ALMA MATER STUDIORUM – UNIVERSITÁ DI BOLOGNA
CAMPUS DI CESENA
SCUOLA DI SCIENZE

CORSO DI LAUREA IN INGEGNERIA E SCIENZE INFORMATICHE

Implementazione di metodologie di rilevamento delle intrusioni con il sistema Bro

Tesi in
Reti di Telecomunicazione

Relatore
Prof. Franco Callegati

Presentata da
Francesco Antimi

Correlatore
Ing. Daniele Bellavista

Sessione Prima
Anno Accademico 2016-2017

Abstract

Il numero di cyber attacchi, che ogni anno colpiscono non solamente le aziende ma più in generale chiunque possa collegarsi ad Internet, è in costante aumento. Da qui la necessità sempre più urgente di dotarsi di sistemi di sicurezza adeguati per proteggere i propri dati. Da qui gli ingenti capitali che vengono investiti annualmente nel settore della cybersecurity dalle grandi organizzazioni.

Gli strumenti di difesa più utilizzati (firewall, antivirus e IDS) presentano tuttavia grossi limiti e possono essere superati con relativa facilità.

Negli ultimi anni stanno perciò prendendo piede nuovi approcci basati sulla threat intelligence che tramite la condivisione di informazioni su minacce note cercano di rilevare e prevenire gli attacchi prima che questi possano colpire.

Generalmente si tratta di metodi piuttosto semplici che si limitano a segnalare che un host si è collegato ad un certo IP, ha risolto un determinato DNS, ha scaricato un file con uno specifico hash e così via, metodi che nella pratica non risultano di poi così grande utilità né per le aziende né per i privati.

Il progetto realizzato per questa tesi si pone l'obiettivo di estendere l'approccio tradizionale, cercando di proporre nuovi modi di utilizzare le informazioni di intelligence e introducendo il concetto di relazione tra indicatori, lasciando la porta aperta a tanti altri possibili sviluppi.

Indice

1 Introduzione

Negli ultimi anni la quantità di traffico malevolo su Internet è aumentata in maniera esponenziale[1][2] e questo sta destando una sempre maggiore preoccupazione in tutte le principali aziende ed organizzazioni del settore.

I cyber attacchi che vengono loro rivolti possono infatti provocare il danneggiamento dei computer e dei vari dispositivi connessi alla rete nonché la perdita di informazioni sensibili di clienti e dipendenti, causando quindi gravi danni alla compagnia.

Ormai non ci si trova più davanti a dei semplici teenager che quasi per gioco si divertono a creare e diffondere malware, ma a veri e propri cyber criminali sempre più organizzati ed attrezzati che conducono azioni mirate volte ad impossessarsi di dati personali quali password o conti bancari, da utilizzare poi per i propri scopi[3].

Ecco quindi che nell'ultimo periodo il tema della **cybersecurity** ha acquisito sempre più importanza, focalizzando l'attenzione dell'opinione pubblica con centinaia di notizie e articoli pubblicati che cercano di stabilire quali siano le tecnologie più efficaci per proteggersi da accessi indesiderati.

Inoltre in questo complesso scenario ognuno, sia privati che organizzazioni che si occupano di sicurezza (Emerging Threats, Shadow Server ecc.), può contribuire in prima persona condividendo con il popolo di Internet informazioni riguardanti eventuali attacchi ricevuti come domini infetti, IP, URL ecc. andando così a costituire una vastissima collezione di dati che risultano di fondamentale importanza per il riconoscimento e la prevenzione di futuri attacchi (si tratta della cosiddetta **Open Source Intelligence**).

1 Introduzione

Con così tante minacce da cui stare in guardia, le aziende necessitano di sempre maggiori garanzie di sicurezza nonché di strumenti che impediscano ai propri sistemi di venire compromessi, strumenti per cui ogni anno vengono spesi miliardi e miliardi di dollari[4].

Di per sé i sistemi di difesa standard messi a disposizione da qualsiasi azienda, ovvero **firewall** e **antivirus**, conferiscono un certo livello di protezione ma la domanda da porsi è se questo sia sempre sufficiente. Gli hacker infatti sono in grado di superare con relativa facilità un firewall e i computer possono essere infettati ancor prima che il programma antivirus lo rilevi.

Le aziende fanno perciò spesso ricorso a software e hardware appositi che sono in grado di monitorare il traffico della rete alla ricerca di attività malevole per poi interrompere il flusso di dati prima che questo possa arrecare dei danni, segnalando la presenza di attività illecite o sospette.

Questi sistemi, che negli ultimi anni stanno prendendo sempre più piede, sono gli **Intrusion Detection System** o **IDS.**

Il mercato degli IDS al giorno d'oggi è estremamente vasto e offre diverse soluzioni, la maggior parte delle quali a pagamento e spesso a costi molto elevati. Accanto ai prodotti commerciali più noti, si è però sviluppata negli ultimi anni anche una comunità molto grande e in continua crescita di **IDS open source**, che offrono a chi sceglie di utilizzarli numerosi vantaggi come un'elevata estensibilità e possibilità pressoché sconfinate di configurazione.

Questa tesi si concentra proprio sullo studio di uno di essi, **Bro**, che si differenzia da tutti gli altri per la presenza di un suo linguaggio di scripting, **il Bro scripting language,** che lo rende estremamente flessibile e adattabile alle più diverse esigenze.

Il Bro scripting language è lo strumento che si è scelto di utilizzare per la

realizzazione del progetto che dà il nome alla tesi e che prevede la creazione di un plugin per Bro per l'applicazione di regole di **Threat Intelligence**, sfruttando appieno tutta la potenza espressiva del linguaggio.

Tale progetto è stato realizzato con la collaborazione e il supporto dell'Ing. Daniele Bellavista presso l'azienda Yoroi, startup innovativa con sede operativa a Cesena che si occupa di sicurezza informatica a vari livelli.

La struttura e il funzionamento del plugin realizzato saranno analizzati nel capitolo 5. I precedenti capitoli sono invece dedicati ad una introduzione alla cybersecurity (capitolo 2), ad uno studio approfondito di Bro (capitolo 3) e al mondo della Threat Intelligence(capitolo 4).

2 La cybersecurity

2.1 I cyber attacchi

I sistemi informatici possono essere attaccati e compromessi in molti modi diversi: hacking, phishing, spam, crimeware, virus, worm e così via. Per fare questo i cyber criminali vanno in cerca di vulnerabilità del sistema già note e le utilizzano poi per accedervi, senza che, nella maggior parte dei casi, gli utenti del sistema neppure se ne accorgano.

Le principali minacce alla sicurezza di un sistema informatico possono essere sostanzialmente classificate in quattro tipologie: **intercettazione**, **interruzione**, **modifica** e **fabbricazione**[5]. Più nello specifico:

- **Intercettazione:** si ha quando una persona o un programma non autorizzati riescono ad avere accesso ad una risorsa del sistema.
  Esempi di attività di intercettazione possono essere la copia illegale dei file di un software o azioni di spionaggio.
- **Interruzione:** quando una risorsa risulta inaccessibile o inutilizzabile si verifica l'interruzione di un servizio.
  Ciò può essere causato dalla distruzione di un dispositivo hardware o dalla cancellazione di un programma o di un file dati ad opera di un attaccante dalle intenzioni malevole.
- **Modifica:** estensione dell'intercettazione, si verifica quando una persona o un programma non autorizzati ottengono l'accesso ad una risorsa del sistema e la manomettono.
  Ciò avviene ad esempio alterando un valore in un database,

modificando il funzionamento di un programma oppure i dati che vengono trasmessi in rete.

- **Fabbricazione:** si parla di fabbricazione quando qualcuno di non autorizzato crea ad hoc una risorsa contraffatta che somiglia ad altre risorse esistenti. L'intruso può ad esempio inserire transazioni sospette all'interno di una comunicazione in rete o aggiungere nuovi record in un database. Queste aggiunte a volte possono essere riconosciute come dei falsi, ma se realizzate da un hacker esperto risultano quasi impossibili da distinguere dalle loro controparti reali.

Per portare a termine una qualsiasi di queste azioni e realizzare così un attacco ad un sistema, l'attaccante deve avere un **metodo** ovvero un piano d'azione, l'**opportunità** di metterlo in pratica e un **motivo** per farlo[5]. In altre parole esso deve avere le conoscenze, le competenze e gli strumenti per portare a termine l'attacco in maniera efficace; il tempo e la possibilità effettiva di eseguire l'attacco; una ragione per voler eseguire questo attacco contro un determinato sistema. Se anche solo una di queste tre caratteristiche viene a mancare, l'attaccante non sarà in grado di eseguire l'attacco. Tuttavia intervenire su di esse allo scopo di rimuoverle non è affatto un compito semplice.

In particolare risulta spesso difficile capire il motivo che spinge un attaccante a colpire uno specifico sistema. Molti di questi vengono attaccati semplicemente in quanto reputati bersagli interessanti dagli attaccanti[5].

I bersagli più popolari per lanciare un attacco includono, ad esempio, i sistemi informatici delle forze dell'ordine o del Ministero della Difesa, poiché considerati ben protetti dagli attacchi(e di conseguenza un attacco portato a termine con successo mostrerebbe al mondo intero le capacità

dell'attaccante). Altri sistemi invece vengono attaccati per il motivo opposto, ovvero perché è possibile accedervi con relativa facilità, come nel caso delle università le cui falle nella sicurezza possono essere sfruttate per espandere una rete criminale.
O ancora, un sistema può essere scelto come bersaglio di un attacco in maniera del tutto casuale diventando così vittima ignara e indifesa dell'attaccante.
In pratica chiunque e in qualsiasi momento può diventare vittima di un cyber attacco, magari senza neanche rendersene conto.
Risulta quindi evidente come di fronte a questa grandissima varietà di scenari proteggersi da un attacco sia un compito estremamente difficile da mettere in pratica.

2.1.1 I Malware

Una menzione particolare spetta ai **malware** che ad oggi rappresentano sicuramente la minaccia più diffusa e pericolosa, non solamente per le aziende ma per chiunque disponga di una connessione a Internet.
Il termine malware, abbreviazione per *malicious software* ovvero software malevolo, si riferisce a qualsiasi programma progettato per danneggiare un sistema informatico o fargli effettuare azioni indesiderate senza il consenso del suo proprietario[6].
Esempi piuttosto comuni di malware comprendono virus, worm, trojan e spyware. I virus ad esempio, una volta eseguiti all'interno di un sistema, sono in grado di replicarsi, infettando eseguibili e librerie in modo che software leciti eseguano codice malevolo ad ogni esecuzione. Gli spyware sono invece in grado di raccogliere di nascosto informazioni di vario tipo, sia commerciali che private, dal sistema di un utente senza che quest'ultimo

lo sappia. E ciò può includere qualsiasi cosa, dalle pagine Web che l'utente visita alle sue informazioni personali, come numeri di carte di credito o password.

Un malware può assumere diverse forme(codice eseguibile, script PHP su una pagina Web, allegato di una mail o altre tipologie di software) e si diffonde principalmente inserendosi all'interno del sistema in maniera celata, cercando di evaderne i sistemi di sicurezza per eseguire azioni malevole.

La diffusione dei malware risulta in continuo aumento: si calcola che nel solo anno 2008 su Internet siano girati circa 15 milioni di malware, di cui quelli circolati tra i mesi di gennaio e agosto sono pari alla somma dei 17 anni precedenti; e tali numeri negli anni successivi non hanno fatto altro che aumentare ancora di più, con l'espansione della Rete e il progressivo diffondersi della cultura informatica[7].

Di grande rilievo a questo proposito è stata senza dubbio l'introduzione degli *Exploit Kit*, framework in grado di analizzare i sistemi alla ricerca di vulnerabilità da poter sfruttare e quindi di generare codice malevolo semplicemente con un paio di click. Questo ha infatti permesso a tantissime persone, anche senza alcuna abilità in ambito informatico, di avere a disposizione i mezzi per poter guadagnare dal cyber crimine in maniera straordinariamente semplice.

In uno scenario di questo tipo in continua e costante evoluzione, acquista dunque ancora più importanza il dotarsi di misure di sicurezza adeguate, volte non solamente ad attuare contromisure per limitare i danni subiti una volta che l'attacco è già andato in porto, quanto piuttosto a rilevare ed identificare in tempo reale le minacce future ad un sistema ed eventualmente prevenirle.

2.2 Sistemi di sicurezza

L'obiettivo principale della sicurezza informatica è quello di prevenire il furto di informazioni, la corruzione dei dati e l'interruzione dei servizi che un sistema offre, facendo però sì che, nonostante tutto, questi siano sempre accessibili agli utenti autorizzati[8].

In generale la salvaguardia delle risorse di un sistema è ottenuta attraverso l'azione combinata di tutta una serie di software, hardware, processi e meccanismi, sia di prevenzione che di protezione, tesi ad assicurare il rispetto di tre proprietà fondamentali[8]:

- **riservatezza**, ovvero un accesso protetto e controllato ai dati in modo tale che solo chi ha ottenuto un'apposita autorizzazione possa accedere ad una risorsa;
- **integrità**, ossia la consistenza dei dati, intesa come correttezza e completezza degli stessi;
- **disponibilità**, cioè la garanzia che sia sempre possibile accedere ai dati nei tempi e nei luoghi previsti.

Le proprietà di integrità, riservatezza e disponibilità dei dati costituiscono l'assunto base sul quale vengono svolte tutte le successive valutazioni di sicurezza, e una delle sfide più difficili quando ci si trova a creare o a scegliere un sistema di sicurezza è proprio quella di trovare il giusto equilibrio tra di esse, in quanto spesso si trovano in conflitto.

Per esempio è facile preservare la riservatezza di un oggetto semplicemente impedendo a tutti di leggerlo. Tuttavia così facendo il file risulterà inaccessibile anche a coloro che invece dovrebbero avervi accesso, venendo quindi a mancare il requisito della disponibilità. Un giusto equilibrio tra la riservatezza e la disponibilità di un risorsa è ciò che si deve cercare di

raggiungere di volta in volta.

Attualmente esistono numerose tecnologie e approcci differenti che è possibile utilizzare per proteggere un sistema e impedire che venga compromesso. Firewall, antivirus, antispyware, Intrusion Detection System, l'applicazione di specifiche politiche di sicurezza, di autenticazione o di controllo degli accessi, la crittografia sono tutti strumenti che impediscono o cercano di impedire ai sistemi informatici di entrare in contatto con malware o traffico malevolo.

Nei paragrafi successivi l'attenzione viene focalizzata su **firewall**, **antivirus** e **IDS**: sono infatti questi quelli che riguardano più da vicino il progetto realizzato per questa tesi.

2.3 I Firewall

Il firewall rappresenta la prima linea di difesa per gli host connessi ad Internet. Si tratta infatti di un dispositivo software, hardware oppure ibrido che si occupa di ispezionare tutti i pacchetti che entrano o escono dalla rete interna, la LAN, e poi a seconda del loro contenuto decide se accettare o meno il pacchetto sulla base di un insieme di regole predefinite che definiscono una policy di sicurezza[9][10].

La politica standard adottata da tutti i firewall è la cosiddetta *default-deny* che consiste nel vietare a priori tutto il traffico, andando a stabilire un insieme di regole per determinare quale sono gli unici tipi di pacchetti che possono essere accettati e fatti passare attraverso il firewall. Se un messaggio non corrisponde a nessuna delle regole prestabilite, verrà bloccato dal firewall.

Attualmente esistono diversi tipi di tecniche che un firewall può utilizzare

per bloccare il traffico indesiderato quali ad esempio il *packet filtering*, l'uso di un server proxy, l'*application-level gateway* o il *NAT.*

Molti di questi metodi filtrano i contenuti analizzando singolarmente ogni pacchetto che attraversa il firewall, senza tenere conto dei precedenti e considerando solamente alcune delle informazioni contenute nell'header(l'indirizzo IP della sorgente, l'indirizzo IP della destinazione, la porta della sorgente, la porta della destinazione e il protocollo di trasporto).

I firewall che costruiscono le proprie policy di sicurezza basandosi su questo tipo di filtraggio dei pacchetti semplice e leggero vengono definiti *packet filter firewall* o *stateless firewall.*

Altra tipologia molto diffusa di firewall sono gli *application firewall* o *proxy firewall*, che operano invece un filtraggio dei pacchetti di tipo applicativo, andando cioè a filtrare tutto il traffico di una singola applicazione sulla base della conoscenza del suo protocollo[11]. Questo tipo di firewall analizza i pacchetti nella loro interezza considerando anche il loro contenuto ed è quindi in grado di distinguere il traffico di un'applicazione da quello di un'altra, cooperando anche molto spesso con altri sistemi per il rilevamento delle intrusioni(IDS).

In base a quanto detto, il firewall è dunque essenzialmente un componente di difesa perimetrale, poichè si trova posto al confine tra due reti per le quali funge da collegamento: una è la rete esterna, che comprende interamente Internet e si suppone non sia né sicura né affidabile; l'altra è la rete interna, o LAN, che può essere invece considerata tale proprio grazie all'azione di difesa del firewall.

Il firewall rappresenta, tuttavia, soltanto uno degli strumenti necessari per costruire una buona strategia di sicurezza e non può in generale considerarsi sufficiente da solo, in quanto presenta diversi limiti e debolezze[12].

Innanzitutto un firewall ha la necessità di essere mantenuto costantemente aggiornato e questo comporta un impiego continuativo di risorse e personale qualificato ad effettuare le dovute configurazioni. Inoltre le continue attività di manutenzione rischiano di renderlo sempre più vulnerabile a causa di possibili errori dovuti all'inesperienza di chi se ne occupa, errori di cui il firewall non è in grado di accorgersi in alcun modo e che lo portano a funzionare in maniera non completamente efficiente.
Per di più, trattandosi di una tecnologia prettamente in-line, esso è in grado di controllare solamente i pacchetti che lo attraversano, mentre tutto il traffico interno ad un segmento di rete su cui il firewall è attestato viene completamente ignorato.
Per sopperire a tutte queste mancanze, il firewall viene perciò affiancato ad almeno un altro dispositivo antivirus che si occupi delle minacce provenienti dalla rete interna.

2.3.1 I Next-Generation Firewall

La tecnologia dei firewall è in continua e costante evoluzione. Risulta infatti sempre più importante al giorno d'oggi riuscire ad adattarsi alle mutevoli esigenze della rete e poter rispondere in maniera efficace alle sempre nuove minacce e attacchi che quasi giornalmente vengono lanciati.
L'ultima generazione sviluppata in ordine di tempo è quella dei cosiddetti **next-generation firewall (NGFW)**[13].
Si tratta di sistemi che combinano le funzionalità tipiche di qualsiasi firewall come il filtraggio dei pacchetti, il NAT o il supporto alle VPN, con le caratteristiche di altre differenti tecnologie per la sicurezza, quali il rilevamento e la prevenzione delle intrusioni (tipiche degli IDPS) o la definizione di policy specifiche per ogni applicazione. In questo modo gli

NGFW possono ispezionare i pacchetti in maniera molto più approfondita rispetto all'analisi effettuata dai firewall tradizionali, andando ad operare anche ai livelli più bassi del modello ISO/OSI e facendo ispezione dei protocolli utilizzati.
L'obiettivo che questa nuova tecnologia di firewall si pone è la semplificazione della configurazione e della gestione di un insieme eterogeneo di strumenti di sicurezza e al tempo stesso il miglioramento del loro impatto sulle performance dell'intero sistema[10].

2.4 Gli Antivirus

Un antivirus è un software utilizzato per prevenire, rilevare e rimuovere programmi malevoli quali virus, worm, trojan, spyware e altri tipi di malware[14]. Esso non svolge quindi solo una funzione di eliminazione dei programmi malevoli ma ha anche una funzione preventiva, impedendo che un malware possa diffondersi in un sistema.
Per gestire tutte queste minacce l'antivirus può ricorrere a diversi metodi. Quello ancora oggi più utilizzato è il *metodo delle signatures* che permette di rilevare la presenza di un software malevolo confrontando il contenuto del file da analizzare con un archivio in cui sono schedati tutti i malware conosciuti, o meglio le loro firme. È evidente però come questa tecnica risulti completamente inefficace con i malware non ancora scoperti o non ancora analizzati, in quanto la loro firma non può ovviamente essere presente nel database.
Per rilevare nuovi malware si utilizza il *metodo delle anomalie*, o *metodo euristico*. Questo approccio si basa sul fatto che molti virus cominciano con una singola infezione e poi attraverso mutazioni e raffinamenti successivi

possono svilupparsi in molte varianti diverse. Ciò che un antivirus fa è quindi andare ad analizzare il contenuto dei vari file alla ricerca di pattern e schemi già noti e che possono identificare virus conosciuti o varianti di essi. Entrambi questi metodi, il metodo delle signature e quello delle anomalie, sono gli stessi che anche gli Intrusion Detection System utilizzano, anche se in modi leggermente differenti.

2.5 Gli Intrusion Detection System

Un intrusion detection system(IDS) è un dispositivo hardware o software(o a volte anche la combinazione di entrambi sotto forma di sistemi Stand-alone) che si occupa di rilevare le intrusioni in un sistema o una rete, andando poi a registrarle in appositi file di log e a segnalarle all'amministratore.

Un'*intrusione* è un tentativo di superare i meccanismi di sicurezza di un computer o di una rete da parte di qualcuno di non autorizzato, allo scopo di andare a compromettere la riservatezza, l'integrità e la disponibilità delle risorse del sistema(e che cosa questo comporti si è già discusso in un precedente paragrafo). Il termine *intrusion detection* indica quindi il processo di monitoraggio degli eventi che si verificano in un sistema alla ricerca di segni di un'intrusione. Perciò, in altre parole, un IDS altro non è che quel dispositivo di cui le varie aziende e organizzazioni che hanno la necessità di proteggere i propri dati si servono per automatizzare il processo di intrusion detection[15].

2.5.1 Modalità d'uso

Per monitorare il traffico in una rete alla ricerca di intrusioni, un IDS può essere utilizzato in due diverse modalità, che si differenziano principalmente in base alla posizione in cui il dispositivo viene installato:

- **modalità mirroring**: utilizzando le funzionalità di un router o di uno switch, il traffico di rete viene copiato e ridirezionato sull'IDS. Questo si occupa quindi di analizzare i pacchetti che gli sono stati inviati e, nel caso in cui venga rilevata un'attività sospetta, provvede a notificarla immediatamente all'amministratore del sistema. Operando solamente su una copia del traffico, un IDS di questo tipo non può in alcun modo bloccare o filtrare i pacchetti in ingresso e in uscita né tanto meno modificarli: esso non cerca di bloccare le eventuali intrusioni come ad esempio fa il firewall, ma si limita a rilevarle laddove si verifichino[16]. Per questo motivo tale modalità viene anche più correttamente definita **passiva** e gli IDS così strutturati sono detti **IDS passivi**;
- **modalità in-line**: in maniera solo apparentemente identica a quanto già visto per i firewall, si impone che tutto il traffico di rete passi attraverso l'IDS. Questo può così avere il pieno controllo su tutti i pacchetti analizzati, svolgendo attività di *prevention* che cercano di bloccare le intrusioni identificate, evitando sul nascere l'attivazione di programmi potenzialmente dannosi. Azioni di questo tipo che possono essere effettuate includono inviare un allarme (*alert*), eliminare pacchetti malevoli, resettare le connessioni o bloccare tutto il traffico proveniente da un determinato indirizzo IP. Ciò viene fatto, solitamente, basandosi su una lista di controllo degli accessi simile a quella utilizzata da un firewall, con la differenza che mentre

quest'ultimo agisce a basso livello lavorando su porte e IP, un IDS è in grado di operare ad un livello più alto (all'incirca fino al livello 7 del modello ISO/OSI) e ha a che fare principalmente con programmi e utenti [17] (Le differenze tra firewall e IDS, qui appena accennate, saranno formalizzate meglio nel paragrafo 2.5.4).

In contrapposizione con gli IDS visti finora, che *passivamente* si limitano a segnalare una sospetta violazione nella rete senza compiere alcuna azione preventiva, si parla in questo caso di **IDS attivi**. Più comunemente però, per riferirsi alla capacità che questi sistemi hanno di rispondere alle intrusioni rilevate tramite attività di prevention, si è soliti parlare di **intrusion detection and prevention system** o **IDPS**.

Gli IDPS sono solitamente considerati un'estensione degli IDS puri: entrambi controllano il traffico e le attività di sistema, ma solo i primi sono abilitati a prevenire e fermare le intrusioni.

In tutte le principali tassonomie dei sistemi di sicurezza, la differenza tra IDS e IDPS è esplicita e resa sempre piuttosto marcata. Ai fini pratici, tuttavia, questi due sistemi possono essere tranquillamente assimilati in un'unica entità poiché alla situazione attuale praticamente ogni IDS utilizzato è dotato di capacità di prevention. D'ora in avanti quindi, quando si parlerà di IDS, faremo implicitamente riferimento sempre ad un IDPS.

Giunti a questo punto, però, risulta abbastanza naturale farsi una domanda: perché non usare sempre la modalità in-line? Stando a quanto detto finora sembrerebbe infatti che questa sia del tutto identica al mirroring, con in più la possibilità di fare prevention.

I motivi alla base di questa differenziazione sono essenzialmente due:

- Un dispositivo in-line necessita di hardware di rete estremamente avanzato e quindi molto più costoso rispetto alla tecnologia necessaria per operare in mirroring;
- Un IDS che analizza il traffico "in linea" non solo deve essere molto potente ma anche a prova di guasto: se, infatti, il dispositivo si danneggia ma sta lavorando in mirroring, non si ha nell'immediato alcuna conseguenza; ma se a smettere di funzionare è un sistema in-line, allora tutto il traffico si bloccherà in quel punto, provocando gravi disservizi e malfunzionamenti all'intera rete.

Di contro la configurazione in mirroring, richiedendo che venga effettuata una copia di tutto il traffico da monitorare, può ritrovarsi a dover sottoporre a sforzi e "stress" notevoli gli apparati di rete di un'azienda, specie se il traffico è molto elevato.

2.5.2 Tipologie di IDS

Oltre alla suddivisione in passivi e attivi a seconda del modo in cui vengono utilizzati, gli IDS possono essere classificati in base a *cosa* analizzano: esistono IDS che analizzano le reti locali(**Network IDS**), quelli che analizzano gli Host(**Host based IDS**) e gli IDS ibridi che analizzano la rete e gli Host(**Hybrid IDS**). Più nello specifico[16]:

- **Network IDS(NIDS):** analizza il traffico di rete per identificare intrusioni, permettendo quindi di monitorare non solo un singolo host ma una rete completa. Solitamente si tratta di un sistema indipendente che sniffa i vari pacchetti che passano sul segmento di rete dov'è attestato, cercando tracce di attacchi e contenuti malevoli;
- **Host based IDS(HIDS):** così come un NIDS provvede ad analizzare il traffico di rete, un HIDS provvede ad analizzare i programmi in

esecuzione su un sistema e il loro utilizzo delle risorse. Si tratta infatti di un software che analizza l'Host su cui è installato alla ricerca di intrusioni. Tali intrusioni vengono rilevate analizzando i file di log del sistema, le system call, le modifiche al file system e altre componenti del computer;

- **Hybrid IDS:** tipologia di IDS che combina i due approcci. Le informazioni recuperate dagli HIDS in esecuzione negli Host vengono integrate con le informazioni prelevate dalla rete locale.

Una ulteriore classificazione può poi essere fatta in base al *come* analizzano, ovvero alla tecnica di intrusion detection utilizzata dall'IDS per rilevare traffico e attività malevole. Si parla così di **signature based IDS** e di **anomaly based IDS**:

- **Signature based IDS:** si tratta della tipologia di IDS in assoluto più diffusa e che fa uso di una tecnica basata su firme(signatures) molto simile a quella utilizzata per il rilevamento dei virus(vedi paragrafo 2.4). Un signature based IDS, infatti, monitora la rete e ne confronta i pacchetti con una serie di firme preimpostate che rappresentano attacchi già conosciuti. Quando poi rileva un pacchetto o del traffico che corrispondono ad una più firme del database analizzato, genera un allarme e lo segnala all'amministratore del sistema.

  Il problema principale con questo tipo di IDS deriva dal fatto che, esattamente come per gli antivirus signature based, non è in grado di rilevare in alcun modo traffico dannoso del quale non è ancora stata fatta alcuna firma; il grande beneficio che invece ha è quello di generare un numero relativamente basso di falsi positivi e di essere affidabile e veloce.

Per ovviare a tale mancanza sono quindi nati gli anomaly based IDS;

- **Anomaly based IDS:** sono IDS che si occupano di tracciare una serie di regole che definiscono lo stato normale del sistema, ricavandole dall'analisi di misure statistiche ed euristiche del sistema stesso come per esempio il carico di rete, il tipo di protocolli utilizzati, la quantità di CPU attiva e così via. Tali regole vengono poi utilizzate per rilevare e identificare eventuali anomalie ossia deviazioni dal comportamento standard del sistema analizzato.
  Le anomalie sono quindi analizzate e il sistema ne stabilisce il grado di pericolosità, andando eventualmente a informare l'amministratore.

2.5.3 IDS signature based vs IDS anomaly based

Abbiamo visto che gli intrusion detection system sono spesso specializzati in particolari settori e utilizzano tecnologie specifiche. Di conseguenza il confronto tra di essi risulterà in una lista di pro e contro che permette di conoscere la forza e le limitazioni dei diversi IDS, risultando quindi di fondamentale importanza per la realizzazione di un sistema di sicurezza completo. I risultati di tale confronto sono stati per praticità riassunti nella seguente tabella[16][18]:

NIDS	
Pro	• Sono in grado di rilevare anche gli attacchi provenienti dalla rete interna. • Permettono di capire quali misure preventive adottare al fine di evitare attacchi futuri. • Difficili da rilevare.
Contro	• Non sono in grado di monitorare protocolli wireless. • Non individuano attacchi che sfruttano vulnerabilità non

	ancora rese pubbliche. • Alto tasso di falsi positivi e falsi negativi.
HIDS	
Pro	• Possono analizzare una comunicazione end-to-end criptata.
Contro	• Consumano le risorse dell'host. • Possono entrare in conflitto con altri sistemi di sicurezza. • Comportano ritardi nella generazione degli allarmi e nelle creazione dei report per l'amministratore
Signature based IDS	
Pro	• Rappresentano il metodo più semplice ed efficace per rilevare attacchi già noti. • Numero relativamente basso di falsi positivi.
Contro	• Inefficaci contro attacchi sconosciuti o varianti di attacchi già noti. • Scarsa conoscenza di stati e protocolli. • Necessità di mantenere le signature e i vari pattern costantemente aggiornati.
Anomaly based IDS	
Pro	• Efficaci contro vulnerabilità non ancora scoperte. • Poco dipendenti dal SO utilizzato.
Contro	• Possono comportare problemi relativi alla selezione delle caratteristiche del sistema da adottare, poiché queste possono variare enormemente a seconda

	dell'ambiente. • Difficoltà a generare allarmi in tempo utile. • La presenza di falsi positivi o falsi negativi può avere gravi conseguenze per il sistema.

2.5.4 Differenze tra Firewall e IDS

Nel paragrafo 2.5.1 si è già accennato al fatto che, sebbene le attività di prevention svolte da un IDS presentino notevoli somiglianze con quelle di un firewall, esse riguardano in realtà ambiti ben diversi: il firewall agisce unicamente ai livelli più bassi del modello ISO/OSI, lavorando con porte e IP, mentre l'IDS è invece in grado di operare fino al *livello di applicazione* e quindi di fare ispezione anche dei protocolli.

Le differenze, tuttavia, non si fermano qui. Diversamente da tutti gli altri sistemi di difesa visti finora(firewall e antivirus), gli IDS agiscono ad un livello inferiore, operando nelle fasi iniziali del ciclo di vita di un cyber attacco, prima ancora che l'attaccante si infiltri all'interno del sistema e cominci l'attacco vero e proprio[16].

Un IDS, per via della possibilità di analizzare traffico in copia (modalità mirroring), è inoltre in grado di rilevare anche gli attacchi che provengono direttamente dall'interno del sistema su cui è attestato, cosa che invece un firewall non può fare in quanto, trattandosi di un dispositivo completamente in-line, riesce ad analizzare unicamente i pacchetti che lo attraversano. Questo viene fatto molto semplicemente controllando lo stato dei pacchetti che viaggiano già all'interno della rete locale e confrontandolo con situazioni pericolose già accadute o situazioni di anomalia definite dall'amministratore di sistema[16].

Tutto ciò significa che un IDS(da solo) e un firewall(da solo) non possono in

alcun modo garantire la completa sicurezza di un sistema: per ottenere risultati soddisfacenti si deve necessariamente combinare il loro utilizzo.

2.6 Firewall, antivirus e IDS: quale scegliere?

Quando si parla delle misure protettive installate in un sistema per garantirne la sicurezza, si ha solitamente a che fare con una grande varietà di opzioni sia da parte dei produttori che dei fornitori. Firewall, antivirus e IDS sono gli strumenti che vengono più spesso menzionati, delle cui funzionalità si è già ampiamente trattato nei precedenti paragrafi.

Una tale varietà lascia molto spesso confusi gli utenti, siano essi privati o amministratori di un'azienda, lasciandoli con una domanda: quale dovrei usare per proteggere il mio sistema? E la risposta a questa domanda è straordinariamente semplice: tutti e tre, in quanto ognuno di essi svolge un'azione complementare agli altri[19].

Un IDS monitora la rete per rilevare se un sistema esegue attività sospette, esaminando il traffico o i processi eseguiti dall'host. Un firewall verrà invece impostato per bloccare suddetto traffico nel caso in cui una connessione tra due o più dispositivi non risulti conforme alla policy di sicurezza stabilita per l'ambiente di rete in questione. Infine l'antivirus può controllare se un dispositivo all'interno della rete locale cerca di eseguire attività dannose che potrebbero compromettere la sicurezza delle sue informazioni.

Se si considerano le varie fasi in cui solitamente si articola il ciclo di vita di un cyber attacco, risulta evidente come ognuno di questi operi ad un differente livello: il primo ad entrare in gioco è l'IDS che permette di

rilevare e segnalare attività malevole, quando ancora l'attaccante non è riuscito ad ottenere accesso alla rete attraverso un sistema compromesso o un furto di identità e non ha quindi ancora potuto dare inizio all'attacco vero e proprio. Solo in un secondo momento agisce il firewall, il quale, in base alle indicazioni fornite dall'IDS, è in grado di rilevare quando l'attaccante tenta di eseguire un'azione malevola e può quindi prendere contromisure per evitarlo. Infine l'antivirus fornisce gli strumenti per impedire che un file ricevuto via mail, tramite un dispositivo USB o scaricato direttamente da Internet possa eseguire codice dannoso che metta a rischio l'integrità dei dati. Quest'ultimo entra quindi in azione soltanto una volta che il sistema è già stato infettato e molto probabilmente compromesso, e rappresenta l'ultima linea di difesa, di cui invece firewall e IDS costituiscono, se così vogliamo vederla, l'avanguardia.

Si può quindi concludere che la sicurezza in una rete può essere garantita solamente attraverso un utilizzo combinato di più meccanismi di cybersecurity: solo infatti servendosi dell'azione congiunta di almeno un firewall, un IDS e un antivirus si può avere un controllo più o meno completo su tutte le forme di attacchi che minacciano la sicurezza di un sistema, anche se, naturalmente, una protezione totale risulta pressoché impossibile da ottenere.

Infine un'ultima osservazione: oltre ad avere una buona infrastruttura di sicurezza risulta fondamentale essere qualificati su come poter operare contro tutte queste attività malevole nonché sensibilizzare gli utenti alle nuove minacce che si sviluppano ogni giorno. In caso contrario, infatti, anche possedere la migliore soluzione antivirus o il firewall più costoso si rivela del tutto inutile, se gli utenti non prestano sufficiente attenzione alle informazioni che condividono su Internet o alle password che

utilizzano[19].
Un uso responsabile delle informazioni e dei dispositivi permetterà agli ambienti di lavoro di essere più produttivi, utilizzando le varie tecnologie a loro disposizione in maniera sicura ed efficiente.

2.7 Il mercato della cybersecurity oggi

Secondo i più aggiornati report sulla sicurezza informatica, il danno economico mondiale causato dai cyber attacchi si aggira tra i 375 e i 575 miliardi di dollari all'anno[20]. Sono questi i soldi persi nelle clonazioni di carte di credito e dei dati dei conti correnti, per i furti di proprietà intellettuali e segreti industriali, a cui vanno poi aggiunti i costi di ripristino dei sistemi violati e tutto quel denaro che le aziende, le imprese e le amministrazioni pubbliche si trovano a dover spendere per ripianare le ingenti perdite finanziarie.
Il flusso di attacchi hacker degli ultimi anni è stato impressionante, continuo e assolutamente globale. E nel 2015 sono stati registrate ben 100 milioni di violazioni di database, un record storico[20].
È evidente quindi che in un contesto del genere il settore della cybersecurity abbia conosciuto nel recente periodo un enorme sviluppo, tanto che secondo i programmi di spesa americani del 2010, il governo avrebbe messo sul piatto ben 13 miliardi all'anno, in un orizzonte quinquennale, proprio per la sicurezza informatica, ovvero per dotarsi di infrastrutture e servizi destinati a proteggere i sistemi, le reti e i dati[21].
Analizzando più nel dettaglio i vari report promulgati, emerge tuttavia anche un dato preoccupante: soltanto meno dell'1% degli attacchi riesce ad essere individuato dai software di sicurezza come i firewall. E addirittura il 95%

delle intrusioni non autorizzate sono possibili semplicemente rubando le credenziali dei dipendenti dell'azienda, un'attività questa che viene spesso effettuata dall'interno della compagnia stessa e non può quindi essere bloccata dal firewall[20].

Ciò significa che non è affidandosi solamente agli antivirus e ai firewall che le compagnie sono in grado di proteggersi dai cyber criminali. Ed è proprio per questo motivo che l'attenzione pubblica si sta focalizzando sempre di più sul settore degli IDS, con le più grandi aziende e organizzazioni che ogni anno si trovano a spendere anche miliardi di dollari per dotarsi dei migliori strumenti disponibili e prevenire le intrusioni.

Il mercato degli IDS è ad oggi decisamente vasto e può contare su diverse decine di prodotti, tutti estremamente validi, tra cui poter scegliere. A contendersi la leadership del settore sono attualmente le grandi aziende americane specializzate in sicurezza informatica come Palo Alto Networks e Sourcefire, le quali sempre più spesso mettono a disposizione non solamente soluzioni a pagamento ma anche alternative open-source altrettanto efficaci.

A fianco di una vasta gamma di prodotti commerciali in continua competizione tra loro, infatti, vi è una grandissima comunità di **IDS open source**, una realtà che negli ultimi tempi sta prendendo sempre più piede.

2.7.1 Il mondo degli IDS open source

Quello degli IDS open source rappresenta ad oggi un settore veramente grande e, soprattutto negli ultimi anni, in continua crescita. Sempre più aziende decidono infatti di adottare queste soluzioni (spesso affiancandole ad altri sistemi di sicurezza), non solamente per una mera questione di costi, ma proprio per via della grande flessibilità e possibilità di configurazione

che questi sistemi offrono.

La comunità open source offre diversi vantaggi di non poco conto rispetto alle soluzioni a pagamento: tutti gli utilizzatori di software open source possono infatti collaborare e contribuire in prima persona, condividendo i propri risultati ed esperienze per aiutarsi a vicenda a risolvere eventuali problemi o bug riscontrati. Quando invece si compra un IDS da una qualche compagnia specializzata questo non avviene: nella maggior parte dei casi si ottiene un dispositivo già configurato da altri che viene installato all'interno della rete e lasciato lì, con nessuna o limitate possibilità di personalizzazione.

Esattamente come per gli IDS commerciali, anche in questo caso le soluzioni disponibili tra cui poter scegliere sono tantissime.

Rimanendo nell'ambito dei NIDS, che è quello che ci interessa più da vicino, i prodotti più diffusi e che hanno riscosso i maggiori consensi tra la community sono senza dubbio **Snort**, **Suricata** e **Bro**.

Quest'ultimo in particolare, in virtù della sua grande flessibilità ed efficienza d'uso, è l'IDS che si è scelto di utilizzare per la realizzazione del progetto di questa tesi e il capitolo successivo sarà interamente dedicato allo studio dettagliato del suo funzionamento.

Prima di passare a Bro però, vale la pena vedere brevemente anche quali sono le caratteristiche di Snort e Suricata, di modo tale da poter poi evidenziare con ancora maggior chiarezza in che modo Bro si differenzi dai suoi simili[22]:

- **Snort:** senza alcun dubbio il leader del settore dei NIDS open source. Nonostante non abbia alcuna GUI o interfaccia di amministrazione, si tratta di uno strumento che ha ottenuto una vastissima accettazione come soluzione NIDS per un'ampia gamma

di scenari e casi d'uso.

Snort è un software leggero e performante, in grado di eseguire in tempo reale l'analisi del traffico di reti IP e grazie a ciò di individuare potenziali minacce ed intrusioni.

Per fare questo esso si serve sia di tecniche di intrusion detection signature-based che di metodi anomaly-based, e può fare affidamento su regole(rules) create dagli utenti e firme(signatures) provenienti da noti database reperibili online come Emerging Threats[23].

- **Suricata:** concorrente diretto di Snort, esamina il traffico di rete utilizzando un potente linguaggio basato su regole e firme, potendo anche contare sul supporto dello scripting Lua per la rilevazione delle minacce più complesse[24]. Sebbene l'architettura di Suricata sia diversa da Snort, esso si comporta esattamente come Snort e può utilizzare le stesse firme.

3 Bro: non il classico IDS signature based

Talvolta denominato anche Bro-IDS, Bro è un intrusion detection system che si mostra subito piuttosto diverso dagli altri. Al tempo stesso sia signature based che anomaly based, Bro non si limita ad un particolare metodo di rilevamento delle intrusioni né tantomeno si basa sulle tradizionali firme come gli altri prodotti visti in precedenza, scegliendo invece di affidarsi ad un approccio peculiare: esso traduce il flusso di pacchetti all'interno di una rete in una serie di *eventi* di vario tipo e applica a ciascuno di questi uno *script*, liberamente personalizzabile dall'utente, che determina le azioni da effettuare in risposta all'evento stesso[25]. Un *evento* può essere qualsiasi cosa che avviene all'interno della rete analizzata come ad esempio una richiesta HTTP, una connessione TCP o una query DNS.

Un approccio di questo tipo è reso possibile dalla presenza di un suo linguaggio di dominio specifico (DLS), chiamato semplicemente *Bro scripting language*, che permette agli utenti di realizzare degli script da associare ai vari eventi, andando così a determinare le attività effettuate dall'IDS in risposta a ciò che accade nella rete, in maniera estremamente flessibile e potente.

Nei paragrafi successivi vengono analizzate le funzionalità principali e la struttura di Bro e viene data un'introduzione al suo linguaggio di script, lo strumento che sarà utilizzato per la realizzazione del nostro progetto.

3.1 Funzionalità

Bro agisce principalmente come un *monitor di sicurezza*, un analizzatore di rete che ispeziona tutto il traffico che passa su un determinato collegamento in cerca di segni di attività sospette.

Il vantaggio più immediato che un sito guadagna dall'usufruire dei suoi servizi è un insieme piuttosto vasto di *file di log* che registrano al loro interno tutte le attività della rete o del segmento di rete analizzati. Questi log includono non solamente un record per ogni connessione stabilita, ma anche una trascrizione accurata di tutto ciò che accade a livello di applicazione, come ad esempio un resoconto di tutte le sessioni HTTP iniziate, con tanto di URI richiesti e relativa risposta del server, delle query DNS effettuate, dei certificati SSL inviati e molto altro ancora[26].

Tutte queste informazioni vengono memorizzate da Bro in appositi file di log ben strutturati e tabulati che possono poi essere utilizzati per successive operazioni di elaborazione da parte di software esterni.

In aggiunta all'attività di logging, Bro dispone poi di una vasta gamma di funzionalità integrate per l'esecuzione di varie operazioni di analisi e rilevamento tra cui l'estrazione di file da sessioni HTTP, l'individuazione di malware, la localizzazione di tipologie di attacco molto diffuse come l'SSH brute-forcing e tanto altro.

Fin qui nulla di particolare, quindi, e nulla che si discosti troppo da quanto già visto per gli altri IDS in commercio.

Tuttavia la chiave per comprendere veramente le potenzialità di Bro consiste nel realizzare che, anche se si tratta di un sistema dotato già di base di funzionalità così vaste, esso rappresenta in realtà una piattaforma completamente personalizzabile ed estendibile. Bro mette infatti a

disposizione un linguaggio di scripting Turing completo, il **Bro scripting language**, che permette agli utenti di esprimere attività di analisi arbitrarie di pressoché qualsiasi tipo[26].

In maniera molto simile a quanto avviene per Python, Bro viene quindi fornito con un gran numero di funzionalità predefinite che possono poi essere facilmente estese scrivendo il proprio codice, andando così a definire un ambiente estremamente flessibile e ampliabile. Le stesse procedure di analisi implementate di default nel programma, infatti, inclusa tutta la parte relativa al logging, non sono altro che il risultato di script di questo tipo.

Tutto ciò dunque rende piuttosto evidente come Bro non sia affatto il classico IDS signature based: sebbene comunque supporti tale funzionalità, l'opportunità di usare un linguaggio di scripting dalle possibilità pressoché infinite permette di ricorrere ad una gamma molto più ampia di approcci diversi per rilevare attività dannose, tra cui l'*anomaly detection* o *la misuse detection*.

3.2 Struttura del sistema

Dal punto di vista puramente strutturale, Bro risulta suddiviso in due componenti principali: **l'Event Engine**, ovvero il motore di eventi, e il **Policy Script Interpreter**, ossia l'interprete degli script[26].

Questi sono evidenziati nella figura 3.1 sottostante:

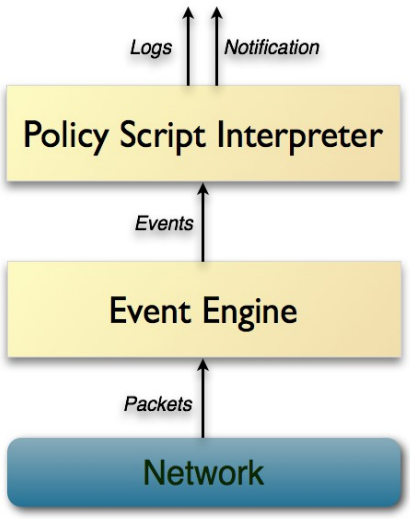


Figura 3.1: L'architettura interna di Bro

Vediamo brevemente il loro funzionamento.

L'Event Engine(detto anche *Bro core*) è il componente che si occupa di ridurre il flusso di pacchetti in entrata in una serie di eventi di alto livello.

Un evento viene generato semplicemente quando qualcosa accade e riflette quindi un'attività della rete in termini assolutamente neutrali. Un evento descrive che cosa è accaduto, non perché né tantomeno se questo sia significativo oppure no. Ad esempio: ogni richiesta HTTP viene trasformata in un corrispondente evento *http_request,* il quale porta con sé tutta una serie di informazioni quali gli indirizzi IP e le porte coinvolte, l'URI che viene richiesto o la versione del protocollo HTTP utilizzato. Tale evento, tuttavia, non fornisce alcuna ulteriore spiegazione sul suo contenuto, come ad esempio se l'URI richiesto corrisponda ad un sito di malware già noto.

Una semantica di questo tipo appartiene invece al secondo componente, lo Script Interpreter, che permette di eseguire una serie di gestori degli eventi(*event handlers*) scritti nel linguaggio di scripting di Bro. Tali script descrivono le azioni da compiere quando l'IDS rileva diversi tipi di attività, e consentono quindi di definire *policy di sicurezza* ad hoc per uno specifico sito. Di default Bro è impostato per registrare informazioni sugli eventi nei file di log, ma può essere configurato per eseguire altre azioni come mandare una mail all'amministratore di sistema, lanciare un allarme, eseguire una system call o anche richiamare un altro Bro script[26].

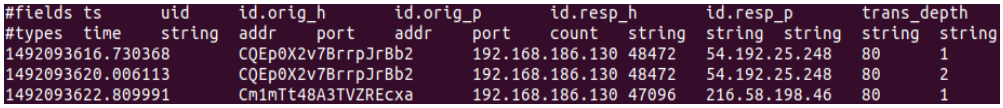
3.3 I file di log

Prima di addentrarsi all'interno del Bro scripting language(di cui si parlerà in dettaglio nel paragrafo seguente), è bene spendere qualche parola sui file di log generati automaticamente dal sistema e che è molto importante saper

leggere, così da poterne trarre informazioni utili per il rilevamento di eventuali attività malevole.

Di default i log di Bro sono file ASCII scritti in un formato che deve essere leggibile sia dall'uomo che dalle macchine, e popolati con dati per lo più *connection-oriented* organizzati in colonne. Generalmente vengono prodotti da uno script corrispondente che ne definisce la struttura, specifica per il protocollo o il tipo di connessione preso in esame. È infatti l'autore dello script a definire i tipi di dati che andranno a riempire i vari campi, ovvero le colonne, del file. Lo stesso autore si occupa poi di decidere quali attività di rete, durante il processo di monitoraggio da parte di Bro, dovranno dar luogo ad una nuove voce del log[27].

Come esempio di quanto detto finora, l'immagine sottostante mostra le prime colonne del file *http.log* che contiene i risultati dell'analisi del protocollo HTTP:



#fields	ts	uid	id.orig_h	id.orig_p	id.resp_h	id.resp_p	trans_depth
#types	time	string	addr	port	addr	port	count	string	string	string	string
1492093616.730368	CQEp0X2v7BrrpJrBb2	192.168.186.130	48472	54.192.25.248	80	1
1492093620.006113	CQEp0X2v7BrrpJrBb2	192.168.186.130	48472	54.192.25.248	80	2
1492093622.809991	Cm1mTt48A3TVZREcxa	192.168.186.130	47096	216.58.198.46	80	1

Figura 3.2: Il file *http.log*

Tutti i log che trattano l'analisi di uno specifico protocollo di rete inizieranno sempre con questi quattro campi, visibili nella figura 3.2: un timestamp(*ts*) che fornisce una collocazione temporale all'evento analizzato, un identificatore univoco(*UID*) della connessione, e una tupla detta *connection 4-tuple* definita su un insieme di quattro attributi(indirizzo IP/ porta dell'origine e indirizzo IP/porta della destinazione). Di fondamentale importanza è soprattutto l'UID, utilizzato per identificare tutte le attività, registrate anche in file di log differenti, associate ad una stessa connection 4-tuple[28].

Le restanti colonne che completano il file contengono invece informazioni specifiche sul protocollo o sull'attività di rete cui il log è collegato.
Nell'esempio di HTTP mostrano, tra le altre cose, il metodo utilizzato per la richiesta, l'URI associato alla richiesta stessa e la versione del protocollo:

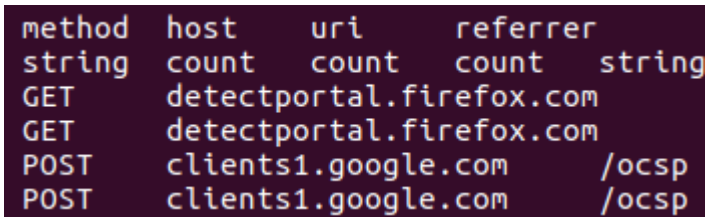


Figura 3.3: Altri campi del file *http.log*

Oltre ad *http.log*, qui riportato come esempio, Bro è in grado di generare molti altri file di log: alcuni di questi sono specifici di un determinato protocollo, mentre altri aggregano informazioni relative a diversi tipi di attività. In questa sede se ne riportano solo alcuni dei più importanti, rimandando alla documentazione ufficiale di Bro per un elenco completo e una descrizione accurata delle informazioni che contengono:

- ***conn.log**:* fornisce una trascrizione completa dell'attività della rete. Contiene una voce per ogni connessione rilevata, memorizzando proprietà di base quali la durata, gli indirizzi IP della sorgente e della destinazione, il tipo di servizio effettuato, la dimensione del payload e altro ancora;
- ***dns.log**:* offre un sommario di tutta l'attività DNS rilevata. Ogni record contiene informazioni attinenti come il dominio soggetto della query DNS e la corrispondente risposta del server;
- ***ftp.log**:* registra le attività di tutte le sessioni FTP rilevate;
- ***files.log**:* fornisce una sintesi di tutti i file trasferiti attraverso la rete. Queste informazioni vengono aggregate dai log di diversi altri

protocolli inclusi HTTP, FTP e SMTP.

- ***intel.log****:* utilizzato per rappresentare una corrispondenza positiva(match) tra l'intelligence caricata nell'IDS e i dati rilevati;
- ***weird.log****:* registra l'eventuale presenza di attività inaspettate a livello di protocollo. Ogni volta che Bro durante il processo di analisi incontra una situazione che normalmente non ci si aspetterebbe(ad esempio una violazione RFC), la registra in questo file;
- ***notice.log****:* identifica e registra attività che Bro riconosce come strane o potenzialmente dannose. Un'attività di questo tipo viene definita *notice*.

3.4 Il Bro Scripting Language

L'obiettivo principale delle varie installazioni di Bro è, nella maggior parte di casi, quello di inviare molto semplicemente mail di allarme ogni qual volta un evento richieda ulteriori investigazioni o l'intervento manuale dell'amministratore. Questo viene fatto cercando sempre di mantenere una politica di sicurezza il più possibile neutrale, dal momento che gli eventi riscontrati su una certa rete potrebbero essere molto meno interessanti degli stessi eventi rilevati all'interno di una rete differente.

Di conseguenza, distribuire Bro all'interno delle varie aziende e organizzazioni che decidono di farne uso può essere visto a tutti gli effetti come un processo di aggiornamento della sua policy di sicurezza, in modo tale da intraprendere azioni differenti in risposta agli eventi notificati, e andando ad utilizzare il **Bro scripting language** per indirizzare in una ben precisa direzione l'analisi del traffico[28].

Il Bro scripting language è un linguaggio di scripting *event-driven*(basato cioè sul *paradigma della programmazione a eventi*) che fornisce allo sviluppatore gli strumenti necessari per estendere e personalizzare le funzionalità di base dell'IDS. Praticamente tutto l'output prodotto da Bro è in realtà generato tramite script scritti in questo linguaggio: Bro può quindi essere considerato come una sorta di semplice entità "dietro le quinte", che processa le connessioni e genera eventi, mentre sono gli script a rappresentarne il cuore, preoccupandosi di gestire tali eventi tramite appositi *event handler* e stabilire quali azioni eseguire in risposta[29].
Nella sua configurazione di default, Bro è impostato per caricare solamente gli script inclusi nella directory *base:* si tratta di script pregenerati e inclusi all'interno del programma che si limitano a gestire tutti gli eventi allo stesso modo, generando un corrispondente file di log che ne memorizza le informazioni principali. Questa è però solamente la funzionalità di base, e la grande flessibilità di Bro risiede appunto nel fatto che gli utenti possono scrivere i propri script definendo una policy di sicurezza personalizzata e andando a gestire come ritengono più opportuno ogni singola tipologia di evento.

3.4.1 Gli Event Handler

Programmare su Bro può causare, soprattutto all'inizio, non poca confusione agli utenti che provengono da linguaggi procedurali o funzionali. Lo scripting in Bro dipende infatti quasi interamente dalla gestione degli eventi generati dall'IDS mentre questo analizza il traffico di rete[29]: anziché attendere che un'istruzione impartisca un comando al programma, come per la programmazione procedurale, il sistema viene predisposto per eseguire un loop infinito che controlla continuamente il verificarsi di un evento e

successivamente manda in esecuzione una o più parti di programma scritti appositamente per gestire l'evento in questione detti *event handler.*
Di base Bro si preoccupa di collocare tutti gli eventi generati in una coda, detta *event queue.* I gestori degli eventi si occupano quindi di processarli seguendo il principio del *first-come-first-served*, in base al quale l'evento inserito prima all'interno della coda è anche il primo ad essere elaborato dai suoi handler corrispondenti[29]. Ogni qual volta si verifica un nuovo evento, **tutti** i gestori associati a quel tipo di evento, anche se presenti su più script diversi, vengono eseguiti da Bro, a seconda del loro grado di priorità.
Strutturalmente qualsiasi script scritto in Bro risulta quindi costituito da un certo numero di event handler, ognuno associato ad un evento specifico e contenente tutte le azioni da effettuare in risposta, azioni che possono essere eventualmente racchiuse all'interno di *funzioni* del tutto simili ai costrutti già visti per molti altri linguaggi di programmazione.
Il modo migliore per comprendere quanto detto finora è senza dubbio quello di prendere uno script già realizzato e implementato su Bro e andare quindi ad analizzarlo nel dettaglio, identificandone i principali componenti. Per fare questo prendiamo ad esempio il file *detect-MHR.bro:* si tratta dello script di cui Bro si serve per controllare l'hash di tipo SHA1 dei vari file estratti dal traffico di rete e verificare che esso non coincida con uno dei valori di hash riportati nel famoso Malware Hash Registry(MHR).
Già dopo una semplice occhiata risulta subito evidente come lo script sia suddiviso in tre sezioni ben distinte, sia strutturalmente che visivamente:

- **load section**: la prima parte dello script consiste in una serie di direttive *@load*, utilizzate per includere librerie e caricare altri Bro script, specificando il filename desiderato. Nell'esempio considerato, le direttive si assicurano che il File Framework, il Notice Framework

e lo script che si occupa di generare l'hash di tutti i file analizzati, siano caricati da Bro (che cosa questo effettivamente comporti verrà spiegato nel paragrafo successivo):

```
@load base/frameworks/files
@load base/frameworks/notice
@load frameworks/files/hash-all-files
```

Figura 3.4: La load section

- **export section**: sezione così chiamata in quanto contiene al suo interno tutte le dichiarazioni di variabili, costanti o tipi che devono essere esportati per poter essere utilizzati anche al di fuori dello script in cui sono definiti. Una delle peculiarità di Bro rispetto ad altri linguaggi è la possibilità di dichiarare costanti ridefinibili, che cioè, anche una volta inizializzate, possono comunque essere modificate tramite una dichiarazione di tipo *redef*.

  Nella pagina seguente viene mostrata l'export section dello script considerato, dove si possono vedere chiaramente le costanti e le redef.

```
export {
    redef enum Notice::Type += {
        ## The hash value of a file transferred over HTTP matched in the
        ## malware hash registry.
        Match
    };

    ## File types to attempt matching against the Malware Hash Registry.
    const match_file_types = /application\/x-dosexec/ |
                             /application\/vnd.ms-cab-compressed/ |
                             /application\/pdf/ |
                             /application\/x-shockwave-flash/ |
                             /application\/x-java-applet/ |
                             /application\/jar/ |
                             /video\/mp4/ &redef;

    ## The Match notice has a sub message with a URL where you can get more
    ## information about the file. The %s will be replaced with the SHA-1
    ## hash of the file.
    const match_sub_url = "https://www.virustotal.com/en/search/?query=%s" &redef;

    ## The malware hash registry runs each malware sample through several
    ## A/V engines.  Team Cymru returns a percentage to indicate how
    ## many A/V engines flagged the sample as malicious. This threshold
    ## allows you to require a minimum detection rate.
    const notice_threshold = 10 &redef;
}
```

Figura 3.5: L'export section

- **instruction section**: fino a questo punto uno script non ha fatto altro che effettuare del semplice setup di base. È solamente in questa sezione, la più importante, che esso definisce le istruzioni e le operazioni effettivamente svolte. Il fulcro di tutto questo è rappresentato, ovviamente, da uno o più event handler che si avvalgono di funzioni di supporto per svolgere buona parte del lavoro. Per quanto riguarda il nostro esempio, il gestore degli eventi in questione è quello relativo all'evento *file_hash*, che consente allo script di accedere a tutta una serie di informazioni associate ad un file per cui Bro, in fase di analisi, ha generato un hash:

```
event file_hash(f: fa_file, kind: string, hash: string)
    {
    if ( kind == "sha1" && f?$info && f$info?$mime_type &&
         match_file_types in f$info$mime_type )
        do_mhr_lookup(hash, Notice::create_file_info(f));
    }
```

Figura 3.6: Un event handler per l'evento *file_hash*

```
function do_mhr_lookup(hash: string, fi: Notice::FileInfo)
    {
    local hash_domain = fmt("%s.malware.hash.cymru.com", hash);

    when ( local MHR_result = lookup_hostname_txt(hash_domain) )
        {
        # Data is returned as "<dateFirstDetected> <detectionRate>"
        local MHR_answer = split_string1(MHR_result, / /);

        if ( |MHR_answer| == 2 )
            {
            local mhr_detect_rate = to_count(MHR_answer[1]);

            if ( mhr_detect_rate >= notice_threshold )
                {
                local mhr_first_detected = double_to_time(to_double(
                local readable_first_detected = strftime("%Y-%m-%d %
                local message = fmt("Malware Hash Registry Detection
                local virustotal_url = fmt(match_sub_url, hash);
                # We don't have the full fa_file record here in orde
                # avoid the "when" statement cloning it (expensive!)
                local n: Notice::Info = Notice::Info($note=Match, $m
                Notice::populate_file_info2(fi, n);
                NOTICE(n);
                }
            }
        }
```

Figura 3.7: Una funzione di supporto per l'*event handler*

In circa poche decine di righe di codice, Bro è quindi in grado di fornire facilmente servizi che utilizzando altri linguaggi risulterebbero molto più difficili da implementare. Anche se, in realtà, dietro a tutto questo si nasconde un gran numero di componenti, di framework e di moduli che agiscono dietro le quinte fornendo all'IDS tutte le funzionalità di cui necessita. È tuttavia proprio per merito del Bro scripting language che gli

analisti possono accedere e servirsi di questi “strati” sottostanti in maniera straordinariamente sintetica ed efficiente[29].

3.4.2 I Framework

Abbiamo visto che tutte le funzionalità di base implementate su Bro vengono realizzate tramite script del tutto analoghi a quelli che qualsiasi utente è in grado di scrivere servendosi del Bro scripting language.

A seconda del loro compito e per praticità, tali script sono raggruppati in *framework*, fornendo così al programmatore degli strumenti di cui potersi servire liberamente, risparmiandogli la necessità di dover riscrivere codice già scritto in precedenza da qualcun altro.

I più importanti, nonché quelli che più spesso ci si trova ad utilizzare quando si programma su Bro, sono:

- **File Analysis Framework**: in passato scrivere script Bro con l'intento di analizzare il contenuto dei file era un compito estremamente scomodo a causa del fatto che tale contenuto poteva essere presentato, tramite eventi, in moltissimi modi diversi a seconda di quale protocollo di rete fosse coinvolto nel suo trasferimento. La situazione imponeva quindi che venissero realizzati script di analisi specifici per un determinato protocollo, che dovevano poi essere copiati e opportunamente modificati per adattarsi anche a tutti gli altri. L'introduzione del File Analysis Framework (FAF), invece, consente una presentazione generalizzata del contenuto di ciascun file. Le informazioni relative al protocollo utilizzato per trasportare il file nella rete sono ancora presenti, tuttavia non determinano più la logica dello script che le gestisce[30];

- **Input Framework:** consente agli utenti di importare all'interno di Bro dati contenuti in file esterni. I dati vengono memorizzati in tabelle oppure convertiti in eventi che possono essere poi gestiti dagli script. Nelle sue impostazioni predefinite, l'input framework è in grado di leggere solamente dati che si presentano nello stesso formato dei file di log generati dal Logging Framework, ma è possibile utilizzare anche altri *reader* per differenti tipologie di file come ad esempio i file binari[31];
- **Intelligence Framework:** fornisce agli utenti tutti gli strumenti necessari per la gestione di dati e informazioni di *cyber threat intelligence*. Questo framework consente infatti di caricare su Bro pressoché qualsiasi tipo di indicatore, confrontarlo con il traffico monitorato e segnalare ogni eventuale corrispondenza[32]. La gestione dell'intelligence su Bro e l'Intel Framework saranno discussi più in dettaglio nel capitolo 4;
- **Logging Framework:** il framework che molto semplicemente è il responsabile di tutti i log generati da Bro. Ciascun componente dell'IDS (analizzatore HTTP, analizzatore DNS, analizzatore FTP ecc.) si preoccupa di inviargli le informazioni raccolte dalla rete, e quest'ultimo genera il corrispondente file di log, applicandovi una serie di filtri che determinano cosa esattamente deve essere registrato e in che modo[33];
- **Notice Framework:** si tratta del framework tramite cui vengono gestite tutte le attività che Bro riconosce e segnala come insolite o potenzialmente dannose. Il meccanismo che permette all'IDS di lanciare una *notice*, vale a dire un avviso o un avvertimento, e le azioni che possono essere effettuate quando questo avviene saranno

spiegate in dettaglio nel paragrafo successivo, data la sua importanza e il ruolo di rilievo che ricopre all'interno del nostro progetto.

3.4.3 Le Notice

Bro viene fornito assieme ad un gran numero di script in grado di eseguire una vasta gamma di tipi differenti di analisi. La maggior parte di questi monitora il traffico della rete alla ricerca di attività che potrebbero essere di un qualche interesse per l'utente. Nessuno di essi, tuttavia, è in grado di determinare l'effettiva importanza di ciò che rileva: uno script si limita a segnalare come potenzialmente interessanti le varie situazioni riscontrate, lasciando alla configurazione locale del sistema l'onere di stabilire su quali di queste valga effettivamente la pena intervenire[34].
Un tale disaccoppiamento tra rilevamento e segnalazione, ovvero tra *detection* e *reporting*, permette a Bro di gestire separatamente le diverse esigenze che i vari siti hanno. La definizione di ciò che rappresenta un attacco o una minaccia, infatti, può differire molto da un ambiente all'altro,
e un'attività considerata malevola su un determinato sito potrebbe essere ritenuta pienamente accettabile in un altro[34].

Lanciare una notice

Ogni volta che uno degli script di analisi di Bro rileva qualcosa di potenzialmente interessante viene lanciata una notice ossia sollevato un allarme. Questo viene fatto richiamando la funzione NOTICE, funzione base del Notice Framework che prende in ingresso un record di tipo *Notice::Info* (tipo di record definito negli script del framework per rappresentare una generica notice; ulteriori dettagli sulla sua struttura possono essere reperiti all'interno della documentazione ufficiale di Bro).

Ciascuna notice è definita da un *tipo*, che riflette la categoria dell'attività rilevata, ed è solitamente accompagnata da ulteriori informazioni circa il contesto e la situazione che l'ha generata. Le tipologie principali di notice che possono essere lanciate sono individuate dai valori di un apposito *enum* definito all'interno del framework chiamato *Notice::Type:* esso rappresenta praticamente tutte le più comuni attività sospette che possono essere rilevate durante una sessione di Bro quali ad esempio un tentativo di lanciare un *attacco di forza bruta* (*FTP::Bruteforcing*) o un certificato SSL che si scopre essere scaduto (*SSL::Certificate_Expired*).
Gli script che creano e lanciano nuovi tipi di notice diversi da quelli predefiniti devono aggiungere all'enum i propri tipi specifici, che saranno poi utilizzati ad ogni chiamata della funzione NOTICE[34].

Le notice policy

Una volta che una notice è stata lanciata, è possibile assegnarle un certo numero di azioni da effettuare in risposta scrivendo uno o più *hook* che definiscono una certa notice policy (*Notice::policy*). Azioni di questo tipo comprendono il mandare una mail ad un indirizzo stabilito o l'ignorare completamente la notice. La figura 3.8 della pagina seguente mostra i principali tipi di azione[34]:

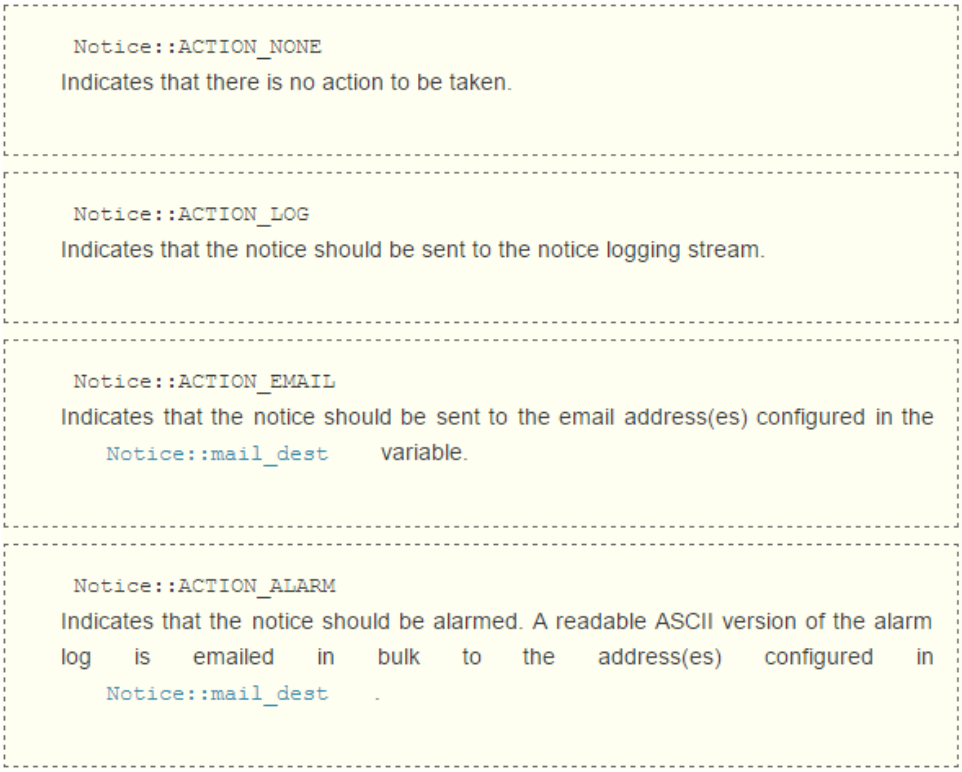


Figura 3.8: Le principali notice action

L'hook Notice::policy fornisce un meccanismo per applicare azioni e generalmente modificare la notice nel momento in cui questa viene lanciata. Un hook può essere considerato come una sorta di *funzione multi-corpo* il cui utilizzo si presenta in maniera molto simile alle gestione degli eventi, con la differenza che diversamente da quest'ultimi l'hook non passa attraverso alcuna coda, ma può essere invocato direttamente come una normale funzione.

In linea con quanto detto precedentemente sulla separazione tra la fase di detection e quella di reporting, il Notice Framework fornisce un hook di notice policy di default che si limita a registrare sul corrispondente file *notice.log* ciascuna notice lanciata, senza dare alcun giudizio sull'attività

rilevata (se sia da considerarsi pericolosa oppure no) né compiere ulteriori azioni. Ancora una volta è poi il grande potere espressivo del Bro scripting language a permettere agli utenti di scrivere le proprie notice policy personalizzate, stabilendo quali sono per uno determinato sito le notice più significative e per cui valga la pena effettuare azioni specifiche[29].

Un semplice esempio

Per meglio comprendere i concetti espressi finora, può risultare utile analizzare personalmente uno script di esempio.

Le figure sottostanti mostrano lo script *interesting-hostnames.bro*, il quale lancia una notice ogni volta che viene rilevato un login SSH e l'hostname di origine risulta sospetto:

```
event ssh_auth_successful(c: connection, auth_method_none: bool)
        {
        for ( host in set(c$id$orig_h, c$id$resp_h) )
                {
                check_ssh_hostname(c$id, c$uid, host);
                }
        }
```

Figura 3.9: Un event handler per il login SSH

```
        ## Strange/bad host names to see successful SSH logins from or to.
        const interesting_hostnames =
                        /^d?ns[0-9]*\./ |
                        /^smtp[0-9]*\./ |
                        /^mail[0-9]*\./ |
                        /^pop[0-9]*\./  |
                        /^imap[0-9]*\./ |
                        /^www[0-9]*\./  |
                        /^ftp[0-9]*\./  &redef;
}

function check_ssh_hostname(id: conn_id, uid: string, host: addr)
        {
        when ( local hostname = lookup_addr(host) )
                {
                if ( interesting_hostnames in hostname )
                        {
                        NOTICE([$note=Interesting_Hostname_Login,
                                        $msg=fmt("Possible SSH login involving a %s %s with an interesting hostname.",
                                                        Site::is_local_addr(host) ? "local" : "remote",
                                                        host == id$orig_h ? "client" : "server"),
                                        $sub=hostname, $id=id, $uid=uid]);
                        }
                }
        }
```

Figura 3.10: Se l'hostname appartiene ad una lista di host sospetti, viene lanciata una nuova notice

Infine vediamo un possibile hook Notice::policy che può essere scritto dall'utente per far sì che, ogni qual volta viene lanciata una notice di questo tipo, venga inviata una mail all'indirizzo specificato:

```
hook Notice::policy(n: Notice::Info)
  {
  if ( n$note == SSH::Interesting_Hostname_Login )
       add n$actions[Notice::ACTION_EMAIL];
  }
```

Figura 3.11: Se la notice lanciata è di un determinato tipo, le viene associata l'azione di inviare una mail

4 Cyber Threat Intelligence

I cybercriminali di oggi sono sempre meno prevedibili, sempre più ostinati e ingegnosi, ben organizzati e supportati da sempre maggiori fondi e risorse. Ciò implica che, anche se un'azienda riesce a contrastare un attacco servendosi dei propri sistemi di sicurezza, il responsabile della minaccia (detto anche *threat actor*) passerà immediatamente e senza perdersi d'animo ad utilizzare nuovi strumenti, nuove tattiche e nuove procedure (in gergo *TTP*, ovvero *Tactics, Techniques and Procedures*), molto più velocemente di quanto il team di sicurezza dell'organizzazione possa rispondere[35].

Per di più oggigiorno è possibile reperire sul Web con estrema facilità avanzati strumenti d'attacco già pronti all'uso come *backdoor* o *keylogger* che permettono anche a chi ha scarse conoscenze informatiche di entrare nel mondo del cyber crimine; senza poi contare la grande diffusione degli Exploit Kit che in pochi semplici click permettono di generare software malevoli di ogni tipo e che dimostrano come infiltrarsi all'interno di un'azienda sia un compito più facile di quanto si possa pensare[36]. E anche una volta che l'attaccante è riuscito ad infiltrarsi, la situazione non migliora di certo: gli attacchi vengono effettuati in forma anonima, con l'aggressore che potrebbe utilizzare le credenziali d'accesso di un dipendente dell'azienda per passare inosservato, rendendo perciò estremamente difficile riconoscere le attività lecite da quelle di un malintenzionato.

Le aziende non possono quindi mai dirsi completamente al sicuro dalle minacce, e questo rende sempre più difficoltoso il processo di

individuazione e protezione dagli attacchi, facendo emergere tutti i limiti degli approcci tradizionali di sicurezza.
Oggigiorno la sicurezza totale di un sistema è qualcosa che non può mai essere garantito con certezza e chi subisce un attacco deve subito preoccuparsi di migliorare le proprie capacità nell'individuare e riconoscere le minacce ai propri sistemi, così da potersi porre in una situazione di vantaggio sull'attaccante e riuscire ad agire in tempo utile e in maniera efficace nei confronti di altri futuri attacchi.
Per fare questo sempre più spesso le aziende ricorrono a nuovi approcci di difesa che si basano sulla **Threat Intelligence**, un concetto che negli ultimi anni sta diventando il cuore pulsante della cybersecurity.

4.1 Il concetto di Threat Intelligence

Con Threat Intelligence, o anche **Cyber Threat Intelligence (CTI)** nel nostro specifico caso, si intende la raccolta e la condivisione di informazioni su minacce già note. Generalmente si fa riferimento ad un database o un insieme di dati da confrontare con gli allarmi di sicurezza e i log generati da un'azienda per capire se quanto rilevato rappresenti o meno una minaccia per il sistema[36]. Nel caso in cui una corrispondenza venga rilevata, ciò può essere usato per proteggere la compagnia da attacchi simili ancora in circolazione. Tali corrispondenze riguardano solitamente indirizzi IP, hostname, URL, hash di file e altri elementi a cui nell'ambito dell'intelligence ci si riferisce col termine di *indicatori,* e che sono stati riconosciuti come associati ad attività malevole e quindi condivisi con il resto del mondo. Ogni volta che un'azienda o un privato diventa il bersaglio di un attacco, infatti, può decidere liberamente di condividere informazioni

riguardanti l'attacco subìto e metterle a disposizione di chiunque possa averne bisogno, molto spesso tramite community di sicurezza create appositamente per favorire lo scambio di intelligence.

Conoscere in anticipo le minacce più rilevanti per la sicurezza di un'azienda permette alle organizzazioni di agire preventivamente, bloccando le intrusioni e correggendo le eventuali vulnerabilità presenti nei propri sistemi, in modo da prevenire possibili attacchi futuri.

Il termine intelligence qui utilizzato non è proprio dell'informatica, ma viene preso in prestito da altri settori come ad esempio quello militare o economico. La sua definizione in tutti questi casi include non solo una serie di informazioni utili alla protezione di un'organizzazione da minacce esterne e interne, come visto finora, ma anche e soprattutto tutti quei processi, politiche e strumenti progettati per raccogliere e analizzare tali informazioni[37].

La grande differenza che separa la threat intelligence dalle signature utilizzate nella maggior parte dei sistemi di sicurezza, infatti, consiste proprio in questo: pur avendo a che fare in entrambi i casi con minacce già note, mentre l'approccio tradizionale si basa su semplici firme che consistono in una serie di indicatori di vario tipo come hash, domini o IP, da inserire all'interno di apposite blacklist per bloccare il traffico (un metodo questo facilmente eludibile e decisamente inefficace, dato che è sufficiente modificare il malware per aggirare il controllo), l'intelligence viene solitamente sottoposta ad un'analisi più approfondita da parte di un team di specialisti, che partendo dai dati raccolti cerca di estrapolare informazioni utili sull'attacco quali ad esempio il modus operandi, gli strumenti utilizzati o l'identità dell'attaccante[38]. Tutto questo viene a costituire un importantissimo bagaglio di informazioni di threat intelligence di cui

un'azienda può servirsi per migliorare i propri sistemi di difesa e proteggersi da futuri attacchi. Essa viene utilizzata durante i processi di analisi in tempo reale delle anomalie riscontrate e, correlando tra loro molteplici indicatori, consente di guidare la compagnia nel prendere le decisioni più opportune, quando gli attacchi sono ancora nella loro fase iniziale e gestibili, prima che la situazione diventi critica e molto più dispendiosa da risolvere in termini sia di tempo che di risorse[39].

Risulta evidente che, affinché un sistema del genere possa funzionare, la condivisione di informazioni di intelligence sulle nuove minacce e attacchi che ogni giorno vengono fuori debba essere effettuata nella maniera più rapida possibile: se infatti una nuova tipologia di attacco impiega dei mesi per essere inserita in threat intelligence, questa informazione non sarà di alcuna utilità nell'immediato, in quanto fino a quel momento le aziende risulteranno vulnerabili a un tale attacco. Oggigiorno fortunatamente l'inclusione di nuove minacce all'interno della threat intelligence può dirsi ormai un procedimento piuttosto veloce e le informazioni su un certo attacco risultano in genere disponibili nell'arco di pochissime ore. Particolarmente utili a questo scopo sono gli *honeypot*, componenti hardware o software che fingendosi host vulnerabili sono soggetti a frequenti attacchi e consentono così di raccogliere preziose informazioni da inserire poi in threat intelligence.

4.2 Open source Intelligence (OSINT)

Quello dell'intelligence è un settore estremamente vasto che oltre all'informatica coinvolge tantissimi altri ambiti, come quello militare, economico, civile e delle comunicazioni. Le attività di intelligence non sono

dunque finalizzate unicamente a garantire la sicurezza dei sistemi informatici, quanto piuttosto a salvaguardare la sicurezza nazionale in generale e su più livelli.

L'intelligence è lo strumento di cui uno Stato si serve per raccogliere, custodire e diffondere ai soggetti interessati, siano essi pubblici o privati, informazioni rilevanti per la tutela della sicurezza delle Istituzioni, dei cittadini e delle imprese. Deputate a tali attività sono solitamente apposite strutture statali, identificate spesso anche come servizi segreti, che si occupano di organizzare e amministrare la raccolta di informazioni di intelligence. Queste possono essere ricavate da una grandissima varietà di fonti diverse(materiale classificato, informazioni sensibili di un'azienda, interrogatori, intercettazioni telefoniche, attività di spionaggio, fotografie, video ecc.), fonti che, come si può vedere, nella grande maggioranza dei casi sono segrete, coperte o addirittura clandestine.

Oggi ci troviamo in un'epoca in cui le informazioni valgono più di ogni altra risorsa, e saperle gestire con oculatezza è il segreto per acquisire conoscenza e di conseguenza potere. Ci troviamo nell'era di Internet, in cui la digitalizzazione delle informazioni e la condivisione in tempo reale, dove tutti possono fotografare, scrivere e segnalare sui social network ciò che sta loro accadendo, si stanno dimostrando armi estremamente potenti che né le aziende né i governi si possono permettere di sottovalutare. Ecco quindi che negli ultimi tempi, accanto ai metodi tradizionali, si è sviluppata sempre più una forma di intelligence che provvede alla raccolta e all'analisi di informazioni reperibili da fonti pubbliche e non classificate, liberamente accessibili a chiunque, una definizione di cui fa pienamente parte anche tutto ciò che viene pubblicato quotidianamente sul Web. Si tratta dell'**Open Source Intelligence (OSINT)** ovvero l'intelligence delle fonti aperte, in

contrapposizione con le fonti segrete e coperte che caratterizzano generalmente l'intelligence tradizionale[40].

L'OSINT, con un nome o con un altro, risulta in circolazione da tantissimi anni. C'è voluto tuttavia diverso tempo affinché le principali agenzie governative si convincessero delle grandi possibilità offerte dalle fonti pubbliche, e oggi, finalmente, i servizi di intelligence sfruttano ampiamente questa metodologia di analisi per accedere ad informazioni per le quali altrimenti dovrebbero sfruttare una notevole quantità di risorse[41].

Le fonti di open source intelligence sono al giorno d'oggi moltissime. Quelle principali e che vale la pena nominare sono senza dubbio[42]:

- **Mass-media**: giornali, riviste, televisione, radio;
- **Dati pubblici dei governi**: rapporti di governo, piani finanziari, dati demografici, conferenze stampa, avvisi di vario tipo ecc. Sebbene provengano da organi ufficiali queste fonti sono accessibili al pubblico e possono essere utilizzate liberamente;
- **Pubblicazioni professionali e accademiche:** riviste, simposi, conferenze, articoli accademici, tesi;
- **Dati commerciali:** valutazioni finanziarie e commerciali, database;
- **Internet:** siti web, pubblicazioni online, articoli su blog, media creati dai cittadini (video e foto scattati col cellulare), post sui social network ecc. Queste fonti hanno il grande vantaggio di essere sempre tempestive (nuovi contenuti vengono creati praticamente in ogni momento) e facilmente accessibili.

Quest'ultima è quella che riguarda più da vicino il tema della cybersecurity trattato in questa tesi, ed è infatti proprio tramite Internet che è stata raccolta tutta l'intelligence utilizzata per la realizzazione del nostro progetto su Bro.

Alcuni esempi di OSINT

Attualmente sul Web è possibile trovare un gran numero di aziende grandi e piccole, organizzazioni no-profit, centri di ricerca ma anche semplici gruppi di appassionati di cyber security che, per rendere Internet un posto più sicuro e cercare di porre un freno al sempre maggior numero di attacchi, danno vita a vere e proprie community di sicurezza, fornendo agli utenti di qualsiasi parte del globo gli strumenti e la possibilità di condividere open source intelligencee andando così a creare un database vastissimo di cui chiunque può liberamente usufruire per proteggere i propri sistemi.

Le immagini seguenti mostrano alcuni esempi di siti di questo genere:

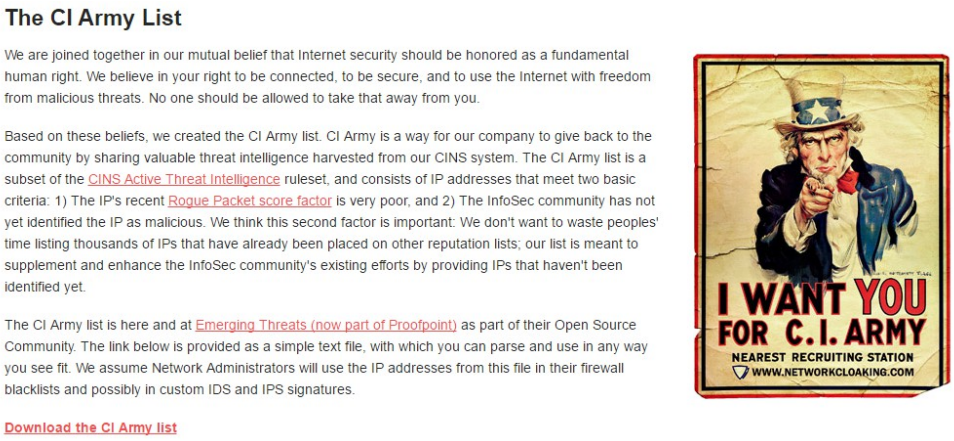


Figura 4.1: La CI Army List - cinsscore.com/#list

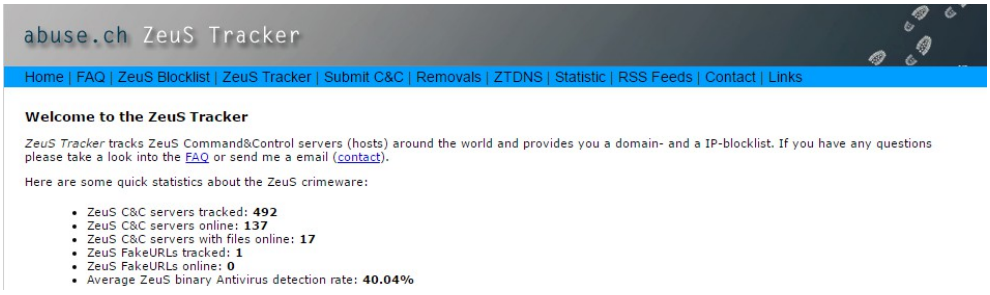


Figura 4.2: ZeuS Tracker - zeustracker.abuse.ch

Figura 4.3: AlienVault - otx.alienvault.com

Figura 4.4: Team Cymru MHR - www.team-cymru.org/MHR.html

4.3 Uno standard per la Threat Intelligence

Per essere le più veloci ed efficienti possibile, le attività di rilevamento e prevenzione di un sistema richiedono una condivisione intelligente, fluida e soprattutto automatica della threat intelligence. Non solo infatti è importante ciò che si condivide, ma anche il modo in cui questo avviene, il tutto sempre con lo scopo di fornire una solida base di conoscenza sui principali avversari di un'azienda e i loro obiettivi[43]. E questo accade solo se si ottengono informazioni da una vasta gamma di utenti, provenienti da ambiti

differenti: nessun singolo partecipante potrà mai essere in grado da solo di fornire tutte le informazioni necessarie.

Il problema che a questo punto solitamente viene fuori non è tanto che le persone non vogliono condividere informazioni sugli attacchi che li riguardano, quanto piuttosto il fatto che il processo di condivisione richiede tempo e risorse notevoli. Al fine di ottimizzare l'intero procedimento risulta quindi molto importante stabilire a priori che cosa si vuole e si deve condividere, andando cioè a definire uno **standard** per la condivisione di informazioni di threat intelligence[43].

Nel corso degli anni numerosi sono stati i tentativi di dar vita a standard industriali in questo settore, non tutti andati a buon fine. Tra questi, una delle soluzioni che specialmente nell'ultimo periodo sta riscuotendo un notevole successo e può contare su una community piuttosto attiva e numerosa è **STIX**, che sta per **Structured Threat Information eXpression**. Questo si trova generalmente accoppiato ad un'altra tecnologia denominata **TAXII**, ovvero **Trusted Automated eXchange of Intelligence Information**, che definisce i servizi e le modalità attraverso cui è possibile condividere informazioni di threat intelligence basate su STIX.

Le funzioni di STIX e TAXII sono strettamente correlate. Di conseguenza vengono solitamente studiati e presi in esame insieme, tenendo sempre bene a mente che non si tratta di software a sé stanti ma di standard di cui i programmi che si occupano di cybersecurity possono servirsi.

4.3.1 STIX e TAXII

STIX è un linguaggio strutturato che mette a disposizione un formato di serializzazione per consentire lo scambio di informazioni di threat intelligence in maniera efficiente. Esso permette infatti alle aziende di

condividere CTI le une con le altre in una maniera consistente e leggibile sia dall'uomo che dalle macchine, consentendo così alle varie community di sicurezza di anticipare o rispondere agli attacchi più velocemente e in modo molto più efficace. STIX è progettato per migliorare molte funzionalità diverse, come l'analisi del traffico, lo scambio automatizzato di threat intelligence, l'intrusion detection e altro ancora[44][45].

Attualmente l'ultima versione rilasciata è la 2.0 che nelle sue specifiche definisce STIX come un linguaggio indipendente da qualsiasi specifica forma di serializzazione, con l'obbligo tuttavia di garantire il pieno supporto alla serializzazione tramite JSON (che rappresenta quindi la serializzazione *mandatory-to-implement* del linguaggio). La figura sottostante mostra un esempio di oggetto in JSON utilizzato per rappresentare un'informazione di threat intelligence, ovvero una campagna di attacchi lanciata dalla compagnia Green Group:

```
{
    "type": "campaign",
    "id": "campaign--8e2e2d2b-17d4-4cbf-938f-98ee46b3cd3f",
    "created": "2016-04-06T20:03:00.000Z",
    "name": "Green Group Attacks Against Finance",
    "description": "Campaign by Green Group against targets in the financial services sector."
}
```

Figura 4.5: Un oggetto STIX in JSON

STIX consente di rappresentare tantissimi dei concetti tipici della CTI utilizzando oggetti JSON appositi denominati *STIX Domain Objects* (*SDO*): questi contengono non solamente semplici indicatori come indirizzi IP o hostname sospetti, ma anche informazioni contestuali e metadati che forniscono un background ulteriore su un dato attacco, informazioni come l'identità dell'attaccante, il nome dell'organizzazione per cui lavora, gli strumenti utilizzati per infiltrarsi nel sistema, i malware impiegati, le

vulnerabilità sfruttate e così via[44].

Tutti questi oggetti, che STIX permette anche di mettere in relazione gli uni con gli altri tramite costrutti appositi detti *STIX Relationship Objects* (*SRO*), vengono poi ripartiti in contenitori chiamati B*undle,* vere e proprie collezioni di SDO che possono essere trasportate e condivise da un sistema ad un altro.

Di base STIX non implementa alcuno specifico meccanismo di trasporto per i propri oggetti, ma si affida solitamente ai servizi messi a disposizione da TAXII, lasciando comunque sempre all'utente la facoltà di scegliere un meccanismo di comunicazione alternativo. Si tratta sostanzialmente di un protocollo a livello di applicazione che permette la comunicazione di informazioni di threat intelligence in modo semplice e veloce su una connessione HTTPS. TAXII è stato appositamente progettato per supportare lo scambio di CTI rappresentata in STIX e ne rappresenta perciò il naturale completamento[46].

4.4 Bro e la Cyber Threat Intelligence

Vista la sempre maggiore importanza che viene data all'intelligence nei moderni sistemi di sicurezza, è prassi comune che gli IDS forniscano degli strumenti che consentano di raccogliere e di gestire agevolmente anche grandi quantità di informazioni di questo tipo, andando poi a confrontarle con ciò che viene rilevato.

Per quanto riguarda Bro esso dispone dell'**Intel Framework**, che ha come obiettivo primario quello di raccogliere e consumare dati di intelligence provenienti da fonti di diverso tipo, verificare eventuali corrispondenze (match) col traffico monitorato dall'IDS e in generale fornire

un'infrastruttura che renda il meccanismo di gestione dell'intelligence semplice ed efficiente, migliorando le prestazioni e l'utilizzo della memoria[32].

I dati utilizzati dal framework rappresentano principalmente parti singole di intelligence come un indirizzo IP o di posta elettronica, un hostname o un URL. Questi vengono forniti generalmente insieme ad una serie di metadati che portano ulteriori informazioni sull'indicatore, come ad esempio l'URL di un sito che è possibile consultare per avere notizie aggiuntive.

Per poter essere utilizzati all'interno dell'Intel Framework tali dati, che possono provenire dai siti e dalle community più diversi (Emerging Threats, AlienVault, ShadowServer ecc.), devono prima di tutto essere caricati su Bro e questo può essere fatto solo se sono strutturati secondo un formato ben preciso, che si mostra del tutto analogo a quello utilizzato per i log: un file di testo suddiviso in un certo numero di colonne o campi, separate da un singolo carattere di tabulazione. La figura sottostante mostra un esempio di file contenente dati di intelligence pronti per essere utilizzati dal framework:

```
#fields indicator       indicator_type  meta.source     meta.url        meta.do_notice  meta.if_in
1.182.117.119   Intel::ADDR     ciarmy  http://www.ciarmy.com/list/ci-badguys.txt       T       -
2.187.28.215    Intel::ADDR     ciarmy  http://www.ciarmy.com/list/ci-badguys.txt       T       -
2.229.117.159   Intel::ADDR     ciarmy  http://www.ciarmy.com/list/ci-badguys.txt       T       -
4.35.96.216     Intel::ADDR     ciarmy  http://www.ciarmy.com/list/ci-badguys.txt       T       -
5.79.69.204     Intel::ADDR     ciarmy  http://www.ciarmy.com/list/ci-badguys.txt       T       -
5.135.146.0     Intel::ADDR     ciarmy  http://www.ciarmy.com/list/ci-badguys.txt       T       -
5.135.240.133   Intel::ADDR     ciarmy  http://www.ciarmy.com/list/ci-badguys.txt       T       -
5.153.54.130    Intel::ADDR     ciarmy  http://www.ciarmy.com/list/ci-badguys.txt       T       -
5.254.101.69    Intel::ADDR     ciarmy  http://www.ciarmy.com/list/ci-badguys.txt       T       -
```

Figura 4.6: Un semplice file di intelligence

I primi due campi, come si può vedere, riportano per intero l'indicatore che si è interessati a ricercare e ne definiscono il tipo. Nel nostro caso si tratta tutti di indirizzi IP (*Intel::ADDR*), ma Bro è in grado di supportare molti altre tipologie di indicatori, come URL (*Intel::URL*), domini

(*Intel::DOMAIN*) o hash di file (*Intel::FILE_HASH*).

Accanto a questi sono poi ben visibili i metadati di cui si è già precedentemente accennato. Un menzione particolare spetta al campo *meta.do_notice*, il quale determina tramite un valore booleano se inviare o meno al Notice Framework ogni eventuale corrispondenza rilevata per l'indicatore a cui si riferisce, così che il sistema lanci una notice di avvertimento.

Una volta che l'intelligence si trova memorizzata in un file con esattamente questa struttura (esistono online anche diversi tool che permettono di formattare automaticamente qualsiasi dato per renderlo compatibile con Bro), essa dev'essere caricata nell'Intel Framework e per fare ciò ci si serve di un altro framework di base, l'Input Framework. Senza stare a dilungarsi troppo in dettagli tecnici, che è comunque possibile approfondire autonomamente nella documentazione ufficiale, è sufficiente specificare il percorso del file di intelligence aggiungendolo all'interno del set *Intel::read_files* che memorizza l'elenco di tutti i file di intelligence che verranno letti dall'Input Framework, similmente a come mostrato in figura:

```
redef Intel::read_files += {
        "/somewhere/feed1.txt",
        "/somewhere/feed2.txt",
};
```

Figura 4.7: Caricare i file di intelligence

A questo punto, caricata l'intelligence, è necessario preoccuparsi di informare il framework ogni volta che vengono rilevati nuovi dati, di modo tale che possano essere confrontati con l'intelligence alla ricerca di corrispondenze[32]. Per fare ciò tutti i componenti di Bro deputati all'analisi e al monitoraggio dei vari protocolli e comunicazioni devono essere istruiti

a inviare i dati visti all'Intel Framework: una tale funzionalità si trova già implementata di default in Bro attraverso una serie di script contenuti nella cartella *seen* e che richiamano la funzione Intel::seen, funzione che dichiara la scoperta di un certo dato (un header HTTP, un hash, una connessione ecc.) per confrontarlo con l'intelligence. Per implementare questa funzione, occorrerà perciò molto semplicemente caricare suddetti script nella load section del proprio script di policy, ovvero:

```
@load policy/frameworks/intel/seen
```

Figura 4.8: Caricare gli script della cartella *seen*

Contro ogni speranza, la stragrande maggioranza delle reti finirà per riscontrare delle corrispondenze tra i dati di intelligence caricati e il traffico monitorato ad indicare una possibile intrusione o attività indesiderata. Ogniqualvolta che questo accade l'Intel Framework genera un evento denominato appunto *Intel::match* [32]. Tale evento provvede a registrare la corrispondenza osservata creando una nuova voce all'interno di un apposito log, il file *intel.log*.

Anche in questo caso, però, come già abbiamo visto per altri event handler, si tratta solamente del comportamento predefinito, che l'utente ha la completa libertà di modificare scrivendo il proprio gestore per l'evento Intel::match all'interno di uno script, andando così a definire nuove azioni da effettuare ogni volta in cui viene rilevata una nuova corrispondenza.

5 Threat Intelligence e relazioni: sviluppo di un plugin per Bro

Dopo aver compreso l'importanza data al giorno d'oggi al tema della cybersecurity, studiato quelli che sono i principali sistemi di sicurezza, con un occhio di particolare riguardo agli IDS, esaminato tutte le grandi possibilità offerte da Bro e infine toccato con mano il crescente settore della threat intelligence, è giunto finalmente il momento di parlare del progetto realizzato presso Yoroi e che ha dato il "la" all'intera tesi.

Nel corso del capitolo viene prima di tutto fornita una panoramica sugli scenari di base che hanno portato alla realizzazione del programma in questione (paragrafo 5.1) e sono quindi definiti gli obiettivi da perseguire e le funzionalità richieste dall'applicativo (paragrafo 5.2). I paragrafi successivi elencano poi gli strumenti che si è scelto di utilizzare per la realizzazione del progetto (paragrafo 5.3) e descrivono l'architettura generale dell'elaborato (paragrafo 5.4). Infine viene fatto un accenno ai possibili futuri sviluppi del progetto verso nuove direzioni (paragrafo 5.5).

5.1 Lo scenario: relazioni nella Threat Intelligence

Fino ad ora nella nostra trattazione, parlando di intelligence, ci siamo concentrati unicamente su indicatori singoli: il sistema viene allertato quando si incontra un determinato indirizzo IP, un DNS, un URL, un hash file e così via. Questo rappresenta il grande limite degli IDS tradizionali

basati su firme, il fatto cioè che le varie signature utilizzate non consentano mai di analizzare un particolare flusso di dati ma semplicemente stabiliscano, a livello generale, che un host si è connesso a quell'IP, ha risolto quel DNS o ha usato un certo payload. Non è possibile, in altre parole, seguire il traffico di rete e il flusso dei dati ad alti livelli, in maniera specifica e dettagliata come molte applicazioni di sicurezza invece richiedono.

Per di più, se si fa uso solamente di indicatori singoli, è molto facile incorrere in una quantità elevata di falsi positivi. Può capitare ad esempio che un indirizzo IP non sia da considerarsi sempre malevolo ma solamente nel caso in cui compaia in corrispondenza di un particolare DNS: in situazioni come questa, basandosi unicamente su quanto visto finora, non esiste un meccanismo in grado di operare una tale distinzione, e potendo contare unicamente sulla propria intelligence qualsiasi IDS si limiterà a lanciare un alert ogniqualvolta l'IP sospetto viene rilevato, generando così frequenti falsi allarmi.

Per evitare situazioni di questo tipo serve stabilire delle **relazioni tra indicatori**, collegamenti interni che permettano, ad esempio, che la connessione ad un determinato indirizzo venga segnalata solo se posta in relazione ad un certo hostname, che uno specifico hash sia considerato malevolo solamente se il file cui appartiene è stato scaricato da uno specifico URL e così via.

I frangenti in cui una funzionalità simile risulta di grande importanza sono innumerevoli. Si consideri, per esempio, un malware che sta girando in incognito su un host e che ad un certo punto cerca di contattare l'infrastruttura *C&C* che lo controlla. Il sistema, per cercare di evitare che

questo avvenga, fa uso della threat intelligence, la quale segnala che un certo hostname è uno dei domini di C&C per quel tipo di malware e che questo risiede su un server con un determinato indirizzo IP. Trattandosi di un provider, tale server ospiterà al suo interno molti siti differenti, di cui il dominio malevolo associato alla C&C del nostro malware risulterà solamente uno dei tanti. Ciò significa che una connessione all'IP riportato nell'intelligence non sempre è da considerasi malevola, ma solamente se seguita alla risoluzione DNS del dominio di C&C: solo in questo caso, se entrambi gli indicatori (hostname e indirizzo IP) sono presenti, i sistemi di sicurezza della macchina dovranno dare l'allarme.

Per poter fare tutto questo è quindi necessario trovare un modo per rappresentare e formalizzare relazioni simili e, come abbiamo già visto, ciò non è possibile con gli IDS tradizionali che nella loro implementazione di base non supportano in alcuna maniera una tale funzionalità. Ma, fortunatamente per noi, Bro non è un IDS come tutti gli altri e grazie alle possibilità offerte dal suo linguaggio di scripting permette di trovare una soluzione al problema in maniera estremamente flessibile e al tempo stesso efficiente. Ed è proprio su quest'ambito che si concentra il progetto analizzato nei successivi paragrafi.

5.2 Obiettivi e funzionalità del progetto

Il progetto prevede la realizzazione di uno script Bro che consenta l'applicazione di regole di threat intelligence create ad hoc per mettere in relazione tra loro più indicatori, laddove una semplice regola signature-based non sia sufficiente. L'obiettivo finale è quello di fornire agli utenti uno

strumento (un *plugin*) capace di integrare le informazioni di intelligence comunemente utilizzate nelle aziende come Yoroi e che fanno largo uso delle relazioni, coi servizi di monitoraggio e rilevamento forniti da Bro, servendosi del suo linguaggio di scripting.

Gli step che in linea di massima si sono seguiti durante il processo di sviluppo e che hanno portato alla realizzazione delle funzionalità principali del plugin sono i seguenti:

- Creazione di uno script che, sfruttando tutta la potenza espressiva data dall'Intel Framework, lanci una nuova notice per ogni corrispondenza tra quanto rilevato e l'intelligence caricata. Si parla in questa fase ancora di intelligence con indicatori singoli: le relazioni, per il momento, non sono prese in considerazione;
- Analisi delle performance dello script realizzato e messa in luce dei suoi limiti, derivanti dall'utilizzo di intelligence con indicatori singoli;
- Definizione del modo più conveniente per rappresentare le relazioni tra due indicatori e poterle poi utilizzare all'interno di Bro;
- Creazione di uno script che lanci una notice differente qualora nello stesso “flusso logico” compaiano entrambi gli indicatori presenti in una relazione prestabilita;
- Analisi delle prestazioni e migliorie al codice;
- Valutazione su possibili futuri sviluppi.

5.3 Strumenti utilizzati

Per la realizzazione del progetto è stato utilizzato l'intrusion detection system **Bro** e in particolare il suo linguaggio di scripting, il **Bro scripting language**. Per funzionare, tuttavia, Bro necessita di una piattaforma Unix (Linux, FreeBSD o Mac OS X). Potendo disporre solamente di ambienti Windows, è stata perciò creata una macchina virtuale con **VMWare** in cui eseguire **Ubuntu 16.10**.

Infine, per la scrittura del codice dello script, come editor di programmazione si è scelto di adottare **Sublime**, che dispone di un comodo plugin di evidenziazione della sintassi di Bro.

5.4 La struttura del progetto

Il progetto si compone di due script differenti: il primo è un semplice script di supporto chiamato *intel_setup* che si occupa della configurazione generale del plugin, mentre il secondo, *intel_check_relations*, costituisce il programma vero e proprio.

Entrambi gli script sono analizzati nel dettaglio in questo paragrafo, mentre in quello successivo viene fornito un tipico caso d'uso, che mostra in maniera pratica come poter utilizzare il plugin realizzato e i risultati che esso permette di ottenere.

5.4.1 Lo script di configurazione: *intel_setup*

Il nome dello script è già di per sé piuttosto esplicativo: si tratta infatti di uno script di setup, all'interno del quale vengono impostate tutte le variabili e le dipendenze che saranno poi richieste in fase di monitoraggio per

controllare la presenza di relazioni tra indicatori. La figura seguente mostra il corpo dello script nella sua interezza:

```
# Setup script where all the variables and dependencies needed to check relations between indicators are setted

@load base/utils/site
@load frameworks/intel/seen
@load frameworks/intel/do_notice

module IntelCorrelation;

export {

    # Definition of local net for generating correlations
    redef Site::local_nets += { 192.168.0.0/16 };

    # Loads needed intelligence data from file into Intel Framework
    redef Intel::read_files += {
    fmt("intel_for_relations.intel")
    };

    # Loads relations between indicators
    const rel_file = fmt("relations.txt", @DIR) &redef;

    # Defines record types for the Input Framework
    # The names of the fields in the record definition must correspond to the column names
    # listed in the '#fields' line of the rel_file
    type Idx: record {
        first_ind: string;
    };

    type Val: record {
        second_ind: string;
    };
}
```

Figura 5.1: Lo script *intel_setup*

Per prima cosa, come si può vedere, la *load section* si preoccupa di caricare gli script della cartella *seen,* che sono responsabili di inviare all'Intel Framework i vari dati rilevati, di modo tale che possano poi essere confrontati con l'intelligence alla ricerca di eventuali corrispondenze da segnalare (vedi paragrafo 4.4). Ad essere caricato all'interno dello script è poi anche il file *do_notice*, il quale segnala all'IDS di lanciare una nuova notice ogniqualvolta viene trovata una corrispondenza.

Il resto dello script si occupa quindi di ridefinire diverse costanti e dichiarare variabili che saranno poi utilizzate in tutto il corso del programma. Ciò viene fatto all'interno di una lunga *export section* in modo tale che, anche all'interno dell'altro script di cui si compone il progetto, sia possibile accedere a tutto ciò che è stato qui definito, evitando inutili

ripetizioni di codice e spreco di risorse.

Analizziamo ora nel dettaglio tali dichiarazioni:

- ***redef Site::local_nets:*** si tratta di un set che si trova già definito negli script base di Bro e che indica quali reti o sottoreti in fase di analisi sono da considerarsi locali, con tutto ciò che ne consegue. Trattandosi di una variabile ridefinibile è possibile modificarla liberamente aggiungendo nuovi elementi al set con l'indirizzo della propria rete locale, come in questo caso;
- ***redef Intel::read_files:*** come già visto nel paragrafo 4.4, è l'elenco dei file di intelligence che vengono letti dall'Input Framework e caricati su Bro. Nel nostro caso tutta l'intelligence necessaria è contenuta in un unico file ma è anche possibile suddividerla tranquillamente su più documenti, se lo si ritiene opportuno. L'importante è che essa rispetti il formato standard stabilito per l'Intel Framework di cui si è già parlato precedentemente (paragrafo 4.4):

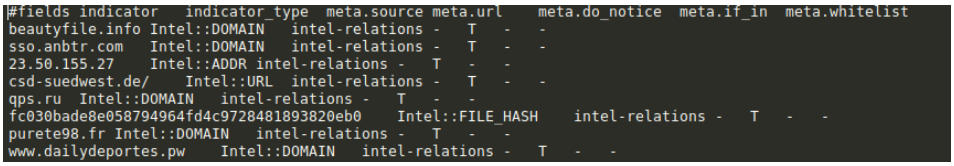


Figura 5.2: Un file di intelligence

- ***const rel_file:*** il nome del file contenente le relazioni tra indicatori che ci interessa considerare. Trattandosi di un concetto nuovo che nella sua implementazione di base Bro non supporta, il come memorizzare queste relazioni è stato oggetto di riflessioni e discussioni durante la fase di progettazione. Alla fine si è optato per

una soluzione che fosse compatibile con l'Input Framework e consentisse quindi di caricare le relazioni in una tabella da poter utilizzare poi nel nostro script. In linea col tipo di file che Bro è in grado di leggere, si è scelto perciò di memorizzare le relazioni in un file di testo che presenta la medesima struttura già vista per i log e l'intelligence, come nella figura seguente:

```
#fields first_ind   second_ind
ssl.wlogin.qq.com   1.1.1.1
beautyfile.info 81.171.14.67
162.87.53.202-in-addr.arpa-nettlinx.com 195.22.26.248
seen-on-screen.thewhizmarketing.com 23.21.207.39
162.87.53.202-in-addr.arpa-nettlinx.com 195.22.26.248
qps.ru  193.124.118.141
37.235.56.180   57ac7884666cd7497f74c12cd43e4c3c00379e4d2316939d2b73224bb787986e
131.107.255.255 f985eb666eb57629cecd3a194d93a3c90174583ede433610c11ac8cf57ab7501
131.107.255.255 bbc391e811347e37a7713d6cf191735ae9c8b15614d4c4aed84d63ff2a37ee37
business-kniga.com/ 194.28.84.75
secretsofmoneyandriches.com/_documents/ii.php?rand=13InboxLightaspxn.1774256418 97.74.144.149
cinemafreetickets.com/  162.255.119.249
achatordijon.fr/    94.23.10.18
162.87.53.202-in-addr.arpa-nettlinx.com 195.22.26.248
```

Figura 5.3: Relazioni tra indicatori

- ***type Idx: record / type Val: record:*** per poter leggere un file e caricarlo in una tabella di Bro occorre definire questi due tipi di record. Il primo contiene i tipi e i nomi delle colonne che andranno a costituire la chiave di ricerca della tabella; il secondo memorizza invece i tipi e i nomi delle colonne che vanno a costituire i valori contenuti in tale tabella. Nel nostro caso si tratta semplicemente di due stringhe che rappresentano gli indicatori messi in relazione.

5.4.2 Il cuore del progetto: lo script *intel_check_relations*

Veniamo ora allo script principale del plugin, quello che realizza tutte le funzionalità che si sono prefissate in fase di progettazione. Il codice completo, per via della sua lunghezza, viene fornito in appendice (Appendice A).

Strutturalmente lo script può essere suddiviso in **tre sezioni** distinte: una prima parte di **inizializzazione e setup** generale, **l'event handler *Intel::match*** (e relative funzioni di supporto), richiamato ogni volta che viene rilevata una corrispondenza, e infine la **funzione *check_relations***, la quale analizza gli indicatori rilevati alla ricerca di relazioni note per poi lanciare, in caso affermativo, una notice.
Analizziamo ognuna di queste tre sezioni più nel dettaglio.

L'inizializzazione dello script

Ciò che occorre fare prima di ogni altra cosa è caricare all'interno di Bro il file contenente le relazioni da utilizzare e il cui filepath è stato già indicato nello script di setup (paragrafo 5.4.1). Per far questo ci si serve dell'Input Framework che, facendo uso della funzione *Input::add_table*, consente di leggere file di testo esterni e memorizzarli in una tabella utilizzabile poi all'interno del programma, tramite poche semplici righe di codice come in figura:

```
# The table where all records read by the Input Framework from relation files are stored
global relations: table[string] of Val = table();

event bro_init() {
  Input::add_table([$source=rel_file, $name="relations", $idx=Idx, $val=Val, $destination=relations, $mode=Input::REREAD])
  Input::remove("relations");
}
```

Figura 5.4: Caricare il file delle relazioni su Bro

Tale funzione indica all'Input Framework di aprire un nuovo flusso di input (*input stream*) chiamato *relations* attraverso il quale leggere il file *rel_file* definito all'interno dello script *intel_setup*, andando poi a memorizzarlo dentro alla tabella dichiarata poco sopra. Denominata anche in questo caso *relations*, tale tabella deve essere per forza una tabella di record dello stesso

tipo dei valori contenuti nel file in cui sono presenti le relazioni.
L'ultima riga si preoccupa poi di rimuovere il flusso di input una volta che non serve più.
La funzione *Input::add_table*, come si può notare, è stata richiamata all'interno di uno specifico event handler chiamato *bro_init*: si tratta di un evento generato automaticamente ogni volta che una istanza di Bro viene mandata in esecuzione e al suo interno è quindi buona norma inserire tutto il codice di inizializzazione e configurazione che si vuole venga eseguito una sola volta appena l'IDS viene fatto partire, come avviene appunto in questo caso.

L'event handler Intel::match

Come già accennato nel paragrafo 4.4, *Intel::match* è il nome dato al gestore dell'evento che viene generato ogni volta in cui Bro rileva una corrispondenza tra l'intelligence caricata e il traffico monitorato.
In questo script l'event handler provvede ad analizzare la corrispondenza rilevata, andando ad individuare tutti gli indicatori coinvolti e aggiungendoli ad una tabella che verrà poi utilizzata per segnalare la presenza di eventuali relazioni. Tale tabella, chiamata *alert_correlation_state*, è indicizzata da un indirizzo che rappresenta l'host coinvolto in una corrispondenza, e per ognuno di questi contiene un set di stringhe che memorizzano tutti gli indicatori associati a quell'host.
Per aggiungere nuovi elementi a questa tabella l'handler si serve di una funzione di supporto, *add_indicator*, che viene però chiamata solamente se l'host in questione appartiene alla propria rete locale, un controllo questo che viene fatto tramite la funzione *correlation_is_local* esaminando il valore

della variabile *Site::local_nets*, definita nello script di setup (vedi paragrafo 5.4.1). La tabella e le funzioni di cui si è parlato finora sono mostrate nella figura seguente:

```
# The table where host addresses and indicator data are dynamically stored.
global alert_correlation_state: table[addr] of set[string] &create_expire=alert_correlation_interval &expire_func=check_relations;

# Function to add host and indicator data to the table above.
function add_indicator(a: addr, ind: string)
{
  if (a !in alert_correlation_state) {
    alert_correlation_state[a] = set();
  }
  if (a in alert_correlation_state) {
    add alert_correlation_state[a][ind];
  }
}

# Function to check if hosts are part ot the local network
function correlation_is_local(a: addr): bool
{
if (Site::is_local_addr(a) && a !in alert_correlation_whitelist)
  return T;
else return F;
}
```

Figura 5.5: Aggiunta di indicatori alla tabella

Guardando la figura, si nota subito la presenza di due particolari attributi in corrispondenza della definizione della tabella: *create_expire* e *expire_func*. Questi vengono utilizzati per definire una sorta di tempo di vita per ciascun elemento della tabella dal momento in cui vengono inseriti nel container (*create_expire*). Una volta che il tempo è scaduto l'elemento in questione viene eliminato, ma prima che questo avvenga viene chiamata la funzione specificata nell'attributo *expire_func.* Nel nostro caso si tratta della funzione *check_relations* che sarà analizzata dettagliatamente nel paragrafo successivo, ma che, come si può intuire dal nome, si occupa di controllare tutti gli indicatori associati ad un certo indirizzo alla ricerca di relazioni. Nella pratica ciò significa che lo script, una volta mandato in esecuzione, si preoccuperà di richiamare continuamente tale funzione, segnalando in tempo reale ogni relazione riscontrata. L'intervallo di tempo in cui questo avviene è definito dal valore della variabile *alert_correlation_interval*, dichiarata nella load section dello script: il valore impostato è in questo caso

pari a 1 msec (in modo tale che gli indicatori vengano analizzati pressoché in tempo reale), ma può essere scelto liberamente dall'utente a seconda degli utilizzi.

Nella figura sottostante è mostrato il restante codice dell'event handler, dove si può notare come la funzione per aggiungere un indicatore alla tabella venga chiamata solamente per gli host appartenenti alla rete che, in fase di setup, si è indicata come locale:

```
# Processing for indicators
event Intel::match(s: Intel::Seen, items: set[Intel::Item])
{
  # In order to detect intel correlations, we have to examine the connection involve
  # However if the indicator is file related(file hashes, file names etc...), connec
  if(s?$f) {
    if(s$f?$conns && |s$f$conns| == 1) {
      for(cid in s$f$conns) {
        s$conn = s$f$conns[cid];
      }
    }
  }

  # Stop processing if connection data does not exist.
  if(! s?$conn) {
    return;
  }

  # Check if the originator or responder is in the local network.
  if(correlation_is_local(s$conn$id$orig_h))
  {
    add_indicator(s$conn$id$orig_h,s$indicator);
  }
  if(correlation_is_local(s$conn$id$resp_h))
  {
    add_indicator(s$conn$id$resp_h,s$indicator);
  }
}
```

Figura 5.6: L'event handler *Intel::match*

La funzione *check_relations*

Vediamo infine come si struttura la funzione *check_relations* che, come già detto, viene richiamata dallo script ciclicamente ogni tot secondi e si preoccupa di rilevare la presenza di eventuali relazioni tra tutti gli indicatori associati ad un certo host. Le successive figure mostrano il codice della funzione nella sua interezza:

```
# Function to build notices from seen indicators
function check_relations(t: table[addr] of set[string], idx: addr): interval
{
  # Local utility variables to store information about the correlation
  local found_first: bool;
  local found_second: bool;
  local rel_second_ind: string;
  local message: string;
  local sub_msg: string;
  local ind1_msg: string;
  local ind2_msg: string;

  local cnt = |t[idx]|;
```

Figura 5.7: La funzione *check_relations* (1)

```
# Continues only if more than one indicator was seen and the threshold has been crossed
if(cnt > 1 && cnt >= alert_correlation_indicator_threshold) {

  #CHECK RELATIONS
  for(rel_first_ind in relations) {
    rel_second_ind = relations[rel_first_ind]$second_ind;
      found_first = F;
      found_second = F;
      for(ind in t[idx]) {
          if(ind == rel_first_ind) {
            found_first = T;
            ind1_msg = ind;
            if(found_first && found_second) {
              break;
            }
          }
        else {
          if(ind == rel_second_ind) {
            found_second = T;
            ind2_msg = ind;
            if(found_first && found_second) {
              break;
            }
          }
        }
      }
      if(found_first && found_second) {
        #A correlation has been found and a new special Notice is raised
        message = fmt("Host %s was involed with multiple indicators that are part of a relation", idx);
        sub_msg = fmt("Indicator: %s Indicator: %s", ind1_msg, ind2_msg);
        NOTICE([$note=Correlation_Alert,
            $src=idx,
            $msg=message,
            $sub=sub_msg,
            $identifier=cat(idx, sub_msg)]);
        print message;
        print sub_msg;
      }
  }
```

Figura 5.8: La funzione *check_relations* (2)

Tralasciando la parte iniziale, in cui viene dichiarata tutta una serie di variabili locali che saranno utilizzate nel seguito, il suo funzionamento è estremamente semplice e al tempo stesso efficace: attraverso un doppio

ciclo lo script attraversa ed esamina simultaneamente sia il file di relazioni che la tabella *alert_correlation_state*; nel caso in cui poi ad uno stesso host corrispondano più indicatori per i quali nel file *relations* è indicata una relazione, allora la funzione lancerà una specifica notice di allarme ad indicare che è stata riscontrata una combinazione di indicatori. Tale notice porta con sé diverse informazioni utili, come l'indirizzo IP dell'host e gli indicatori coinvolti nella correlazione, e viene resa disponibile all'amministratore e agli analisti tramite il consueto file *notice.log.*

Tornando al codice mostrato, si può notare la presenza di un if che precede tutta la procedura di cui si è appena parlato, andando ad effettuare un controllo sul numero di indicatori riscontrati: questi devono infatti essere sempre maggiori o uguali ad una certa soglia, definita dalla variabile *alert_correlation_treshold*, la quale determina il numero minimo di indicatori che devono essere stati rilevati dall'IDS prima che si possa dare inizio alla procedura di ricerca delle corrispondenze.

Nel caso del nostro progetto questa soglia è stata impostata a 2, dato che consideriamo relazioni che coinvolgono due soli indicatori per volta. Per l'utente è però possibile aumentare liberamente questo numero a seconda delle proprie esigenze: in tal caso, però, è poi ovviamente necessario anche modificare il file contenente le relazioni, in modo tale che ognuna di esse contenga il numero di indicatori desiderato.

5.5 Un esempio di utilizzo

Questo progetto è stato realizzato per funzionare sia monitorando il traffico in **tempo reale** su una certa interfaccia di rete sia analizzando **file pcap** in cui è stata precedentemente salvata una traccia del traffico: i risultati ottenuti in entrambe le modalità sono esattamente gli stessi, con lo script che si occupa di generare file di log e lanciare notice mano a mano che l'IDS esamina il flusso dei dati o analizza il pcap.

Eseguire lo script è al momento possibile solamente da linea di comando tramite shell, sia per le attività di analisi in tempo reale che per quelle offline con i pcap. In futuro si prevede però di rendere il tutto molto più comodo e snello con la realizzazione di un vero e proprio plugin in grado di funzionare in maniera a sé stante, obiettivo che non rientra nei tempi previsti per questa tesi (vedi capitolo 6 sui futuri sviluppi del progetto).

I comandi utilizzati per mandare in esecuzione il programma in entrambe le modalità presentate sono mostrati nelle due figure sottostanti:

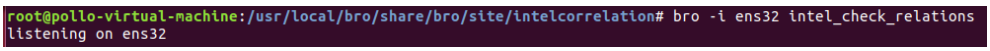


Figura 5.9: Utilizzo dello script in tempo reale

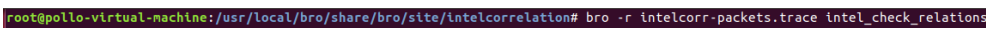


Figura 5.10: Utilizzo dello script con file pcap

Nel primo caso il comando *bro -i ens32* indica all'IDS di mettersi in ascolto sull'interfaccia di rete denominata *ens32* (per conoscere come viene chiamata la propria è sufficiente eseguire il comando *ifconfig* su qualunque terminale), monitorando tutto il traffico sia in entrata che in uscita, producendo i consueti file di log in output e applicandovi poi lo script di

policy specificato, ovvero il nostro *intel_check_relations*.

Se si vogliono invece analizzare in "modalità offline" pacchetti che sono già stati precedentemente catturati e scritti su un file pcap, il comando da utilizzare è *bro -r* seguito dal nome del file che si vuol leggere e, anche in questo caso, dallo script di policy di applicare.

Quale che sia la soluzione scelta, l'output risultante dall'esecuzione dello script *intel_check_relations* sarà nel file *notice.log* che, oltre alle normali notice lanciate da Bro quando viene rilevata un'attività sospetta, conterrà anche delle notice più specifiche ad indicare la presenza di una certa combinazione di indicatori. Un esempio di file di log prodotto in questo modo è visibile nella figura seguente:

Figura 5.11: L'output dello script

Per comodità nell'immagine vengono mostrati solamente i campi *note*, indicante il tipo di notice lanciata, e *msg* che contiene un breve messaggio esplicativo della notice. Già da qui si può notare che, oltre alle consuete notice generate dal Notice Framework, ne compaiono anche due "speciali" di tipo *IntelCorrelation::Correlation_Alert*: si tratta del tipo di notice definito e lanciato all'interno dello script del nostro progetto e che fornisce agli analisti tutte le informazioni necessarie per identificare una specifica combinazione di indicatori.

6 Conclusione e futuri sviluppi

I cyber attacchi rappresentano ad oggi una seria minaccia, non solamente per le aziende ma in generale per chiunque disponga di un accesso a Internet: senza neanche la necessità di utilizzare strumenti particolarmente costosi, un aggressore può causare danni gravissimi ad un sistema, provocando la perdita di informazioni sensibili o l'interruzione dei suoi servizi.

Cercare di proteggere i propri dispositivi e dotarsi dei più efficaci sistemi di sicurezza disponibili è diventata quindi una necessità sempre più urgente che specialmente negli ultimi anni ha portato sia le grandi aziende che quelle più piccole a servirsi di firewall, antivirus, IDS e moltissimi altri strumenti che operano nel settore della cybersecurity.

La sicurezza totale, tuttavia, è qualcosa che in informatica non può mai essere garantita: per quanto infatti si possa proteggere i propri sistemi, un attaccante troverà sempre il modo di bypassare i controlli e infiltrarsi all'interno della rete. Per questo motivo risulta sempre più di fondamentale importanza riuscire a raccogliere un ampio ventaglio di informazioni di threat intelligence sugli attacchi che minacciano un sistema, in modo tale da porsi in una situazione di vantaggio sull'attaccante e poter così agire in tempo utile.

Le tecniche di difesa basate sull'intelligence fanno generalmente affidamento su liste di singoli indicatori e portano alla definizione di filtri e policy da applicare ai sistemi di sicurezza di un'azienda. Nella maggior parte

dei casi, tuttavia, queste si dimostrano piuttosto inaffidabili: gli indicatori possono risultare in realtà falsi positivi, fuorvianti oppure semplicemente poco informativi e quindi di scarsa utilità.

Il progetto realizzato per questa tesi ha cercato di porre una soluzione al suddetto problema, introducendo delle combinazioni tra indicatori molto semplici ma che permettono comunque di migliorare in maniera esponenziale la qualità delle attività di intrusion detection di un sistema. Sfruttando tutta la flessibilità di Bro si è introdotta la possibilità di mettere in relazione tra loro più elementi di threat intelligence, permettendo così di definire vere e proprie logiche e policy di sicurezza di complessità arbitraria. Si è proposto, in altre parole, un nuovo tentativo di approccio alla cybersecurity basato sull'intelligence che cerca di superare i limiti delle tecniche tradizionali.

Tutto ciò rappresenta in realtà solamente l'inizio di una lunga strada, costellata da tante possibili migliorie al codice, estensioni che prevedono l'aggiunta di ulteriori funzionalità e la definizione di nuove combinazioni tra indicatori più complesse ed elaborate. Attualmente lo script risulta utilizzabile soltanto da linea di comando e uno dei più naturali sviluppi del progetto è quindi quello di costruire un vero e proprio plugin in grado di essere eseguito e funzionare autonomamente. Piuttosto scomoda inoltre è la necessità di dover caricare manualmente di volta in volta l'intelligence e le corrispondenti relazioni all'interno del programma: un altro obiettivo futuro può quindi essere quello di garantire il pieno supporto ai vari standard per la condivisione di informazioni di threat intelligence che si sono affermati negli ultimi anni, come ad esempio STIX. Un tale supporto contribuirebbe notevolmente ad automatizzare il processo di scambio di intelligence, rendendo quindi molto più efficiente lo script e andando ad aumentarne la

portabilità, facendo sì che possa essere utilizzato da qualsiasi azienda che ha scelto di fare uso di un determinato standard.

7 Appendice A

```
@load base/frameworks/notice
@load base/frameworks/intel
@load frameworks/files/hash-all-files
@load site/intelcorrelation/intel_setup

module IntelCorrelation;

# The type of the notice that will be raised when a new indicator correlation
is seen
redef enum Notice::Type += {
  Correlation_Alert
};

#Amount of time to watch for indicator correlations
const alert_correlation_interval = 1msec &redef;

# Number of indicators to see before alerting on correlations
const alert_correlation_indicator_threshold = 2 &redef;

# Hosts and servers to exclude from correlations
const alert_correlation_whitelist: set[addr] = {} &redef;

# The table where all records read by the Input Framework from relation
files are stored
global relations: table[string] of Val = table();
```

7 Appendice A

```
# Function to build notices from seen indicators
function check_relations(t: table[addr] of set[string], idx: addr): interval {
# Local utility variables to store information about the correlation
  local found_first: bool;
  local found_second: bool;
  local rel_second_ind: string;
  local message: string;
  local sub_msg: string;
  local ind1_msg: string;
  local ind2_msg: string;
  local cnt = |t[idx]|;
# Continues only if more than one indicator was seen and the threshold has
been crossed
  if(cnt > 1 && cnt >= alert_correlation_indicator_threshold) {
    #CHECK RELATIONS
    for(rel_first_ind in relations) {
      rel_second_ind = relations[rel_first_ind]$second_ind;
        found_first = F;
        found_second = F;
        for(ind in t[idx]) {
           if(ind == rel_first_ind) {
             found_first = T;
             ind1_msg = ind;
             if(found_first && found_second) {
               break;
             }
           }
```

```
        else {
          if(ind == rel_second_ind) {
            found_second = T;
            ind2_msg = ind;
            if(found_first && found_second) {
              break;
            }
          }
        }
      }
      if(found_first && found_second) {
        #A correlation has been found and a new special Notice is raised
        message = fmt("Host %s was involed with multiple indicators that are
                        part of a relation", idx);
        sub_msg = fmt("Indicator: %s Indicator: %s", ind1_msg, ind2_msg);
        NOTICE([$note=Correlation_Alert,
          $src=idx,
          $msg=message,
          $sub=sub_msg,
          $identifier=cat(idx, sub_msg)]);
        print message;
        print sub_msg;
      }
    }
  }
return 0sec;
}
```

7 Appendice A

```
# The table where host addresses and indicator data are dynamically stored.
global alert_correlation_state: table[addr] of set[string]
&create_expire=alert_correlation_interval &expire_func=check_relations;

# Function to add host and indicator data to the table above.
function add_indicator(a: addr, ind: string)
{
 if (a !in alert_correlation_state) {
   alert_correlation_state[a] = set();
 }
 if (a in alert_correlation_state) {
   add alert_correlation_state[a][ind];
         }
}

# Function to check if hosts are part ot the local network
function correlation_is_local(a: addr): bool
{
if (Site::is_local_addr(a) && a !in alert_correlation_whitelist)
 return T;
else return F;
}
# Processing for indicators
event Intel::match(s: Intel::Seen, items: set[Intel::Item])
{
 # In order to detect intel correlations, we have to examine the connection
involved in the matching.
```

```
# However if the indicator is file related(file hashes, file names etc...),
connection data must be extracted from the file record
 if(s?$f) {
  if(s$f?$conns && |s$f$conns| == 1) {
   for(cid in s$f$conns) {
    s$conn = s$f$conns[cid];
   }
  }
 }
 # Stop processing if connection data does not exist.
 if(! s?$conn) {
  return;
 }
 # Check if the originator or responder is in the local network.
 if(correlation_is_local(s$conn$id$orig_h))
        {
                add_indicator(s$conn$id$orig_h,s$indicator);
        }
 if(correlation_is_local(s$conn$id$resp_h))
        {
                add_indicator(s$conn$id$resp_h,s$indicator);
        }
}
event bro_init() {
 Input::add_table([$source=rel_file, $name="relations", $idx=Idx,
$val=Val, $destination=relations, $mode=Input::REREAD]);
 Input::remove("relations"); }
```

8 Bibliografia

[1] Symantec, Symantec Internet Security Threat Report, 2017, https://www.symantec.com/security-center/threat-report

[2] Zafirah Salim, Imperva: malicious traffic and web attacks have significantly increased, 2014, https://www.computerworld.com.my/resource/security/malicious-traffic-and-web-attacks-have-significantly-increased-imperva/

[3] Kelly Batke, 7 Types of Cyber Crimes and Criminals, 2011, http://www.faronics.com/news/blog/7-types-of-cyber-criminals/

[4] SANS Institute, IT Security Spending Trends, 2017, https://www.sans.org/reading-room/whitepapers/analyst/security-spending-trends-36697

[5] Charles P. Pfleeger, Shari Lawrence Pfleeger, *Security in computing 4th edition*, 2006

[6] TechTerms, Malware definition, 2017, https://techterms.com/definition/malware

[7] PuntoInformatico, Sicurezza, in una classifica i bug più pericolosi, 2009, http://punto-informatico.it/2523404/PI/News/sicurezza-una-classifica-bug-piu-pericolosi.aspx

8 Bibliografia

[8] Wikipedia, Computer security, 2017, https://en.wikipedia.org/wiki/Computer_security

[9]Charles P. Pfleeger, Shari Lawrence Pfleeger, *Security in computing 4th edition*, 2006

[10] Wikipedia, Firewall, 2017, https://it.wikipedia.org/wiki/Firewall

[11] CCM, Firewall, 2016, http://it.ccm.net/contents/568-firewall

[12] , Firewall limitations, 2008, http://www.sandiegopchelp.com/firewall-limitations/

[13] Judy Thompson-Melanson, Firewall Evolution from Packet Filter to Next Generation, 2014, http://www.juniper.net/documentation/en_US/learn-about/LA_FIrewallEvolution.pdf

[14] Wikipedia, Antivirus, 2017, https://it.wikipedia.org/wiki/Antivirus

[15] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung, *Intrusion detection system: A comprehensive review*, 2012

[16] Wikipedia, Intrusion Detection System, 2017, https://en.wikipedia.org/wiki/Intrusion_detection_system

[17] Wikipedia, Intrusion Prevention System, 2017, https://it.wikipedia.org/wiki/Intrusion_prevention_system

[18] Daniele Bellavista, *ICT Security: defence strategies against targeted attacks*, 2012-2013

[19] Camilo Gutiérrez Amaya, IDS, Firewall and Antivirus: what you need to have installed?, 2015, https://www.welivesecurity.com/2015/04/30/ids-firewall-antivirus-need-installed/

[20] Fabio Tonacci, Cybercrime, il web sotto attacco per banche, aziende e Stati un conto da 575 miliardi, 2015, http://www.repubblica.it/economia/affari-e-finanza/2015/11/09/news/cybercrime_il_web_sotto_attacco_per_banche_aziende_e_stati_un_conto_da_575_miliardi-127010094/

[21] Repubblica, Cybersecurity: danni da 400 mld l'anno, il mercato della difesa informatica ne varrà 170 nel 2020, 2015, http://www.repubblica.it/economia/finanza/2015/11/17/news/cybersecurity_etf_sicurezza-127545099/

[22] Joe Schreiber, Open Source Intrusion Detection Tools: A Quick Overview, 2014, https://www.alienvault.com/blogs/security-essentials/open-source-intrusion-detection-tools-a-quick-overview

[23] CISCO, Snort website, 2017, https://www.snort.org/

[24] OISF, Suricata website, 2017, https://suricata-ids.org/news/

[25] Bro Team, Why choose Bro?, 2017, https://www.bro.org/why_choose_bro.pdf

[26] Bro Team, Bro 2.5 documentation: Introduction, 2017, https://www.bro.org/sphinx/intro/index.html

[27] Bro Team, Bro 2.5 documentation: Bro logging, 2017, https://www.bro.org/sphinx/logs/index.html

[28] 2017, Bro 2.5 documentation: Quick Start Guide, Bro Team, https://www.bro.org/sphinx/quickstart/index.html

[29] Bro Team, Bro 2.5 documentation: Writing Bro Scripts, 2017, https://www.bro.org/sphinx/scripting/index.html

[30] Bro Team, Bro 2.5 documentation: File Analysis, 2017, https://www.bro.org/sphinx/frameworks/file-analysis.html

[31] Bro Team, Bro 2.5 documentation: Input Framework, 2017, https://www.bro.org/sphinx/frameworks/input.html

[32] Bro Team, Bro 2.5 documentation: Intelligence Framework, 2017, https://www.bro.org/sphinx/frameworks/intel.html

[33] Bro Team, Bro 2.5 documentation: Logging Framework, 2017, https://www.bro.org/sphinx/frameworks/logging.html

[34] Bro Team, Bro 2.5 documentation: Notice Framework, 2017, https://www.bro.org/sphinx/frameworks/notice.html

[35] Secure Works, Secure Works Global Threat Intelligence, 2017, https://www.secureworks.com/capabilities/threat-intelligence/global-threats

[36] Dark Space Blogspot, Cos'è la Cyber Threat Intelligence? Sicurezza aziendale, 2016, http://darkwhite666.blogspot.it/2016/09/cose-la-cyber-threat-intelligence.html

[37] TechTarget, Threat Intelligence definition, 2015, http://whatis.techtarget.com/definition/threat-intelligence-cyber-threat-intelligence

[38] Wikipedia, Cyber Threat Intelligence, 2017, https://en.wikipedia.org/wiki/Cyber_threat_intelligence

[39] Darktrace, Sicurezza aziendale, cyber intelligence o threat Intelligence?, 2016, https://www.wired.it/attualita/tech/2016/01/22/sicurezza-aziendale-cyber-intelligence-threat-intelligence/

[40] Wikipedia, Open Source Intelligence, 2017, https://en.wikipedia.org/wiki/Open-source_intelligence

[41] Kriptia - ICT Security Magazine, OSINT (Open Source INTelligence) tra metodologia e funzione vitale per le aziende, 2016, https://www.ictsecuritymagazine.com/articoli/osint-open-source-intelligence-tra-metodologia-e-funzione-vitale-per-le-aziende/

[42]Jeffrey T. Richelson, *The U.S. Intelligence Community 7th edition*, 2016

[43] Koen Van Impe, How STIX, TAXII and CybOX Can Help With Standardizing Threat Information, 2015, https://securityintelligence.com/how-stix-taxii-and-cybox-can-help-with-standardizing-threat-information/

[44] OASIS, About STIX, 2017, https://oasis-open.github.io/cti-documentation/

8 Bibliografia

[45] OASIS, STIX specifications, STIX Version 2.1, part 1: STIX Core Concepts, 2017, https://docs.google.com/document/d/1HRVFn2kAxBOTMbEb3KRu8tjMoHm-KRAI-2R8CTzGil4/edit

[46] OASIS, TAXII specifications, TAXII Version 2.0, 2017, https://docs.google.com/document/d/1eyhS3-fOlRkDB6N39Md6KZbvbCe3CjQlampiZPg-5u4/edit#heading=h.bb39tz293a6l

9 Indice delle figure