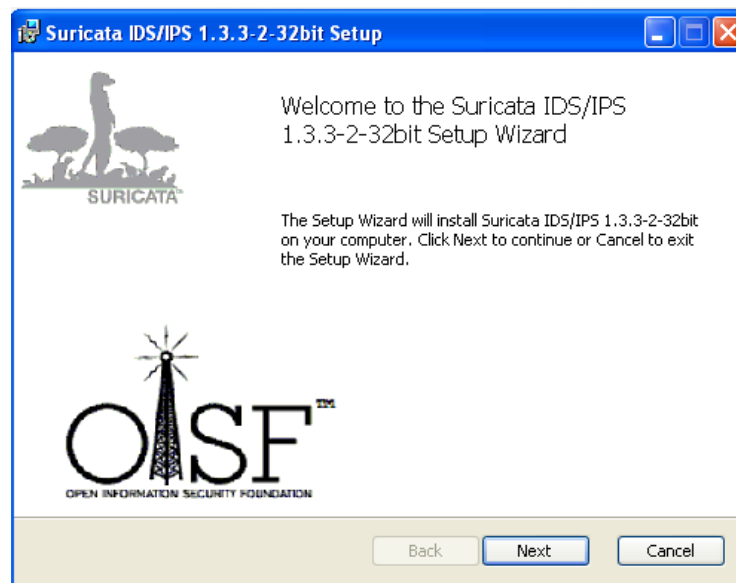


Figura 2.4: Finestra di installazione di Suricata per Win32

Dopodiché basterà eseguire il file di installazione e seguire le istruzioni a

¹⁵ Suricata non è al momento disponibile in versione 64bit per Microsoft Windows.

schermo.

Nota: di default, il percorso di installazione scelto dal Windows Installer è “C:\Program Files (x86)\Suricata 1.3.3-2-32bit”, dove al posto di 1.3.3-2-32bit c'è il numero della versione di Suricata che è stata installata. Poiché nel file di configurazione `suricata.yaml` (che verrà esaminato in seguito) distribuito con il file `.msi`, i percorsi di default per file di log, regole e altro hanno come percorso “C:\Program Files (x86)\Suricata”, Suricata non riuscirà ad avviarsi correttamente. Per ovviare a questo problema è sufficiente cambiare o i percorsi all'interno del file di configurazione o il nome della cartella di installazione di Suricata da “Suricata 1.3.3-2-32bit” a “Suricata”.

2.3.2 Compilazione sorgenti

I sorgenti di Suricata sono disponibili in versione Stable e Preview dalla pagina dei download di OISF (<http://www.openinfosecfoundation.org/index.php/download-suricata>) e sono gli stessi, essendo il progetto di tipo cross-platform, adoperati in ambiente GNU/Linux. In alternativa è possibile recuperare gli ultimi sorgenti dal repository `git://phalanx.openinfosecfoundation.org/oisf.git` .

Poiché nel suo codice, Suricata, si avvale di chiamate standard *POSIX*, in ambiente Windows è obbligatorio l'utilizzo di un ambiente *Unix-like* per la compilazione. È possibile utilizzare a tal proposito il software *Cygwin*¹⁶ della *Cygnus Solutions* (ora acquistata da *Red Hat*).

Tramite Cygwin si debbono installare i seguenti pacchetti necessari alla compilazione:

- *w32api* dal ramo *Libs*.
- *mpfr* dai rami *Libs* e *Maths*.
- *pthread* dal ramo *Devel*.

¹⁶ <http://cygwin.com/setup.exe>

- *gcc-core* e *gcc4-core* dal ramo *Devel*.
- *make*, *autoconf* ed *automake* dal ramo *Devel*.
- *glib* dai rami *Gnome* e *Libs*.
- *libtool* dal ramo *Devel*.
- *zlib* dai rami *Devel* e *Libs*.
- *pkg-config* dal ramo *Devel*.
- Facoltativo per l'installazione da repository git: *git* dal ramo *Devel*.

Scaricare le librerie per lo sviluppo di Winpcap da <http://www.winpcap.org/devel.htm> e spostare il contenuto dalla cartella Lib nella cartella Lib di Cygwin, rinominando poi il file *libwpcap.a* in *libpcap.a*. Copiare poi i file header dalla cartella Include nella cartella include di Cygwin (`cygwin\usr\include`).

Per rendere raggiungibili dal sistema le librerie di Cygwin si devono aggiungere i seguenti percorsi (assumendo che Cygwin sia installato nella cartella `C:\cygwin` come da installazione di default) “`C:\cygwin\bin;C:\cygwin\lib\pkgconfig;`” in append alla variabile di sistema PATH. (tasto destro su Computer → Proprietà → Avanzate → Variabili D'Ambiente).

Procedere poi con l'installazione di yam1 (sfruttato da Suricata per il suo file di configurazione) come segue:

Scaricare il pacchetto più recente da <http://pyyaml.org/download/libyaml> e decomprimerlo nella cartella *tmp* di Cygwin. Avviare Cygwin ed impartire il comando

```
$ cd /tmp/directory_di_yam1
$ ./configure --prefix=/usr && make && make install
```

Nel caso si volesse procedere con la compilazione dei sorgenti Stable/Preview, estrarre il contenuto del file precedentemente scaricato dal sito di OISF nella cartella *tmp* di Cygwin. Nel caso invece si voglia utilizzare il codice del repository git impartire i seguenti comandi da terminale:

```
$ cd /tmp
$ git clone git://phalanx.openinfosecfoundation.org/oisf.git
```

A questo punto, spostarsi con il comando `cd` nella directory con i file appena estratti o scaricati ed impartire

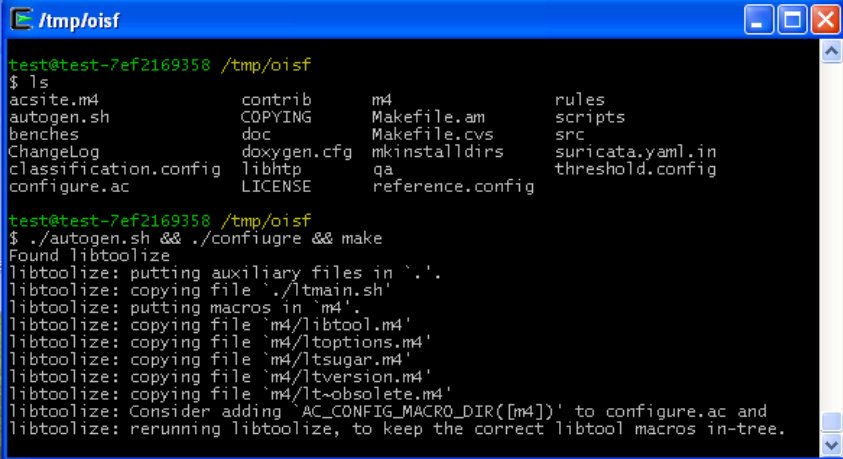
```
$ dos2unix.exe libhttp/configure.ac && dos2unix.exe
libhttp/http.pc.in && dos2unix.exe libhttp/Makefile.am
```

Infine per compilare nel caso dei file Stable o Preview, impartire

```
$ libtoolize -c && autoreconf -fv --install && ./configure &&
make
```

nel caso i sorgenti provenissero dal repository git, dare i comandi

```
$ ./autogen.sh && ./configure && make
```



```
test@test-7ef2169358 /tmp/oisf
$ ls
acsite.m4          contrib          m4              rules
autogen.sh        COPYING         Makefile.am     scripts
benches           doc             Makefile.cvs   src
ChangeLog         doxygen.cfg    m4installdirs  suricata.yaml.in
classification.config libhttp        qa              threshold.config
configure.ac      LICENSE        reference.config

test@test-7ef2169358 /tmp/oisf
$ ./autogen.sh && ./configure && make
Found libtoolize
libtoolize: putting auxiliary files in `.'.
libtoolize: copying file `./ltmain.sh'
libtoolize: putting macros in `m4'.
libtoolize: copying file `m4/libtool.m4'
libtoolize: copying file `m4/ltoptions.m4'
libtoolize: copying file `m4/ltsugar.m4'
libtoolize: copying file `m4/ltversion.m4'
libtoolize: copying file `m4/lt-obsolete.m4'
libtoolize: Consider adding `AC_CONFIG_MACRO_DIR([m4])' to configure.ac and
libtoolize: rerunning libtoolize, to keep the correct libtool macros in-tree.
```

Figura 2.5: Fase di compilazione di Suricata su Windows tramite Cygwin

A questo punto nella cartella `src/.lib` della cartella dei sorgenti di Suricata presente in `cygwin/tmp/`, è possibile trovare l'eseguibile appena compilato. Creare una cartella dove si desidera disporre Suricata (Es. "C:\Program Files (x86)\Suricata") e copiarvi l'eseguibile. Nella suddetta cartella:

- copiare le DLL `cyggcc_s-1.dll`, `cygmagic-1.dll`, `cygpcrc-1.dll`, `cygwin1.dll`, `cygz.dll` dalla cartella `bin` di Cygwin.
- copiare i file di configurazione `reference.config`, `suricata.yaml`, `classification.config` dalla cartella dei sorgenti di Suricata in `cygwin\tmp`.
- Copiare il file `magic.mgc` dalla cartella `cygwin/usr/share/misc`.

- Creare due sottocartelle *log* e *rules*.

Nota: accertarsi che nel file di configurazione *suricata.yaml*, il percorso dell'opzione *default-rule-path* termini con uno backslash.

2.3.3 Verifica dell'installazione

Per verificare che tutto funzioni correttamente, aprire il *prompt dei comandi* (Win+R → cmd → OK), spostarsi nella cartella di Suricata ed eseguire

```
> Suricata.exe -V
```

Se tutto è andato per il verso giusto l'output dovrebbe essere qualcosa del genere:

```
C:\Program Files\Suricata>suricata.exe -V
This is Suricata version 1.4dev (rev 40d067e)
```

```
C:\WINDOWS\system32\cmd.exe - suricata.exe -i 192.168.1.3
- stream "max-sessions": 262144
[1492] 1/12/2012 -- 19:19:32 - (stream-tcp.c:349) <Info> <StreamTcpInitConfig>
- stream "prealloc-sessions": 32768
[1492] 1/12/2012 -- 19:19:32 - (stream-tcp.c:365) <Info> <StreamTcpInitConfig>
- stream "memcap": 33554432
[1492] 1/12/2012 -- 19:19:32 - (stream-tcp.c:371) <Info> <StreamTcpInitConfig>
- stream "midstream" session pickups: disabled
[1492] 1/12/2012 -- 19:19:32 - (stream-tcp.c:377) <Info> <StreamTcpInitConfig>
- stream "async-oneside": disabled
[1492] 1/12/2012 -- 19:19:32 - (stream-tcp.c:394) <Info> <StreamTcpInitConfig>
- stream "checksum-validation": enabled
[1492] 1/12/2012 -- 19:19:32 - (stream-tcp.c:416) <Info> <StreamTcpInitConfig>
- stream "inline": disabled
[1492] 1/12/2012 -- 19:19:32 - (stream-tcp.c:434) <Info> <StreamTcpInitConfig>
- stream.reassembly "memcap": 67108864
[1492] 1/12/2012 -- 19:19:32 - (stream-tcp.c:452) <Info> <StreamTcpInitConfig>
- stream.reassembly "depth": 1048576
[1492] 1/12/2012 -- 19:19:32 - (stream-tcp.c:493) <Info> <StreamTcpInitConfig>
- stream.reassembly "toserver-chunk-size": 2560
[1492] 1/12/2012 -- 19:19:32 - (stream-tcp.c:495) <Info> <StreamTcpInitConfig>
- stream.reassembly "toclient-chunk-size": 2560
[1492] 1/12/2012 -- 19:19:32 - (tm-threads.c:2150) <Info> <InThreadWaitOnThreadI
nit> -- all 2 packet processing threads, 3 management threads initialized, engin
e started.
```

Figura 2.6: Suricata in funzione su Windows XP

2.3.4 Differenze dalla versione Linux

Come si è visto nei paragrafi 2.3.1 e 2.3.2 l'installazione di Suricata in ambiente Microsoft Windows è leggermente più complicata ma comunque possibile. È possibile eseguire Suricata in modalità IDS e, con maggiore sforzo, IPS ed è possibile usare lo stesso set di regole della versione Linux. Tuttavia, data

la natura prettamente Linux-based di Suricata, l'esperienza d'uso in ambiente Windows ne risente al punto da non poter sfruttare alcune particolarità quali il supporto a PF_RING (vedi paragrafo 5.1) o di usare alcune interfacce di gestione o, ancora, di disporre di determinate modalità di funzionamento quale l'esecuzione come demone di sistema (2.5).

2.4 Aggiornamento di Suricata

Poiché Suricata è un software ancora in sviluppo, può essere necessario aggiornarlo ad una versione più recente quando questa viene rilasciata. Questo perché potrebbero essere necessarie nuove features aggiunte, si potrebbe giocare di miglorie prestazionali e ancor più importante, si potrebbero veder risolti problemi di compatibilità o bug del software.

L'aggiornamento di Suricata è piuttosto semplice, poiché basta affidarsi al gestore pacchetti (Linux) o scaricare ed installare il nuovo pacchetto qualora queste opzioni fossero possibili (Windows e Linux). Altrimenti, sarà necessario ricompilare i nuovi sorgenti e sostituire il nuovo file eseguibile prodotto al vecchio. A volte potrebbero essere apportate delle modifiche ad altri file come il file di configurazione *suricata.yaml*, è bene quindi effettuare sempre un backup delle proprie impostazioni prima di procedere all'upgrade.

Coloro che vogliono essere sempre aggiornati alle ultime modifiche al codice di Suricata e che quindi fanno uso del repository git, sono obbligati a ricompilare il codice ad ogni update dei sorgenti. Per ricevere gli ultimi aggiornamenti da repository git, è sufficiente spostarsi nella directory dei sorgenti da riga di comando ed impartire:

```
$ git pull
```

Per essere sempre aggiornati sulle nuove versioni di Suricata è consigliabile registrarsi alla mailing-list “Announce”¹⁷ di OISF e visitare spesso il loro sito.

¹⁷ <https://lists.openinfosecfoundation.org/mailman/listinfo>

2.5 Opzioni da riga di comando

Per specificare in che modo Suricata debba funzionare nella macchina si fa ricorso alle opzioni d'avvio. Queste sono dei parametri sotto forma di stringhe da passare all'eseguibile di Suricata da linea di comando:

```
suricata [Parametro 1] [Parametro 2] ... [Parametro n]
```

In questo paragrafo si esaminano il funzionamento dei più importanti.

In precedenza, in realtà, è già stato presentato un parametro: `-V`. Avviando Suricata con questo parametro, ci viene risposta la versione della build di Suricata attualmente installata.

Per ottenere una lista dei parametri con cui Suricata può essere avviato basta eseguire il comando

```
# suricata
```

Analogamente in ambiente Windows basta spostarsi nella cartella di Suricata da prompt dei comandi ed impartire il comando equivalente “suricata.exe”.

Poiché con il suddetto comando Suricata non riceve alcun parametro per determinare come debba comportarsi nel sistema, risponderà in output con una lista di possibili parametri. È possibile consultare la lista nell'appendice Lista di parametri per l'avvio di Suricata.

I parametri più importanti sono quelli che servono a specificare il path ai file di configurazione `.yaml` (`-c path`), ai file delle regole `.rules` aggiuntivi a quelli indicati dal file di configurazione (`-s path`) e l'interfaccia di rete dalla quale catturare i pacchetti in transito (`-i interface`). Il parametro essenziale perché Suricata entri in funzione è quello che definisce l'interfaccia su cui effettuare l'analisi dei pacchetti in transito. In assenza dei parametri `-c` e `-s` verranno utilizzati i percorsi di default.

Al parametro `-i` possono seguire l'indirizzo IP assegnato all'interfaccia di rete o il suo nome (utile nel caso in cui l'interfaccia non disponga di un indirizzo IP). In ambiente GNU/Linux la lista delle interfacce e' consultabile tra-

mite comando *ifconfig*. In ambiente Microsoft Windows si fa ricorso all'UUID¹⁸ della NIC con il comando *wmic nicconfig get ipaddress,SettingID* . All'UUID della NIC è poi sufficiente anteporre `\\DEVICE\\NPF_` .

Riassumendo, eseguire Suricata in modo che analizzi il traffico su una determinata interfaccia è possibile nei modi seguenti:

```
suricata -i eth0
suricata -i 192.168.1.2
suricata -i \\DEVICE\\NPF_{04D7BF4B-B66F-4336-B8B4-
C25D21569AF0}
```

È anche possibile ascoltare più interfacce in contemporanea specificando ognuna¹⁹:

```
suricata -i eth0 -i ppp0
```

Un parametro degno di nota è il `-q <queue id>` per l'esecuzione di Suricata in modalità in-line. Il Queue ID è l'identificativo di una coda definita in *netfilter*, il modulo del kernel Linux che manipola e gestisce i pacchetti in transito sulla macchina. Quando si dispone Suricata in modalità in-line non è necessario fornire l'interfaccia d'ascolto in quanto i pacchetti verranno prelevati dalla coda. Saranno le regole definite per netfilter ad instradare i pacchetti su quella coda. Per maggiori dettagli si rimanda al paragrafo 4.3.1 Modalità in-line ed IPTables

Un altro parametro da poter passare a Suricata nella sua fase di avvio è quello per avviare il programma come demone, tenendolo così in esecuzione in background. Il parametro in questione è `-D`. Nel caso in cui si voglia avviare Suricata come demone di sistema può venire in aiuto anche il parametro `--pidfile <file>`. Per mezzo di questo parametro, Suricata scriverà in `<file>` il suo PID (o Process ID) lasciandolo disponibile all'utente per successivi interventi quale potrebbe essere la terminazione del processo demone. Esempio:

```
# suricata -i eth0 -D --pidfile /var/run/suricata.pid
```

18 Universal Unique Identifier.

19 <http://blog.inliniac.net/2010/12/24/listening-on-multiple-interfaces-with-suricata/>

Chiudere il processo diviene poi possibile impartendo:

```
# kill -9 `cat /var/run/suricata.pid`
```

Nota: Suricata deve avere i permessi di scrittura su `<file>` o nella sua directory di appartenenza se `<file>` non esiste.

2.6 File di configurazione `suricata.yaml`

In questo paragrafo verrà trattato il già menzionato file di configurazione `suricata.yaml`. In questo file sono contenute tutte le voci di configurazione di Suricata che ne determinano il funzionamento. Le voci meno importanti o di poco frequente utilizzo verranno tralasciate, mentre altre verranno trattate nello specifico più avanti nel testo; tuttavia, è consigliabile leggere all'intero file nel complesso poiché, essendo ben commentato, risulta di facile comprensione.

Il team OISF ha deciso di fare un passo in avanti nella gestione delle configurazioni e piuttosto che utilizzare una struttura di salvataggio delle opzioni di funzionamento mediante i più tradizionali file `.config` ha abbracciato l'uso dei file di tipo *YAML*.

YAML²⁰ (acronimo ricorsivo che sta per “YAML Ain't a Markup Language”) è un formato per la serializzazione di dati che riesce a trovare un ottimo compromesso fra la leggibilità agli occhi di un umano e la rappresentazione di strutture dati nei confronti dei calcolatori. La strutturazione del file è di tipo gerarchico ed è implementata per via dell'indentazione. Consente di rappresentare agevolmente strutture dati quali *liste*, *array associativi* oltre che dati tipizzati.

Segue una analisi del file di configurazione[6]:

²⁰ <http://en.wikipedia.org/wiki/YAML>

- *pid-file*, se decommentata, definisce un percorso di default per il file contenente il pid di Suricata, utilizzato in assenza del comando `--pidfile <file>`

```
# Default pid file.
# Will use this file if no --pidfile in command options.
#pid-file: /var/run/suricata.pid
```

- *default-log-dir*, specifica in quale directory suricata andrà a scrivere i propri file di log

```
default-log-dir: /var/log/suricata/
```

- *outputs*, è una lista di metodi di output di cui si può avvalere Suricata, ogni sottovoce ha delle sue proprietà, fra cui *enabled: yes/no* che permette di regolare l'attivazione o disattivazione dello specifico metodo di output.

```
# Configure the type of alert (and other) logging you would
like.
outputs:

# a line based alerts log similar to Snort's fast.log
- fast:
    enabled: yes
    filename: fast.log
    append: yes
    filetype: regular # 'regular', 'unix_stream' or
'unix_dgram'

# alert output for use with Barnyard2
- unified2-alert:
    enabled: yes
    filename: unified2.alert

# File size limit. Can be specified in kb, mb, gb.
Just a number
# is parsed as bytes.
#limit: 32mb

# a line based log of HTTP requests (no alerts)
```

```

- http-log:
    enabled: yes
    filename: http.log
    append: yes
    #extended: yes      # enable this for extended logging
information
    #custom: yes        # enabled the custom logging format
(defined by customformat)
    #customformat: "%{%D-%H:%M:%S}t.%z %{X-Forwarded-For}i
%H %m %h %u %s %B %a:%p -> %A:%P"
    #filetype: regular # 'regular', 'unix_stream' or
'unix_dgram'

# a line based log of TLS handshake parameters (no alerts)
- tls-log:
    enabled: no # Log TLS connections.
    filename: tls.log # File to store TLS logs.
    #extended: yes # Log extended information like
fingerprint
    certs-log-dir: certs # directory to store the
certificates files

# a line based log to used with pcap file study.
# this module is dedicated to offline pcap parsing (empty
output
# if used with another kind of input). It can interoperate
with
# pcap parser like wireshark via the suriwire plugin.
- pcap-info:
    enabled: no

# Packet log... log packets in pcap format. 2 modes of
operation: "normal"
# and "sguil".
#
# In normal mode a pcap file "filename" is created in the
default-log-dir,
# or are as specified by "dir". In Sguil mode "dir"
indicates the base directory.
# In this base dir the pcaps are created in th directory
structure Sguil expects:
#

```

```
# $sguil-base-dir/YYYY-MM-DD/$filename.<timestamp>
#
# By default all packets are logged except:
# - TCP streams beyond stream.reassembly.depth
# - encrypted streams after the key exchange
#
- pcap-log:
    enabled: no
    filename: log.pcap

    # File size limit. Can be specified in kb, mb, gb.
Just a number
    # is parsed as bytes.
    limit: 1000mb

    # If set to a value will enable ring buffer mode. Will
keep Maximum of "max-files" of size "limit"
    max-files: 2000

    mode: normal # normal or sguil.
    #sguil-base-dir: /nsm_data/
    #ts-format: usec # sec or usec second format (default)
is filename.sec usec is filename.sec.usec
    use-stream-depth: no #If set to "yes" packets seen
after reaching stream inspection depth are ignored. "no" logs
all packets

    # a full alerts log containing much information for
signature writers
    # or for investigating suspected false positives.
- alert-debug:
    enabled: no
    filename: alert-debug.log
    append: yes
    #filetype: regular # 'regular', 'unix_stream' or
'unix_dgram'

    # alert output to prelude (http://www.prelude-technologies.com/) only
    # available if Suricata has been compiled with --enable-
prelude
```

```
- alert-prelude:
    enabled: no
    profile: suricata
    log-packet-content: no
    log-packet-header: yes

# Stats.log contains data from various counters of the
suricata engine.
# The interval field (in seconds) tells after how long
output will be written
# on the log file.
- stats:
    enabled: yes
    filename: stats.log
    interval: 8

# a line based alerts log similar to fast.log into syslog
- syslog:
    enabled: no
    # reported identity to syslog. If omitted the program
name (usually
    # suricata) will be used.
    #identity: "suricata"
    facility: local5
    #level: Info ## possible levels: Emergency, Alert,
Critical,
        ## Error, Warning, Notice, Info, Debug

# a line based information for dropped packets in IPS mode
- drop:
    enabled: yes
    filename: drop.log
    append: yes
    #filetype: regular # 'regular', 'unix_stream' or
'unix_dgram'

# output module to store extracted files to disk
#
# The files are stored to the log-dir in a format
"file.<id>" where <id> is
# an incrementing number starting at 1. For each file
```

```
"file.<id>" a meta
# file "file.<id>.meta" is created.
#
# File extraction depends on a lot of things to be fully
done:
# - stream reassembly depth. For optimal results, set this
to 0 (unlimited)
# - http request / response body sizes. Again set to 0 for
optimal results.
# - rules that contain the "filestore" keyword.
- file-store:
    enabled: no          # set to yes to enable
    log-dir: files      # directory to store the files
    force-magic: no     # force logging magic on all stored
files
    force-md5: no       # force logging of md5 checksums
    #waldo: file.waldo # waldo file to store the file_id
across runs

# output module to log files tracked in a easily parsable
json format
- file-log:
    enabled: no
    filename: files-json.log
    append: yes
    #filetype: regular # 'regular', 'unix_stream' or
'unix_dgram'

    force-magic: no     # force logging magic on all logged
files
    force-md5: no       # force logging of md5 checksums
```

- *default-rule-path*, indica il percorso predefinito dove Suricata si aspetta di trovare i file *.rules*.

```
# Set the default rule path here to search for the files.
# if not set, it will look at the current working dir
default-rule-path: /etc/suricata/rules
```

- *rule-files*, precede la lista di tutti i file di regole che Suricata caricherà all'avvio. I file qui indicati devono essere presenti nella directory specificata da *default-rule-path*. Qualora si volessero caricare regole esterne

alla directory di default è possibile utilizzare il parametro `-s` come specificato nel paragrafo 2.5 Opzioni da riga di comando. Qualora si volesse disabilitare le regole scritte in un file elencato nella lista, basterebbe scrivere `#` come primo carattere della relativa riga.

```
[...]
- emerging-trojan.rules
- emerging-user_agents.rules
# - emerging-virus.rules
- emerging-voip.rules
- emerging-web_client.rules
- emerging-web_server.rules
[...]
```

- *classification-file* e *reference-config-file*, indicano i percorsi ai file di classificazione e riferimento per le regole. Sono una sorta di macro d'abbreviazione che vengono utilizzate nelle regole. *classification-file* contiene le macro per classificare il tipo di alert generato sia per quanto ne concerne la categoria sia per quanto ne concerne la priorità. *reference-config-file* contiene i riferimenti a siti esterni inerenti la specifica regola.

```
classification-file: /etc/suricata/classification.config
reference-config-file: /etc/suricata/reference.config
```

- *vars* è una lista che definisce delle macro da poter utilizzare nella definizione di regole utili perché facilitano sia la lettura che la scrittura delle stesse e permettono di rendere le regole il più generali possibile.

```
# Holds variables that would be used by the engine.
vars:

    # Holds the address group vars that would be passed in a
    Signature.
    # These would be retrieved during the Signature address
    parsing stage.
    address-groups:
```

```
HOME_NET: "[192.168.0.0/16,10.0.0.0/8,172.16.0.0/12]"
EXTERNAL_NET: "!$HOME_NET"
HTTP_SERVERS: "$HOME_NET"
SMTP_SERVERS: "$HOME_NET"
SQL_SERVERS: "$HOME_NET"
DNS_SERVERS: "$HOME_NET"
TELNET_SERVERS: "$HOME_NET"
AIM_SERVERS: "$EXTERNAL_NET"
DNP3_SERVER: "$HOME_NET"
  DNP3_CLIENT: "$HOME_NET"
MODBUS_CLIENT: "$HOME_NET"
MODBUS_SERVER: "$HOME_NET"
ENIP_CLIENT: "$HOME_NET"
ENIP_SERVER: "$HOME_NET"

# Holds the port group vars that would be passed in a
Signature.

# These would be retrieved during the Signature port
parsing stage.

port-groups:
  HTTP_PORTS: "80"
  SHELLCODE_PORTS: "!80"
  ORACLE_PORTS: 1521
  SSH_PORTS: 22
  DNP3_PORTS: 20000
```

- *action-order*, indica l'ordine con cui vengono effettuate le azioni. Quando un pacchetto sollecita più di una regola, e queste impongono al motore IDPS decisioni diverse come per esempio di bloccare e di far passare il pacchetto, entrano in conflitto. Per risolvere tale situazione si applica la regola che applica l'azione che viene prima secondo l'ordine qui definito.

```
# Set the order of alerts based on actions
# The default order is pass, drop, reject, alert
action-order:
  - pass
  - drop
  - reject
  - alert
```

2.7 Le regole

Le *rules* (*regole*) o *signatures* sono delle istruzioni fornite all'IDPS che definiscono come il sistema debba comportarsi al presentarsi di determinati pacchetti o situazioni. Più regole inerenti un determinato argomento di sicurezza vengono raccolte all'interno di file con estensione *.rules*. L'insieme di file *.rules* che vengono fatti caricare da un IDPS prendono il nome di set di regole o *ruleset*.

All'avvio dell'IDPS, le regole vengono lette e controllate che siano scritte in modo sintatticamente consistente e successivamente vengono caricate all'interno dell'engine IDPS. Al passaggio di ogni pacchetto, il motore verifica se questo soddisfa o meno i requisiti di qualche regola; se il pacchetto ha qualche match con una o più regole, le azioni definite in esse vengono eseguite dall'IDPS.

Suricata sfrutta la stessa sintassi e struttura dei file di regole di Snort²¹. Le regole sono composte da tre parti[7].

```
drop tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"ET
TROJAN Likely Bot
Nick in IRC (USA +..)"; flow:established,to_server;
flowbits:isset,is_proto_irc; content:"NICK "; pcre:"/NICK
.*USA.*[0-9]{3,}/i"; classtype:trojan-activity;
reference:url,doc.emergingthreats.net/2008124;
reference:url,www.emergingthreats.net/cgi-
bin/cvswweb.cgi/sigs/VIRUS/TROJAN_IRC_Bots;
sid:2008124; rev:2;)
```



Figura 2.7: Suddivisione di una regola

²¹ In realtà, essendo Snort e Suricata due progetti distinti e in sviluppo, alcune determinate funzioni all'interno delle regole possono essere supportate esclusivamente da l'uno o dall'altro, rendendo tali regole incompatibili fra i due IDPS.

- **Azione:** è l'azione che Suricata deve intraprendere al verificarsi delle condizioni definite a seguire nella regola. Le azioni che Suricata può intraprendere sono
 - **pass:** se la regola corrisponde con il pacchetto in transito, Suricata smette di esaminare il pacchetto e lo lascia passare.
 - **alert:** quando c'è un match fra la regola che fa uso dell>alert ed un pacchetto, questo verrà lasciato passare ma genererà un alert sotto forma di log, disponibile alla lettura di un amministratore di sistema. Sia il drop che il reject generano degli alert. È l'azione più comune intrapresa nell'intrusion detection.
 - **drop:** disponibile solo quando si dispone Suricata in modalità in-line, effettua il drop del pacchetto evitando che proceda oltre. È alla radice della funzionalità di prevention di Suricata poiché blocca i pacchetti che corrispondono con la regola. Chi ha inviato il pacchetto che subisce il drop non verrà avvisato di nulla, conseguentemente riceverà una notifica (se in uso il protocollo TCP) di time-out. Suricata genererà poi un alert per il pacchetto.
 - **reject:** quando c'è un match fra la regola che fa uso del reject ed un pacchetto, verrà inviato sia al destinatario che al mittente un pacchetto di reject. Questo può essere un *reset-packet* nel caso di protocollo TCP o un *ICMP-error* nel caso di altri protocolli. Se Suricata è disposto in modalità in-line, procederà al drop del pacchetto. Ad ogni modo verrà infine generato un alert al riguardo.
- **Intestazione:** definisce il dominio di azione della regola, rendendone possibile l'innescò (*trigger*) solo quando un pacchetto cade all'interno di tale dominio. A sua volta, l'intestazione, è composta da:

- **Protocollo:** indica il protocollo a cui il pacchetto deve appartenere perché la regola venga innescata. Parole chiave per il protocollo sono *tcp*, *icmp*, *ip*, *http*.
- **Sorgente e Destinazione:** Sono due coppie composte da Indirizzo IP di un host o di una rete e da una porta. La prima coppia (IP, porta) è separata dalla seconda coppia (IP, porta) da un *operatore direzionale* che stabilisce quale dei due si comporterà da sorgente e quale da destinazione. È ammesso l'uso della notazione CIDR e di parole chiave quale *any* o macro definite nel file `suricata.yaml` quali `$HOME_NET` ed `$EXTERNAL_NET`.
- **Direzione:** questo operatore si interpone fra le due coppie (IP, porta). La direzione “->” indica che la coppia (IP, porta) a sinistra funge da sorgente, mentre quella a destra dell'operatore funge da destinazione. Viceversa per l'operatore “<-”. Esiste inoltre l'operatore “<>” per l'evenienza in cui la regola debba essere processata in entrambi i casi.
- **Opzioni:** sono parole chiave che permettono di specificare ulteriormente il comportamento di una regola. Possono essere semplici come *msg* che definisce il messaggio da scrivere nei log e *content* che definisce lo scattare della regola alla presenza di una determinata stringa contenuta nel pacchetto; ci possono anche essere opzioni più complicate come quelle che fanno uso dei *flow*²².

Per una lista completa di parole chiave supportate nelle regole da Suricata è possibile eseguire il comando

```
# suricata --list-keywords
```

²² Il meccanismo dei flow è spiegato nelle pagine

<https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Flow-keywords> e

<https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Flowint> . È un

meccanismo complesso ma che permette la creazione di regole sofisticate.

È possibile far riferimento alla Wiki di OISF[7] per comprendere tutti i dettagli sul funzionamento e la creazione di regole.

Una regola d'esempio è la seguente:

```
alert http $HOME_NET any -> any $HTTP_PORTS (msg:"Suricata!";
content:"suricata"; nocase; classtype:not-suspicious; sid:1;
rev:1;)
```

La regola definisce che venga generato un **alert** ogniqualvolta un pacchetto del protocollo **http** proveniente da un qualunque host della rete interna **\$HOME_NET** originato da una porta qualsiasi (**any**), sia destinato ad un qualunque host (**any**) sulla porta 80 (**\$HTTP_PORTS**) e che contenga al suo interno la parola “suricata” indipendentemente dalla sua composizione in lettere maiuscole o minuscole (**content: “suricata”; nocase;**). Tale regola ha come ID il valore 1 come definito da **sid: 1;** e si tratta della sua prima revisione **rev: 1**. La regola è definita come attività non sospetta **classtype:not-suspicious**, non c'è nulla di male a consultare una pagina che parla di questo simpatico mammifero dopotutto, e genererà un messaggio nei log con scritto “Suricata!” (**msg:”Suricata”**).

È possibile mettere alla prova questa regola avviando Suricata. Visitando successivamente il sito http://it.wikipedia.org/wiki/Suricata_suricata e consultando il file di log *fast.log* compariranno messaggi del tipo:

```
12/03/2012-04:05:45.999712  [**] [1:1:1] Suricata! [**]
[Classification: Not Suspicious Traffic] [Priority: 3] {TCP}
192.168.1.2:58236 -> 91.198.174.224:80
```

I file *.rules* come detto, sono collezioni di regole generalmente organizzati secondo classi di appartenenza nel settore della sicurezza informatica. All'interno di tale file possono esserci una o più regole; qualora si volessero disabilitare delle regole, basterebbe anteporre alle regole in questione il simbolo di **#** (*sharp*). Per abilitarle, basterebbe invece rimuovere tale simbolo ad inizio riga. Le righe dei file *.rules* che iniziano per **#** non vengono processate

dall'engine di Suricata ed hanno funzione di commento. Infatti, oltre che per abilitare/disabilitare una regola, il simbolo `#` può essere utilizzato ad inizio riga/e per scrivere dei commenti, ad esempio per spiegare di cosa si occupano le regole in un determinato file o come funziona una determinata regola.

2.7.1 Live Rules Swap

Suricata IDPS, carica le regole una volta soltanto, all'avvio. Nel caso in cui si cambiassero le regole, se ne inserissero di nuove o si abilitassero/disabilitassero regole esistenti, mentre Suricata è in esecuzione, queste non verrebbero (ri)caricate fino al successivo avvio del software. Per ovviare a questo scomodo problema, il team OISF ha introdotto una funzionalità a partire dalla versione 1.3dev (rev 7109a05) che prende il nome di *Live Rules Swap*²³. Tramite questa funzionalità è possibile forzare Suricata mentre è in esecuzione a ricaricare il suo set di regole. Questo processo è possibile mediante l'uso delle *signal* in ambiente Unix-like. Per procedere con il Live Rules Swap, bisogna che tale funzionalità sia abilitata nel file di configurazione `suricata.yaml`, come segue:

```
# When rule-reload is enabled, sending a USR2 signal to the
Suricata process
# will trigger a live rule reload. Experimental feature,
use with care.
- rule-reload: true
```

A tal punto è necessario trovare il PID relativo al processo di Suricata caricato in memoria. Nel caso in cui Suricata fosse stato lanciato con il parametro `--pidfile` basterebbe consultare il file in questione, altrimenti si può ricorrere alla ricerca fra i processi di sistema per esempio con il comando

```
# ps aux | grep suricata
```

Una volta che il PID di `suricata` è noto, si può procedere con l'invio del si-

²³[https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Live Rule Swap](https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Live_Rule_Swap)

gnal al processo:

```
# kill -USR2 <PID>
```

Ottenendo come output:

```
[28330] 3/12/2012 -- 04:30:40 - (detect-engine.c:353) <Info>  
(DetectEngineLiveRuleSwap) -- ===== Starting live rule swap  
triggered by user signal USR2 =====
```

```
[...]
```

```
[28330] 3/12/2012 -- 04:32:41 - (detect-engine.c:549) <Info>  
(DetectEngineLiveRuleSwap) -- ===== Live rule swap DONE =====
```


2.7.2 Emerging Threats



Il pacchetto di Suricata che si installa nella propria macchina può essere o meno fornito di un set di regole. Il set di regole scelto da OISF per Suricata è quello della compagnia Emergin Threats²⁴. Emerging Threats è una compagnia nel settore della sicurezza delle reti che si occupa di detection e prevention systems. Fra le varie cose, la compagnia si occupa di ricercare e sviluppare nuove regole, mantenendole al contempo aggiornate giorno per giorno per affrontare le ultime minacce note. La compagnia mette a disposizione dei suoi utenti due possibili repository di regole, uno open ed uno a pagamento. Il repository open copre una ristretta serie di minacce e viene aggiornata più lentamente rispetto alla controparte a pagamento.

	ETPro RULESET	ETOpen RULESET	Other VENDORS
Extensive malware coverage	✓		
Basic malware coverage	✓		✓
Platform independent	✓		✓
Snort 2.4 to current compatible	✓		✓
Suricata compatible	✓		✓
Daily updates	✓		✓
Major vulnerability coverage	✓	✓	
Full-time research team	✓	✓	
Support	✓	✓	

Purchase

Figura 2.8: Confronto fra ruleset a pagamento e gratuite

²⁴ <http://www.emergingthreats.net/>

2.7.3 Vulnerability Research Team



Così come per Emerging Threats anche VRT²⁵ (Vulnerability Research Team) gestisce dei ruleset. VRT è un gruppo di esperti in sicurezza delle reti della SourceFire che sviluppa il software IDPS Snort. Come per Emerging Threats essa offre due piani a disposizione dei propri utenti, uno open ed uno a pagamento. Anche in questo caso, per avere le regole in grado di competere con le ultime minacce si deve necessariamente ricorrere al piano a pagamento. In alternativa sono disponibili gratuitamente per gli utenti registrati, i ruleset aggiornati a 30 giorni prima delle ultime novità del ramo a pagamento. Tuttavia è bene notare che questi ruleset nascono principalmente per l'IDPS Snort, <http://www.snort.org/snort-rules/>, e non tutte le regole vengono correttamente lette da Suricata. Per gli utenti Suricata sono dunque consigliabili le regole proposte da Emerging Threats.

2.7.4 Rule Manager

OinkMaster²⁶ e PulledPork²⁷ sono due Rule Manager per la gestione dei ruleset. Poiché risulta arduo eseguire manualmente l'aggiornamento delle regole di un IDPS, questi due software automatizzano tale processo. Entrambi sono nati per Snort ma funzionano correttamente anche con Suricata. Una volta configurati correttamente, si occuperanno di tenere aggiornate le regole, gestendo eventuali conflitti.

²⁵ <http://www.sourcefire.com/security-technologies/snort/vulnerability-research-team>

²⁶ <http://oinkmaster.sourceforge.net/>

²⁷ <http://code.google.com/p/pulledpork/>

I seguenti passi di configurazione valgono per OinkMaster ma sono facilmente riadattabili per PulledPork[8].

Una volta installato sul proprio sistema il software OinkMaster da pacchetto o da repository, si può passare a specificare nel suo file di configurazione *oinkmaster.conf* l'url da cui effettuare il download del ruleset aggiungendo la riga

```
url =
http://rules.emergingthreats.net/open/suricata/emerging.rules.t
ar.gz
```

o decommentandola se già presente.

Fatto ciò, sarà sufficiente impartire il comando

```
#sudo oinkmaster -C <path al file oinkmaster.conf> -o <path
alla directory delle regole di suricata>
```

ed OinkMaster si occuperà di scaricare ed estrarre le ultime regole disponibili. È consigliato eseguire questo comando ogni giorno per essere certi di avere sempre le ultime definizioni.

2.8 Distribuzioni IDS/IPS pronte all'uso

In questo capitolo è stato spiegato come installare e configurare Suricata. Sulla rete è possibile tuttavia trovare delle soluzioni già pronte all'uso, configurate e con software di gestione e reportistica pre-installate. Queste soluzioni sono genericamente rilasciate come distribuzioni di derivazione GNU/Linux.

2.8.1 Smooth-Sec

Smooth-Sec²⁸ è una distribuzione IDS/IPS pronta all'uso rilasciata da Philip Bailey e basata, alla corrente versione 2.1, su Debian 6.0 Squeeze 64-bit. Integra al suo interno Suricata alla versione 1.3, console web Snorby per la reportistica, e Barnyard2 come interprete dei log.

28 <http://bailey.st/blog/smooth-sec/>



Figura 2.9: Schermata iniziale della live di Smooth-Sec 2.1

Smooth-Sec viene distribuito come Live-CD; questo significa che è possibile scaricare l'immagine del disco, eseguirne il boot da file iso o da disco, ed iniziare ad usarlo senza necessità di installarlo sulla macchina. Qualunque modifica, configurazione o altro apportata al sistema andrà persa al riavvio della macchina. In alternativa, per memorizzare i dati in maniera persistente, è possibile installare Smooth-Sec su disco.

Quale che sia la scelta, una volta eseguito il boot del disco per la prima volta, verrà richiesto un minimo di configurazione, come la password di root ed il timezone.

Smooth-Sec non dispone di alcun desktop environment ed è utilizzabile tramite riga di comando. Di default Smooth-Sec ha il servizio SSH abilitato permettendo la connessione alla macchina per la gestione da remoto.

Per accedere all'interfaccia web Snorby basta collegarsi da browser, via *https* all'indirizzo della macchina Smooth-Sec.

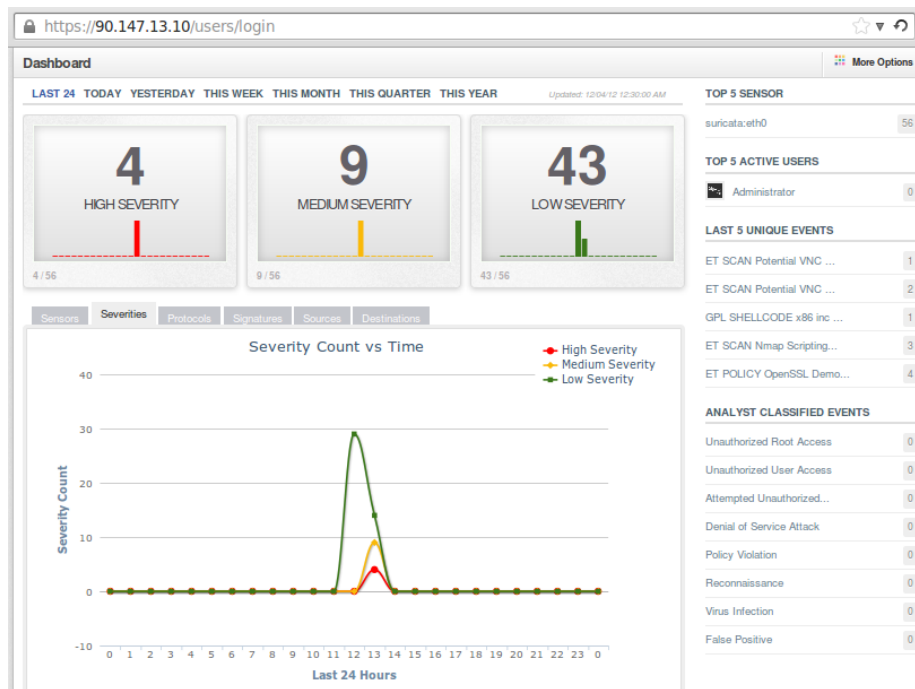


Figura 2.10: Snorby da Smooth-Sec

2.8.2 Security Onion

Security Onion è una distribuzione Linux per IDS/IPS pronta all'uso dopo una minima configurazione. La distribuzione è basata su XUbuntu 10.04 e 12.04 e contiene al suo interno Suricata e Snort quali IDPS, mentre come interfacce di gestione usa Snorby, Squert e Sguil. Dispone inoltre di qualche utile tool quale WireShark, ZenMap (nmap), EtherApe e molte altre utility di networking²⁹. Security Onion, dispone del desktop environment XFCE, e può essere gestita perlopiù da esso.

²⁹ <http://code.google.com/p/security-onion/wiki/Tools>



Figura 2.11: Schermata del desktop di Security Onion

Questa distribuzione può essere eseguita in modalità Live CD o può essere installata sulla macchina a seconda delle necessità.

La versione di Suricata disponibile con l'ultima ISO rilasciata è la poco recente stable 1.2.1. Tuttavia è possibile procedere con l'upgrade ad una versione più recente come spiegato nel paragrafo 2.4 Aggiornamento di Suricata oppure sfruttando il sistema di aggiornamento della distribuzione. Doug Burks, l'autore di Security Onion, non rilascia nuove ISO ad ogni aggiornamento bensì dei pacchetti non cumulativi, per i quali ha pubblicato uno script che automatizza il processo di installazione ed aggiornamento. Per proseguire con l'aggiornamento di Security Onion, e quindi dei software di cui fa uso, è sufficiente impartire il seguente comando:

```
$ sudo -i "curl -L http://sourceforge.net/projects/security-onion/files/security-onion-upgrade.sh > ~/security-onion-upgrade.sh && bash ~/security-onion-upgrade.sh"
```

A partire dal rilascio di Ubuntu 12.04 (e tutte le distribuzioni da esso ricavate), i pacchetti di installazione di Security Onion sono disponibili in un *Ubuntu Launchpad PPA*. È dunque possibile rendere IDPS una macchina con Ubuntu in pochi passi.³⁰

³⁰ <http://code.google.com/p/security-onion/wiki/Beta>

Di default, questa distribuzione usa Snort come software IDPS. Per sfruttare Suricata sarà sufficiente eseguire lo *script di setup* disponibile sin dal primo avvio sul desktop. Seguendo le istruzioni a video e scegliendo la modalità di configurazione avanzata, sarà possibile scegliere Suricata quale sistema di IDPS e sarà possibile impostare le password per l'uso degli strumenti di reportistica.

Security Onion è disponibile al download dalla piattaforma sourceforce al seguente indirizzo <http://sourceforge.net/projects/security-onion/files/> mentre il blog dell'autore è disponibile al seguente indirizzo <http://securityonion.blogspot.it/> . Qualora si decidesse di scegliere questa distribuzione può far altresì comodo la wiki, raggiungibile al seguente indirizzo: <http://code.google.com/p/security-onion/w/list> .

3 Report, segnalazioni e tools

Suricata viene distribuito come solo software per la prevenzione ed individuazione di intrusioni, eseguibile da riga di comando ed in modalità di processo demone per essere disposto su uno o più sensori all'interno di una rete. È pertanto normale che non vi sia un'interfaccia grafica di gestione interna a Suricata, o che non vi sia integrata un'interfaccia per la lettura degli alert. Più sensori Suricata IDPS possono essere dislocati all'interno della rete, generando alert distribuiti. La collezione delle informazioni che ogni sensore produce può essere delegato a management server centralizzati, o possono essere salvate all'interno dei sensori stessi, in modo individuale, come in un tipico scenario di network-based IDPS (Figura 1.2: Esempio di network-based IDPS).

A sopperire la mancanza di funzionalità di gestione e reportistica integrate in Suricata, possono essere usati alcuni tool esterni. In tal senso Suricata è stato ben progettato permettendo l'integrazione con i suddetti; la scelta di at-

tenersi alla strada intrapresa da Snort, consente (talvolta previo riadattamento, talvolta parzialmente) l'uso dei tool sviluppati per quest'ultimo.

Di seguito verranno analizzati alcuni strumenti esterni a Suricata che ne consentono una più facile o migliore gestione e consultazione degli output prodotti.

3.1 File di log

Di default, Suricata, salva i propri output (ed alert) su file di log. La posizione su disco dei log è dettata dal valore di *default-log-dir* nel file *suricata.yaml*. Su Windows genericamente viene usata la cartella “log” all'interno della cartella di installazione di Suricata. Su Linux viene spesso utilizzata la directory */var/log/suricata*.

Sotto la voce *outputs* del file di configurazione sono elencati i vari tipi di log che Suricata può salvare.

- **fast**: sono i log mono-riga, prodotto degli alert generati dalle regole di Suricata. È possibile scegliere se abilitare o meno questo tipo di log e se scrivere o meno in append al file. Questo genere di log si presentano con una specifica formattazione e contengono al loro interno quello che eventualmente nelle regole è specificato dalla parola chiave *msg*.

```
12/03/2012-04:05:45.999712  [**] [1:1:1] Suricata! [**]  
[Classification: Not Suspicious Traffic] [Priority: 3] {TCP}  
192.168.1.2:58236 -> 91.198.174.224:80
```

- **http-log**: si tratta del file di log che registra le informazioni riguardo al traffico HTTP di passaggio attraverso il sensore. È possibile specificare il formato del log, se scrivere in append, e se abilitarlo o meno.

Un esempio di output è:

```
11/07/2012-01:55:30.337469 www.google.com [**]  
/logos/Logo_25wht.gif [**] Mozilla/5.0 (X11; Ubuntu;  
Linux x86_64; rv:16.0) Gecko/20100101 Firefox/16.0  
[**] 90.147.13.136:46933 -> 173.194.35.177:80
```


- **pcap-log**: se abilitato salva al suo interno una copia di tutti i pacchetti passanti per il sensore. Questa operazione può risultare costosa in termini di *cicli di CPU per pacchetto* e di spazio su disco. Tuttavia è possibile specificare una dimensione massima che il file può raggiungere prima di accodare nuovi pacchetti nel file di log ed eliminare i vecchi pacchetti. Il formato di salvataggio dei pacchetti è di tipo *pcap* e può essere utilizzato per l'analisi in un secondo momento³¹. Di norma in questo file non vengono salvati i pacchetti TCP che formano file di dimensioni superiori alla costante *stream.reassembly.depth* o flussi di pacchetti dopo lo scambio delle chiavi qualora fossero criptati.
- **alert-debug**: particolarmente utile per chi testa e crea le regole, fornisce i dettagli del pacchetto che ha generato un determinato alert e lo stato dei flow di Suricata.

```

TIME:                12/06/2012-00:53:52.490970
SRC IP:              91.198.174.234
DST IP:              90.147.12.168
PROTO:               6
SRC PORT:            80
DST PORT:            32876
TCP SEQ:             1926837078
TCP ACK:             665649665
FLOW:                to_server: FALSE, to_client: TRUE
FLOW Start TS:      12/06/2012-00:53:52.299716
FLOW IPONLY SET:    TOSERVER: TRUE, TOCLIENT: TRUE
FLOW ACTION:        DROP: FALSE, PASS FALSE
FLOW NOINSPECTION: PACKET: FALSE, PAYLOAD: FALSE,
APP_LAYER: FALSE
FLOW APP_LAYER:     DETECTED: TRUE, PROTO 1
PACKET LEN:         1434

```

³¹ I file *pcap* contengono pacchetti catturati che possono essere processati ed analizzati in qualunque momento ed in modalità offline. Questo tipo di file può essere letto da programmi fra i quali WireShark, TCPdump, Suricata o Snort. Per eseguire l'analisi di un file *pcap* in Suricata senza che questo monitori alcuna interfaccia è sufficiente eseguire Suricata con il parametro `-r <file pcap>`.

PACKET: [...]

- **stats**: su questo file di log vengono scritti i valori di vari contatori interni all'engine di Suricata. È usato soprattutto per trarre delle statistiche o per conoscere i dettagli del flusso di rete che attraversa il sensore.

```
-----
Date: 11/19/2012 -- 01:01:43 (uptime: 1d, 00h 48m 44s)
-----
```

Counter	TM Name	Value
decoder.pkts	RxPcapwlan0	223725
decoder.bytes	RxPcapwlan0	142281676
decoder.ipv4	RxPcapwlan0	223038
decoder.ipv6	RxPcapwlan0	154
decoder.ethernet	RxPcapwlan0	223725
decoder.raw	RxPcapwlan0	0
decoder.sll	RxPcapwlan0	0
decoder.tcp	RxPcapwlan0	215768
decoder.udp	RxPcapwlan0	7392
decoder.sctp	RxPcapwlan0	0
decoder.icmpv4	RxPcapwlan0	0
decoder.icmpv6	RxPcapwlan0	17
decoder.ppp	RxPcapwlan0	0
decoder.pppoe	RxPcapwlan0	0
decoder.gre	RxPcapwlan0	0
decoder.vlan	RxPcapwlan0	0
decoder.teredo	RxPcapwlan0	11
decoder.ipv4_in_ipv6	RxPcapwlan0	0
decoder.ipv6_in_ipv6	RxPcapwlan0	0
decoder.avg_pkt_size	RxPcapwlan0	636
decoder.max_pkt_size	RxPcapwlan0	1506
defrag.ipv4.fragments	RxPcapwlan0	0
defrag.ipv4.reassembled	RxPcapwlan0	0
defrag.ipv4.timeouts	RxPcapwlan0	0
defrag.ipv6.fragments	RxPcapwlan0	0
defrag.ipv6.reassembled	RxPcapwlan0	0
defrag.ipv6.timeouts	RxPcapwlan0	0
defrag.max_frag_hits	RxPcapwlan0	0
tcp.sessions	Detect	3090
tcp.ssn_memcap_drop	Detect	0

Figura 3.1: Parte di un log di tipo stats.

- **syslog**: funziona come i log di tipo *fast* ma invia l>alert al file di log del sistema (*syslog*). È possibile definire l'identità nei confronti di *syslog*, la facility ed il livello di pericolo che avranno le entry di Suricata in esso.

- **drop**: vengono salvati in questo tipo di file di log (se abilitato), i pacchetti che subiscono il drop qualora Suricata venga disposto in modalità IPS. Il formato dei pacchetti salvati è compatibile con *Netfilter*, per poter essere eventualmente elaborati successivamente.

3.2 Unified2 e Barnyard2

Un'altra modalità di output per gli alert di Suricata consiste nell'uso di file *Unified2*. Questo formato nasce dalle ceneri di *Unified*, sviluppato dal team Sourcefire di Snort per salvare i propri output. Utilizzare Unified2 comporta alcuni vantaggi. Si tratta infatti di un formato unificato, uguale per diversi tools, e si tratta del mezzo di output più rapido di cui dispone Suricata, riducendo al minimo l'uso di CPU e disco grazie un'accurata ottimizzazione della formattazione dei dati e conseguente diminuzione di overhead; è l'unico mezzo di comunicazione a disposizione di Suricata nei riguardi di diversi programmi esterni.

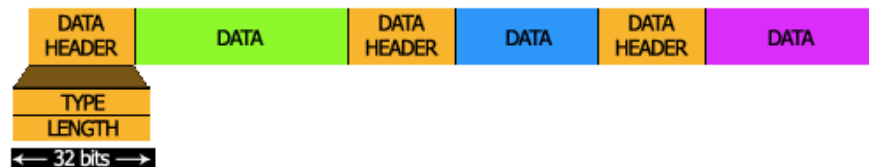


Figura 3.2: Struttura del formato Unified2

Ogni blocco dati all'interno del *file binario* Unified2 è preceduto da un *header* di 64bit, di cui 4 byte rappresentano il tipo di dato che seguirà nel blocco *data* ed altri 4 byte rappresentano la lunghezza di tale blocco.[9]

Barnyard2 è un software open source appositamente sviluppato per la lettura dei file Unified2. È il successore il Barnyard che operava con i file di tipo Unified. Usualmente questo software viene lasciato lavorare in background sugli output Unified2 prodotti in una directory da Suricata. Barnyard2 si occuperà di “consumare” questi output in real-time e ridirigerli in un database o,

eventualmente, in file o syslog. Un vantaggio nell'uso di Barnyard2 è la sua capacità di mantenere consistenza fra i dati prodotti da Suricata e quelli salvati in un database. Qualora venisse a mancare la connessione con il database, per esempio, esso accoderebbe gli alert da inviare fino a connessione ripristinata.

Per configurare Barnyard2 versione 1.9 è sufficiente scaricarne i sorgenti e compilarli, come segue^{32 33}:

```
# cd /tmp && wget
http://www.securixlive.com/download/barnyard2/barnyard2-
1.9.tar.gz && tar xvfz barnyard2-1.9.tar.gz && cd barnyard2-1.9
# ./configure --with-mysql && make && make install
```

Copiare poi il file di configurazione dalla cartella temporanea utilizzata per la compilazione in qualche altra directory come quella di Suricata:

```
# cp /tmp/barnyard2-1.9/etc/barnyard2.conf /etc/suricata/
```

Editare il file appena copiato impostando i percorsi ai file di Suricata:

```
config reference_file:      <path>/reference.config
config classification_file: <path>/classification.config
config gen_file:           <path>/rules/gen-msg.map
config sid_file:           <path>/rules/sid-msg.map
```

Scrivere poi l'indirizzo e le credenziali del server MySQL che ospita il database su cui si desidera salvare gli alert come segue

```
output database: log, mysql, user=<MYSQL_USER>
password=<PASSWORD> dbname=<DBNAME> host=<IP>
sensor_name=<NOME_SENSORE>
```

Bisogna ora abilitare Suricata all'uso dei file di log Unified2. Modificare, sotto la sezione *outputs* del file *suricata.yaml* la categoria *unified2-alert* ponendo

```
enabled: yes
```

Non resta che avviare Suricata in modo tradizionale, il quale ora genererà output di tipo Unified2 (in aggiunta alle modalità descritte in 3.1) e Barnyard2 come segue:

³² Verrà spiegato come installare Barnyard2 per funzionare con MySQL, tuttavia è possibile utilizzare anche PostgreSQL.

³³ È possibile installare e configurare Barnyard2 anche su sistemi Windows.

```
# barnyard2 -c <path al file di conf>/barnyard2.conf -d <path
log Suricata>/suricata -f unified2.alert -w <path log
Suricata>/suricata.waldo --pid-path=<path> -D
```

Barnyard2, leggerà le informazioni necessarie e la stringa di connessione al database dal proprio file di configurazione (specificato tramite parametro *-c*); cercherà i file di tipo Unified2 nella directory specificata da *-d* aspettandosi di leggere file dal nome *unified2.alert.<timestamp>*³⁴ ed userà un bookmark³⁵ dal nome *suricata.waldo*. Infine il processo viene lanciato come demone tramite *-D* ed il suo PID viene salvato file specificato da *--pid-path*.

Barnyard2 adesso *consumerà* gli output prodotti da Suricata inviando i singoli alert al database specificato.

3.3 Prelude e Prewikka

*Prelude-IDS*³⁶ è un sistema di *Security Information Management (SIM)* Il suo scopo è raccogliere ed organizzare i dati provenienti da uno o più sensori disposti in una rete. È in grado di interfacciarsi con file di log provenienti da syslog o altri file, come anche da altri software in grado di interagire con esso quali Suricata o Snort. I messaggi che vengono usati da Prelude vengono formattati secondo uno standard chiamato *IDMEF* o *Intrusion Detection Message Exchange Format*. Si tratta di uno standard creato dal team di Prelude in collaborazione con l'IETF[10]. Prelude ha raggiunto la versione 1.0.1 nel corso di luglio 2012 ed i sorgenti disponibili per la compilazione dei suoi vari componenti sono disponibili dal sito web³⁷. Prelude è composto da diverse compo-

34 Il prefisso “unified2.alert” deve coincidere con quello specificato nella sezione unified2-alert del file suricata.yaml. Suricata si occuperà di posporre a tale prefisso il timestamp dell'attimo in cui viene generato ogni file unified2.

35 I bookmark, o file waldo nel mondo Snort, sono dei file che permettono a Barnyard2 di avere una sorta di journaling. Qualora Barnyard2 venisse chiuso prima che tutti gli alert in esame siano processati, il programma ricomincerà, se specificato lo stesso file waldo, da dove si era fermato al successivo avvio.

36 <https://www.prelude-ids.org/>

37 <https://www.prelude-ids.org/projects/prelude/files>

nenti che interagiscono fra loro e possono essere installate su una stessa macchina o dislocate su più macchine.

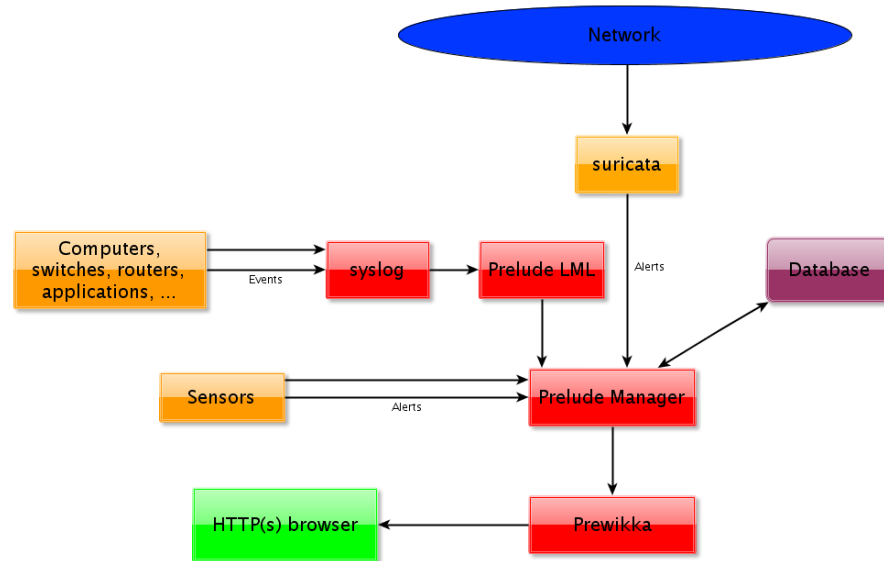


Figura 3.3: Schema delle componenti di Prelude-IDS

- **Prelude-manager:** è un server che accetta delle connessioni sicure (previa generazione di chiavi) da parte di sensori e/o altri manager nella rete. Il suo scopo è salvare gli alert ricevuti dove specificato dall'amministratore di rete; di solito i dati vengono salvati su un database, ma nulla vieta l'utilizzo di file di log o email.
- **Prelude-LML:** è la componente di Prelude che si occupa di scansionare i log di sistema del singolo host in cui viene installato e di inviarli ad un prelude-manager. Svolge l'attività di host-based IDS (cfr. Panorama sulle tecnologie IDS/IPS).
- **Sensori o Agenti:** sono applicazioni che fanno uso delle librerie di prelude (*libprelude*) e che sono dunque in grado di comunicare tramite il protocollo IDMEF con un prelude-manager, inviando ad esso i propri alert. Suricata rientra in questa categoria.
- **Prewikka:** interfaccia web per la consultazione degli eventi salvati dal prelude-manager.

Affinché Suricata faccia uso di Prelude, è necessario che siano disponibili le librerie libprelude e che sia stato compilato passando il parametro `--enalbe-pre-`

lude:

```
$ ./configure --enable-prelude
```

Una volta compilato ed installato Suricata, bisognerà abilitarlo a comportarsi da sensore per conto di Prelude perché generi ed invii gli opportuni output. Questo passaggio è reso possibile modificando il file di configurazione *suricata.yaml* nella sezione *alert-prelude*.

```
- alert-prelude:
  enabled: yes
  profile: suricata
  log-packet-content: no
  log-packet-header: yes
```

In questa sezione andrà modificato lo stato di *enabled* a *yes* ed impostato un valore per il profilo (di default è *suricata*). Il nome del profilo è importante in quanto richiesto dal prelude-manager per identificare il sensore quando questo richiede di stabilire una connessione. Le opzioni booleane *log-packet-content* e *log-packet-header* stabiliscono rispettivamente se si vuole che vengano inviati anche il payload o lo header dei pacchetti che generano gli alert. L'attivazione del packet-content può comportare carichi maggiori di dati da inviare nella rete e da salvare da parte del prelude-manager.

A questo punto Suricata è pronto a stabilire una connessione con Prelude al suo prossimo avvio.

Non è altrettanto pronto il prelude-manager verso cui bisogna effettuare la connessione. Nella macchina che ospita Suricata è necessario sia installato prelude-admin e che venga lanciata un'istanza di quest'ultimo impartendo da terminale il seguente comando:

```
$ prelude-admin register <profile name> <requested
permission> <manager address> --uid <uid> --gid <gid>
```

dove il *profile name* coincide con il *profile* impostato nel file di configurazione di Suricata. I permessi richiesti sono, nel caso di un sensore come Suricata, di poter scrivere (modalità *w*) messaggi IDMEF ("idmef:w"). Il *manager address* è l'indirizzo della macchina che ospita il prelude-manager mentre *uid* e *gid*

sono User ID e Group ID dello user e group di cui fa parte l'eseguibile di Suricata.

Una specifica interpretazione del comando potrebbe essere:

```
$ prelude-admin register "suricata" "idmef:w" localhost --uid 0 --gid 0.
```

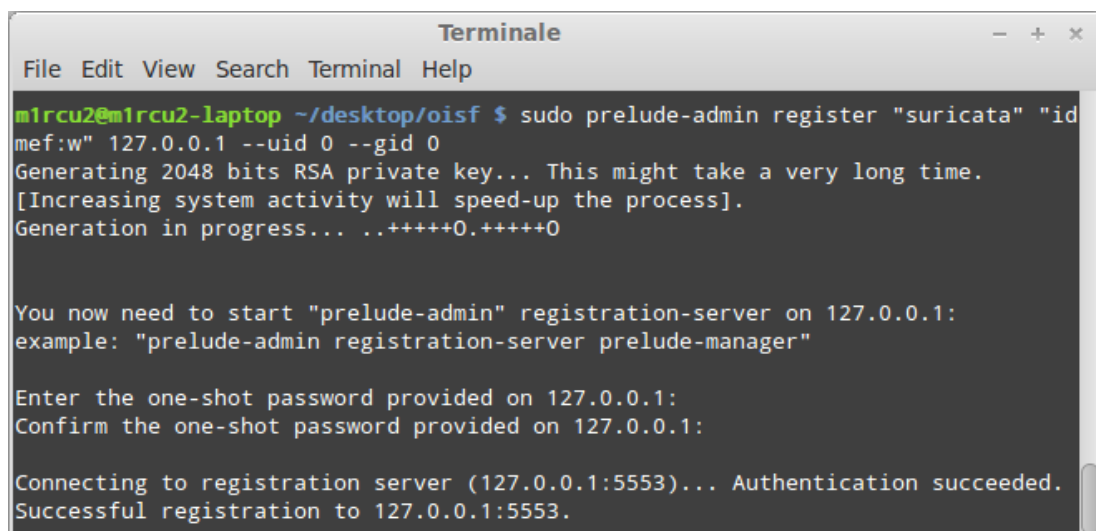
Nota: La prima volta che questo comando viene impartito verrà generata una chiave privata RSA a 2048bit per il profilo indicato; tale processo può richiedere parecchio tempo.

Eseguito il comando e generata la chiave privata (se necessario) l'istanza di prelude-admin della macchina del sensore rimarrà in attesa che venga aperta una corrispettiva istanza di prelude-admin anche sulla macchina che ospita il prelude-manager³⁸ e che gli sia fornita la password da questo generata.

Sul server in cui è in esecuzione prelude-manager, impartire il seguente comando:

```
$ prelude-admin registration-server prelude-manager
```

prelude-admin risponderà con la password che andrà digitata sulla macchina con Suricata. In ultimo, prelude-admin sulla macchina di prelude-manager chiederà di approvare la registrazione del sensore; rispondendo di sì ed eseguendo Suricata, il sistema di Prelude entrerà in funzione.



```
Terminale
File Edit View Search Terminal Help
m1rcu2@m1rcu2-laptop ~/desktop/oisf $ sudo prelude-admin register "suricata" "idmef:w" 127.0.0.1 --uid 0 --gid 0
Generating 2048 bits RSA private key... This might take a very long time.
[Increasing system activity will speed-up the process].
Generation in progress... ..+++++0.+++++0

You now need to start "prelude-admin" registration-server on 127.0.0.1:
example: "prelude-admin registration-server prelude-manager"

Enter the one-shot password provided on 127.0.0.1:
Confirm the one-shot password provided on 127.0.0.1:

Connecting to registration server (127.0.0.1:5553)... Authentication succeeded.
Successful registration to 127.0.0.1:5553.
```

Figura 3.4: Fase di richiesta di registrazione del sensore

³⁸ Le due macchine possono coincidere.


```

Terminale
File Edit View Search Terminal Help
* be using it.
*
* Your sensor WILL NOT START without sufficient permission to load the profile.
* [Please press enter if this is what you intend to do]

Generating 2048 bits RSA private key... This might take a very long time.
[Increasing system activity will speed-up the process].
Generation in progress... .+++++0.....+++++0

The "6z64frmz" password will be requested by "prelude-admin register"
in order to connect. Please remove the quotes before using it.

Generating 1024 bits Diffie-Hellman key for anonymous authentication...+++++
+++++
+++++>+++++
+++++>+++++
+++++O+++++
+++++^^^
Waiting for peers install request on 0.0.0.0:5553...

Connection from 127.0.0.1:51482...
Registration request for analyzerID="1479750729527086" permission="idmef:w".
Approve registration? [y/n]: y
127.0.0.1:51482 successfully registered.

```

Figura 3.5: Registrazione del sensore da parte di prelude-manager

Con Suricata e Prelude-IDS in funzione, gli alert generati dal primo finiranno nel database (o dove specificato) per mezzo del secondo. Per disporre delle informazioni salvate sul database, è possibile fare ricorso a Prewikka, l'interfaccia web di Prelude. Una volta installato Prewikka da pacchetto basterà specificare le informazioni per la connessione al database di Prelude nel file *prewikka.conf* ed avviare il processo *prewikka-httpd* affinché siano consultabili gli alert e le statistiche da pagina web.

L'uso di Prewikka nella consultazione degli alert permette di sfruttare il servizio *whois* di Prelude³⁹, per avere dettagli aggiuntivi sulla provenienza dei pacchetti che generano gli alert.

³⁹ https://www.prelude-ids.com/host_details.php

3 Report, segnalazioni e tools

The screenshot shows a web browser window displaying the Prewikka alert listing interface. The browser title is "[PREWIKKA] - Mozilla Firefox" and the address bar shows "localhost:8000/?view=alert_listing". The page header includes "Prewikka company ltd." and "Prelude console". A navigation menu on the left includes "Events", "Agents", "Statistics", "Settings", and "About". The main content area is a table of alerts with columns for "Alerts", "CorrelationAlerts", "ToolAlerts", and a timeline. The table contains multiple rows of alert data, including "ET TOR Known Tor Exit Node Traffic (49)", "SURICATA TCPv4 invalid checksum", and "ET POLICY Dropbox Client Broadcasting". A filter sidebar is visible on the left, and a URL bar at the bottom shows the full page URL.

Figura 3.6: Pagina dei report degli alert di Prewikka

The detailed alert view shows the following information:

- Alert:** COMMUNITY SIP TCP/IP message flooding directed to SIP proxy
- Ident:** 1:100000160
- Severity:** medium
- Type:** other
- Description:** Attempted Denial of Service
- Analyzer #1:**
 - Model: Snort
 - Name: snort
 - Analyzerid: 4064401856936935
 - Version: 2.8.5.2
 - Class: NIDS
 - Manufacturer: http://www.snort.org
 - Node name: localhost
 - Operating System: Linux 2.6.32-5-486
 - Process: (empty)
 - Process PID: 2294
 - Analyzer Path (1 not shown): (empty)
- Source(0):**
 - Node address: 90.147.12.204
 - Port: 37156
 - ip_version: 4
 - Protocol: tcp
- Target(0):**
 - Node address: 90.147.12.94
 - Port: 80
 - ip_version: 4
 - Protocol: tcp

Figura 3.7: Particolare di un alert

3.4 Squert, Snorby e Sguil

Squert, Snorby e Sguil sono tre GUI di reportistica che un analista della sicurezza di rete può sfruttare per consultare e studiare gli alert generati da Suricata, ben organizzati in appositi report eventualmente forniti di grafici e dati statistici. Tutte e tre le GUI sono accomunate dal fatto di appoggiarsi ad un database. Il database può essere opportunamente fornito di dati sugli alert tramite Barnyard2 (3.2 - Unified2 e Barnyard2) nel caso di Snorby e Sguil, mentre Squert si basa a sua volta sui dati forniti da Sguil. Tutti e tre i software sono nati appositamente per Snort, ma funzionano piuttosto bene anche con Suricata⁴⁰. Squert e Snorby sono front-end web mentre Sguil è più vicino alla struttura di Prelude ed è programmato come applicazione client/server. Squert e Snorby offrono entrambi rappresentazioni grafiche dei dati raccolti, funzionalità che manca a Sguil, il quale invece dispone della possibilità di inviare mail di reportistica, della messaggistica fra più amministratori che effettuano log-in allo stesso server Sguil ed altre funzionalità di amministrazione. Un confronto fra Snorby e Squert è già stato trattato.[11]

3.4.1 Sguil

Sguil è un programma multiplatforma scritto in Tcl/Tk “da analisti della sicurezza delle reti per analisti della sicurezza delle reti”. [12] Si presenta come una interfaccia grafica che fornisce accesso in real-time ad eventi, allarmi e pacchetti catturati. Il client Sguil necessita di un server Sguil a cui connettersi. Il sito ufficiale di Sguil è <http://sguil.sourceforge.net/> .

40 Alcune features sono state tuttavia progettate appositamente per funzionalità proprie di Snort, e sono inutilizzabili con Suricata (<https://lists.openinfosecfoundation.org/pipermail/oisf-users/2011-March/000549.html>)

3 Report, segnalazioni e tools

The screenshot shows the Sguil client interface. At the top, it displays 'SGUIL-0.8.0 - Connected To 90.147.13.44' and user information: 'ServerName: 90.147.13.44', 'UserName: admin', 'UserID: 2', and the date '2012-12-06 18:14:23 GMT'. Below this is a table of 'RealTime Events' with columns for CNT, Sensor, Alert ID, Date/Time, Src IP, SPort, Dst IP, DPort, Pr, and Event Message. The table lists several security alerts related to SURICATA stream events and checksum errors.

CNT	Sensor	Alert ID	Date/Time	Src IP	SPort	Dst IP	DPort	Pr	Event Message	
RT	22	securityo...	3.3	2012-12-06 17:34:42	90.147.13.44	54446	196.216.2.3	21	6	SURICATA STREAM Last ACK with wrong seq
RT	9	securityo...	3.8	2012-12-06 17:36:31	196.216.2.3	21	90.147.13.44	54465	6	SURICATA STREAM ESTABLISHED SYNACK resend
RT	1	securityo...	3.18	2012-12-06 17:40:54	196.216.2.3	31342	90.147.13.44	36447	6	SURICATA STREAM FIN1 FIN with wrong seq
RT	5	securityo...	3.39	2012-12-06 17:48:32	90.147.13.44	53475	196.216.2.3	30084	6	SURICATA STREAM TIMEWAIT ACK with wrong seq
RT	1	securityo...	3.41	2012-12-06 17:49:15	196.216.2.3	21	90.147.13.44	40705	6	SURICATA STREAM Packet with invalid timestamp
RT	1	securityo...	3.50	2012-12-06 17:51:08	90.147.12.41	1900	239.255.255.250	1900	17	SURICATA IPv4 Invalid checksum
RT	1	securityo...	3.61	2012-12-06 17:53:34	90.147.12.156	60461	239.255.255.250	1900	17	SURICATA IPv4 Invalid checksum
RT	5	securityo...	3.68	2012-12-06 17:57:51						SURICATA UDPv6 Invalid checksum
RT	1	securityo...	3.87	2012-12-06 18:07:51						SURICATA ICMPv6 Invalid checksum

Below the table, there are tabs for 'IP Resolution', 'Agent Status', 'Snort Statistics', 'System Msgs', and 'User Msgs'. The 'System Msgs' tab is active, showing a rule configuration for 'alert tcp any any -> any (msg:"SURICATA STREAM FIN1 FIN with wrong seq"; stream-event:fin1_fin_wrong_seq; sid:2210032; rev:1;)'. To the right, a packet capture view shows the details of a packet, including IP, TCP, and DATA sections.

Figura 3.9: Interfaccia del client Sguil

3.4.2 Squert

È un'interfaccia web progettata per fare query ai database Sguil e visualizzarne i dati registrati. Il programmatore di Squert afferma che la sua interfaccia web non intende essere un rimpiazzo per il client Sguil e che non pretende essere una console di consultazione eventi in *real-time*, né tantomeno *quasi real-time*.^[13]

Il sito ufficiale del progetto di Squert è <http://www.squertproject.org/>.

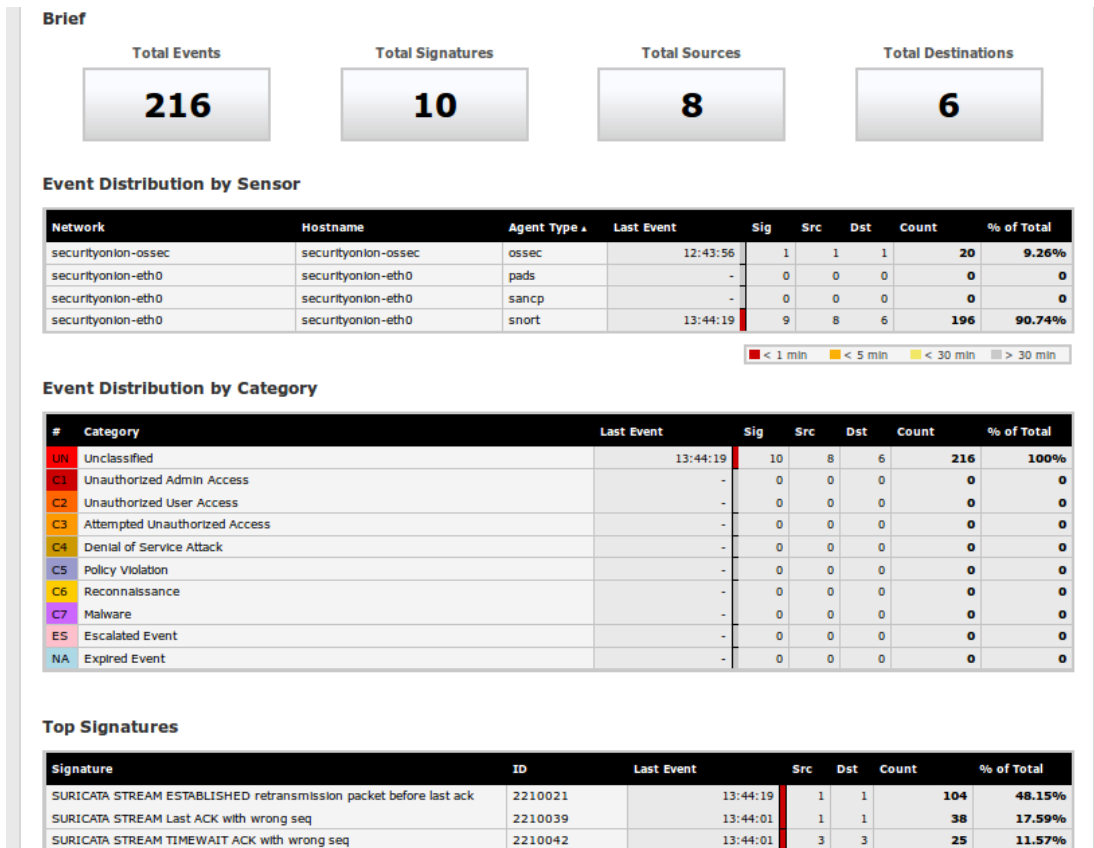


Figura 3.10: Home page di SQueRT

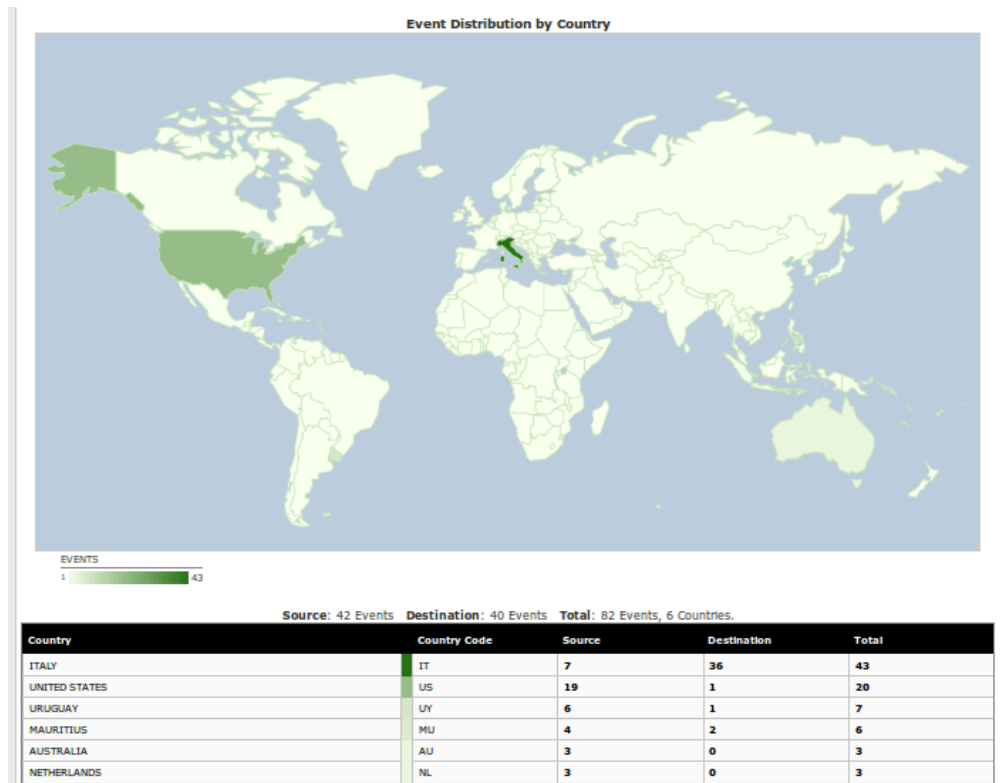


Figura 3.11: Squert. Provenienza geografica dei pacchetti che hanno generato alert

3.4.3 Snorby

Snorby è un'interfaccia web per l'analisi degli alert. Snorby non ha bisogno di affidarsi ad un altro software di intermezzo (come Squert necessita di Sguil); esso può attingere direttamente alle informazioni inviate al database da Barnyard2⁴¹. Ha un'interfaccia grafica ben curata. Benché tramite Snorby sia possibile consultare gli alert di Suricata ottimamente formattati ed in qualunque momento, nella *Dashboard* (la sua homepage) le statistiche vengono aggiornate di tanto in tanto (di default ogni 30 minuti) mancando così il real-time. Snorby permette di salvare tutte le statistiche dell'arco di tempo selezionato, in un comodo file .pdf per la consultazione e la stampa.

41 https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Suricata_Snorby_and_Barnyard2_set_up_guide

Il sito ufficiale di Snorby è <https://snorby.org/>. È disponibile una demo online all'indirizzo <http://demo.snorby.org/> (username: *demo@snorby.org* – password: *demo*).

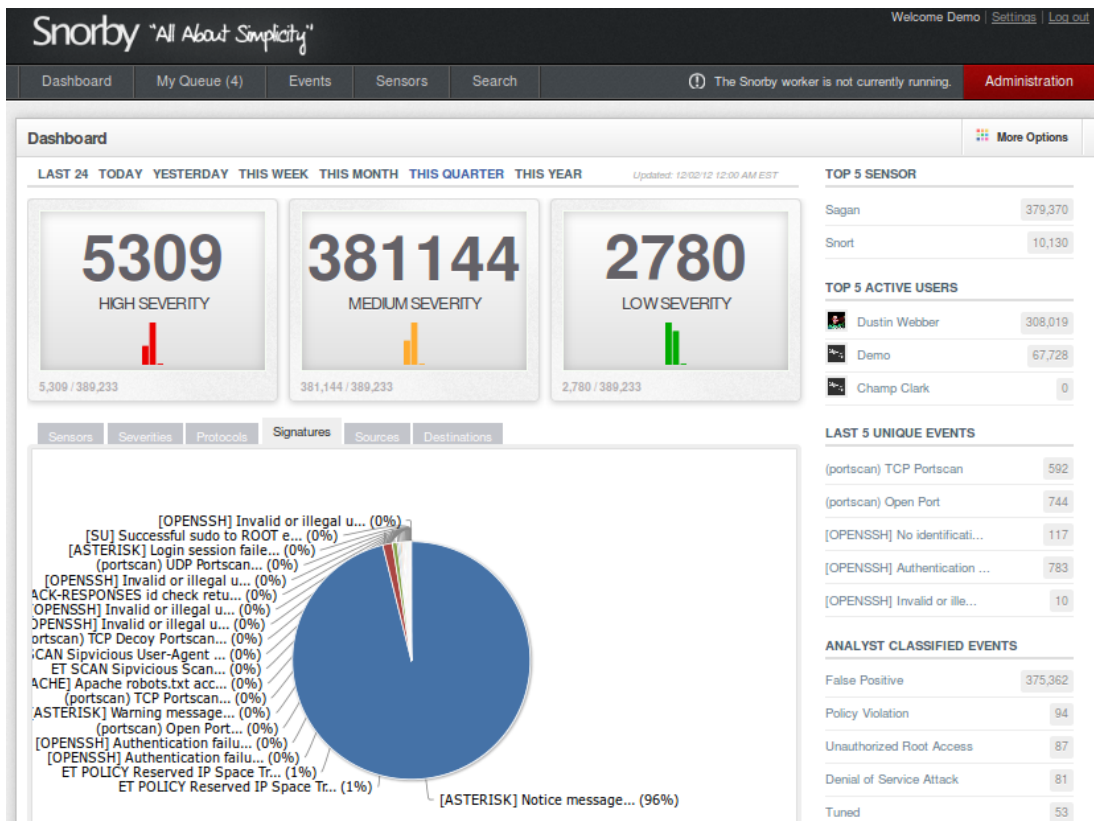


Figura 3.12: Pagina di report di Snorby da <http://demo.snorby.org/>

<input type="checkbox"/>	Sev.	Sensor	Source IP	Destination IP	Event Signature	Timestamp
<input type="checkbox"/>	3	Snort	US 65.204.173.194	US 173.255.236.165	(portscan) Open Port	5:55 PM
<input type="checkbox"/>	3	Snort	US 65.204.173.194	US 173.255.236.165	(portscan) Open Port	5:55 PM
<input type="checkbox"/>	3	Snort	US 65.204.173.194	US 173.255.236.165	(portscan) TCP Portscan	5:55 PM
<input type="checkbox"/>	3	Snort	US 65.204.173.194	US 173.255.236.165	(portscan) TCP Portscan	5:55 PM
<input type="checkbox"/>	3	Snort	US 156.40.117.1	US 173.255.236.165	(portscan) Open Port	4:45 PM
<input type="checkbox"/>	3	Snort	US 156.40.117.1	US 173.255.236.165	(portscan) Open Port	4:45 PM
<input type="checkbox"/>	3	Snort	US 156.40.117.1	US 173.255.236.165	(portscan) TCP Portscan	4:45 PM
<input type="checkbox"/>	3	Snort	US 156.40.117.1	US 173.255.236.165	(portscan) TCP Portscan	4:45 PM
<input type="checkbox"/>	3	Sagan	IN 121.242.134.150	US 173.255.236.165	[OPENSsh] No identification string - possible scan	4:33 PM
<input type="checkbox"/>	3	Snort	US 74.219.115.58	US 173.255.236.165	(portscan) Open Port	4:16 PM
<input type="checkbox"/>	3	Snort	US 76.182.149.254	US 173.255.236.165	(portscan) Open Port	4:15 PM
<input type="checkbox"/>	3	Snort	US 66.228.32.211	US 173.255.236.165	(portscan) TCP Portscan	4:15 PM
<input type="checkbox"/>	3	Snort	US 66.228.32.211	US 173.255.236.165	(portscan) TCP Portscan	4:15 PM
<input type="checkbox"/>	1	Sagan	KR 118.216.255.102	US 173.255.236.165	[OPENSsh] Authentication failure - Brute force [uid: 0]	3:51 PM
<input type="checkbox"/>	1	Sagan	KR 118.216.255.102	US 173.255.236.165	[OPENSsh] Authentication failure - Brute force [uid: 0]	3:51 PM
<input type="checkbox"/>	1	Sagan	KR 118.216.255.102	US 173.255.236.165	[OPENSsh] Authentication failure - Brute force [uid: 0]	3:51 PM

Figura 3.13: Pagina di report degli alert da <http://demo.snorby.org/>

3.5 Rules Editor

Un *rule editor* è un programma che permette di gestire e scrivere le regole in maniera grafica allo stesso modo in cui Snorby, Sguil e Squert gestiscono gli alert. Purtroppo, nella panorama di Suricata mancano dei punti di riferimento in questo ambito, nemmeno facendo riferimento a software per Snort. Infatti una fra le differenze fra Snort e Suricata è la struttura del file di configurazione (*suricata.yaml* nel primo, *snort.conf* nel secondo). I percorsi di sistema alle regole, le regole abilitate e disabilitate, etc. sono tutti valori fissati nei file di configurazione, che differiscono fra i due software.

Un'interfaccia web che ha prodotto risultati tutto sommato accettabili è *Snort Rule Manager*⁴². Il front-end web (eventualmente raggiungibile anche da remoto) permette, dopo un import iniziale dei ruleset, la creazione, modifica

⁴² Il progetto è in fase di sviluppo presso la piattaforma Google Project Hosting ed è raggiungibile all'indirizzo <http://code.google.com/p/snort-rule-manager/>.

o rimozione di regole dai file *.rules*. Snort Rule Manager è programmato in PHP, sfruttando il *framework Yii* ⁴³, appoggiandosi ad un database MySQL per operare sulle regole. Il codice è in licenza *GNU GPLv3*.

Come si può notare dal nome del progetto, il software è pensato per Snort e configurarlo per lavorare con Suricata è possibile ma ad un prezzo da pagare: poiché il software si aspetta di interpretare il file di configurazione di Snort per trovare la posizione delle regole su disco e determinare quali regole sono abilitate, sarà necessario creare un fittizio file di configurazione *snort.conf* in cui riadattare le informazioni contenute nel *suricata.yaml*. Questo ovviamente si ripercuote negativamente nella gestione delle regole. Qualora di sovente venissero aggiunti o rimossi nuovi file *.rules*; non c'è ancora un automatismo che permetta di aggiornare in contemporanea il file di configurazione di Suricata con quello fittizio di Snort.

Per procedere all'installazione di Snort Rule Manager, bisogna accertarsi di avere un server web pronto ad ospitare la piattaforma e le componenti descritte precedentemente (PHP con Yii e MySQL). I file sorgenti sono disponibili tramite SVN:

```
$ svn co http://snort-rule-manager.googlecode.com/svn/ .
```

Impartendo il comando precedente verrà scaricato il necessario nella working directory. Sarà necessario poi (se già non lo fosse) spostare i file in una directory che sia di DocumentRoot per il web server. Si può procedere poi con la configurazione.

Creare un database per ospitare le tabelle di Snort Rule Manager ed eventualmente creare un apposito nuovo utente. Successivamente, eseguire l'import di *IMPORT_THIS.sql* dalla root del progetto nel database appena creato.

Modificare il file *protected/config/main.php* specificando la stringa di connessione al DB , username e password secondo le proprie impostazioni:

⁴³ <http://www.yiiframework.com/>

```
'db'=>array(
    'connectionString' =>
'mysql:host=localhost;dbname=snortRules',
    'emulatePrepare' => true,
    'username' => 'snort',
    'password' => 'snort',
    'charset' => 'utf8',
),
```

Modificare infine il path ai file di configurazione di Snort (non ancora creato) ed alle regole *.rules* similmente all'esempio:

```
'params'=>array(
    // this is used in contact page
    'adminEmail'=>'webmaster@example.com',
    'confPath' => "/etc/suricata/snort.conf",
    'rulePath' => "/etc/suricata/rules/",
),
```

Come ultimo passo prima dell'importazione delle regole, bisogna creare il fittizio file *snort.conf* nel path appena specificato.

Il file dovrà contenere una prima riga che dichiari la variabile `RULE_PATH` in modo che contenga il path alle regole di Suricata.

```
var RULE_PATH /etc/suricata/rules
```

A seguire dovranno esserci, uno per riga i nomi dei singoli file *.rules*:

```
[...]
include $RULE_PATH/emerging-inappropriate.rules
include $RULE_PATH/emerging-malware.rules
include $RULE_PATH/emerging-misc.rules
[...]
```

Per velocizzare la procedura è possibile ricorrere a qualche script. Ad esempio, posizionandosi da terminale nella directory delle regole:

```
# ls -lS | awk {'print$9'} | while read line; do echo
"include \"$RULE_PATH/$line"; done >> /etc/suricata/snort.conf
```

Scriverà in append nel file */etc/suricata/snort.conf* i percorsi alle regole nel formato Snort.

A questo punto tutto è pronto per l'uso della piattaforma. Aprire un browser e connettersi al web server che ospita Snort Rule Manager. Cliccare sulla tab

Manage e premere il pulsante *ReloadFromCacheFile*. L'operazione richiederà il tempo necessario a popolare il database con le regole. Qualora non si volessero far processare determinati file *.rules*, andrebbe anteposto alla corrispondente riga nel file *snort.conf* un *#* come commento.

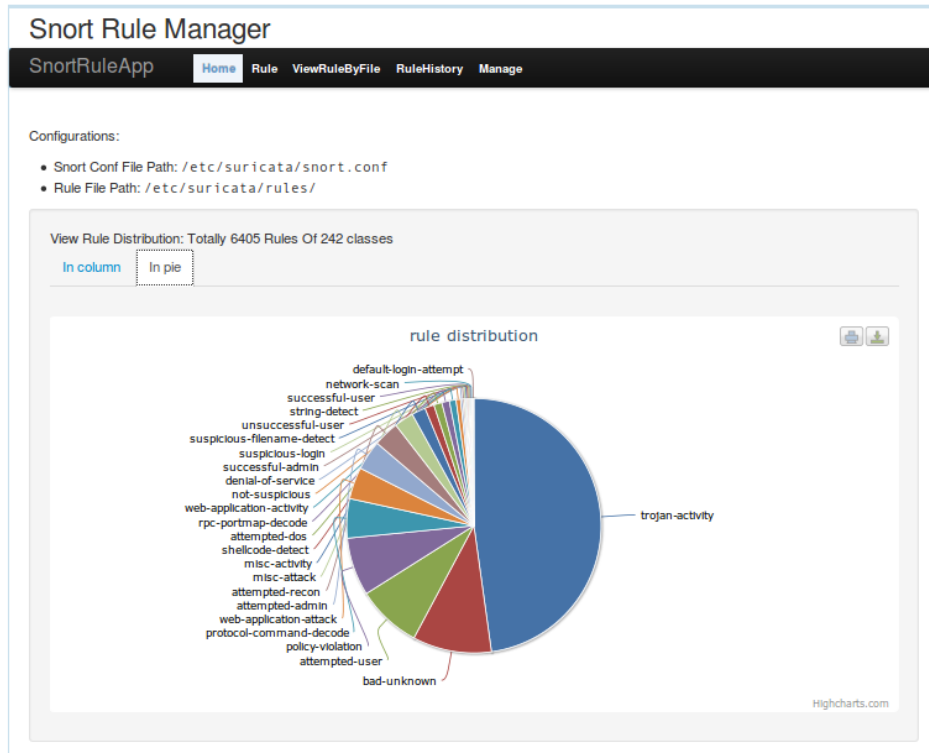


Figura 3.14: Home page di Snort Rule Manager

ID	gid	sid	Fullcontent	Rule Class	Category
2	1	2404001	alert ip \$HOME_NET any -> [109.236.81.141,109.70.71.150,109.74.195.116,109.74.200.40,109.74.206.120,111.67.17.204,113.212.96.205,113.35.224.60,114.112.255.81,114.207.246.138,116.123.41.170,116.229.239.244,117.20.11.38,118.218.219.175,119.10.181.1,119.59.99.160,119.59.99.52,119.82.231.212] any (msg "ET CNC Shadowserver Reported CNC Server IP (group 2)";reference url,doc:emergingthreats.net/bin/view/Main/BoTCC;reference url,www.ahadosever.org;threshold: type limit, track by_src, seconds 3600, count 1;flowbits:set,ET.Evil;flowbits:unset,BoTCCP;class:trojan-activity;sid:2404001;rev:978)	trojan-activity	botcc.rules
3	1	2404002	alert ip \$HOME_NET any -> [119.59.99.52,119.82.231.212,122.248.218.2,125.160.17.72,128.194.112.48,128.39.65.226,128.39.65.230,130.237.188.216,130.240.22.202,140.211.166.64,145.89.150.59,147.32.127.200,149.255.99.157,149.255.99.168,149.255.99.214,149.3.131.6,149.47.133.128,150.254.65.42] any (msg "ET CNC Shadowserver Reported CNC Server IP (group 3)";reference url,doc:emergingthreats.net/bin/view/Main/BoTCC;reference url,www.ahadosever.org;threshold: type limit, track by_src, seconds 3600, count 1;flowbits:set,ET.Evil;flowbits:unset,BoTCCP;class:trojan-activity;sid:2404002;rev:978)	trojan-activity	botcc.rules
4	1	2404003	alert ip \$HOME_NET any -> [149.47.133.128,150.254.65.42,154.35.136.34,154.35.64.10,154.35.64.100,154.35.64.101,154.35.64.104,154.35.64.106,154.35.64.107,154.35.64.113,154.35.64.115,154.35.64.119,154.35.64.122,154.35.64.123,154.35.64.129,154.35.64.132,154.35.64.151] any (msg "ET CNC Shadowserver Reported CNC Server IP (group 4)";reference url,doc:emergingthreats.net/bin/view/Main/BoTCC;reference url,www.ahadosever.org;threshold: type limit, track by_src, seconds 3600, count 1;flowbits:set,ET.Evil;flowbits:unset,BoTCCP;class:trojan-activity;sid:2404003;rev:2878)	trojan-activity	botcc.rules
5	1	2404004	alert ip \$HOME_NET any -> [154.35.64.132,154.35.64.15,154.35.64.16,154.35.64.17,154.35.64.19,154.35.64.25,154.35.64.26,154.35.64.29,154.35.64.30,154.35.64.31,154.35.64.32,154.35.64.34,154.35.64.35,154.35.64.42,154.35.64.46,154.35.64.49,154.35.64.50,154.35.64.53] any (msg "ET CNC Shadowserver Reported CNC Server IP (group 5)";reference url,doc:emergingthreats.net/bin/view/Main/BoTCC;reference url,www.ahadosever.org;threshold: type limit, track by_src, seconds 3600, count 1;flowbits:set,ET.Evil;flowbits:unset,BoTCCP;class:trojan-activity;sid:2404004;rev:2878)	trojan-activity	botcc.rules
6	1	2404005	alert ip \$HOME_NET any -> [154.35.64.50,154.35.64.53,154.35.64.54,154.35.64.55,154.35.64.62,154.35.64.65,154.35.64.69,154.35.64.70,154.35.64.72,154.35.64.76,154.35.64.78,154.35.64.81,154.35.64.82,154.35.64.83,154.35.64.89,154.35.64.92,154.35.64.97,154.35.64.99] any (msg "ET CNC Shadowserver Reported CNC Server IP (group 6)";reference url,doc:emergingthreats.net/bin/view/Main/BoTCC;reference url,www.ahadosever.org;threshold: type limit, track by_src, seconds 3600, count 1;flowbits:set,ET.Evil;flowbits:unset,BoTCCP;class:trojan-activity;sid:2404005;rev:2878)	trojan-activity	botcc.rules

Figura 3.15: Tabella delle regole in cui è possibile leggere, modificare ed eliminare le regole

Nota: la piattaforma non permette ancora di disabilitare o abilitare le regole. Le regole a cui è anteposto il simbolo di *#* non verranno lette dal sistema, ed anteporlo manualmente nella modifica genererà un errore in fase di salvataggio.

4 Caso di studio

In questo capitolo verrà presentato uno scenario reale di utilizzo di Suricata IDPS. Verranno indicati gli elementi della rete in esame, come disporli e come configurarli. Infine verrà diretto un *port-scan* ad un host interno alla rete, dietro Suricata, per mostrare come si comporta il software, prima in modalità di detection system e poi in modalità di prevention system.

I test sono stati condotti utilizzando delle macchine virtuali, mediante *Oracle VM Virtualbox*. Il sistema operativo della macchina host è Linux Mint 13 a 64bit. La versione del kernel GNU/Linux è la 3.6.6-030606-generic x86_64.

4.1 La topologia della rete

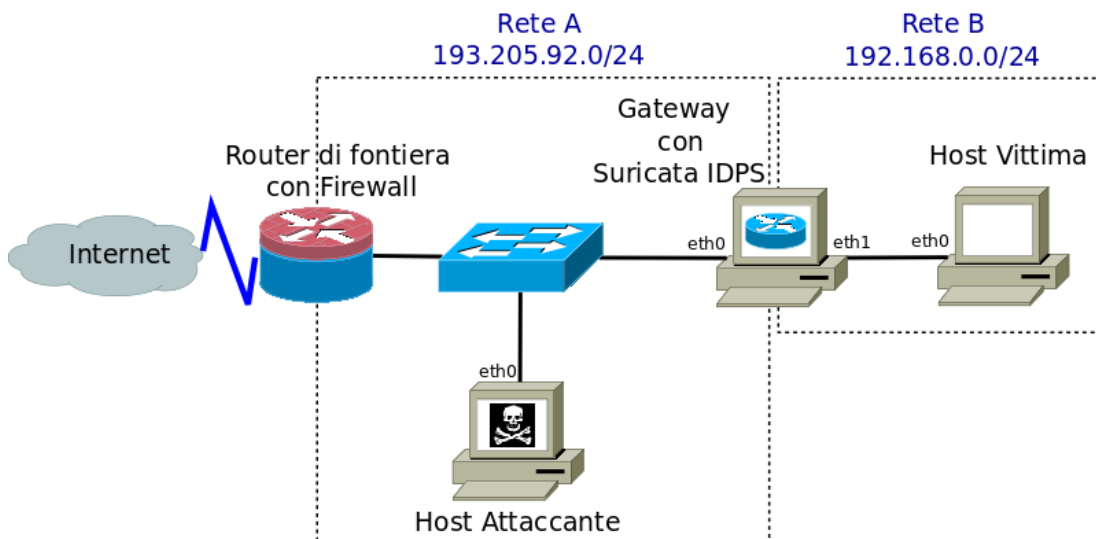


Figura 4.1: Topologia di rete

La topologia di riferimento nel capitolo sarà quella in figura 4.1. I test in realtà sono stati svolti sul PC attaccante come macchina host mentre il PC Gateway e l'host vittima sono entrambi macchine guest appositamente virtualizzate e configurate.

4.1.1 Host Vittima

Il PC vittima è un host (**guest** in riferimento all'ambiente virtualizzato)

della rete B 192.168.0.0/24. Il suo indirizzo IP è impostato manualmente a 192.168.1.100 ed ha come indirizzo di gateway 192.168.1.1. È stato inoltre fornito l'indirizzo di un DNS perché potesse risolvere i nomi a dominio. Il sistema operativo installato sulla macchina è Microsoft Windows XP (Service Pack 3). Il firewall sul sistema è stato disattivato per apprezzare, in questo caso di test, gli effetti di Suricata quale IDPS.

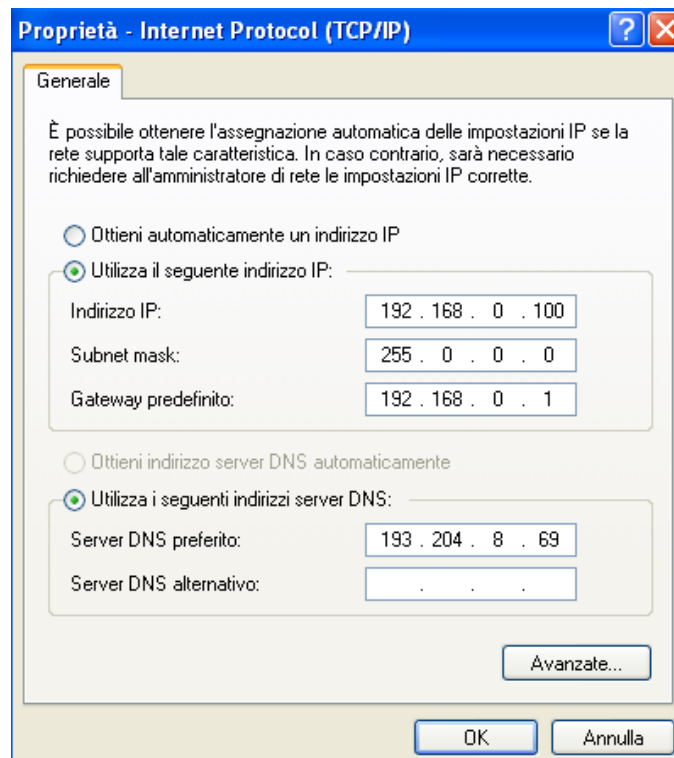


Figura 4.2: Configurazione *eth0* del PC vittima

La macchina virtuale è stata fornita di una scheda di rete di tipo PCnet-FAST III connessa ad una rete interna a cui è stato assegnato il nome *intnet*.

4.1.2 Gateway con Suricata IDPS

Questa macchina virtuale monta la distribuzione Smooth-Sec 1.2, precedentemente trattata nel capitolo 2.8.1. La distribuzione non è eseguita in modalità live ma è stata installata, in quanto è importante che le modifiche ap-

plicate ad essa rimangano persistenti. Suricata è stato aggiornato manualmente alla versione 1.3.5 e compilato con il parametro `--enable-nfqueue`.

La macchina dispone di due schede di rete Intel Pro/1000 MT Desktop per svolgere il compito di gateway fra la rete A (*eth0*) e la rete B (*eth1*).

All'interfaccia *eth0* è lasciato acquisire l'indirizzo IP, netmask, default gateway ed indirizzo DNS tramite DHCP. Questa scheda infatti è impostata in modalità *bridge* con la scheda di rete della macchina host.

L'interfaccia *eth1* invece è stata configurata manualmente utilizzando il menù all'avvio di Smooth-Sec. L'IP per questa interfaccia sarà 192.168.0.1 e nelle impostazioni della macchina virtuale, sarà associata anch'essa alla rete di nome *intnet*.

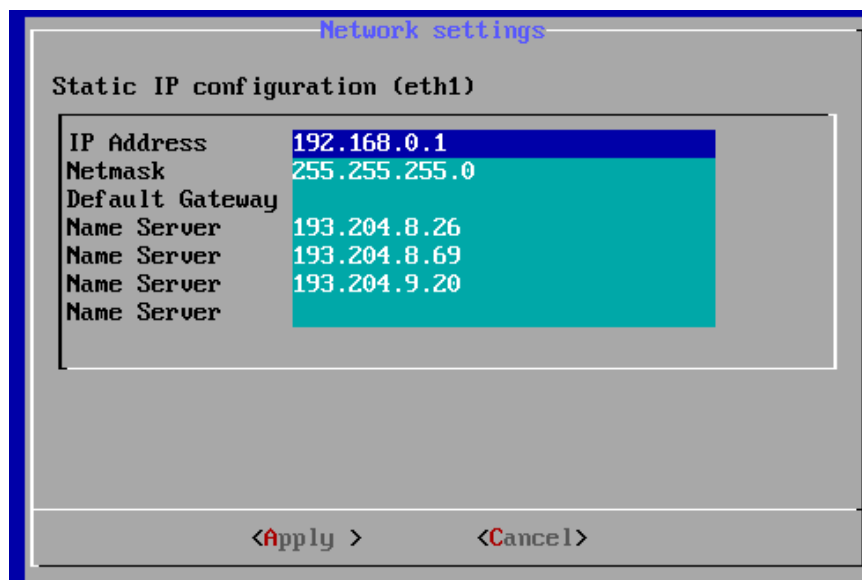


Figura 4.3: Configurazione *eth1* del gateway

La configurazione della macchina gateway non è però conclusa: bisogna abilitarla al forwarding dei pacchetti da e verso la rete interna B poiché in caso contrario l'host vittima non potrà comunicare con l'esterno. Per questo genere di configurazioni si è fatto ricorso all'uso di IPTables.

Di seguito viene illustrato come abilitare il NATting ed impostare le regole di forwarding dei pacchetti:

```
# iptables -A FORWARD -o eth0 -i eth1 -s 192.168.0.0/24 -m
contrack --ctstate NEW -j ACCEPT

# iptables -A FORWARD -m contrack --ctstate
ESTABLISHED,RELATED -j ACCEPT

# iptables -t nat -F POSTROUTING

# iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Per abilitare il forwarding è stato inoltre necessario eseguire

```
# sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
```

e decommentare la riga `net.ipv4.ip_forward=1` del file `/etc/sysctl.conf`.

Sulla macchina non sono state definite altre regole IPTables, così da non interferire con il test dell'IDPS⁴⁴.

4.1.3 Host Attaccante

L'attaccante (o macchina **host** in riferimento allo scenario di virtualizzazione) è un host della rete 193.205.92.0/24 ed acquisisce il suo indirizzo IP sulla interfaccia `eth0` tramite DHCP; mette inoltre la propria scheda di rete fisica a disposizione del bridging per conto della macchina gateway. L'host monta Linux Mint 13 a 64bit con kernel GNU/Linux 3.6.6-030606-generic x86_64.

La macchina così configurata riesce a comunicare con l'interfaccia `eth0` della macchina gateway; tuttavia il router a monte della rete non essendo a conoscenza dell'esistenza di un'altra rete (la 192.168.0.0/24) non permette all'attaccante di comunicare con la rete interna. Per aggirare questo problema è stato sufficiente definire una *static route*:

```
# ip route add 192.168.0.0/24 via 193.205.92.135
```

supponendo che all'interfaccia `eth0` del gateway fosse stato attribuito l'indiriz-

44 In uno scenario reale è d'obbligo la disposizione di (almeno) un firewall opportunamente configurato all'interno della rete, specie se la macchina è un gateway o si trova immediatamente dietro di esso. Un IDPS non deve mai sostituire un firewall, bensì affiancarlo.

zo IP 193.205.92.135.

4.1.4 Test di connettività

Ecco uno schema che riassume gli indirizzi associati alle varie interfacce ed ai vari host:

Host	Intf	IP Addr	Netmask CIDR	Gateway
Host Attaccante	eth0	193.205.92.130	/24	193.205.92.2
Gateway IDPS	eth0	193.205.92.135	/24	193.205.92.2
	eth1	192.168.0.1	/24	
Host Vittima	eth0	192.168.0.100	/24	192.168.0.1

Come verifica della corretta connettività fra i vari host sono stati eseguiti una serie di ping da e verso ciascuno di essi.

- Host Vittima → Gateway

```
C:\Documents and Settings\test>ping 192.168.0.1

Pinging 192.168.0.1 with 32 bytes of data:

Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64
Reply from 192.168.0.1: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.0.1:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
```


- Host Vittima → Host Attacacnte

```
C:\Documents and Settings\test>ping 193.205.92.130

Pinging 193.205.92.130 with 32 bytes of data:

Reply from 193.205.92.130: bytes=32 time<1ms TTL=63
Reply from 193.205.92.130: bytes=32 time<1ms TTL=63
Reply from 193.205.92.130: bytes=32 time<1ms TTL=63
Reply from 193.205.92.130: bytes=32 time<1ms TTL=63

Ping statistics for 193.205.92.130:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

- Gateway → Host Vittima

```
root@Smooth-Sec ~# ping 192.168.0.100
PING 192.168.0.100 (192.168.0.100) 56(84) bytes of data.
 64 bytes from 192.168.0.100: icmp_seq=1 ttl=128
time=0.096 ms
 64 bytes from 192.168.0.100: icmp_seq=2 ttl=128
time=0.361 ms
 64 bytes from 192.168.0.100: icmp_seq=3 ttl=128
time=0.321 ms
 64 bytes from 192.168.0.100: icmp_seq=4 ttl=128
time=0.248 ms
^C
--- 192.168.0.100 ping statistics ---
 4 packets transmitted, 4 received, 0% packet loss, time
2999ms
 rtt min/avg/max/mdev = 0.096/0.256/0.361/0.102 ms
```

- Gateway → Host Attaccante

```
root@Smooth-Sec ~# ping 193.205.92.130
PING 193.205.92.130 (193.205.92.130) 56(84) bytes of
data.
 64 bytes from 193.205.92.130: icmp_seq=1 ttl=64
time=0.094 ms
 64 bytes from 193.205.92.130: icmp_seq=2 ttl=64
time=0.125 ms
 64 bytes from 193.205.92.130: icmp_seq=3 ttl=64
time=0.136 ms
 64 bytes from 193.205.92.130: icmp_seq=4 ttl=64
time=0.131 ms
^C
--- 193.205.92.130 ping statistics ---
 4 packets transmitted, 4 received, 0% packet loss, time
2999ms
 rtt min/avg/max/mdev = 0.094/0.121/0.136/0.019 ms
```

- Host Attaccante → Gateway

```
mlrcu2-laptop client # ping 193.205.92.135
PING 193.205.92.135 (193.205.92.135) 56(84) bytes of
data.
 64 bytes from 193.205.92.135: icmp_req=1 ttl=64
time=0.143 ms
 64 bytes from 193.205.92.135: icmp_req=2 ttl=64
time=0.168 ms
 64 bytes from 193.205.92.135: icmp_req=3 ttl=64
time=0.133 ms
 64 bytes from 193.205.92.135: icmp_req=4 ttl=64
time=0.145 ms
^C
--- 193.205.92.135 ping statistics ---
 4 packets transmitted, 4 received, 0% packet loss, time
3000ms
 rtt min/avg/max/mdev = 0.133/0.147/0.168/0.015 ms
```

- Host Attaccante → Host Vittima

```
mlrcu2-laptop client # ping 192.168.0.100
PING 192.168.0.100 (192.168.0.100) 56(84) bytes of data.
 64 bytes from 192.168.0.100: icmp_req=1 ttl=127
time=0.388 ms
 64 bytes from 192.168.0.100: icmp_req=2 ttl=127
time=0.370 ms
 64 bytes from 192.168.0.100: icmp_req=3 ttl=127
time=0.477 ms
 64 bytes from 192.168.0.100: icmp_req=4 ttl=127
time=0.380 ms
^C
--- 192.168.0.100 ping statistics ---
 4 packets transmitted, 4 received, 0% packet loss, time
2997ms
 rtt min/avg/max/mdev = 0.370/0.403/0.477/0.049 ms
```

4.2 Suricata come Intrusion Detection System

Nella distribuzione Smooth-Sec, Suricata è in esecuzione dall'avvio monitorando l'interfaccia eth0. Qualora non fosse stato così, basandosi su quanto scritto nei capitoli precedenti, basterebbe eseguire:

```
# suricata -D --pidfile=/var/run/suricata.pid -c
/etc/suricata/suricata.yaml -i eth0
```

Le regole pre-caricate sono quelle di Emerging Threats⁴⁵.

Il test di intrusion detection è stato effettuato eseguendo un port-scan dal PC attaccante nella rete A al PC vittima della rete B per mezzo del software *nmap* (usando *zenmap* per l'interfaccia grafica).

⁴⁵ In Smooth-Sec 1.2 è disponibile uno script per eseguire l'aggiornamento del set di regole.

Il path verso tale script è `/root/script.utils/rules.update` e sarebbe opportuno aggiungere questo script in esecuzione automatica tramite Cron.

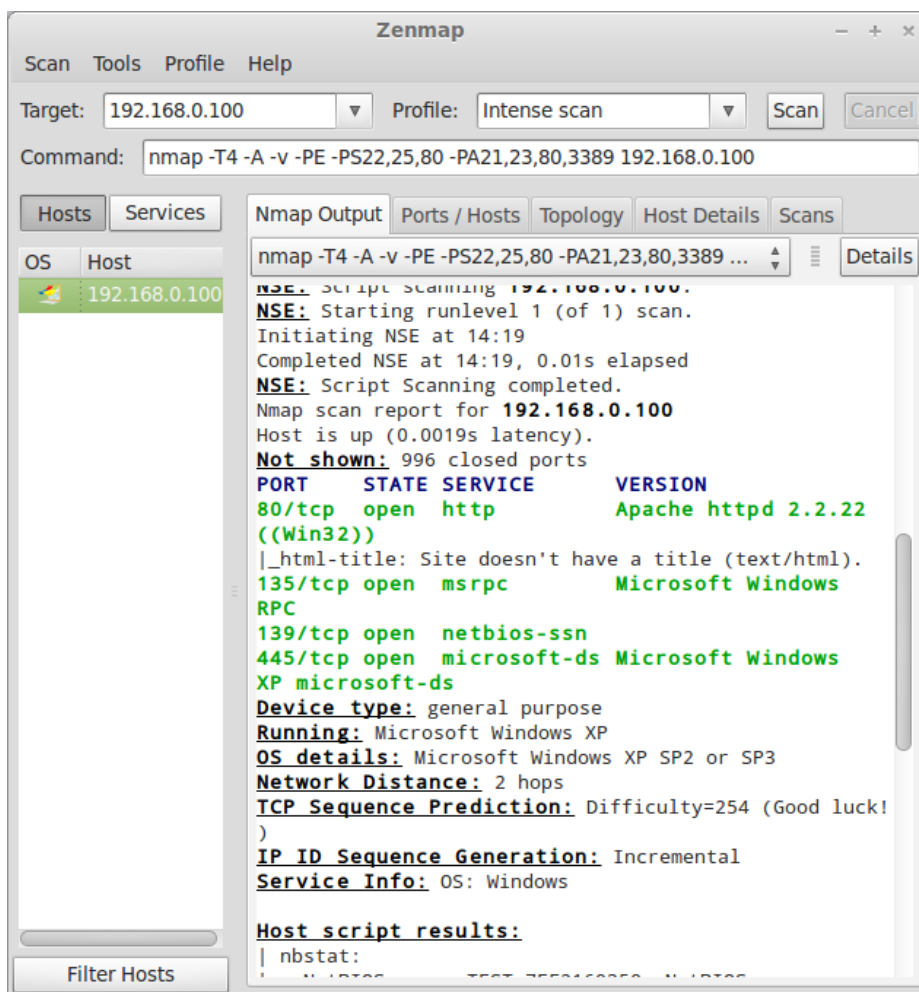


Figura 4.4: Port-scan verso l'host vittima

In Figura 4.2 si può notare come il port-scan sia andato a buon fine da parte dell'host attaccante rilevando le porte 80, 135, 139, 445 come aperte e fornendo dettagli sull'host grazie ai servizi msrpc, NetBIOS e microsoft-ds.

Ecco però cosa Suricata, sul PC gateway è riuscito a monitorare:

```

12/07/2012-14:58:47.072509  [**] [1:22109360:2] ET POLICY
Suspicious inbound to NetBIOS port 139 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP}
193.205.92.130:41616 -> 192.168.0.100:139

12/07/2012-14:58:47.072616  [**] [1:22109359:2] ET POLICY
Suspicious inbound to MSRPC port 135 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP}
193.205.92.130:41616 -> 192.168.0.100:135

12/07/2012-14:58:47.073604  [**] [1:2210935:2] ET POLICY
Suspicious inbound to Microsoft-DS port 445 [**]
[Classification: Potentially Bad Traffic] [Priority: 2] {TCP}
193.205.92.130:41616 -> 192.168.0.100:445

```

```
12/07/2012-14:58:47.073894  [**] [1:2010937:2] ET POLICY
Suspicious inbound to MySQL port 3306 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP}
193.205.92.130:41616 -> 192.168.0.100:3306

12/07/2012-14:58:47.087453  [**] [1:2010935:2] ET POLICY
Suspicious inbound to MSSQL port 1433 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP}
193.205.92.130:41616 -> 192.168.0.100:1433

12/07/2012-14:58:47.087701  [**] [1:2010939:2] ET POLICY
Suspicious inbound to PostgreSQL port 5432 [**]
[Classification: Potentially Bad Traffic] [Priority: 2] {TCP}
193.205.92.130:41616 -> 192.168.0.100:5432

12/07/2012-14:58:47.097649  [**] [1:2002911:4] ET SCAN
Potential VNC Scan 5900-5920 [**] [Classification: Attempted
Information Leak] [Priority: 2] {TCP} 193.205.92.130:41616 ->
192.168.0.100:5901

12/07/2012-14:58:47.100471  [**] [1:2002910:4] ET SCAN
Potential VNC Scan 5800-5820 [**] [Classification: Attempted
Information Leak] [Priority: 2] {TCP} 193.205.92.130:41616 ->
192.168.0.100:5802

12/07/2012-14:58:48.235446  [**] [1:2002911:4] ET SCAN
Potential VNC Scan 5900-5920 [**] [Classification: Attempted
Information Leak] [Priority: 2] {TCP} 193.205.92.130:41616 ->
192.168.0.100:5904

12/07/2012-14:58:48.238111  [**] [1:2010936:2] ET POLICY
Suspicious inbound to Oracle SQL port 1521 [**]
[Classification: Potentially Bad Traffic] [Priority: 2] {TCP}
193.205.92.130:41616 -> 192.168.0.100:1521

12/07/2012-14:58:48.298914  [**] [1:22109359:2] ET POLICY
Suspicious inbound to MSRPC port 135 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP}
193.205.92.130:53065 -> 192.168.0.100:135

12/07/2012-14:58:48.298932  [**] [1:22109360:2] ET POLICY
Suspicious inbound to NetBIOS port 139 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP}
193.205.92.130:58116 -> 192.168.0.100:139

12/07/2012-14:58:48.298953  [**] [1:2210935:2] ET POLICY
Suspicious inbound to Microsoft-DS port 445 [**]
[Classification: Potentially Bad Traffic] [Priority: 2] {TCP}
193.205.92.130:54260 -> 192.168.0.100:445

12/07/2012-14:58:54.324607  [**] [1:22109359:2] ET POLICY
Suspicious inbound to MSRPC port 135 [**] [Classification:
Potentially Bad Traffic] [Priority: 2] {TCP}
193.205.92.130:53068 -> 192.168.0.100:135

12/07/2012-14:58:55.001578  [**] [1:2101390:6] GPL SHELLCODE
x86 inc ebx NOOP [**] [Classification: Executable code was
detected] [Priority: 1] {UDP} 193.205.92.130:41766 ->
192.168.0.100:38475

12/07/2012-14:58:55.001868  [**] [1:2101390:6] GPL SHELLCODE
x86 inc ebx NOOP [**] [Classification: Executable code was
detected] [Priority: 1] {ICMP} 192.168.0.100:3 ->
```

```
193.205.92.130:3
 12/07/2012-14:58:55.447739  [**] [1:2009358:4] ET SCAN Nmap
Scripting Engine User-Agent Detected (Nmap Scripting Engine)
[**] [Classification: Web Application Attack] [Priority: 1]
{TCP} 193.205.92.130:41818 -> 192.168.0.100:80

 12/07/2012-14:58:55.448284  [**] [1:2009358:4] ET SCAN Nmap
Scripting Engine User-Agent Detected (Nmap Scripting Engine)
[**] [Classification: Web Application Attack] [Priority: 1]
{TCP} 193.205.92.130:41819 -> 192.168.0.100:80

 12/07/2012-14:58:55.449350  [**] [1:2210935:2] ET POLICY
Suspicious inbound to Microsoft-DS port 445 [**]
[Classification: Potentially Bad Traffic] [Priority: 2] {TCP}
193.205.92.130:54265 -> 192.168.0.100:445

 12/07/2012-14:58:55.451459  [**] [1:2009358:4] ET SCAN Nmap
Scripting Engine User-Agent Detected (Nmap Scripting Engine)
[**] [Classification: Web Application Attack] [Priority: 1]
{TCP} 193.205.92.130:41823 -> 192.168.0.100:80

 12/07/2012-14:58:55.451965  [**] [1:2210935:2] ET POLICY
Suspicious inbound to Microsoft-DS port 445 [**]
[Classification: Potentially Bad Traffic] [Priority: 2] {TCP}
193.205.92.130:54268 -> 192.168.0.100:445
```

Questi sono i log che Suricata ha registrato sul file `/var/log/suricata/fast.log`.

A seguire, una delle regole che sono entrate in gioco nell'azione di detection dello scan, dal file `emerging-scan.rule`:

```
alert http $EXTERNAL_NET any -> $HOME_NET any (msg:"ET SCAN
Nmap Scripting Engine User-Agent Detected (Nmap Scripting
Engine)"; flow:to_server,established; content:"|0d 0a|User-
Agent|3a| Mozilla/5.0 (compatible|3b| Nmap Scripting Engine";
nocase; reference:url,doc.emergingthreats.net/2009358;
classtype:web-application-attack; sid:2009358; rev:4;)
```

Poichè Smooth-Sec dispone dell'interfaccia web Snorby, è possibile consultare i log in modo più semplice ed organizzato del tradizionale file di log connettendosi, per esempio, dal PC vittima all'indirizzo <https://192.168.0.1> .

The screenshot displays a detailed report for a scan event in the Snorby web interface. The event is identified as 'ET SCAN Nmap Scripting Engine User-Agent Detected (Nmap Scripting Engine)' occurring at 2:58 PM. The source IP is 193.205.92.130 and the destination is 192.168.0.100.

IP Header Information:

Source	Destination	Ver	Hlen	Tos	Len	ID	Flags	Off	TTL	Proto	Csum
193.205.92.130	192.168.0.100	4	5	0	190	0	0	0	0	6	14212

Signature Information:

Generator ID	Signature ID	Signature Revision	Activity (72/608)
1	2009358	4	12%

TCP Header Information:

Src Port	Dest Port	Seq	Ack	Off	Res	Flags	Win	Csum	URP
41823	80	0	0	5	0	0	0	64050	0

Payload:

```

00000000: 47 54 2f 20 48 54 54 50 2f 31 2e 31 0d 0a 55 73 65 72 2d 41 67 65 6e 74  GET./..HTTP/1.1..User-Agent
0000001A: 3a 20 4d 6f 7a 69 6c 6c 61 2f 35 2e 30 20 28 63 6f 6d 70 61 74 69 62 6c 65 3b  :.Mozilla/5.0.(compatible;
00000034: 20 4e 6d 61 70 20 53 63 72 69 70 74 69 6e 67 20 45 6e 67 69 6e 65 3b 20 68 74  .Nmap.Scripting.Engine;.ht
0000004E: 74 70 3a 2f 2f 6e 6d 61 70 2e 6f 72 67 2f 62 6f 6f 6b 2f 6e 73 65 2e 68 74 6d  tp://nmap.org/book/nse.htm
00000068: 6c 29 0d 0a 43 6f 6e 6e 65 63 74 69 6f 6e 3a 20 43 6c 6f 73 65 0d 0a 48 6f 73  l)..Connection:.Close..Hos
00000082: 74 3a 20 31 39 32 2e 31 36 38 2e 30 2e 31 30 30 0d 0a 0d 0a  t:.192.168.0.100....

```

Notes: This event currently has zero notes - You can add a note by clicking the button below.

Other Scan Events:

- ET SCAN Potential VNC Scan 5900-5920
- ET POLICY Suspicious inbound to Oracle SQL port 1521
- ET POLICY Suspicious inbound to MySQL port 3306
- ET POLICY Suspicious inbound to MSSQL port 1433
- ET POLICY Suspicious inbound to PostgreSQL port 5432
- ET SCAN Potential VNC Scan 5900-5920
- ET SCAN Potential VNC Scan 5800-5820

Figura 4.5: Report dello scan da interfaccia web Snorby

4.3 Suricata come Intrusion Prevention System

Nel precedente paragrafo è stato dimostrato come l'azione di detection di un port-scan da parte di Suricata sia un'azione non troppo difficile, una volta che si dispone di un ambiente configurato in maniera appropriata e si conoscono gli elementi in gioco. Disporre Suricata come sistema di prevention richiede invece qualche passaggio aggiuntivo.

4.3.1 Modalità in-line ed IPTables

Normalmente Suricata monitora il traffico su una interfaccia (parametro `-i <intf>` all'avvio) effettuando una copia dei pacchetti in transito per mezzo delle librerie pcap lasciando tuttavia il pacchetto originale proseguire inalterato verso la sua destinazione. Affinchè si possa garantire che i pacchetti che matchano determinate regole subiscano il drop/reject e ne venga quindi impedito il proseguimento nella rete bisogna disporre Suricata in modalità in-line.

Quando Suricata lavora in qualità di solo IDS, può essere disposto su un gateway o su un sensore in cui viene dirottato in copia il traffico, usando eventualmente la monitor port di uno switch o una network-tap. Perché Suricata lavori come IPS è invece necessario che sia disposto come agente posto nel mezzo, cioè su una macchina che riceve ed inoltra il traffico di rete. Questo è possibile sfruttando una macchina che disponga di due interfacce di rete come il PC gateway nel caso di test di questo capitolo.

Avendo a disposizione due interfacce di rete per le quali passano i pacchetti originali, un IPS si interpone fra le due ed acquista la capacità di gestire il traffico stesso dei pacchetti⁴⁶. Questa disposizione prende il nome di modalità **in-line**.

Affinchè Suricata venga disposto come in-line, il traffico in entrata e in uscita deve essere deviato e fatto passare per una coda di analisi. Questo è possibile in ambiente GNU/Linux utilizzando *Netfilter*⁴⁷ e quindi *IPTables*⁴⁸. Tramite precise regole si può inviare il traffico in un'apposita coda chiamata **NFQueue** (da *NetFilter Queue*) ed impostare Suricata per lavorare su tale

46 Se un IPS come Suricata viene disposto su un singolo host e solo su questo deve lavorare come tale è sufficiente la sola interfaccia di rete che collega l'host alla rete per effettuare il drop o l'accept dei pacchetti secondo le regole.

47 Con qualche passaggio in più, si può fare altrettanto in ambiente Windows.
<http://sourceforge.net/projects/netfilterforwin/>

48 Si possono utilizzare altri firewall al di fuori di IPTables, come Vuurmuur
<http://www.vuurmuur.org/trac/wiki/Suricata>

coda piuttosto che su una interfaccia di rete, garantendogli così le funzionalità di *drop* e di *accept* per i pacchetti in coda.

Dirottare il traffico sulla NFQueue è possibile nel caso di test utilizzando la chain di FORWARD per la quale funzionano le regole definite nel paragrafo 4.1.2:

```
# iptables -I FORWARD -j NFQUEUE
```

Nel caso in cui si volesse disporre Suricata solo su un host si sarebbe dovuto specificare il dirottamento del traffico nella NFQueue sia per la chain INPUT che nella chain OUTPUT:

```
# iptables -I INPUT -j NFQUEUE
# iptables -I OUTPUT -j NFQUEUE
```

Dal momento in cui viene effettuato il dirottamento dei pacchetti alla coda NFQueue risulterà impossibile per gli host dietro il gateway comunicare con l'esterno fintanto che qualcosa non processi la coda. Questo qualcosa è nel nostro caso Suricata, al quale dovrà essere specificato di prelevare i pacchetti dalla coda di Netfilter piuttosto che mettersi in ascolto su una interfaccia di rete. Questo è possibile per mezzo del parametro all'avvio `-q <#nfqueue>` dove `#nfqueue` è l'identificativo della coda da processare⁴⁹.

Ecco allora che il comando per avviare Suricata cambia e diviene:

```
suricata --user suricata -D --pidfile=/var/run/suricata.pid
-c /etc/suricata/suricata.yaml -q 0
```

Se Suricata è già in esecuzione monitorando un'interfaccia, sarà obbligatorio riavviarlo:

```
kill -9 `cat /var/run/suricata.pid` && suricata --user
suricata -D --pidfile=/var/run/suricata.pid -c
/etc/suricata/suricata.yaml -q 0
```

Qualora si volesse interrompere la modalità in-line, sarà sufficiente fermare l'esecuzione di Suricata e riavviarlo in modalità di monitoring su un'interfaccia. Bisogna poi rimuovere la regola da IPTables perchè il traffico torni a fluire senza uso della coda NFQUEUE:

49 È possibile specificare più di una coda tramite IPTables. Di default la coda in uso è quella con id 0.

```
# iptables -D FORWARD -j NFQUEUE
```

O nel caso del singolo host:

```
# iptables -D INPUT -j NFQUEUE
# iptables -D OUTPUT -j NFQUEUE
```

4.3.2 Regole per il drop dei pacchetti

Le regole in uso da parte di Suricata prevedono tutte di intraprendere l'azione di *alert* nell'ipotesi di match con un pacchetto. Se però il nostro obiettivo è quello di sfruttare le capacità IPS di Suricata, andranno modificate le regole di nostro interesse cambiando l'azione da semplice *alert* a *drop*. Per fare questo o si modificano manualmente i file delle regole cercando il SID della regola che ha generato un determinato alert, o si sfrutta qualche tool esterno (paragrafo 3.5 - Rules Editor).

Ad ogni modo, una volta individuate (o scritte, se necessario) le regole che si vuole abbiano un comportamento di prevenzione, sarà sufficiente adottare l'azione di drop. Nel caso della regola presentata precedentemente:

```
alert http $EXTERNAL_NET any -> $HOME_NET any (msg:"ET SCAN
Nmap Scripting Engine User-Agent Detected (Nmap Scripting
Engine)"; flow:to_server,established; content:"|0d 0a|User-
Agent|3a| Mozilla/5.0 (compatible|3b| Nmap Scripting Engine";
nocase; reference:url,doc.emergingthreats.net/2009358;
classtype:web-application-attack; sid:2009358; rev:4;)
```

la regola diverrà

```
drop http $EXTERNAL_NET any -> $HOME_NET any (msg:"ET SCAN
Nmap Scripting Engine User-Agent Detected (Nmap Scripting
Engine)"; flow:to_server,established; content:"|0d 0a|User-
Agent|3a| Mozilla/5.0 (compatible|3b| Nmap Scripting Engine";
nocase; reference:url,doc.emergingthreats.net/2009358;
classtype:web-application-attack; sid:2009358; rev:4;)
```

4.3.3 Test di prevenzione

Una volta disposto Suricata in modalità in-line e cambiate le regole che nel paragrafo 4.2 avevano generato alert in seguito al port-scan in modo tale che

effettivo ora un drop, lo scenario è pronto per il test di prevenzione.

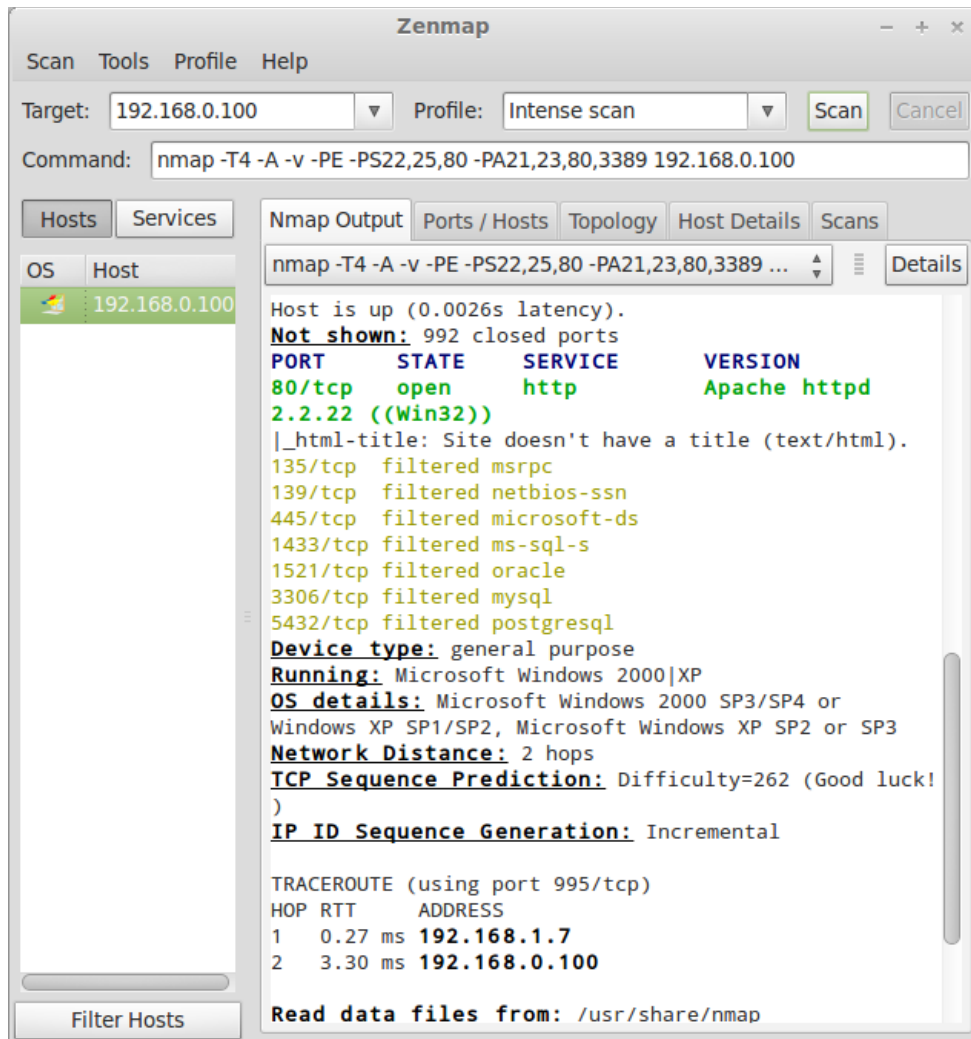


Figura 4.6: Risultato del port-scan verso la vittima con Suricata in modalità in-line

Come si può evincere dalla figura 4.6, il tentativo di port-scan verso il PC vittima è stato correttamente bloccato (a meno della porta 80 del web server, la quale deve risultare raggiungibile). I log tracciati da Suricata durante lo scan sono stati:

```
12/07/2012-17:50:04.333518 [Drop] [**] [1:2002911:4] ET SCAN
Potential VNC Scan 5900-5920 [**] [Classification: Attempted
Information Leak] [Priority: 2] {TCP} 193.205.92.130:39791 ->
192.168.0.100:5910

12/07/2012-17:50:05.336906 [Drop] [**] [1:22109360:2] ET
POLICY Suspicious inbound to NetBIOS port 139 [**]
[Classification: Potentially Bad Traffic] [Priority: 2] {TCP}
193.205.92.130:39793 -> 192.168.0.100:139
```

```
12/07/2012-17:50:05.336913 [Drop] [**] [1:2210935:2] ET
POLICY Suspicious inbound to Microsoft-DS port 445 [**]
[Classification: Potentially Bad Traffic] [Priority: 2] {TCP}
193.205.92.130:39793 -> 192.168.0.100:445

12/07/2012-17:50:05.336919 [Drop] [**] [1:22109359:2] ET
POLICY Suspicious inbound to MSRPC port 135 [**]
[Classification: Potentially Bad Traffic] [Priority: 2] {TCP}
193.205.92.130:39793 -> 192.168.0.100:135

12/07/2012-17:50:05.336924 [Drop] [**] [1:2010937:2] ET
POLICY Suspicious inbound to mySQL port 3306 [**]
[Classification: Potentially Bad Traffic] [Priority: 2] {TCP}
193.205.92.130:39793 -> 192.168.0.100:3306

12/07/2012-17:50:05.336929 [Drop] [**] [1:2010935:2] ET
POLICY Suspicious inbound to MSSQL port 1433 [**]
[Classification: Potentially Bad Traffic] [Priority: 2] {TCP}
193.205.92.130:39793 -> 192.168.0.100:1433

12/07/2012-17:50:05.336934 [Drop] [**] [1:2010939:2] ET
POLICY Suspicious inbound to PostgreSQL port 5432 [**]
[Classification: Potentially Bad Traffic] [Priority: 2] {TCP}
193.205.92.130:39793 -> 192.168.0.100:5432

12/07/2012-17:50:05.336940 [Drop] [**] [1:2010936:2] ET
POLICY Suspicious inbound to Oracle SQL port 1521 [**]
[Classification: Potentially Bad Traffic] [Priority: 2] {TCP}
193.205.92.130:39793 -> 192.168.0.100:1521

12/07/2012-17:50:12.234101 [Drop] [**] [1:2101390:6] GPL
SHELLCODE x86 inc ebx NOOP [**] [Classification: Executable
code was detected] [Priority: 1] {UDP} 193.205.92.130:39842 ->
192.168.0.100:30313

12/07/2012-17:50:12.434409 [Drop] [**] [1:2101390:6] GPL
SHELLCODE x86 inc ebx NOOP [**] [Classification: Executable
code was detected] [Priority: 1] {UDP} 193.205.92.130:39842 ->
192.168.0.100:30313

12/07/2012-17:50:13.008157 [Drop] [**] [1:2009358:4] ET SCAN
Nmap Scripting Engine User-Agent Detected (Nmap Scripting
Engine) [**] [Classification: Web Application Attack]
[Priority: 1] {TCP} 193.205.92.130:43143 -> 192.168.0.100:80

12/07/2012-17:50:13.008985 [Drop] [**] [1:2009358:4] ET SCAN
Nmap Scripting Engine User-Agent Detected (Nmap Scripting
Engine) [**]
```

Nota: quando Suricata è disposto in modalità in-line non abbandona la sua funzionalità di IDS continuando infatti a comportarsi come tale con tutte le regole che mantengono l'azione di *alert*.

4.4 Esempio aggiuntivo: Suricata come proxy

Nei precedenti paragrafi si è potuto verificare il funzionamento di Suricata quale IDPS nei confronti di un tentativo di port-scan da parte di un PC at-

taccante dall'esterno. Suricata grazie alle potenzialità del linguaggio di definizione delle regole, può essere usato per molte altre situazioni. In questo paragrafo verrà dimostrato un semplice esempio in cui sarà possibile tracciare e bloccare le consultazioni di siti contenenti una determinata parola chiave (la parola "suricata"), da parte degli host interni alla rete (come il PC vittima dello scenario in uso in questo capitolo).

La regola in questione verrà definita all'interno di un file di regole personalizzate *custom.rules*. Questo file andrà aggiunto al file *suricata.yaml* nella lista *rule-files*. La regola per generare l'alert sarà:

```
alert http $HOME_NET any -> any $HTTP_PORTS (msg:"Suricata!";
content:"suricata"; nocase; classtype:policy-violation; sid:1;
rev:1;)
```

La regola per il drop, quando disposto Suricata in modalità IPS, sarà invece la seguente:

```
drop http $HOME_NET any -> any $HTTP_PORTS (msg:"Suricata!";
content:"suricata"; nocase; classtype:policy-violation; sid:2;
rev:1;)
```

In modalità IDS, visitando il sito <http://www.treccani.it/enciclopedia/suricata/> dalla macchina host dietro al gateway che dispone di Suricata, verrà registrato nei log quanto segue:

```
12/07/2012-18:20:58.244662  [**] [1:1:1] Suricata! [**]
[Classification: Potential Corporate Privacy Violation]
[Priority: 1] {TCP} 193.205.92.135:1250 -> 151.1.175.115:80

12/07/2012-18:20:59.659404  [**] [1:1:1] Suricata! [**]
[Classification: Potential Corporate Privacy Violation]
[Priority: 1] {TCP} 193.205.92.135:1251 -> 206.165.250.90:80

12/07/2012-18:20:59.863923  [**] [1:1:1] Suricata! [**]
[Classification: Potential Corporate Privacy Violation]
[Priority: 1] {TCP} 193.205.92.135:1252 -> 206.165.250.244:80
```

Eseguito suricata in modalità IPS verrà invece bloccato il traffico da quell'indirizzo per ciò che concerne i pacchetti che includono la keyword *suricata*, generando gli alert di avvenuto drop come segue:

```
12/07/2012-18:39:47.863898  [Drop] [**] [1:2:1] Suricata!
[**] [Classification: Potential Corporate Privacy Violation]
[Priority: 1] {TCP} 192.168.0.100:1429 -> 212.31.249.104:80
```

```
12/07/2012-18:39:47.863907 [Drop] [**] [1:2:1] Suricata!  
[**] [Classification: Potential Corporate Privacy Violation]  
[Priority: 1] {TCP} 192.168.0.100:1430 -> 212.31.249.104:80  
  
12/07/2012-18:39:48.859568 [Drop] [**] [1:2:1] Suricata!  
[**] [Classification: Potential Corporate Privacy Violation]  
[Priority: 1] {TCP} 192.168.0.100:1456 -> 206.165.250.243:80
```

Inoltre sul PC verrà segnalato dal browser che la richiesta per la pagina è andata in time-out, rendendo di fatto impossibile la navigazione secondo quanto stabilito dalla regola.

Nota: la regola descritta in questo capitolo andrebbe rivista nel caso in cui si volesse tracciare anche il traffico criptato come in pagine incapsulate nel protocollo *https*.

5 Funzionalità

Di default con i pacchetti pre-costruiti di Suricata non ci sono funzionalità aggiuntive oltre a quelle standard eccezion fatta per la funzionalità di NF-Queue. Quando si compila Suricata invece (capitolo 2, Installazione ed implementazione) è possibile scegliere di quali funzionalità si vorrà disporre una volta installato il software. Per una lista completa di moduli e funzionalità aggiuntive che si possono abilitare via ricompilazione si rimanda all'appendice . Per sapere di quali funzionalità dispone la propria versione di Suricata è possibile passare `--build-info` come parametro a Suricata:

```
# suricata --build-info
```

Un possibile output sarà

```
[28158] 5/12/2012 -- 00:20:20 - (suricata.c:540) <Info>
(SCPrintBuildInfo) -- This is Suricata version 1.4dev (rev
02874a1)

[28158] 5/12/2012 -- 00:20:20 - (suricata.c:613) <Info>
(SCPrintBuildInfo) -- Features: NFQ PCAP_SET_BUFF
LIBPCAP_VERSION_MAJOR=1 AF_PACKET HAVE_PACKET_FANOUT LIBCAP_NG
LIBNET1.1 HAVE_HTTP_URI_NORMALIZE_HOOK
HAVE_HTTP_TX_GET_RESPONSE_HEADERS_RAW PCRE_JIT
```

È bene notare che di seguito verranno trattate solo le funzionalità di spicco e che OISF continua ad aggiungerne di versione in versione. Si noti anche che più funzionalità possono essere abilitate in contemporanea laddove una non vada in conflitto con l'altra.

5.1 PF_RING

PF_RING⁵⁰ è un modulo per il kernel GNU/Linux in sviluppo presso il team italiano *ntop*⁵¹. Questo modulo permette di aumentare sensibilmente la velocità di cattura dei pacchetti implementando un nuovo tipo di socket di rete. L'aumento di velocità di cattura è ottenuto in realtà cercando di diminuire il più possibile il numero di pacchetti persi quando il traffico in transito presso un'interfaccia di rete diviene insostenibile. Il vantaggio di utilizzare PF_RING è infatti apprezzabile solo se l'interfaccia di rete è sotto grosso carico.

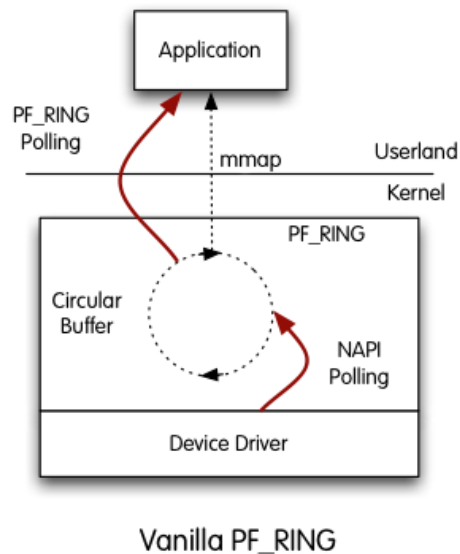


Figura 5.1: Schema di funzionamento di PF_RING

Alcuni test effettuati da *ntop*, mostrano l'aumento di banda passante per un'interfaccia di rete utilizzando PF_RING⁵².

⁵⁰ http://www.ntop.org/products/pf_ring/

⁵¹ <http://www.ntop.org/home/about-us/>

⁵² In questo paragrafo non viene trattato nello specifico il funzionamento di PF_RING, pertanto sono stati trascurati alcuni dettagli nel riproporre la tabella, quale la *Transparent Mode*.

Applicazione	Senza PF_RING	Con PF_RING
pfcount (with -a flag)	757 Kpps	843 Kpps
pfcount (without -a flag)	661 Kpps	762 Kpps
pcount (with PF_RING-aware libpcap)	730 Kpps	830 Kpps

Per ulteriori dettagli sui test o su PF_RING in generale, si rimanda alla pagina di ntop[14].

PF_RING è disponibile per kernel GNU/Linux a partire dalla versione 2.6.18, dimostrando tuttavia di essere non ancora pienamente compatibile con kernel superiori alla versione 3.2. È bene notare che questo modulo è ancora in sviluppo (ha raggiunto la versione 5.5.1 in data 24 novembre 2012) e che si tratta di un progetto completamente open source. PF_RING supporta pienamente le librerie pcap utilizzate da Suricata per la cattura.

Per abilitare PF_RING in Suricata è necessario che il modulo sia correttamente installato nel kernel del sistema operativo⁵³.

Successivamente si deve procedere con la compilazione di Suricata come già spiegato nel capitolo 2, passando come parametro di compilazione *--enable-pfring*.

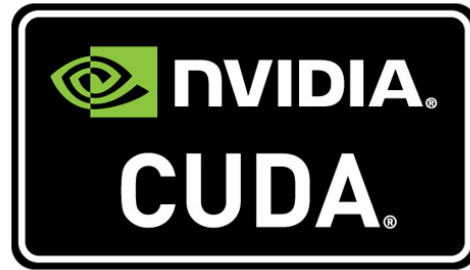
```
$ ./configure --prefix=/usr --sysconfdir=/etc --
localstatedir=/var --enable-pfring
```

Per eseguire Suricata con il modulo PF_RING basterà passargli come parametro d'avvio *--pfring-int=<interfaccia di rete>*.

```
# suricata --pfring-int=eth0
```

5.2 NVIDIA CUDA

⁵³ http://www.ntop.org/pf_ring/installation-guide-for-pf_ring/



CUDA è una architettura hardware sviluppata da NVIDIA Corporation nelle proprie schede video o GPGPU dedicate, per il calcolo parallelo ed il GPGPU⁵⁴. *CUDA* sta per *Compute Unified Device Architecture* ed è una recente tecnologia utilizzabile in tutte le attività in cui l'esecuzione in parallelo di medesime istruzioni su dati differenti (architetture SIMD), producono un notevole aumento della velocità di computazione. *CUDA* è spesso usata in ambito scientifico per la ricerca, in supercomputer e grandi cluster GPGPU⁵⁵.

Al momento l'uso di *CUDA* in Suricata è relegato al solo pattern matching fra regole e pacchetti in transito. È tuttavia nei piani di OISF continuare a sfruttare la tecnologia del GPGPU il più possibile per ottenere migliori prestazioni, integrandola il più possibile nell'engine di Suricata. L'uso della tecnologia *CUDA* al momento, dunque, non offre grandi vantaggi[15][16] e come rovescio della medaglia la tecnologia è proprietaria NVIDIA ed implementata solo sui suoi prodotti. Un'alternativa a *CUDA* è data da *OpenCL*, librerie open e supportate anche da schede non NVIDIA. Poiché il team OISF ha inizialmente trovato problemi nello sfruttare tali librerie, ha optato per *CUDA*, ma è loro intenzione tornare sulle *OpenCL* quando possibile.[17]

Per abilitare Suricata all'utilizzo di *CUDA* è necessario che sulla propria macchina siano correttamente installati gli appositi driver. Successivamente basterà compilare Suricata specificando il parametro `--enable-cuda` in fase di

54 General-Purpose Graphics Processing Unit. <http://it.wikipedia.org/wiki/GPGPU>

55 http://www.nvidia.it/object/cuda_clusters_it.html

configurazione del make.

```
$ ./configure --prefix=/usr --sysconfdir=/etc --  
localstatedir=/var --enable-cuda
```

5.3 LUA



A partire dalla versione 1.4beta1, è stato reso possibile dal team di OISF, l'uso di script in *linguaggio LUA*⁵⁶; tale funzionalità non ha ancora raggiunto una fase stabile e può mutare nel tempo o essere rimossa, pertanto è sconsigliabile farne pieno affidamento per le regole più delicate.

LUA è un linguaggio di scripting e permette di aggiungere maggiore espressività alle regole per Suricata sia in congiunzione con classici pre-filtri ed espressioni regolari *pcr* sia usato singolarmente. Per abilitare Suricata all'uso di LUA è necessario installare nel proprio sistema le librerie *liblua*. Una volta installate le librerie da repository o manualmente da pacchetto, è possibile procedere con la configurazione del file di make. Dalla root directory dei sorgenti di Suricata, impartire:

```
# ./configure --enable-luajit
```

Il codice eseguibile generato dalla successiva compilazione sarà una versione di Suricata con il supporto a regole con riferimenti a script LUA abilitato.

In generale, uno script invocato da una regola, deve ritornare un valore booleano che affermi il successo del match della regola con il pacchetto o meno. Per invocare lo script, nella definizione delle opzioni si deve far ricorso alla parola chiave *luajit* come nel seguente esempio:

```
alert tcp any any -> any any (msg:"LUAJIT test";  
luajit:test.lua; sid:1;)
```

Qualora un pacchetto dovesse fare match con l'header della regola e con le al-

⁵⁶ <http://www.lua.org/>

tre opzioni eventualmente specificate, il successivo controllo di matching passerebbe allo script *test.lua*.

Lo script LUA deve contenere al suo interno due funzioni: la *function init(args)* e la *function match(args)*.

L'uso di LUA permette di estendere le capacità di Suricata senza incidere molto nelle prestazioni⁵⁷. È stato addirittura provato che le regole in LUA possono essere più veloci della loro controparte con l'uso dei prefilter.[18]

Sono stati effettuati dei test con il seguente set di regole:

```
# Regola con LUA e prefilter content.
alert http any any -> any any (msg:"LUAJIT HTTP POST test,
pure lua"; luajit:test.lua; sid:1;)

#Regola con LUA e con prefilter content.
alert http any any -> any any (msg:"LUAJIT HTTP POST test,
content prefilter"; content:"POST"; http_method;
content:".php"; http_uri; luajit:test.lua; sid:2;)

#Regola con LUA e con prefilter content ed uso di pcre.
alert http any any -> any any (msg:"LUAJIT HTTP POST test,
pcre prefilter"; content:"POST"; http_method; content:".php";
http_uri; pcre:"/^POST\s+\./.*\.php\s+HTTP\/1\.0\r\n/m";
luajit:test.lua; sid:3;)

#Regola con pcre e prefilter content (no LUA).
alert http any any -> any any (msg:"LUAJIT HTTP POST test,
pcre no lua"; content:"POST"; http_method; content:".php";
http_uri; pcre:"/^POST\s+\./.*\.php\s+HTTP\/1\.0\r\n/m"; sid:4;)

#Regola con solo pcre (no LUA).
alert http any any -> any any (msg:"LUAJIT HTTP POST test,
pure pcre"; pcre:"/^POST\s+\./.*\.php\s+HTTP\/1\.0\r\n/m";
sid:5;)
```

⁵⁷ Regole troppo complesse, richiedono molti cicli di CPU per essere processate, andando così a gravare sul bandwidth di pacchetti esaminati ed incidendo sulla percentuale di pacchetti persi.

ed il seguente script di test:

```
function init (args)
    local needs = {}
    needs["http.request_line"] = tostring(true)
    return needs
end

-- match if packet and payload both contain HTTP
function match(args)
    a = tostring(args["http.request_line"])
    if #a > 0 then
        if a:find("^POST%s+/.*%.php%s+HTTP/1.0$") then
            return 1
        end
    end
end

return 0
end

return 0
```

Segue una tabella riassuntiva dei risultati ottenuti:

Regola	Media dei Tick	Ticks medi in caso di match	Ticks medi in caso di non match
LUA	8812.31	15841.85	8807.01
LUA + content prefilter	10682.71	35497.08	10194.56
LUA + content prefilter + pcre	11638.15	39842.23	9424.83
Content prefilter + pcre	8536.46	20074.97	7630.97
pcre	12113.53	7198.08	12114.28

La regola che fa ricorso allo script LUA senza uso dei pre-filtri risulta essere più veloce della regola che fa ricorso ai pre-filtri. Tale regola viene superata in velocità solo dal caso di content prefilter + pcre.

L'uso di LUA nella scrittura di regole sembra essere molto promettente, e potrebbe divenire un punto di forza di Suricata IDPS.

Conclusioni

Alla luce di quanto trattato nel corso dei precedenti capitoli, Suricata si dimostra un progetto decisamente interessante in ambito di sicurezza delle reti. Nel panorama dei progetti di Intrusion Detection and Prevention System open source introduce delle novità degne di nota come il supporto al multithreading, o all'utilizzo di script in linguaggio LUA per la definizione di regole o, ancora, all'uso del GPGPU per mezzo di CUDA ed il monitoring velocizzato per mezzo di PF_RING. Laddove non innova, mantiene una certa compatibilità con il percorso già intrapreso da altri progetti come Snort, rendendo possibile una buona interoperabilità fra i due sistemi. Ad ogni modo, Suricata è ancora un progetto in fase di pieno sviluppo e testing, lungi dall'essere considerabile come completo ed insostituibile.

Diversi strumenti o interfacce sono compatibili con Suricata, tuttavia alcuni di questi solo parzialmente. Non sono ancora presenti strumenti appositamente creati per Suricata, come i rule editor, o la gestione intelligente delle regole in modo tale che il passaggio da detection a prevention risulti semplificato.

Per quanto il software sia concepito come multiplatforma, nella realtà, complici il focus di sviluppo da parte di OISF per sistemi GNU/Linux, e lo sviluppo di tool esterni da parte della community in tale ambiente, il supporto ad altri sistemi è piuttosto ridotto rispetto alla controparte del sistema del pinguino.

Questi sono dunque i punti sui quali bisognerebbe lavorare in futuro perché Suricata possa affermarsi come punto di riferimento nel panorama IDPS: tools di gestione e consultazione ad-hoc per Suricata e maggiore supporto e facilità d'uso in tutte le piattaforme da esso supportate. Poiché il progetto e ciò che vi ruota intorno è spesso software open source è pensabile che chiunque possa sviluppare e contribuire nei punti evidenziati.

I due casi di test che sono stati trattati sono relativamente semplici in

un'ottica di sicurezza di rete, per esempio, aziendale ma comunque esemplificativi. Le potenzialità dei sistemi IDPS sono enormi e Suricata, per quanto uno dei più giovani, non è da meno⁵⁸.

In definitiva, Suricata si presenta come un software molto *interessante e promettente* e che cerca di conseguire il suo scopo di innovare nel settore della sicurezza delle reti, versione per versione, nel rispetto della definizione da parte dello stesso team OISF di **next generation IDS/IPS engine**.

⁵⁸ Consultare ad esempio <https://home.regit.org/2012/10/defend-your-network-from-word/>

Appendice

A- Parametri di compilazione

Posizionandosi nella root directory dei sorgenti di Suricata ed impartendo il comando:

```
./configure --help
```

è possibile consultare una lista di parametri per configurare il file di make di Suricata. Segue l'output fornito da Suricata in versione 1.4rc1.

```
`configure' configures this package to adapt to many kinds of
systems.

Usage: ./configure [OPTION]... [VAR=VALUE]...

To assign environment variables (e.g., CC, CFLAGS...),
specify them as

VAR=VALUE. See below for descriptions of some of the useful
variables.

Defaults for the options are specified in brackets.

Configuration:
  -h, --help                display this help and exit
  --help=short              display options specific to this
package
  --help=recursive          display the short help of all the
included packages
  -V, --version             display version information and
exit
  -q, --quiet, --silent    do not print `checking ...'
messages
  --cache-file=FILE        cache test results in FILE
[disabled]
  -C, --config-cache        alias for `--cache-
file=config.cache'
  -n, --no-create           do not create output files
  --srcdir=DIR              find the sources in DIR [configure
dir or `..']
```

Installation directories:

```

--prefix=PREFIX      install architecture-independent
files in PREFIX

                        [/usr/local]

--exec-prefix=EPREFIX install architecture-dependent
files in EPREFIX

                        [PREFIX]

```

By default, ``make install'` will install all the files in ``/usr/local/bin'`, ``/usr/local/lib'` etc. You can specify an installation prefix other than ``/usr/local'` using ``--prefix'`, for instance ``--prefix=$HOME'`.

For better control, use the options below.

Fine tuning of the installation directories:

```

--bindir=DIR          user executables [EPREFIX/bin]
--sbindir=DIR         system admin executables
[EPREFIX/sbin]
--libexecdir=DIR     program executables
[EPREFIX/libexec]
--sysconfdir=DIR     read-only single-machine data
[PREFIX/etc]
--sharedstatedir=DIR modifiable architecture-independent
data [PREFIX/com]
--localstatedir=DIR  modifiable single-machine data
[PREFIX/var]
--libdir=DIR          object code libraries [EPREFIX/lib]
--includedir=DIR     C header files [PREFIX/include]
--oldincludedir=DIR  C header files for non-gcc
[/usr/include]
--datarootdir=DIR    read-only arch.-independent data
root [PREFIX/share]
--datadir=DIR        read-only architecture-independent
data [DATAROOTDIR]
--infodir=DIR        info documentation
[DATAROOTDIR/info]
--localedir=DIR      locale-dependent data
[DATAROOTDIR/locale]
--mandir=DIR         man documentation [DATAROOTDIR/man]
--docdir=DIR         documentation root
[DATAROOTDIR/doc/PACKAGE]

```

```

--htmldir=DIR          html documentation [DOCDIR]
--dvidir=DIR           dvi documentation [DOCDIR]
--pdfdir=DIR           pdf documentation [DOCDIR]
--psdir=DIR            ps documentation [DOCDIR]

Program names:
  --program-prefix=PREFIX      prepend PREFIX to
installed program names
  --program-suffix=SUFFIX      append SUFFIX to
installed program names
  --program-transform-name=PROGRAM  run sed PROGRAM on
installed program names

System types:
  --build=BUILD            configure for building on BUILD [guessed]
  --host=HOST              cross-compile to build programs to run on
HOST [BUILD]

Optional Features:
  --disable-option-checking  ignore unrecognized --enable/--
with options
  --disable-FEATURE          do not include FEATURE (same as
--enable-FEATURE=no)
  --enable-FEATURE[=ARG]    include FEATURE [ARG=yes]
  --disable-dependency-tracking  speeds up one-time build
  --enable-dependency-tracking  do not reject slow
dependency extractors
  --enable-shared[=PKGS]    build shared libraries
[default=yes]
  --enable-static[=PKGS]    build static libraries
[default=yes]
  --enable-fast-install[=PKGS]
                                optimize for fast installation
[default=yes]
  --disable-libtool-lock    avoid locking (might break parallel
builds)
  --disable-largefile       omit support for large files
  --enable-gccprotect        Detect and use gcc hardening
options
  --enable-gccprofile        Enable gcc profile info i.e -pg
flag is set
  --enable-gccmarch-native
                                Enable gcc march=native gcc 4.2 and

```

```

later only
    --enable-uitests      Enable compilation of the unit
tests
    --enable-old-barnyard2 Use workaround for old barnyard2 in
unified2 output
    --enable-debug        Enable debug output
    --enable-debug-validation
                                Enable (debug) validation code
output
    --enable-profiling    Enable performance profiling
    --enable-profiling-locks
                                Enable performance profiling for
locks
    --enable-ipfw         Enable FreeBSD IPFW support for
inline IDP
    --enable-nfqueue      Enable NFQUEUE support for inline
IDP
    --enable-prelude      Enable Prelude support for alerts
    --enable-pfring       Enable Native PF_RING support
    --enable-af-packet    Enable AF_PACKET support
[default=yes]
    --enable-non-bundled-htp
                                Enable the use of an already
installed version of
                                htp
    --enable-cuda         Enable experimental CUDA pattern
matching
    --enable-dag          Enable DAG capture
    --enable-napatech     Enabled Napatech Devices
    --enable-luajit       Enable Luajit support

Optional Packages:
    --with-PACKAGE[=ARG]  use PACKAGE [ARG=yes]
    --without-PACKAGE     do not use PACKAGE (same as --with-
PACKAGE=no)
    --with-pic[=PKGS]    try to use only PIC/non-PIC objects
[default=use
                                both]
    --with-gnu-ld         assume the C compiler uses GNU ld
[default=no]
    --with-sysroot=DIR   Search for dependent libraries within
DIR
                                (or the compiler's sysroot if not

```

```

specified).
    --with-libpcre-includes=DIR  libpcre include directory
    --with-libpcre-libraries=DIR  libpcre library directory
    --with-libyaml-includes=DIR  libyaml include directory
    --with-libyaml-libraries=DIR  libyaml library directory
    --with-libpthread-includes=DIR  libpthread include
directory
    --with-libpthread-libraries=DIR  libpthread library
directory
    --with-libjansson-includes=DIR  libjansson include
directory
    --with-libjansson-libraries=DIR  libjansson library
directory
    --with-libnfnetworklink-includes=DIR  libnfnetworklink include
directory
    --with-libnfnetworklink-libraries=DIR  libnfnetworklink library
directory
    --with-libnetfilter_queue-includes=DIR  libnetfilter_queue
include directory
    --with-libnetfilter_queue-libraries=DIR
libnetfilter_queue library directory
    --with-netfilterforwin-includes=DIR  netfilterforwin
include directory
    --with-libprelude-prefix=PREFIX
                                Prefix where libprelude is
installed (optional)
    --with-libnet-includes=DIR  libnet include directory
    --with-libnet-libraries=DIR  libnet library directory
    --with-libpfring-includes=DIR  libpfring include directory
    --with-libpfring-libraries=DIR  libpfring library
directory
    --with-libpcap-includes=DIR  libpcap include directory
    --with-libpcap-libraries=DIR  libpcap library directory
    --with-libhttp-includes=DIR  libhttp include directory
    --with-libhttp-libraries=DIR  libhttp library directory
    --with-cuda-includes=DIR  cuda include directory
    --with-cuda-libraries=DIR  cuda library directory
    --with-cuda-nvcc=DIR  cuda nvcc compiler directory
    --with-libcap_ng-includes=DIR  libcap_ng include directory
    --with-libcap_ng-libraries=DIR  libcap_ng library
directory
    --with-dag-includes=DIR  dagapi include directory

```

```

--with-dag-libraries=DIR  dagapi library directory
--with-libnspr-includes=DIR  libnspr include directory
--with-libnspr-libraries=DIR  libnspr library directory
--with-libnss-includes=DIR  libnss include directory
--with-libnss-libraries=DIR  libnss library directory
--with-libmagic-includes=DIR  libmagic include directory
--with-libmagic-libraries=DIR  libmagic library directory
--with-napatech-includes=DIR  napatech include directory
--with-napatech-libraries=DIR  napatech library directory
--with-libluajit-includes=DIR  libluajit include directory
--with-libluajit-libraries=DIR  libluajit library
directory

```

Some influential environment variables:

```

CC          C compiler command
CFLAGS      C compiler flags
LDFLAGS     linker flags, e.g. -L<lib dir> if you have
libraries in a
            nonstandard directory <lib dir>
LIBS        libraries to pass to the linker, e.g.
-l<library>
CPPFLAGS    (Objective) C/C++ preprocessor flags, e.g.
-I<include dir> if
            you have headers in a nonstandard directory
<include dir>
CPP         C preprocessor
PKG_CONFIG  path to pkg-config utility
PKG_CONFIG_PATH
            directories to add to pkg-config's search path
PKG_CONFIG_LIBDIR
path
            path overriding pkg-config's built-in search
LIBHTPMINVERSION_CFLAGS
            C compiler flags for LIBHTPMINVERSION,
overriding pkg-config
LIBHTPMINVERSION_LIBS
            linker flags for LIBHTPMINVERSION, overriding
pkg-config
PYTHON      the Python interpreter
LUAJIT_CFLAGS
            C compiler flags for LUAJIT, overriding pkg-

```

```

config
    LUAJIT_LIBS linker flags for LUAJIT, overriding pkg-config

    Use these variables to override the choices made by
    `configure' or to help
    it to find libraries and programs with nonstandard
    names/locations.

    Report bugs to the package provider.

```

B- Lista di parametri per l'avvio di Suricata

```

USAGE: suricata
    -c <path>                : path to configuration file
    -T                        : test configuration file
(use with -c)
    -i <dev or ip>           : run in pcap live mode
    -F <bpf filter file>     : bpf filter file
mode
    -r <path>                : run in pcap file/offline
mode
    -q <qid>                 : run in inline nfqueue mode
    -s <path>                : path to signature file
loaded in addition to suricata.yaml settings (optional)
    -S <path>                : path to signature file
loaded exclusively (optional)
    -l <dir>                 : default log directory
    -D                        : run as daemon
    --list-keywords          : list all keywords
implemented by the engine
    --list-runmodes          : list supported runmodes
    --runmode <runmode_id>  : specific runmode
modification the engine should run. The argument
supplied should be the id
for the runmode obtained by running
    --list-runmodes
    --engine-analysis        : print reports on analysis
of different sections in the engine and exit.
Please have a look at the
conf parameter engine-analysis on what reports
can be printed
    --pidfile <file>        : write pid to this file
(only for daemon mode)

```

```
--init-errors-fatal      : enable fatal failure on
signature init error
--dump-config            : show the running
configuration
--pcap[=<dev>]          : run in pcap mode, no value
select interfaces from suricata.yaml
--pcap-buffer-size      : size of the pcap buffer
value from 0 - 2147483647
--af-packet[=<dev>]     : run in af-packet mode, no
value select interfaces from suricata.yaml
--user <user>           : run suricata as this user
after init
--group <group>         : run suricata as this group
after init
--erf-in <path>         : process an ERF file
```

To run the engine with default configuration on interface eth0 with signature file "signatures.rules", run the command as:

```
suricata -c suricata.yaml -s signatures.rules -i eth0
```

Bibliografia

- [1] http://en.wikipedia.org/wiki/Intrusion_detection_system
- [2] Karen Scarfone, Peter Mell, Guide to Intrusion Detection and Prevention Systems (IDPS) (Draft), 2012, <http://csrc.nist.gov/publications/nistpubs/800-94/SP800-94.pdf>
- [3] http://wiki.aanval.com/wiki/Snort_vs_Suricata
- [4] <http://www.aldeid.com/wiki/Suricata-vs-snort>
- [5] http://www.thinkmind.org/download.php?articleid=icds_2011_7_40_90007
- [6] <https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Suricatayaml>
- [7] https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Suricata_Rules
- [8] https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Rule_Management_with_Oinkmaster
- [9] <http://www.securixlive.com/barnyard2/docs/unified.php>
- [10] <http://www.ietf.org/rfc/rfc4765.txt>
- [11] http://www.snort.org/assets/187/Snort_Frontend_Compare.pdf
- [12] <http://sguil.sourceforge.net/>
- [13] <http://www.squertproject.org/>
- [14] http://www.ntop.org/products/pf_ring/
- [15] http://2011.rml.info/IMG/pdf/2011_rml_suricata.pdf
- [16] <https://home.regit.org/2010/05/using-suricata-with-cuda/>
- [17] <http://blog.inliniac.net/2010/02/20/suricata-has-experimental-cuda-support/>
- [18] <http://blog.inliniac.net/2012/09/08/first-impressions-of-luajit-performance-in-suricata/>

Ryan Russell; James C. Foster, Jeffrey Posluns; Snort 2.0 Intrusion Detection; 2003; Syngress Media Inc