Università degli Studi di Modena e
Reggio Emilia

Facoltà di Ingegneria di Modena
Corso di Laurea Specialistica in Ingegneria Informatica

Architettura peer-to-peer per la rilevazione e la prevenzione dal malware

Relatore
Prof. Michele Colajanni

Correlatore
Ing. Mirco Marchetti

Tesi di Laurea di
Michele Messori

Anno Accademico 2007/2008

“Se pensi di potercela fare perchè non provarci?”

A tutte le persone che hanno creduto in me e che mi hanno supportato. Grazie!

Indice

1 Introduzione **7**

2 Tecnologie per la rilevazione e la prevenzione dal malware **10**

2.1 Malware e minacce dalla rete 10

2.2 Honeypot 22

2.3 Intrusion Detection System 27

2.3.1 Intrusion Prevention System 32

2.3.2 Distribuited Intrusion Detection System 34

2.4 Reti peer-to-peer 36

3 Architettura proposta **40**

3.1 Elementi principali e loro disposizione 40

3.1.1 I sensori 41

3.1.2 I manager 44

3.1.3 Il collettore 45

3.2 Raccolta gerarchica 47

3.3 Raccolta distribuita su rete peer-to-peer 49

3.3.1 Funzionamento 51

3.3.2 Confronto architetture: distribuita vs. gerarchica . . . 54

4 Realizzazione dell'architettura **60**

4.1 Strumenti software utilizzati 60

4.1.1 Prelude 60

4.1.1.1 Componenti di prelude 61
4.1.1.2 Architettura di prelude 62
4.1.1.3 Lo standard IDMEF 63
4.1.2 Prewikka 64
4.1.3 Nepenthes 64
4.1.4 Ossec 68
4.1.5 Snort 69
4.1.6 MySQL 71
4.1.7 Pastry 72
4.1.7.1 PAST 76
4.1.7.2 Scribe 79
4.2 Realizzazione dell'architettura distribuita p2p 82
4.2.1 Architettura 84
4.2.2 Implementazione 86

5 Sperimentazione dell'architettura distribuita 95
5.1 Validazione dei singoli componenti 95
5.1.1 Sensori 96
5.1.2 Manager 97
5.2 Validazione dell'intera architettura 100
5.2.1 Analisi di un singolo attacco 101
5.2.2 Analisi di un attacco a più vittime dal medesimo attaccante 104
5.2.3 Analisi della diffusione di una tipologia d'attacco . . . 109
5.2.4 Analisi bilanciamento del carico 111
5.2.5 Analisi tolleranza ai guasti 113

6 Conclusioni 118

Elenco delle figure

3.1 Esempio di distribuzione sensori all'interno di una singola rete 46
3.2 Sbilanciamento del carico su architettura gerarchica. Il ramo evidenziato in rosso sarà sovraccaricato. 56
3.3 Bilanciamento del carico su architettura distribuita peer-to-peer. I rami evidenziati in rosso **non** sono sovraccaricati . . . 57

4.1 Esempio di routing table in Pastry per un nodo con nodeId $65a1x$ e $b = 4$. Le cifre sono in base 16 e x rappresenta il suffisso. Non sono mostrati gli indirizzi IP associati ad ogni entry. 73
4.2 Indirizzamento di un messaggio con chiave $d46a1c$ a partire dal nodo $65a1fc$. I punti sul cerchio rappresentano i nodi attivi all'interno del dominio circolare di Pastry 75

5.1 Prewikka - Schermata degli Heartbeats 99
5.2 Prewikka - Schermata degli eventi 100
5.3 Output della lettura ed inserimento dei dati del relativi al malware 101
5.4 Cartelle dei nodi contenenti il file relativo al malware 104
5.5 Cartelle dei nodi contenenti il file relativo alla connessione avvenuta tra l'host attaccante e l'honeypot 105
5.6 Output relativo all'inserimento di 10 eventi provocati dal medesimo IP sorgente 107

5.7 Cartelle dei nodi contenenti il file relativo all'indirizzo IP rilevato e segnalato da 10 nodi differenti 108
5.8 Output relativo all'inserimento di 10 eventi generati dalla medesima signature 110
5.9 Cartelle dei nodi contenenti il file relativo alla signature segnalata da 10 nodi differenti 111
5.10 CDF di 200 nodi con 40000 messaggi memorizzati e fattore di replica = 3. Media = 200, deviazione standard = 75,56 113
5.11 CDF di 500 nodi con 75000 messaggi memorizzati con fattore di replica = 4. Media 150, deviazioni standard 43,4 114
5.12 CDF di 500 nodi con 100000 messaggi memorizzati e fattore di replica = 4. Media 200, deviazione standard = 71,61 115
5.13 CDF di 1000 nodi con 160000 messaggi memorizzati e fattore di replica = 3. Media 160, deviazione standard 76,38 116
5.14 CDF di 3000 nodi con numero messaggi = 30000 e due differenti fattori di replica 117

Capitolo 1

Introduzione

Negli ultimi anni si è assistito ad un aumento vertiginoso delle minacce provenienti dalla rete, tra le quali la diffusione del malware rappresenta sicuramente una fetta importante. La causa di questo aumento esponenziale è da ricercarsi nel profitto che al giorno d'oggi portano la diffusione di queste minacce. Dallo Spam alla creazione di botnet, a stimolarne la diffusione sono organizzazioni criminali di diverse dimensioni che cercano una fonte di denaro alternativa per sovvenzionare le altre attività criminose svolte o addirittura per andarle a sostituire grazie al maggior profitto.

Ovviamente questo processo non è passato inosservato e grazie al contributo di molti attori, tra i quali figurano certamente le università, si sono evolute anche le tecniche di difesa dalle diverse minacce, con l'utilizzo di nuovi strumenti e lo scambio di informazione per accrescerne l'efficacia. Proprio da questo punto ci si è resi conto che l'unica soluzione per potersi difendere dall'incessante aumento delle minacce provenienti dalla rete è la collaborazione. A prima vista potrebbe sembrare un'affermazione scontata, ma la sua messa in atto non è certo tale e ha portato allo studio delle forme più adatte per unire le forze contro il nemico comune. Nascono così le prime architetture collaborative, basate principalmente su tipici approcci gerarchici, già utilizzati all'interno delle singole organizzazioni. I benefici sono notevoli, il

numero di macchine monitorate cresce, aumentando di pari passo la capacità e la prontezza di analisi delle nuove minacce. Non mancano però i lati negativi: il verificarsi di guasti ai livelli superiori porta a danni tanto più elevati quanto più alto è il livello di appartenenza del nodo, mentre all'aumentare del numero di nodi e del traffico generato si fa più pesante l'impossibilità di bilanciare il carico.

Proprio con l'intenzione di risolvere questi problemi, viene presentata in questa tesi un'architettura di nuova generazione, nella quale per la prima volta viene utilizzata una rete peer-to-peer per la raccolta, la gestione e la difesa dal malware. Come mostrerò nei seguenti capitoli, l'utilizzo di una collaborazione distribuita mediante l'utilizzo di tale rete, riesce a risolvere i problemi finora riscontrati, offrendo un'elevata affidabilità e un efficiente utilizzo delle risorse disponibili. Inoltre, grazie al prototipo realizzato e alla sua integrazione nell'ambiente di simulazione, verranno presentati i risultati di prove sperimentali a supporto delle analisi teoriche proposte. Tali simulazioni dimostreranno l'elevata tolleranza ai guasti, l'effettivo bilanciamento del carico, nonché l'ottimizzazione dello spazio di storage, ottenuti grazie all'approccio innovativo utilizzato da quest'architettura.

Nel secondo capitolo, verranno innanzitutto presentate le diverse minacce provenienti dalla rete, tra le quali verrà data una maggiore enfasi alle diverse tipologie di malware oggi presenti in rete. Dopo di ché, verranno presentati due dei principali strumenti di difesa e il relativo stato dell'arte. In conclusione di capitolo, verranno descritte brevemente le caratteristiche comuni e i punti salienti delle reti peer-to-peer

Nel terzo capitolo verranno mostrati gli elementi e le idee alla base dell'architettura proposta. Dopo una descrizione dei vari componenti alla base di essa, che andranno installati all'interno delle singole reti, verranno accennate le attuali soluzioni basate su architetture gerarchiche. Infine troverà largo spazio la presentazione dell'architettura distribuita su rete peer-to-peer, che

mostrerà le caratteristiche principali e i benefici che ne scaturiscono da un confronto con gli approcci gerarchici.

Nel quarto capitolo verrà presentata la realizzazione dell'architettura. Verranno mostrati i diversi dettagli a partire dai singoli strumenti utilizzati fino ad arrivare al software realizzato, passando per la descrizione della piattaforma peer-to-peer scelta per l'implementazione.

Nel quinto capitolo verranno mostrati i risultati delle prove e delle simulazioni effettuate con lo scopo di confermare al livello pratico, quanto mostrato sotto forma teorica. Dopo una prima parte di validazione dei singoli componenti, verranno descritte le diverse simulazioni accompagnate da figure esplicative.

Nel sesto capitolo infine, verranno presentate le conclusioni con riferimento al lavoro svolto e ai risultati ottenuti. Verranno inoltre proposti i possibili sviluppi futuri del progetto riguardante l'architettura presentata.

Capitolo 2

Tecnologie per la rilevazione e la prevenzione dal malware

In questo capitolo verrà presentata inizialmente una panoramica sulle principali minacce provenienti dalla rete, con particolare attenzione per il malware. Successivamente verranno illustrate due delle principali armi di difesa a livello di Network, più precisamente gli Honeypots e gli Intrusion Detection System, con relative varianti. A chiudere il capitolo sarà una breve presentazione sulle reti peer-to-peer, che mostrerà i punti chiave e le diverse tipologie.

2.1 Malware e minacce dalla rete

Il termine malware deriva dalla contrazione di due termini inglesi, rispettivamente MALicious e softWARE, e viene utilizzato per indicare tutti quei programmi realizzati per danneggiare le macchine che li eseguono, da qui il nome di software malevolo. I tre obiettivi alla base di tutti i programmi all'interno di questa categoria sono:

- **Installarsi** sul dispositivo (es. computer o smartphone) al verificarsi di un certo evento. Per avere una maggior probabilità di successo è neces-

sario che questi programmi abbiano la più alta compatibilità possibile con le piattaforme esistenti.

- **Nascondersi** dall'utente, in modo da poter sopravvivere il più a lungo possibile. Per fare questo vengono utilizzate spesso tecniche di mascheramento molto sofisticate che li rendono praticamente invisibili.
- **Propagarsi** il più possibile aumentando in questo modo il numero di successi. A tal fine si sono da sempre appoggiati ai principali mezzi di comunicazione informatici e se in origine tali mezzi si riconducevano principalmente ai floppy disk, ora il ventaglio è ben più ampio e partendo da email e pagine web modificate arriva alle sempre più diffuse piattaforme di file sharing o di instant messaging.

Prendendo in considerazione le differenti caratteristiche di questi programmi è possibile classificarli come segue:

Virus Si tratta sicuramente della classe più conosciuta. Una volta lanciato in esecuzione, il programma è in grado di replicarsi infettando altri file. Uno dei metodi è quello di copiare il proprio codice all'interno del file eseguibile di un'altra applicazione, questo è possibile inserendosi o all'inizio, col rischio però di sovrascrivere l'header, o in coda in modo da essere eseguito dopo l'esecuzione del programma originale. Altri file particolarmente vulnerabili sono i documenti di applicazioni che permettono di utilizzare linguaggi di programmazione più o meno complessi per la realizzazione di alcune operazioni (Macro). La loro diffusione avviene prevalentemente per mezzo di supporti fisici (floppy disk prima, chiavette USB oggi), email e programmi di file sharing, ma finché non vengono caricati in memoria non arrecano alcun danno. Per sopravvivere tendono a replicarsi in zone nascoste del sistema, alcuni arrivano fino al boot sector dell'hard disk, e cercano di disabilitare eventuali software antivirus attivi su quel sistema. Per favorire la prolificazione invece, sono nati i virus polimorfici e metamorfici. Queste

nuove generazioni sono in grado di criptare il proprio codice binario all'atto della replicazione in modo da sfuggire ai sistemi antivirus basati su firme e mentre i primi mantengono comunque una prima parte in chiaro (la routine di decriptazione) i secondi sono in grado di mutare completamente il proprio codice arrivando anche a scomporlo in più parti da inserire in zone separate del file infettato.

Worm A differenza dei virus, i worm non hanno bisogno di infettare altri file per propagarsi, poiché sono in grado di modificare il sistema operativo che li ospita in modo da venire eseguiti ogni volta che la macchina si avvia e rimanere attivi fino al suo spegnimento o fino ad un intervento esterno che termini il relativo processo. Spesso questi programmi non arrecano danni direttamente, il loro scopo infatti è passare inosservati, ma possono avere effetti collaterali quali consumo di risorse e introduzione di ulteriore malware. I principali mezzi di trasmissione sono email, file sarong e sfruttamento dei bug di software diffusi. Per diffondersi tramite email possono integrare al loro interno tutto il necessario per generarle autonomamente (con un motore SMTP) oltre ovviamente ad implementare i metodi di difesa dagli antivirus visti in precedenza: disabilitazione delle suite si sicurezza, polimorfismo e metamorfismo. Inoltre procedendo autonomamente possono portare ad un notevole traffico di email che spesso riportano un indirizzo del mittente falsificato con la conseguenza che i provvedimenti presi dai vari server di posta, all'atto della rilevazione del worm, risultano inefficaci se non addirittura controproducenti (es. inserimento in black-list di indirizzi estranei ai fatti). Un'alternativa molto comune e anche più difficile da contrastare è la diffusione mediante lo sfruttamento di bug, spesso noti, di programmi molto diffusi. In questi casi l'utente non ha nessuna possibilità di controllo poichè il meccanismo è completamente automatico e l'unica soluzione possibile è l'aggiornamento costante sia del sistema operativo che dei software installati. Di notevole rilievo è il

ruolo di questa classe di malware nella proliferazione di botnet, ovvero di reti di computer infetti alle dipendenze di gruppi criminali e alla base di alcune categorie di attacchi informatici. Una descrizione più dettagliata verrà data in seguito.

Trojan Diminutivo di **Trojan Horse** ovvero *Cavallo di troia*. Il nome deriva dalla somiglianza del suo funzionamento con quello dell'antico "dono" che permise ai greci di espugnare Troia. Questo tipo di malware infatti si presenta come un programma, solitamente gratuito, che mette a disposizione dell'utente funzioni più o meno utili invogliandolo ad eseguire il codice, celando però le vere finalità. L'attrazione delle funzionalità offerte ha un ruolo molto importante, questi programmi infatti spesso non sono in grado di auto replicarsi, di conseguenza le uniche vie per una larga diffusione sono l'inserimento del codice all'interno di worm o convincere il maggior numero di persone a scaricare il programma dal web. Questo malware solitamente è costituito da due file, quello che viene diffuso è il *file server*, mentre quello che rimane in mano all'attaccante è il *file client* che permette di comandare a distanza le macchine vittime del file server. Le principali finalità sono la creazione di *backdoor* o l'installazione di *keylogger*, entrambi analizzati in seguito.

Backdoor Letteralmente *porta sul retro* o più correttamente *porta di servizio*, permettono di aggirare, completamente o in parte, le procedure di sicurezza del sistema su cui sono installate. In alcuni casi possono essere già presenti nei programmi originali o per dimenticanza (spesso vengono utilizzate in fase sviluppo) o volutamente (in forma non sempre legale), ma spesso vengono installate da altri malware, come virus, worm o trojan, per garantire un accesso da remoto all'attaccante che ne conosca l'indirizzo IP.

Rootkit Il nome deriva dall'utente amministratore di sistemi Unix, chiamato *root*, e può assumere due significati. Il primo, e anche il più ap-

propriato, è quello di un pacchetto di funzioni che permettano di avere il completo controllo sulla macchina su cui viene installato, il secondo viene utilizzato invece per indicare altre tipologie di malware in grado di usare avanzate tecniche di mascheramento. Entrambi i casi hanno in comune l'utilizzo di moduli del kernel (solo unix), librerie o driver di sistema che rendono molto difficile sia l'individuazione che la rimozione. La loro tecnica è semplice quanto efficace, ogni software installato su un sistema, compresi gli antivirus, basano il proprio funzionamento sulle chiamate a funzioni messe a disposizione dal sistema operativo ospitante. Il rootkit interviene modificando o sostituendo i moduli che mettono a disposizione le suddette funzioni, in questo modo possono assicurarsi che la loro presenza non venga rivelata e bloccare eventuali chiamate che volessero porre termine alla loro esistenza.

Keylogger Sono strumenti il cui scopo è quello di registrare tutto ciò che un utente digita sulla tastiera del proprio computer con la speranza di intercettare informazioni preziose come password e numeri di carte di credito. Possono essere sia hardware che software, ma ovviamente solo i secondi rientrano nella categoria dei malware. Per svolgere il loro compito usano alcune tecniche tipiche dei rootkit, come la modifica dei driver delle tastiere o la modifica delle specifiche librerie del S.O. e i più avanzati permettono di registrare anche alcuni screenshot (immagini dello schermo) in modo da agevolare l'operato dell'attaccante. Nonostante siano di difficile rilevazione esistono due tecniche di difesa. La prima è quella di installare plugin in alcuni dei software più critici (come i browser) che vanno ad installare un secondo driver in grado di creare un collegamento sicuro tra tastiera e browser mediante l'utilizzo di comunicazioni crittografate. La seconda tecnica, che sta cominciando a diffondersi anche in parecchie suite di sicurezza, è quella di utilizzare una tastiera virtuale presentata a schermo e utilizzabile mediante il mouse, in questo modo non vengono premuti tasti della

tastiera fisica e il keylogger non registrerà alcuna attività. Questo tipo di malware può presentarsi solo o assieme ad un pacchetto più completo che prende il nome di *spyware*.

Spyware Esistono due categorie identificabili con questo termine che deriva dall'unione dei termini *SPY softWARE*. La prima, e probabilmente più innocua, raccoglie informazioni riguardanti l'attività online di un utente (siti web visitati, iscrizioni a servizi, acquisti etc) senza il suo consenso, trasmettendole tramite Internet ad un'organizzazione che le utilizzerà per trarne profitto, solitamente attraverso l'invio di pubblicità mirata. La seconda, e più pericolosa, oltre alle consuetudini dell'utente, tiene traccia delle informazioni inserite nei diversi moduli come password e numeri di carte di credito. Quest'ultima categoria è possibile trovarla accoppiata ai keylogger descritti sopra, ma non si tratta di un vincolo. In entrambi i casi si tratta di software non in grado di propagarsi autonomamente e per installarsi richiedono l'intervento dell'utente, quindi un funzionamento simile a quello dei trojan, con la differenza che in alcuni casi vengono aggiunti appositamente dalle softwarehouse per ottenere un ritorno economico. In questi casi è possibile che l'utente venga informato in modo più o meno trasparente prima di procedere con l'installazione, col vincolo che nel caso si rifiuti questa componente non è possibile proseguire con l'installazione del resto del programma. Dove esiste quest'informazione non è più possibile parlare di malware quanto piuttosto di *Adware*.

Adware Il termine deriva dalla contrazione inglese di *ADvertising-supported softWARE*, che può essere tradotto come Software supportato dalla pubblicità, ed indica quei programmi che durante il loro utilizzo mostrano avvisi pubblicitari permettendone la distribuzione gratuita o comunque a prezzi ridotti. Successivamente l'utente potrà poi decidere di rimuovere tali avvisi mediante il pagamento di una licenza. Di per se

questi software non rientrerebbero nella categoria dei malware se non fosse che spesso implementano alcune caratteristiche tipiche degli spyware per poter presentare all'utente avvisi pubblicitari focalizzati sui suoi interessi.

hijacker o **browser hijacker**, sono software in grado di modificare la *homepage* dei browser con lo scopo di indirizzarli su pagine contenenti altri tipi di malware in grado di diffondersi mediante i bug dei browser stessi o dei plugin in essi installati.

Ransomware Si tratta di una classe di malware abbastanza recente e che può essere suddivisa ancor più nello specifico in *cryptovirus*, *cryptotrojan* or *cryptoworm*. Come si può intuire dai nomi si tratta di particolari versioni di virus, trojan e worm in grado di criptare il contenuto dei più diffusi tipi di file contenuti sull'hard disk dell'utente. Lo scopo è quello di poter successivamente ricattare la vittima chiedendo un riscatto in cambio della password per decriptare i documenti. L'utente sprovvisto di copie di backup dei file più importanti che intendesse pagare come da richiesta, si troverebbe comunque nella spiacevole condizione tipica di tutti i casi di ricatto, quella cioè di non avere nessuna garanzia che la disavventura termini dopo il pagamento della somma pattuita.

Dialer Questa categoria sta lentamente scomparendo grazie all'avvento delle linee xDSL e fibra ottica. In origine il loro unico scopo era quello di creare connessioni telefoniche per potersi connettere ad internet, quindi un atteggiamento lecito, ma in seguito cominciano a diffondersi versioni programmate per contattare numeri di particolari servizi telefonici caratterizzati da costi molto elevati e distinguibili dal loro prefisso, in origine 144, poi 166 e infine 899 e 892. Grazie all'utilizzo di modem ADSL, il computer non ha più la possibilità di effettuare tali tipi di connessioni, di conseguenza il bacino di potenziali vittime si sta riducendo rapidamente.

Rabbit Anche questa classe di malware sta perdendo popolarità, ma a differenza dei dialer non per particolari innovazioni tecnologiche, ma a causa di un cambiamento radicale nelle motivazioni che portano alla diffusione di malware. Oggi infatti la stragrande maggioranza di programmi malevoli in circolazione ha lo scopo di far ottenere un guadagno alle organizzazioni criminali, mentre i Rabbit hanno il solo scopo di riprodursi in continuazione fino a saturare le risorse del sistema e a differenza dei Virus non infettano alcun file.

Dopo questa panoramica sulle varie tipologie di malware, verranno ora mostrati i principali impieghi nell'ambito di attacchi più complessi.

Man in the middle In questo caso l'attaccante si intromette nella comunicazione tra due terminali intercettandone il contenuto. In base alle caratteristiche della comunicazione quest'attività può essere più o meno complessa, ma la procedura standard prevede che l'attaccante faccia da intermediario nella comunicazione tra i due terminali senza che quest'ultimi se ne accorgano e per fare questo deve riuscire ad ingannare entrambi spacciandosi per l'utente all'altro terminale grazie a credenziali false. In questo caso un malware potrebbe servire per reindirizzare le connessioni verso la macchina dell'attaccante senza che l'utente vittima se ne possa accorgere.

Denial of Service (DoS) Si traduce in *Negazione di Servizio* e si può dividere in due categorie: sfruttamento di bug e esaurimento delle risorse. Nella prima categorie rientrano diverse tecniche che si basano su bug software (WinNuke), protocolli insicuri (Ping of Death) o cattiva implementazione (Teardrop, Land). Ultimamente però si tende ad utilizzare il termine DoS per indicare la seconda categoria, cioè quella in cui, lavorando su uno dei parametri d'ingresso, si cerca di portare il funzionamento di un sistema informatico che fornisce un servizio, ad esempio un sito web, al limite delle prestazioni fino a renderlo non più in

grado di erogare il servizio. Solitamente i due parametri su cui si lavora sono la saturazione della banda o il riempimento delle code di gestione delle richieste. Tipiche tecniche di attacco per esaurimento delle risorse sono quelle basate sul Flooding, e ad esempio il *TCP SYN Flooding* che inonda il server di pacchetti TCP con flag SYN per inizializzare le connessioni. Il server dovrà mantenere in coda tutte le richieste fino allo scadere del timeout, mentre l'attaccante continuerà a mandare richieste senza finalizzare nessuna connessione. L'aumento esponenziale del traffico sulla rete Internet, ha portato ad un proporzionale (rispetto al traffico, quindi esponenziale in senso assoluto) aumento della banda disponibile, soprattutto lato server, visto che la maggior parte dei collegamenti domestici sono di tipo asimmetrico (*Assymmetric*-DSL), cioè con una limitata banda in upload. Questa evoluzione ha reso irrealizzabili attacchi DoS lanciati da singoli terminali o da piccoli gruppi di utenti. Nascono così gli attacchi DDoS.

Distribuited Denial of Service (DDoS) Come si intuisce dal nome, si tratta dello stesso tipo di attacco sopra definito, ma realizzato in modo *distribuito*. Esistono alcuni attacchi DDoS basati su vulnerabilità dei protocolli (Smurf), ma solitamente per realizzare questo tipo di attacco è prima necessario procurarsi un vasto bacino di *zombie*, termine usato per indicare macchine infette da malware e sotto il controllo degli potenziali attaccanti. Come visto in precedenza esistono vari tipi di malware, in questi casi si fa spesso riferimento a worm, grazie alla loro capacità di diffondersi molto rapidamente, incaricati di aprire backdoor, in questo modo si otterrà una *botnet*, cioè una rete di zombie pronti ad eseguire i comandi che gli verranno impartiti dai server *C&C (Command & Control)*. Spesso quest'ultimi sono dei semplici server IRC a cui i malware si andranno a connettere una volta installati e che possono essere gestiti direttamente dai diffusori del worm o indirettamente da bande criminali che ne facessero richiesta, esiste infatti un vero e proprio mercato nero di

botnet, utilizzate per lo più per l'invio di *spam*, ma anche per realizzare attacchi DDoS. Per rendersi conto della portata di queste botnet è interessante citare qualche numero. Nell'agosto del 2007 col worm *STORM* in piena espansione, Peter Gutmann fece alcune osservazioni[2] sulla capacità di calcolo della botnet, basandosi su alcune stime e il risultato fu sbalorditivo: il numero di computer infetti stimato era compreso tra il milione e i 10 milioni, prendendo la stima più bassa e moltiplicandola per le prestazioni medie di una macchina collegata ad internet in quel periodo[1], si otteneva una capacità di calcolo superiore alla somma dei primi 10 supercomputer classificati su www.top500.org. Prendendo invece in considerazione il numero di attacchi provenienti da botnet si ottengono valori altrettanto impressionanti. Il 22 Settembre 2008 SecureWorks, azienda leader nella fornitura di servizi per la sicurezza, ha pubblicato[3] alcuni dati statistici riguardanti gli attacchi subiti dalle macchine da lei controllate, raggruppandoli per zone di provenienza e se i nomi delle prime due nazioni non hanno stupito, USA e Cina, l'hanno fatto i loro numeri, rispettivamente con 20.6 milioni e 7.7 milioni di attacchi. Esempi famosi di attacchi DDoS sono quello subito da eBay, Buy.com, CNN.com e Amazon nel 2000[4], con un danno stimato di oltre mezzo milione di dollari, e quello più recente di Scientology[5] che nel Gennaio 2008 è rimasta sotto attacco per un'intera settimana da parte di Anonymous.

Distribuited Reflection Denial of Service (DRDoS) Rappresenta l'evoluzione degli attacchi DDoS. Gli effetti finali sono gli stessi, ma per rendere più difficoltoso il rintracciamento delle macchine attaccanti (che potrebbero quindi essere bloccate), ci si appoggia a server che mettono a disposizione servizi pubblici. Lo scopo è di far riflettere i propri attacchi da questi server, da qui il nome *Reflection*, utilizzando la tecnica dello *spoofing*. In poche parole l'attaccante, o gli zombie a sua disposizione, mandano pacchetti TCP ai server pubblici utilizzando come

indirizzo sorgente spoofato (contraffatto) quello della vittima che si vuol colpire, in questo modo i server pubblici risponderanno a quest'ultima che si ritroverà improvvisamente sommersa di richieste. Inoltre per evitare di sovraccaricare i *reflection server*, l'attaccante utilizzerà quest'ultimi a turno in modo da non superare eventuali livelli di soglia che porterebbero a notificare l'evento ai rispettivi amministratori.

Antisocial network Questo termine rappresenta semplicemente una provocazione ed è stato coniato da alcuni ricercatori che hanno messo in evidenza come la notevole diffusione dei social network possa essere un potenziale rischio per la sicurezza [7]. Il titolo della ricerca è eloquente: "Antisocial Networks: Turning a Social Network into a Botnet", la ricerca infatti propone un punto di vista critico su queste nuove piattaforme già note per la loro vulnerabilità a codice maligno (basti pensare a al worm Samy[6][1]). Cosa accadrebbe infatti, se una delle applicazioni più diffuse, inserite dagli utenti nelle proprie pagine, fosse in realtà studiata per sovraccaricare di richieste i server della vittima prescelta? Una risposta la danno i ricercatori stessi che pubblicano i risultati di un test da loro effettuato[7]. Dopo aver scelto facebook.com come piattaforma di partenza, hanno messo a disposizione un programma che prometteva di visualizzare la foto del giorno del National Geographic[8] sulla pagine personale dell'utente che l'avrebbe installato. Nella realtà il software nascondeva altre 4 richieste che andavano a prelevare un'immagine da 600KB presso un host vittima senza che l'utente se ne accorgesse. Nel caso in questione, l'host vittima era una macchina controllata dai ricercatori che nell'arco di pochi giorni hanno visto crescere il traffico generato dalle mille installazioni fino a raggiungere il picco di 300 richieste/ora e di 6Mbit al secondo di banda. Se a prima vista

[1] Il worm Samy può essere considerato il primo worm a larga diffusione su un social network. Fu creato da un ragazzo di 19 anni e inserito nel proprio profilo ospitato su MySpace.com con lo scopo di aumentare la propria popolarità . In meno di 24 ore dall'inserimento i profili infettati superarono il milione.

questi numeri potrebbero non sembrare così impressionanti, si sottolinea come il programma sia stato rimosso dopo pochi giorni e come le richieste fossero limitate a 4, mentre un ipotetico malware potrebbe generare centinaia di richieste e rimanere online molto più tempo con conseguenze devastanti, alcuni degli applicativi più diffusi di Facebook oltrepassano infatti il milione di utenti giornalieri.

Phishing Questa tipo di attacchi, non si basa direttamente sulla distribuzione di malware, ma sullo *Spam* che questo può generare. L'interesse crescente per questa categoria è legato alla sua diffusione in continua crescita. Lo scopo è quello di ottenere i dati relativi ai conti correnti, sia bancari che postali, delle vittime. Per raggiungerlo, viene inizialmente creata una pagina web che rispecchi fedelmente la pagina di login della banca scelta come esca e che sia in grado di registrare i dati inseriti dagli utenti, dopo di chè si pubblica ad un indirizzo che contenga alcuni nomi che richiamino l'url della pagina originale, anche se ovviamente il dominio sarà diverso. A questo punto parte una campagna di *Spam* focalizzata il più possibile sull'area geografica interessata all'ente utilizzato come esca. Il termine Spam sta ad indicare email indesiderate, spesso contenenti pubblicità, ma che nel caso del phishing contengono messaggi appositamente studiati per allarmare l'utente invitandolo poi a risolvere eventuali problemi dopo aver effettuato il login. Dopo questi avvisi viene riportato il link alla pagina web creata precedentemente, mascherato da sotto il titolo dell'url originale. L'utente sprovveduto che dovesse cadere nella trappola si ritroverebbe con le credenziali rubate e ben presto col contocorrente svuotato. In questi casi è facile difendersi, innanzitutto è sempre bene diffidare di certi tipi di email, spesso la lingua utilizzata presenta diversi errori grammaticali (soprattutto in Italia), poi è buona abitutine diffidare dei link proposti, quindi aprire un nuova finestra dove poter digitare l'indirizzo che già si conosce.

Pharming Molto simile al phishing con cui condivide le finalità e anche tutta la parte di preparazione. Fino alla realizzazione della pagine web contraffata infatti, le tue tecniche di attacco sono identiche, la differenza sta nel come si attirano le potenziali vittime. A differenza del phishing, nel pharming non si mandano email di spam, ma si attaccano i server DNS sparsi per la rete. Qusti server sono di fondamentale importanza e servono per fornire ai computer la traduzione delle url in indirizzi ip. L'attacco a questi server è molto più complesso rispetto all'inoltro di spam e ha come scopo sostituire l'indirizzo ip originale della banca con quello dei propri server. A questo punto l'utente che tentasse di connettersi per esigenze personali (e non più su richieste via email) verrebbe indirizzato sulla pagina contraffatta senza accorgersene. Questi attacchi sono di difficile realizzazione, ma se portati a termine mieteranno vittime anche tra gli utenti più esperti, questo perchè non esiste modo di rendersi conto di quello che sta accadendo, salvo quello di far precedere l'autenticazione da un'approfondita navigazione, in questo caso infatti potrebbero uscire allo scoperto lacune nella contraffazione del sito originale.

2.2 Honeypot

Gli honeypot sono strumenti preziosissimi per la raccolta del malware ed il tracciamento delle attività malevole svolte da eventuali aggressori. La base comune di queste applicazioni è quella di mettere a disposizione dell'utente uno strumento sicuro ed isolato da utilizzare come trappola in cui circoscrivere le attività degli aggressori. Proprio dallo scopo che si prefigge nasce il suo nome, che dalla lingua inglese può essere tradotto letteralmente come "vasetto del miele", usato anche in ambiti non informatici, come quello poliziesco ad esempio, dove viene associato a tutti gli stratagemmi messi in atto per cogliere i malviventi in flagranza di reato. Inoltre uno dei principali

vantaggi derivante dalla struttura e dall'implementazione degli honeypot è che qualsiasi attività registrata può essere considerata a priori come ostile, riducendo al minimo, o perfino azzerando, il numero di falsi positivi.

Dopo questa introduzione si può procedere con la loro classificazione. Il primo passo è quello di distinguerli in base al *deployment* (spiegamento) ottenendo così gli honeypot di **produzione** e quelli di **ricerca**.

- Gli **honeypots di produzione** sono più semplici da usare, ma hanno anche una limitata capacità di raccolta di informazioni. Vengono impiegati principalemente presso aziende e grandi compagnie che li dispongono nella propria rete affianco ai propri server. L'obiettivo è quello di aumentare la sicurezza della rete aziendale, cercando di rallentare l'operato degli aggressori. A tale scopo vengono utilizzati honeypot a bassa interazione su cui reindirizzare i tentativi di attacco, che nonostante le limitate informazioni raccolte permettono di raggiungere l'obiettivo prefissato.

- Gli **honeypots di ricerca** vengono impiegati principalmente da istituzioni accademiche, enti no-profit ed enti governativi. Si tratta di sistemi complessi, che richiedono notevoli conoscenze tecniche sia per l'installazione che per la manutenzione, ma che permettono di raccogliere informazioni molto dettagliate su tutte le operazioni eseguite al loro interno. Lo scopo è quello di poter identificare nuove tecniche d'attacco, mediante l'analisi dei dati raccolti per poi studiare e divulgare le relative contromisure atte a garantire il maggior livello di sicurezza possibile.

L'utilizzo di questo strumento ha diversi vantaggi.

- L'honeypot non eroga servizi pubblici, di conseguenza ogni connessione ricevuta può essere considerata malevola senza il rischio di falsi positivi. Un utente comune, infatti, non può connettersi per sbaglio, perchè non esistono informazioni pubbliche che riconducano all'indirizzo

dell'honeypot, quindi chi riesce a contattare tale servizio, deve averle ottenute grazie ad una qualche tecnica di attacco.

- Considerando che, come appena detto, ogni connessione è malevola, se ne deduce che anche ogni attività svolta all'interno dell'honeypot risulta tale. Questo risultato è molto importante, poichè semplifica notevolmente le operazioni di analisi. Ogni modifica apportata al sistema infatti, è parte di un tentativo di attacco e come tale dev'essere presa in considerazione senza il rischio di apessantire il lavoro da svolgere con informazioni superflue.

- Altro vantaggio importante è che le attività svolte vengono registrate indipendentemente dalle techniche di offuscamento adottate dall'attaccante per raggiungere l'honeypot. Quest'ultimo infatti rappresenta il punto finale (in gergo *end point*) di conseguenza le operazioni si svolgeranno in chiaro, sia che le connessioni utilizzate siano crittografate (tecnica usata per bypassare i controlli effettuati sul traffico che attraversa la rete locale), sia che provengano da sorgenti multiple (tecnica usata dall'attaccante per operare nell'anonimato).

- Inoltre, nel caso si utilizzino honeypot a bassa interazione, a questi vantaggi si aggiunge la relativa economicità del deployment di questi strumenti. Oltre alla possibilità di appoggiarsi a piattaforme open source senza dover rinunciare alle funzioni fondamentali, si hanno costi ridotti anche per quanto riguarda l'hardware da utilizzare. Un moderno computer è infatti sufficiente per gestire migliaia di indirizzi ip e per implementare svariati honeypots, mentre se per singole installazioni ci si può appoggiare anche ad hardware più datato.

Tutti questi vantaggi sono accompagnati da due svantaggi che devono essere tenuti in considerazione all'atto dell'installazione.

1. L'honeypot è in grado di registrare e tenere traccia esclusivamente del traffico ad esso indirizzato. Di conseguenza attacchi rivolti ad altre non verrebbero rilevati. Per questo vanno ben studiati gli indirizzi da associare all'honeypot e soprattutto non bisogna sciarlo "solo" sperando che basti a limitare i danni, ma affiancarlo ad altri strumenti di rilevazione ed analisi.

2. Essendo vulnerabile di sua natura, bisogna prestare attenzione all'eventualità che un aggressore si renda conto di essere finito in un honeypot. In questo caso infatti potrebbe cercare di prendere il controllo della macchina per lanciari ulteriori attacchi agendo stavolta dall'interno della rete.

Per questi due motivi occore studiare attentamente la disposizione di questi strumenti cercando di mantenerli isolati dalle altre macchine connesse alla rete aziendale, mantenendoli però accessibili in modo da poter dirottare il traffico su di essi.

Nella classificazione precedente si è fatto riferimento anche al diverso livello di interazione degli honeypot di produzione da quelli di ricerca e la distinzione fatta era tra alta e bassa.

Bassa interazione Si tratta di software installati su sistemi operativi standard e che spesso impiegano un singolo processo per simulare dal singolo computer alla complessa rete di host. Qust'ultimo è il caso di *honeyd*, un demone in grado di emulare l'esecuzione di svariati servizi su diversi host a cui vengono associati gli indirizzi ip disponibili all'interno della rete reale. Questa possibilità permette di simulare un elevato numero di server tra i quali nascondere quelli reali, rendendo più difficili le attività di ricerca di un attaccante. Ha inoltre il vantaggio di poter essere personalizzato mediante script creati dall'utente che potrà in questo modo adattarlo alle proprie esigenze. Altri esempi di honeypots diffusi sono *Nepenthes*[9] e *mwcollect*[10], entrambi in grado di

simulare alcuni dei servizi più diffusi e di scaricare il contenuto di eventuali malware proposti dall'attaccante, che verranno poi archiviati per una futura analisi. I vantaggi pricipali di questa categoria di prodotti risiedono nella semplicità di utilizzo e di installazione (si tratta di semplici programmi), nonchè nella bassa probabilità di compromissione delle macchine ospitanti.

Alta interazione Si tratta di sistemi molto complessi basati su computer dedicati a tale scopo o in alternativa su macchine virtuali, anche se questo rappresenta più un vantaggio economico che tecnico. La caratteristica di questa tipologia consiste nella raccolta di una notevole quantità di dati da cui estrapolare informazioni importanti per lo studio di nuove tecniche di attacco. La complessità di tali implementazioni, sia in termini di installazione che di mantenimento, risulta tale da limitare il loro utilizzo a personale esperto in grado di studiare piani di spiegamento che tengano conto dei rischi derivanti da possibili infezioni o dalla possibilità che uno degli host cada nelle mani di un attaccante. Inoltre questi sistemi risultano solitamente statici e di conseguenza identificabili dagli aggressori che scambiandosi informazioni realizzano vere e proprie *blacklist*. Uno dei progetti più importanti di questa categoria è *Honeynet,* che appoggiandosi su una grande rete di macchine dedicate è in grado di raccogliere e analizzare un'imponente mole di dati.

Oltre alla raccolta di malware ed al rallentamento degli attacchi subiti, l'utilizzo di honeypots risulta molto utile ai fini di combattere lo spam. Una delle tecniche principali degli spammers è infatti lo sfruttamento di *open mail relay*, cioè di server SMTP impostati in modo da permettere a chiunque l'invio di email, senza controllare che il mittente o il destinatario siano utenti registrati. Purtroppo questa è l'impostazione di default di molti server, con la conseguenza che molti di essi vengono sfruttati appunto a fine di spam. La

tecnologia honeypot può intervenire simulando uno di questi server, in questo modo nel caso si riesca ad ingannare lo spammer si otterranno due grandi benefici: primo, le email di spam non verranno inoltrate; secondo, sarà possibile risalire allo spammer, mediante le tracce da lui lasciate. Interessante è notare come all'introduzione di questi sistemi la maggior parte degli spammer si collegasse direttamente ai mail server senza alcun timore, mentre col tempo e la presa di coscienza del rischio di cadere in una trappola, il comportamento è cambiato e l'accesso avviene solo mediante l'utilizzo di una catena di sistemi precedentemente compromessi. Questo significa che l'introduzione degli honeypot in questo campo ha reso più difficoltose le operazioni degli spammer.

In conclusione questa tecnologia risulta utile al fine di rilevare e reagire agli attacchi provenienti dalla rete. Offre un ampio ventaglio di opzioni permettendo all'utente di scegliere quelle che meglio si adattano alle proprie esigenze, sia a livello di funzionalità offerte che di capacità richieste.

2.3 Intrusion Detection System

Gli *Intrusion Detection System* o *IDS* vengono utilizzati per rilevare accessi non autorizzati alle reti locali o ad i computer ad esse colegate. Si tratta di dispositivi software o hardware (a volte la combinazione di tutti e due) che dislocati in punti strategici della rete permettono di fornire dettaggli sugli attacchi sfuggiti al controllo del firewall. Le intrusioni rilevate possono essere di varia natura, partendo da quelle prodotte da cracker esperti si passa a quelle di utenti inesperti che utilizzano programmi semiautomatici, fino ad arrivare a tool automatici (vedi malware).

Come indica il nome stesso, si tratta di sistemi in grado di segnalare intrusioni già avvenute o ancora in atto, ma non di prevenirle, per cercare di rendere più chiara la sua funzione, è possibile paragonare l'IDS ad un antifurto. Inoltre si possono utilizzare le informazioni raccolte sia per poter

aggiornare la propria rete rendendola di volta in volta sempre più sicura, sia a scopo legale, nel caso si volessero intraprendere cause nei confronti degli intrusori.

Un IDS è composto da 4 componenti fondamentali:

- i **sensori**, utilizzati per ricevere le informazioni dalla reti o dai computer;
- una **console**, che serve per monitorare lo stato della rete;
- un **engine** o **motore**, che analizza i dati prelevati dai sensori e provvede a individuare eventuali anomalie;
- un **database**, contenente le regole e le firme utilizzate dal engine per l'analisi.

Solitamente i quattro componenti si trovano in un unico pacchetto, ma non è escluso trovarli separatamente, soprattutto nel caso dei sensori che possono essere caratterizzati da funzioni specifiche.

Una prima classificazione di questi strumenti si può fare in base all'obiettivo da monitorare ottenendo le seguenti categorie:

Host Intrusion Detection System (HIDS) consiste in un agente che analizza l'Host alla ricerca di intrusioni mediante l'analisi dei file di log del sistema, delle system call, delle modifiche al file system del computer (modifiche nel file delle password, nel database degli utenti e della gestione dei privilegi, ecc), e anche di altre componenti del computer. Come per gli honeypots, questa classe di IDS ha il vantaggio di essere installata sulla macchina vista dall'aggressore come end-point, di conseguenza le tecniche di offuscamento degli attacchi non hanno alcun effetto. Lo svantaggio è di avere una visione limitata al traffico rivolto alla singola macchina. Un esempio di questa tipologia è *Ossec*[11].

Network Intrusion Detection System (NIDS) ha l'obiettivo di analizzare il traffico di rete per identificare le intrusioni e per fare ciò tiene sotto monitoraggio non solo un singolo host ma una rete completa. Si tratta di un sistema che ispeziona (in gergo *sniffa*) il traffico che passa sul segmento di rete a cui sono connessi i suoi sensori, cercando tracce di attacchi. A tale scopo il dislocamento di quest'ultimi è di fondamentale importanza e dev'essere studiato attentamente per garantire una quantità di traffico adeguata. Questa categoria, anche se soggetta a tecniche di offuscamento degli attacchi, consente una visione generale dello stato della rete. Un esempio di Network Intrusion Detection System è *Snort*, si tratta di un prodotto opensource ed è probabilmente il più diffuso.

Application Protocol Intrusion Detection System (APIDS) analizza il traffico concentrandosi su specifici protocolli applicativi, dei quali conosce le diverse implementazioni ed è in grado di notare eventuali anomalie.

Hybrid Intrusion Detection System Come indica il nome si tratta di un modello ibrido, ovvero implementa più tecniche cercando di sfruttare i vantaggi di ognuna. I modelli più diffusi si basano su HIDS e NIDS e offrono sia una panramica sullo stato della rete che un'analisi più specifica per ogni singolo host. Un esempio di questa categora di IDS è rappresentato da *Prelude*.

Una seconda classificazione può essere effettuata sulla base delle metodologie di analisi dei dati raccolti. In questo caso si hanno 3 classi:

Signature Analysis hanno un funzionamento simile a quello della maggior parte degli antivirus, ovvero si appoggiano ad un database contenente le firme di tutti i tipi di attacchi conosciuti che dev'essere costantemente aggiornato. Oltre alle signature vere e proprie si possono utilizzare set

di regole che spesso si possono personalizzare mediante il linguaggio proprio del software scelto. Il vantaggio principale è che se le firme sono scritte correttamente non esistono falsi positivi, ogni segnalazione infatti, sarà basata un tipo di attacco conosciuto e di cui sono disponibili informazioni a proposito, come l'elenco dei sistemi vulnerabili e la descrizione dell'attacco completo. Un altro vantaggio è la velocità di risposta, che risulta elevata grazie alle ridotte operazioni da svolgere. Lo svantaggio principale è l'impossibilità di rilevare attacchi basati su tecniche non ancora conosciute o non ancora inserite nel database delle firme. Si ha in oltre una notevole difficoltà nel rilevamento di worm polimorfici, mentre quelli metamorfici risultano impossibili da rilevare.

Anomaly Detection richiedono lo studio preventivo della reti in cui andranno a lavorare, in modo da ricavare un modello di traffico standard. Dal momento dell'entrata in funzione, tale modello verrà considerato come riferimento e le attività che dovessero discostarsi da esso andrebbero a generare le dovute segnalazioni. Il vantaggio principale di questa classe di IDS è rappresentato dalla possibilità di rilevare tentativi di attacco basati anche su tecniche sconosciute, mentre gli svantaggi sono dati dalla complessità della realizzazione di un modello standard e dall'imprecisione di questo sistema che genera una notevole quantità di falsi positivi, o di falsi negativi nel caso le regole siano più permissive.

Analisi dei grafi (GrIDS) Si realizzano grafi associati a vari tipi di attività, considerando gli host come nodi e le connessioni come rami. L'ipotesi alla base è che attività simili tra loro generino traffici simili tra loro, quindi grafi simili tra loro. Di conseguenza dopo aver realizzato i grafi di diverse tecniche di attacco si studiano le attività correnti per verificare eventuali corrispondenze. Purtroppo non esistono modelli per tutti i tipi di attacco, inoltre la rilevazione non è perfetta e alcune tecniche possono mascherarsi facilmente per uscire dagli schemi, al con-

trario alcune attività benigne potrebbero attivare la segnalazione. Per questi motivi, questa classe risulta quella meno adoperata.

Mentre gli IDS si evolvono per cercare di garantire risultati migliori, di pari passo si evolvono le tecniche di *evasione* adoperate dagli attaccanti per eludere questi strumenti di difesa.

In riferimento ai *NIDS*, si è accennato alla loro vulnerabilità a tecniche di *offuscamento*. La prima di queste consiste nell'utilizzo di connessioni criptate, in questo modo infatti, l'engine non è in grado di analizzare i dati raccolti dai sensori, o in alcuni casi, sono i sensori stessi a non inoltrare il traffico criptato. Una sottogategoria risulta quella dei malware polimorfici e metamorfici che applicano la crittografia a se stessi per eludere gli IDS basati su signature. Fa parte delle tecniche di offuscamento anche l'utilizzo di caratteri Unicode. Questi infatti non vengono riconosciuti dall'IDS, ma vengono interpretati dalle applicazioni destinatarie, come ad esempio alcuni web server.

Esistono anche altre tecniche di elusione più ingegnose[12]. Tra queste troviamo la *frammentazione* dei pacchetti, che risulta efficace nel caso gli IDS non effettuino la ricostruzione dei pacchetti frammentati. Basate invece sulle diverse implementazioni dei protocolli che hanno i sistemi operativi sono le tecniche basate sull 'utilizzo di *sequence number* appositi e quelle basate sulla violazione degli standard relativi ai protocolli. Nel primo caso, si inviano pacchetti contenenti *sequnce number* che all'atto della ricostruzione risultano sovrapposti. Il comportamento della macchina colpita dipende dall'implementazione che utilizza il sistema operativo per ricostruire lo stream TCP che in alcuni casi può portare al *crash* dei sistemi. Anche nel secondo caso occorre conoscere le reazioni dei S.O. a comandi fuori standard. L'ultima tecnica appartenente a questa categoria si basa sul settaggio di valori differenti nel campo *TTL* (Time To Live) dei pacchetti TCP. In questo modo è possibile mandare pacchetti che non arriveranno agli host, ma che verranno comunque letti dall'IDS. Questa tecnica unita alla framentazione dei pacchet-

ti permette di inserire dati casuali per eludere le regole, o le signature, usate dell'engine.

Tutte le tecniche finqui viste si basano sulla differenza tra i dati che giungono agli host e quelli che giungono al NIDS (gli HIDS risultano solitamente invulnerabili a questi attacchi), ma esiste un'altra tipologia di attacco che si basa sui limiti delle risorse a disposizione degli IDS. Si tratta di veri e propri attacchi DoS, realizzati inondando la rete con pacchetti che risultino positivi ai controlli, in questo modo l'IDS esaurirà la propria capacità di analisi ed essendo questi strumenti Fail Open[2], da quel momento l'attaccante avrà via libera per operare senza controlli.

Tornando alla classificazione degli IDS, esiste un ultima diferenziazione e viene fatta tra gli IDS passivi e quelli attivi:

passivi sono la versione standard, si limitano alla segnalazione degli eventi rilevati e per loro vale tutto quello scritto in precedenza.

attivi oltre alle funzioni di segnalazione, implementano metodi per intervenire sul sistema in modo da porre fine alle minacce rilevate. Gli interventi più diffusi riguardano la modifiche delle *access list* dei firewall, in modo da interrompere le connessioni con indirizzi esterni alla rete locale. Ulteriori miglioramenti hanno reso disponibile il blocco anche del traffico interno alla rete, potendo cosi interrompere anche la diffusione di malware tra host interni. Tutte queste funzionalità hanno portato alla creazione degli IPS (Intrusion Prevention System).

2.3.1 Intrusion Prevention System

Rappresentano l'evoluzione degli IDS e non esiste in realtà una linea di demarcazione netta nei confronti delle versioni attive di quest'ultimi. In alcu-

[2]Col termine *fail open*, si intendono i sistemi che in caso di guasto (fail), lasciano aperte (open) le comunicazioni. Al contrario con *fail close*, si intendono i sistemi che in caso di guasto (fail), chiudono (close) le comunicazioni. Due esempi di questi comportamenti sono rispettivamente gli *IDS* e i *Firewall*.

ni casi si tende ad identifcare col termine IDS quelli esclusivamente passivi, mentre con IPS tutti gli strumenti che permettono di intervenire attivamente, in altri si aggira la questione utilizzando il termine IDPS, Intrusion Detection and Prevention System. Hanno molte similitudini col funzionamento dei firewall: sono entrambi *in line*, quindi *fail close*, e agiscono entrambi mediante l'utilizzo di liste di controllo degli accessi. Gli IPS, però sono più spostati a livello applicativo e nelle liste citate tendono ad inserire coppie utente-programma, piuttosto che idirizzo IP-porta. Esistono due categorie principali di questi strumenti.

Host IPS (HIPS) come si deduce dal nome si tratta di implementazioni a livello di singolo host. Il controllo può essere molto dettagliato e per ogni programma che dovrà essere eseguito sul sistema è possibile definire specifiche policy. Se non impostato in modo preciso, può risultare particolarmente invadente per l'utente dell'host.

Network IPS (NIPS) hanno una visione a livello di rete, quindi più ampia, e si dividono in tre principali sottocategorie.

Protocol analysis IPS (PIPS) ispezionano il traffico a livello di protocolli applicativi, come HTTP e FTP, cercando anomalie o violazioni degli standard.

Content Based IPS (CBIPS) analizzano il *payload* dei pacchetti utilizzando solitamente tecniche basate su *signature analysis*, con tutti i vantaggi e svantaggi tipici di tale approccio.

Rate Based IPS (RBIPS) si basano principalmente su tecniche di anomaly detection. Esiste una fase iniziale di studio delle statistiche sul traffico relativo alla rete monitorata, in particolare si osservano le percentuali (rate) dei pacchetti divisi per protocolli (TCP, UDP, ARP, ICMP). Finito lo studio iniziale l'IPS entra in funzione affiancato da un sistema di autoapprendimento, in modo

da garantire un miglioramento delle prestazioni. Questo approccio risulta particolarmente efficace nel rilevamento di attacchi DoS o DDoS.

La scelta tra un HIPS o un NIDS va effettuata in base alle proprie esigenze, ma occorre tenere in considerazione come la seconda opzione richieda maggiori risorse e risulti un punto critico del sistema (in gergo *single point of failure*) a causa del suo comportamento *fail close* citato in precedenza.

2.3.2 Distribuited Intrusion Detection System

Un IDS distribuito (dIDS) consiste nel deployment di più IDS su larga scala, i quali sono in grado di comunicare tra loro oppure con un'unità centrale. Questa comunicazione facilita il monitoraggio avanzato della rete, l'analisi di incidenti e la raccolta istantanea di dati riguardanti gli attacchi. Inoltre la cooperazione permette agli amministratori di avere una visione più ampia delle proprie reti nel complesso. Un dIDS permette inoltre un'analisi più efficiente degli attacchi ricevuti, grazie alla raccolta centralizzata dei rispettivi dati, e la possibilità di rilevare eventuali trend a livello globale.

Gli elementi caratterizzanti i dIDS sono 3.

Server centrale di analisi Come dice il nome stesso rappresenta l'unità centrale. Svolge le operazioni di analisi sui dati raccolti e memorizza le informazioni ottenute nel proprio database, che viene reso disponibile agli agenti distribuiti per eventuali interrogazioni. Lo scopo è quello di condividere le informazioni per migliorare i risultati ottenibili. Spesso viene affiancato da un server web che mette a disposizione degli utenti un interfaccia grafica user friendly.

Agenti cooperanti distribuiti Si tratta di IDS completi ed installati presso le reti che aderiscono al dIDS e il loro incarico è di raccogliere i dati

dalle proprie reti per poi inoltrarli al server centrale per farli analizzare. Per ottenere i migliori risultati le reti devono esser distribuite geograficamente in modo da coprire la più vasta area possibile.

Aggregazione dei dati Non si tratta di un componente software o hardware, ma di un processo fondamentale per l'ottenimento degli obiettivi preposti. Questa operazione viene effettuata sul server centrale e ha lo scopo di aumentare il valore dei dati raccolti dai singoli agenti. Infatti è grazie all'aggregazione dei dati che è possibile rilevare la diffusione su larga scala delle minacce identificate a livello di singole reti. Si pensi alla segnalazione di un malware, rilevarlo 100 volte all'interno della stessa rete e rilevarlo una volta in 100 reti distinte ha significati ben differenti. Nel primo casi si tratta di un infezione locale, probabilmente un host compromesso ha infettato le macchine all'interno della propria. I danni risultano circoscritti alla rete in questione e l'intervento potrà solo salvare eventuali macchine ancora pulite. Nel secondo caso invece, esiste una nuova minaccia che ha già raggiunto diverse zone del mondo e che se non bloccata tempestivamente rischia di portarte a conseguenze devastanti. L'intervento dev'essere immediato. Per invogliare la partecipazione a questi progetti, vengono garantiti adeguati livelli di privacy. I dati raccolti infatti, verranno elaborati in forma anonima.

Lo svantaggio nell'utilizzo di dIDS sta nella complessità di tali sistemi. Devono essere studiati attentamente per garantire la massima sicurezza, inoltre richiedono la collaborazione tra enti differenti, quindi la scelta di uno standard comune. D'altra parte i benefici sono notevoli e possono essere paragonati a quelli che si ottengono passando da un HIDS ad un NIDS. Inoltre una visione più ampia migliora i tempi di reazione alle nuove minacce e la gestione più efficiente permette anche il risparmio di risorse. Quest'ultimo punto aumenta d'importanza in considerazione del crescente numero di minacce da cui difensersi e di conseguenza del costante aumento di risorse richieste per difendersi appunto da tali minacce.

2.4 Reti peer-to-peer

Le reti peer-to-peer, indicate anche con la sigla p2p, sono costituite da nodi aventi diverse connessioni dirette verso gli altri partecipanti alla rete al contrario dei tipici sistemi centralizzati dove i collegamenti sono unicamente tra client e server. Per funzionare ogni nodo deve implementare sia le funzioni del server che quelle del client a cui ricorrerà di volta in volta a seconda della situzione. I vantaggi di questa tipologia di rete sono molti.

efficienza di rete L'utilizzo di nodi che svolgono funzioni sia id client che di server, permette una migliore gestione della banda disbonibile, poichè si può contare su tutta quella messa a disposizione dai singoli collegamenti. In un sistema centralizzato invece la banda sarebbe limitata alla capacità offerta dal nodo centrale.

maggiore capacità di storage Ogni nodo che partecipa alla reteporta in dote una certa quantità di storage che sarà a disposizione di tutti. Quindi anche se l'investimento per un singolo nodo fosse ridotto, all'aumentare del numero di nodi si raggiungerebbero comunque capacità notevoli. Oltre al minore costo unitario, vi è anche la possibilità di raggiungere livelli superiori a quelli possibili in caso di unità centralizzata.

maggiore potenza computazionale Oltre alla capacità di storage, ogni nodo della rete rende disponibile una certa potenza computazionale proporzionale alle proprie caratteristiche hardware. Anche ipotizzando che della potenza totale venga resa disponibile una piccola percentuale, è possibile ottenere risorse complessive molto elevate. Ne sono un chiaro esempio i diversi progetti di calcolo ditribuito aperti al pubblico, che grazie a questa tecnica sono in grado di ovviare all'utilizzo di potenti e costosi supercomputer[3].

[3]Qusta affermazione non intende supporre che l'utilizzo di reti di calcolo distribuito

migliore resistenza ai guasti La possibilità di distribuire il carico di lavoro evita di avere un unico point of failure, di conseguenza la resistenza ai guasti risulta superiore. Ovviamente l'aumento degli elementi in gioco aumenta anche la probabilità di riscontrare malfunzionamenti, ma quest'ultimi risultano circoscritti e non influenzano l'intera rete. Con uno studio accurato dei vari parametri in gioco, sarà inoltre possibile raggiungere valori di affidabilità molto elevati, che in caso di sistemi centralizzati risulterebbero difficilmente ottenibili e molto costosi.

Un altro elemento che trova terreno fertile in questa tipologia di reti, senza però rappresentarne un vantaggio intrinseco, è quello dell'anonimato. A seconda dell'implementazione utilizzata è possibile per i nodi cominicare in forma anonima. Le cause principali si riscontrano nel'assenza di un punto centrale, che possa tenere traccia delle operazioni svolte dai singoli nodi, e dalla possibilità di assegnare ai nodi riferimenti indipendenti dagli indirizzi IP. Un esempio di rete p2p fortemente orientata all'anonimato è sicuramente Freenet.

In linea di massima le reti p2p si possono distinguere tra pure e ibride. Le prime sono costiutite da nodi tutti considerati allo stesso livello, *peer* appunto, mentre le seconde, nonostante si basino sugli stessi principi, si appogiano a nodi che svolgono esclusivamente le funzioni di server.

Una caratteristica importante delle diverse implementazioni di reti p2p è l'*overlay routing*. L'idea alle sue spalle è quella di astrarre il concetto di routing, in questo modo le risorse cercate possono essere di diversa natura (e non solo indirizzi). L'evoluzione storica dell'overlay routing è la seguente.

Sistemi con directory centralizzata Inizialmente le reti p2p venivano implementate appoggiandosi a server incaricati di tenere traccia dei contenuti resi disponibili da ogni nodo partecipante. In questo modo le

possa sostituire quello di potenti supercalcolatori, ma che accettandone le limitazioni può risultare un'economica alternativa.

ricerche risultavano particolarmente efficienti mentre le reti risultavano di più semplice realizzazione. Lo svantaggio era legato alla vulnerabilità dei nodi centrali che raprresentavano un punto critico per l'intero sistema. Infatti in caso di guasto, i dati sui nodi rimanevano salvi, ma risultavano irreperibili.

Sistemi con ricerca Flood-based Rientra nella categoria di reti p2p pure. Il concetto alla sua base è semplice: ogni nodo inoltra le richieste ai suoi vicini. Il vantaggio principale è quello di non avere single point of failure, ma purtroppo gli svantaggi sono notevoli. Primo tra tutti la scarsa scalabilità, dovuta principalmente a due motivi: crescita esponenziale del numero di messaggi scambiati, col rischio di rimanere vittime di DoS involontari, e crescita lineare del costo di lookup. Questo significa che al crescere della rete aumentano le probabilità di congestione e i tempi di attesa.

Distribuited Hash Tables (DHT) Sfrutta il vantaggio delle tabelle basate su hash, già conosciute da tempo (vengono utilizzate in diversi sistemi tra i quali alcuni database), applicandolo in forma distribuita. Le tabelle hash si basano su funzioni (hash) in grado di associare un indirizzo ai parametri passati (chiave). Questo permette, al momento del recupero, di saltare, o meglio accelerare la fase di ricerca, poichè è possibile ottenere l'indirizzo dell'oggeto sempicemente calcolandone la sua funzione hash. Nel caso distribuito si associa ad ogni nodo un range di chiavi hash delle quali risulta direttamente responsabile (l'associazione avviene in base all'ID del nodo), in questo modo sia per l'inserimento che per il successivo prelievo è possibile indirizzarsi istantantaneamente verso il nodo responsabile, se lo si conosce, o verso un suo vicino. Questo sistema fornisce un'ottima scalabilità. Alcune tra le più note implementazioni di DHT sono: Chord[22], Pastry[25], Tapestry[24] e CAN[23].

Nonostante questo tipo di reti abbia raggiunto la notorietà presso il grande pubblico grazie agli applicativi di file sharing (a partire dal celeberrimo Napster), le applicazioni basate su questi protocolli sono parecchie e tra loro troviamo: file system distribuiti, piattaforme VoIP, piattaforme di streaming e i già citati sistemi di calcolo distribuito.

Capitolo 3

Architettura proposta

3.1 Elementi principali e loro disposizione

Nel capitolo precedente sono stati introdotti i principali strumenti di difesa attualmente a disposizione degli amministratori di rete. Il livello base è rappresentato dall'installazione di un IDS che integri al suo interno tutte le principali funzioni. Questo approccio può risultare sufficiente per piccole realtà lavorative, soprattutto se di basso profilo tecnologico, ma come si esce da questa scena, risulta necessario dotarsi di strumenti più complessi. I due principali ostacoli che si incontrano col crescere delle dimensioni, sono la diversificazione degli attacchi e soprattutto la quantità di traffico da analizzare, in entrambi i casi è richiesto il passaggio da un pacchetto integrato, ad un sistema che preveda l'utilizzo di più componenti, a partire dai sensori fino ad arrivare all'unità di raccolta e analisi dei dati. In base alle esigenze specifiche della rete monitorata si possono adottare diverse configurazioni, ma i punti chiave rimangono sempre gli stessi. Innanzitutto occorre sistemare almeno un sensore per ogni sottorete, in modo da non avere zone d'ombra, dopodichè si può sceglierne il tipo e il grado si sensibilità (un sensore potrebbe essere impostato per inoltrare solo determinati tipi di eventi). Tutti i sensori dovranno successivamente essere collegati ad uno o più manager. Esistono

diverse soluzioni commerciali in cui i sensori sono configurati per operare solo con determinati manager, mentre nel mondo open source spesso è sufficiente impostare semplici parametri all'atto dell'installazione per far comunicare prodotti differenti, questa possibilità semplifica notevolmente le operazioni di deployment, tuttavia per i più esperti è possibile ampliare ulteriormente la scelta ricorrendo ad estensioni ed a plug in di terze parti o addirittura scritti in casa.

I manager a loro volta si appoggeranno ad un database per la memorizzazione delle informazioni raccolte. Per fare questo occore scegliere un DBMS (DataBase Management System) compatibile con il manager installato. L'utilizzo di un database permette una memorizzazione ben schematizzata in grado di facilitare il successivo recupero delle informazioni mediante chiavi o anche utilizzando complesse query, che possono rappresentare una prima elaborazione dei dati (raggruppamento, ordinamento, controlli incrociati). Nonostante i vantaggi appena citati, l'utilizzo di un database non è necessario e può essere sostituito, o affiancato, dalla scrittura su file, sia semplici sia in formato XML.

Per quanto riguarda i manager, esistono anche interfacce grafiche in grado di offrire un front end user friendly all'amministratore. Quest'ultimo potrà così visualizzare le notifiche all'interno di una finestra invece che da terminale, con tutti i benefici che ne seguono: immediatezza, differenziazione basata su colori e/o figure, interazione mediante link e pulsanti.

3.1.1 I sensori

La loro scelta e la successiva installazione richiedono una particolare attenzione. Prima di tutto occorre decidere il tipo di minacce che si vogliono rilevare, poi cosa si vuole ottenere dai dati raccolti. Da queste considerazioni emergeranno le esigenze dalle quali dipenderà la scelta del prodotto che meglio le soddisfa. Esistono 3 categorie in cui possono rientrare i sensori.

Host based Devono essere installati su ogni singolo host e permettono un controllo molto dettagliato. Richiedono molte risorse sia all'atto dell'installazione, che può richiede una minuziosa configurazione, sia durante il normale funzionamento, inoltre generano un'elevata quantità di dati che richiede altre risorse per essere analizzata e spesso fornisce informazioni di poco rilievo. Per questo motivo il loro utilizzo è spesso limitato alle postazioni più importanti o addirittura assente[1].

Network based Hanno il compito di sniffare tutto il traffico che atrraversa il loro segmento di rete. Di solito vengono installati con il set completo di regole, ma è possibile impostarli in modo che rilevino solo determinati tipi di attacchi. Quest'operazione apparentemente abbassa il livello di sicurezza, ma se impostata correttamente è invece in grado di diminuire il numero di avvisi superflui senza effetti collaterali. L'importante è rimuovere le regole riguardanti vulnerabilità legate a componenti non presenti all'interno della rete analizzata. Per esempio in un segmento di rete in cui sono presenti server web basati su apache è possibile rimuovere le firme riguardanti attacchi a server IIS. Ovviamente occore fare attenzione a rimuovere solo ciò che è effettivamente superfluo, inoltre è necessario segnare queste modifiche in modo da tenerne traccia ed eventulmente inoltrarla assieme alle altre informazioni ottenute dalla raccolta dei dati.

Honeypot Già descritti in precedenza, questi strumenti usati come sensori offrono principalmente tre benefici.

Raccolta del malware Il motore interno è in grado di scaricare i file binari proposti e di memorizzarli all'interno della directory prestabilita. Una volta scaricati possono essere inoltrati al manager per

[1]Nell'architettura sperimentata sono stati utilizzati tutti è tre i tipi di sensori anche se quelli provenienti da quello *Host based* non sono rientrati nelle analisi ed elaborazioni successive alla raccolta

farli analizzare in modo da ottenere informazioni utili alla difesa da questo tipo di minaccia.

Riduzione falsi positivi I dati raccolti da questi sensori possono essere considerati come malevoli a priori. Si ricorda infatti che questi nodi non offrono servizi reali di conseguenza tutti i dati raccolti sono frutto di attacchi riusciti o tentati, come detto in sezione 2.2.

Traffico limitato Gli honeypots registrano esclusivamente il traffico ad essi destinato, quindi la quantità di dati raccolti risulta sensibilmente inferiore rispetto ai sensori di rete.

L'importanza di poter raccogliere malware risulta ancora più significativa all'interno della rete collaborativa. La maggior parte dei sensori di rete si basa infatti su regole o firme e sono quindi impotenti nei confronti di codici polimorfici o metamorfici. Grazie agli honeypot è però possibile raccogliere i binari, analizzarli ed estrapolarne linee guida per potersi difendere. Questo perchè anche se il codice cambia, i suoi riferimenti interni, come gli indirizzi a cui connettersi per ricevere ordini o i numeri di porta da aprire, rimangono gli stessi, quindi un'analisi che ricavi questi valori potrà suggerire di difendersi da essi, per esempio agendo sul firewall.

Sensori manager La maggior parte dei manager permette di inoltrare i dati raccolti ai manager di livello superiore, quindi si è in presenza di elementi che possono essere visti sia come manager sia come sensori. In questo secondo caso si tratta di sensori generici, cioè non appartenenti ad una delle tre categorie appena elencate, che forniranno dati di diversa natura.

3.1.2 I manager

I manager hanno il compito di raccogliere i dati provenienti dai sensori ad essi collegati, e a loro volta possono inoltrare il traffico ad altri manager di livello superiore. L'ambivalenza di questi elementi aumenta notevolmente la scalabilità del sistema è infatti possibile aumentare il numero di sensori, il numero di manager ed eventualmente il numero di livelli, senza influire più di tanto sulle prestazioni. Inoltre ogni rete può presentarsi all'esterno come un'unica fonte di dati nascondendo quindi la struttura interna e la relativa suddivisione in sottoreti.

Le principali funzioni messe a disposizione dai manager sono:

- Raccolta delle segnalazioni provenienti dai sensori. Come già detto, si tratta della funzione principale. I dati raccolti potranno essere aggregati in modo da ottenere diversi benefici, tra i quali: diminuzione del traffico in caso di successivo inoltro ed estrapolazione di informazioni aggiuntive.

- Memorizzazione delle informazioni raccolte su file, sia in semplice formato testo che in xml, o inserimento in un database d'appoggio.

- Inoltro dei dati mediante diverse forme di comunicazione tra le quali: socket, xmlrpc e http.

- Instaurazione di connessioni sicure verso i sensori o verso manager di livello superiore. Per raggiungere tale obiettivo è prima necessaria un'operazione di registrazione da effettuare tra sensore e manager in cui è possibile associare nome e descrizione al primo in modo da poterne agevolare l'identificazione da parte dell'utente. Durante la fase di registrazione avviene anche lo scambio delle chiavi pubbliche. Queste verranno utilizzate per criptare il contenuto di tutte le comunicazioni e per garantirne l'autenticità, elemento fondamentale per evitare che soggetti estranei possano recuperare informazioni sensibili e/o inserirne di

non veritiere per falsificare i risultati (per esempio sarebbe possibile realizzare un attacco DoS inserendo come malevolo l'indirizzo dell'host bersaglio).

- Mantenimento di un elenco aggiornato coi nomi dei sensori attivi e loro locazione. Questo è possibile grazie alla funzione dei sensori che ad intervalli regolari manda un segnale per confermare il proprio stato di attività (*heartbeat*).

3.1.3 Il collettore

Con questo termine si vuole indicare quel componente che rappresenta la destinazione di tutte le informazioni raccolte e che ha il compito di elaborarle. La sua presenza sarà unica nel caso di reti gerarchiche mentre vi saranno più istanze nel caso di rete p2p, ma come si vedrà il suo compito non cambia. Come primo passaggio il collettore dovrà raccogliere le informazioni provenienti dai managar ad esso connessi e dovrà essere in grado di disnguerle in base alla loro tipologia in modo da processarle nella maniera più adatta. Nel caso infatti i dati raccolti siano costituiti da codice binario di malware, le operazioni da compiere saranno quelle di analisi, da svolgere o localmente, nel caso siano presenti gli strumenti adeguati, o appoggiandosi a servizi esterni disponibili in rete. Se invece di codice binario si è in presenza di informazioni riguardanti tentativi di intrusione, di DoS o di altri attacchi in generale, il collettore avrà il compito di memorizzare tali informazioni localmente e verificare se siano stati raggiunti i valori di soglia stabiliti in precedenza, in caso affermativo dovrà diramare un messaggio che notifichi i dati in questione, eventualmente suggerendo le operazioni di difesa consigliate. Si pensi ad esempio alla rilevazione di un indirizzo ip che abbia già provocato svariate segnalazioni, è ipotizzabile che per precauzione questo indirizzo venga inserito in una blacklist provvisoria per evitare spiacevoli inconvenienti. Altro

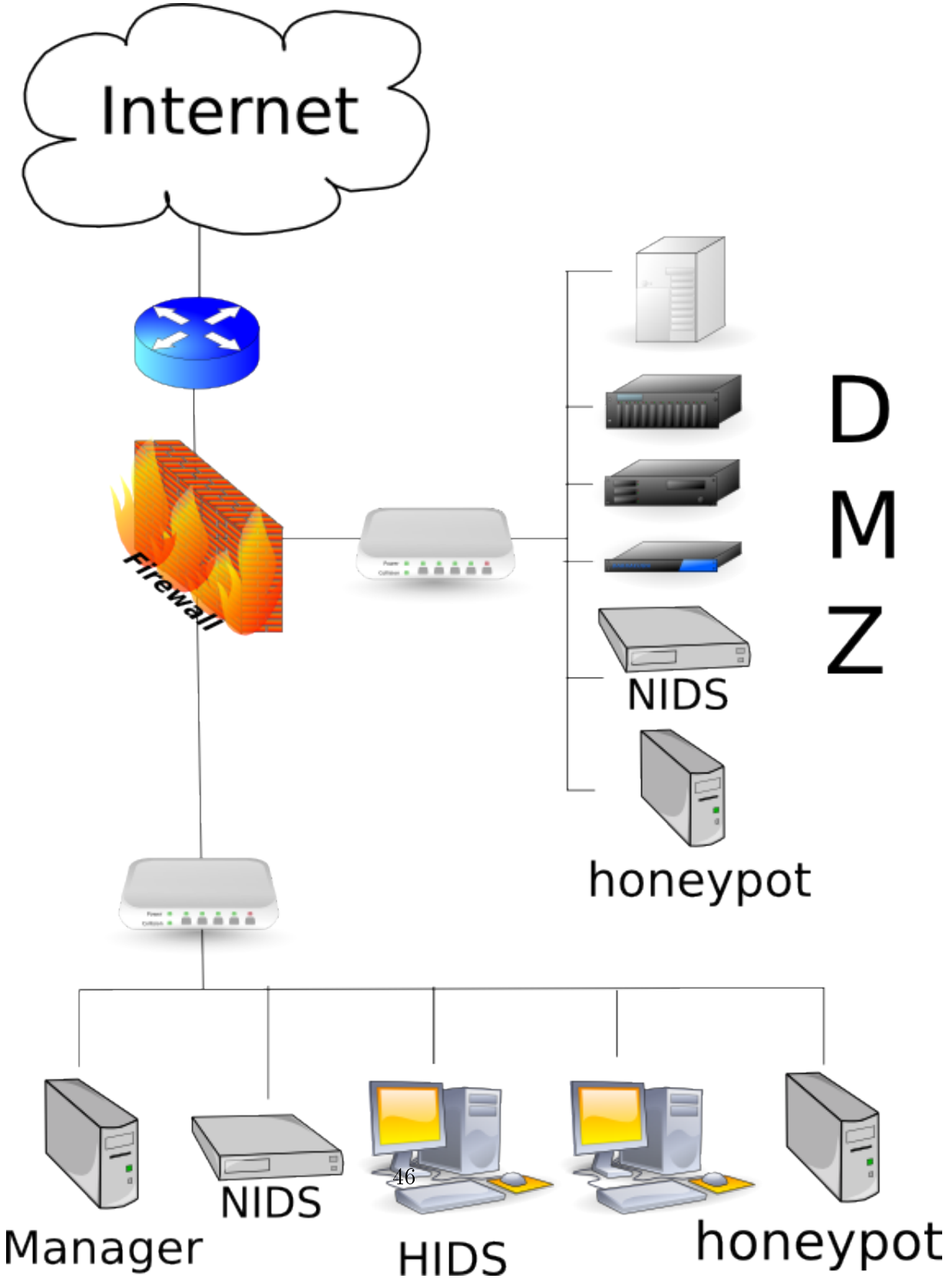

Figura 3.1: Esempio di distribuzione sensori all'interno di una singola rete

esempio può essere la notifica di diversi attacchi mirati verso una determinata porta, in questo caso gli amministratori di rete potrebbero essere avvertiti e verificare lo stato di eventualo servizi ad essa associata.

3.2 Raccolta gerarchica

Il primo approccio alle architetture distribuite è stato quello di tipo gerarchico. L'utilizzo di tali architetture ha permesso la collaborazione di più soggetti aventi il medesimo fine: aumentare la sicurezza delle proprie reti. Questo passaggio può essere visto come la trasposizione su larga scala di quello che già avveniva nelle singole reti e che già dava buoni risultati, ovvero il collegamento ad un manager centrale di più sensori installati in sottoreti differenti, eventualmente mediante manager intermediari. Come si è già detto in precedenza, una rete gerarchica risulta scalabile grazie all'utilizzo di manager disposti su più livelli in grado di trasportare tutti i dati raccolti dai sensori fino al collettore centrale. In questo modo una volta registrati i singoli manager, sarà possibile gestire il numero e il tipo di sensori locali in modo totalmente autonomo. La registrazione serve infatti per poter stabilire connessioni sicure tra i sensori e i manager o tra i manager stessi, utilizzando la crittografia assimmetrica basata sullo scambio di chiavi pubbliche, risulta così possibile garantire la segretezza e l'autenticità dei dati trasmessi. Questo significa che registrando un manager ci si fida dei dati da lui trasmessi e lo si lascia libero di gestire i livelli sottostanti.

Il collettore, risulterà l'utlimo anello della catena, o meglio l'ultimo gradino della scala gerarchica, e riceverà i dati dai manager ad esso collegati o a flusso continuo o a intervalli prestabiliti (polling). Dovrà essere dotato di risorse proprorzionate alle dimensioni delle reti monitorate e dovrà essere controllato costantemente. Il vantaggio di questa disposizione risiede nella possibilità di sgravare tutti i nodi sottostanti dalle pesanti operazioni di analisi approfondita dei dati, i manager infatti, potranno limitarsi ad effettuare

semplici operazioni di aggregazione giusto per ottimizzare il traffico di rete, ma il grosso del lavoro verrà svolto dal collettore. Quest'ultimo avrà anche l'incarico di informare i nodi sottostanti riguardo i rischi riscontrati e le eventuali contromisure. Per farlo potrà implementare un sistema di comunicazione diretta verso i manager ad esso collegati che a loro volta inoltreranno i messaggi ricevuti verso i livelli più bassi, in alternativa potrà appoggiarsi a sistemi di comunicazione paralleli, ovvero esterni all'architettura realizzata, per esempio mailing list o avvisi su siti web consultabili liberamente o dietro registrazione.

La vulnerabilità principale di questo sistema, risiede nel *single point of failure* costituito dal collettore. Nel caso infatti in cui questo componente dovesse guastarsi, o risultasse inadeguato al carico di lavoro richiesto, ne soffrirebbe l'intera rete collegata ad esso. Tutte le informazioni raccolte rimarrebbero in attesa di essere elaborate, o peggio ancora potrebbero andare perse, e tutte le reti che facessero affidamento ad esso si ritroverebbero improvvisamente vulnerabili come se non stessero collaborando. La situzione sarebbe aggravata dalla possibilità che quest'utlime, rimanendo all'oscuro della caduta del collettore, continuassero la loro normale attività senza nessuna particolare attenzione, che verrebbe invece presa se aggiornate sullo stato del sistema. Fortunatamente una buona progettazzione e un controllo adeguato rendono quest'eventualità abbastanza remota, ma sicuramente non trascurabile e neanche l'unica situazione negativa. Oltre alla peggiore delle ipotesi appena descritta, esistono altri casi decisamente seri, ovvero la caduta dei manager intermedi. Se infatti il collettore risulta il componente più importante e di conseguenza il più curato, non è detto che avvenga altrettanto per i nodi ai livelli intermedi. In questi casi la gravità del danno risulta porporzionale al numero di reti dipendenti da quel manager, sarà quindi maggiore nel caso si tratti di un collegamento diretto col collettore, mentre risulterà ininfluente (per gli altri partecipanti) nel caso rappresenti il punto di collegamento per una singola rete.

Esiste anche un secondo grande svantaggio, che consiste nel possibile sbilanciamento del carico. Ipotizzando infatti che una rete si trovi sotto attacco, il nodo corrispondente genererà un quantitativo di alert molto superiore alla norma, col risultato che il ramo a cui è collegato verrà sovraccaricato. Il problema risulterebbe aggravato nel caso finissero sotto attacco più nodi appartenenti alla stessa diramazione, ipotesi non certo remota. La conseguenza principale sarà un rallentamento delle operazioni svolte dal manager di riferimento, fino ad arrivare, nel peggiore dei casi, al suo collasso. Questo problema è dovuto al mancato supporto del bilanciamento del carico da parte delle architetture gerarchiche e rappresenta forse il principale vincolo di progettazione per una loro reale scalabilità.

3.3 Raccolta distribuita su rete peer-to-peer

Rappresenta l'evoluzione della raccolta gerarchica e ne condivide la struttura del livello più basso. Ogni rete che intenda partecipare alla collaborazione dovrà affacciarsi mediante un'unico punto d'accesso[2] a cui farà confluire tutti i dati raccolti dai sensori locali. La grande differenza sta nel metodo di collegamento delle diverse reti. In questo caso non si avranno diversi livelli composti da manager che raccolgono e aggregano i dati fino ad inoltrarli al collettore centrale, i nodi infatti saranno tutti collegati tra loro mediante una rete peer-to-peer, le cui caratteristiche son già state accennate in precedenza. L'idea alle spalle di quest'architettura è quella di sfruttare i benefici di questa tipologia di reti, in particolare la migliore resistenza ai guasti e il bilanciamento del carico. Le conseguenze sono notevoli e vanno a incidere sia sulle singole implementazioni sia sul funzionamento comlessivo. Vediamo ora nel dettaglio l'organizzazione di qust'architettura.

Abbiamo detto che ogni singola rete partecipante si affaccerà con un unico punto d'accesso, questi punti all'interno dell'architettura verranno con-

[2]L'utilizzo di più punti d'accesso è previsto solo in determinati casi discussi in seguito

siderati come nodi che, trattandosi di peer-to-peer, non dovranno subire distinzioni e non verranno suddivisi in livelli, tutti i nodi infatti saranno equivalenti e svolgeranno le stesse funzioni.

Dal punto di vista dell'amministratore che intenda partecipare alla collaborazione con la propria rete, saranno necessari tre passaggi.

1. Il primo passaggio consisterà nell'installazione di un componente software che lo metta in grado di partecipare attivamente alla rete p2p. Essa potrà avvenire su una macchina indipendente, nel caso sia consentito un accesso remoto ai dati, altrimenti l'utente sarà vincolato ad effettuarla sulla macchina su cui viene effettuata la raccolta degli stessi.

2. Successivamente, perchè il programma possa funzionare, sarà necessario configurarlo in modo che possa accedere ai dati raccolti dal manager. Questo significa che andrà indicata la sorgente (per esempio DBMS) e che dovranno essergli accreditati i privilegi necessari.

3. Infine, occorrerà dedicare al programma una parte delle risorse offerte dalla macchina su cui è installato.

Per quanto riguarda l'utimo punto, ci si riferisce in modo particolare al libero utilizzo della cpu per poter elaborare i messaggi scambiati con gli altri nodi e ad uno spazio per lo storage dei dati ricevuti. Quest'ultimo dovrà avere una dimensione che rientri nei limiti concordati dal'insieme dei partecipanti, in modo da garantire la maggiore omogeneità possibile. Nel caso la dimensione disponibile fosse superiore al limite massimo si potrà aggirare il problema lanciando due o più istanze del programma, in questo modo la rete distribuita vedrà più nodi senza accorgersi che appartengono alla stessa rete locale o addirittura alla stesso host, l'importante è tenere in considerazione e soddisfare l'aumento delle richieste di utilizzo della CPU, nonchè assegnare alle varie istanze distinti numeri di porta e distinte directory di memorizzazione. Un altro requisito importante è la garanzia di una collegamento alla rete

sufficientemente stabile possibile onde evitare perdita di informazioni ed un eccessivo traffico di overhead legato alla sincronizzazione dei nodi per ogni nuovo nodo entrato/uscito. Considerando l'ambito professionale a cui è rivolta quest'architettura si tratta di un requisito che non dovrebbe sollevare grosse difficoltà, verranno comunque fornite maggiori informazioni **in seguito**.

3.3.1 Funzionamento

Completate le procedure di installazione e configurazione, analizziamo ora il funzionamento del programa. La prima operazione sarà quella di leggere i dati raccolti. Le possibilità sono essenzialmente due: o la lettura viene effettuata ad intervalli regolari (modalità pull o polling) o viene sollecitata da chiamate esterne. Una volta acquisiti, i dati verranno elaborati singolarmente e per ogni categoria prevista sarà creato un messaggio apposito che verrà poi inserito nella rete. Per la precisione, la parte riguardante lo storage e la condivisione dei dati raccolti sarà basata su una DHT e ogni messaggio avrà una chiave su cui verrà effettuato l'inserimento. In questo modo vi sarà un determinato responsabile per ogni contenuto inserito e sarà quello con il nodeId più simile alla chiave. Una corretta costruzione delle chiavi permetterà inoltre di ottenere un controllo più fine sulle segnalazioni. Infatti l'inserimento di un messaggio contenente determinati dati, potrà essere effettuato più volte con chiavi differenti in modo che vada ad incrementare eslcusivamente i contatori desiderati (Si potrebbe voler segnalare la porta da cui è stato scaricato un malware, ma non quella da cui si è ricevuta una connessione ssh[3]). Tornando al nodo responsabile, una volta ricevuto il messaggio, questo svolgerà le funzioni descritte per il collettore, ovvero raccolta ed elaborazione dei dati in esso contenuti e successiva informazione di tutti i nodi partecipanti alla rete. Per quanto riguarda quest'ultima operazione

[3]La porta di download di un malware potrebbe essere la stessa da macchina a macchina, mentre un client ssh aprirà una porta alta più o meno casualmente.

sarà possibile implementare un'applicazione di comunicazione brodcast. Le reti p2p infatti si prestano bene per le comunicaizoni su larga scala e vengono spesso utilizzate in caso di trasmissioni multicast, come per esempio lo streaming di eventi live o on-demand[33]. Questa possibilità può essere sfruttata all'interno dell'architettura offrendo ai partecipanti un componente che integri tale funzione, in particolare ogni nodo sottoscriverà i temi sui quali vorrà rimanere aggiornato e l'applicazione gestirà in automatico trasmissione e ricezione dei messaggi[4]

Ecco un paio di esempi di funzionamento tipico dell'architettura proposta.

- Quando viene salvato il codice binario di un malware lo si inserisce nella rete adoperando come chiave il relativo hash. A questo punto vi sarà un nodo con id vicino all'hash del malware che si occuperà di analizzarlo e successivamente di inoltrare a livello broadcast le informazioni ottenute. Gli altri nodi sapranno quindi come difendersi da quel malware senza doverlo inserire nuovamente nella rete ottenendo vantaggi in termini di tempo e di banda risparmiata. Nel caso invece un altro nodo ricevesse ed inoltrasse il codice sospetto, prima di ottenere i risultati di un'analisi già in corso, il ricevente sarebbe sempre lo stesso nodo, di conseguenza non verrebbe effettuata nuovamente l'analisi.

- Se invece l'attività sospetta registrata riguarda un particolare indirizzo IP, sarà questo che verrà utilizzato per la creazione della chiave. In questo modo il nodo responsabile riceverà le segnalazioni provenienti dai diversi nodi (eventualmente anche più segnalazioni dallo stesso nodo) e una volta raggiunto il valore di soglia provvederà ad informare tutta la rete mediante un messaggio broadcast.

[4]La possibilità di sottoscrivere argomenti differenti avrà effetti esclusivamente sulla ricezione, per la trasmissione infatti ogni nodo dovrà comunicare ogni informazione ricavata dall'elaborazione dai dati, indipendentemente dagli interessi locali. Questo per garantire una corretta e completa informazione.

Con riferimento al primo dei due esempi viene di seguito descritta la serie completa dei diversi passaggi compiuti dall'architettura in risposta ad un'infezione da malware.

1. L'attacante riesce a contattare l'host vittima, impersonato dall'honeypot.

2. L'host scarica il codice binario del malware e lo salva su disco.

3. La segnalazione viene inoltrata fino al manager di riferimento di quella rete.

4. Il programma realizzato legge i dati raccolti e crea un messaggio con una struttura ben definita in cui son presenti diverse informazioni oltre ovviamente al codice binario. La chiave utilizzata per l'inserimento sarà derivata dall'hash del binario.

5. Prima di inviare il messaggio, controllerà che la chiave non sia già stata inserita, in questo caso infatti non sarà necessario spedire il messaggio, perchè il malware raccolto è già stato inserito da un'altro nodo e al momento sarà sotto analisi. Supponendo che la chiave non sia presente, viene effettuato l'invio.

6. Il messaggio attraversa più nodi, fino ad arrivare a destinazione, cioè al nodo con nodeId più simile. Le repliche avranno come destinazione i nodi vicini.

7. Il nodo responsabile effettuerà l'analisi del messaggio, che potrà avvenire localmente o mediante l'utilizzo di servizi remoti.

8. Una volta ottenute le informazioni dettagliate riguardanti il funzionamento del malware, verranno selezionate quelle utilizzabili per la realizzazione di apposite regole di difesa.

9. Il nodo provvederà a pubblicare le informazioni selezionate mediante un canale broadcast

3.3.2 Confronto architetture: distribuita vs. gerarchica

Dopo aver visto l'architettura distribuita e il suo funzionamento è possibile effettuare un confronto diretto con l'architettura gerarchica descritta nella sezione 3.2, dal quale emergono principalmente due vantaggi: tolleranza ai guasti e bilanciamento del carico.

Tolleranza ai guasti Il collegamento di tutti i nodi allo stesso livello e la distribuzione su di essi dei compiti di analisi evita il problema del *single point of failure* tipico delle architetture gerarchiche. Infatti come descritto nella sezione a loro dedicata, il guasto di un nodo col ruolo di manager comporta l'isolamento di tutti i nodi appartenenti a quel ramo della rete, nell'architettura distribuita invece, le conseguenze sono limitate agli attacchi gestiti da quel nodo, che in media saranno $1/N$ (con N=numero dei nodi). Inoltre non si perde la visione panoramica, poichè ogni nodo elabora dati provenienti da tutte le reti colpite, per ottenere questo risultato è importante scegliere correttamente i valori su cui costruire la chiave del messaggio in modo che per segnalazioni identiche vi sia un unico nodo responsabile.

Considerando inoltre la maggiore probabilità di failure del singolo nodo, è stata implementata una procedura di inserimento dei dati che permette di creare delle repliche dei file, in modo da diminuire i rischi di perdita e di aumentrare l'efficienza del recupero degli stessi. Il funzionamento è semplice, oltre alla copia primaria gestita dal nodo con id più simile[5] alla chiave, verrebbero create delle copie sui nodi vicini[6] in modo che in caso di fallimento

[5]La similarità si riferisce solo al prefisso dell'id, più è lungo il prefisso in comune più i due nodi sono simili, o vicini. Al contrario due id identici ad esclusione della prima cifrà, risulteranno molto distanti tra loro.

[6]Nel campo delle DHT il termine *vicinanza*, o il più preciso *vicinanza numerica*, viene usato in riferimento alla similarità degli id

del nodo principale rimarrebbero comunque presenti nel sistema delle copie dei dati raccolti. Inoltre la rete manterebbe costante il numero di repliche grazie a controlli periodici, in modo da evitare che il fallimento di più nodi possa compromettere di volta in volta un numero crescente di copie fino a farle scomparire. Si fa notare inoltre che l'assegnazione degli id ai nodi non ha relazioni col collocamento geografico, di conseguenza nodi vicini nella DHT potrebbero essere a miglia di kilometri di distanza quindi tra loro indipendenti dal punto di vista di eventuali malfunzionamenti della rete Internet, a partire dai problemi riscontrabili sui server del rispettivo ISP fino ad arrivare a casi più seri come il troncamento di dorsali in fibra ottica [34].

L'aumento dell'efficienza nel recupero dei file introdotto dalle repliche è dovuto invece al sistema implementato dalla DHT che nel segnalare al client il nodo contenente il file richiesto, sceglie tra le repliche il più vicino in senso assoluto[7]. Il messaggio di richiesta verrà inoltrato normalmente, ma invece di avere come destinatario il nodo con nodeId più simile, verrà scelto quello che tra possessori del file cercato, risulta meno distante. In questo modo il trasferimento del file sarà più veloce e si consumerà meno banda.

Un altro vantaggio derivante dalla distribuzione dell'elaborazione dei dati su più nodi è che permette ai diversi amministratori di partecipare alla rete senza richiedere requisiti d'accesso particolarmente elevati, soprattutto se il compito di analisi, cioè quello che richiede maggiori risorse, viene affidato a servizi esterni raggiungibili via web. Nel caso invece dell'architettura gerarchica risulta necessario prestare particolare attenzione ai nodi situati ai livelli superiori, mentre per il collettore centrale sarà richiesto un vero e proprio investimento in termini di progettazione e assegnazione delle risorse necessarie.

Bilanciamento del carico Il secondo grande vantaggio nell'utilizzo di un'architettura distribuita, consiste nel bilanciamento del carico. Ogni nodo

[7] I parametri utilizzati per determinare la distanza assoluta possono essere diversi, esempi tipici sono il numero di hop e il Round Trip Time (RTT)

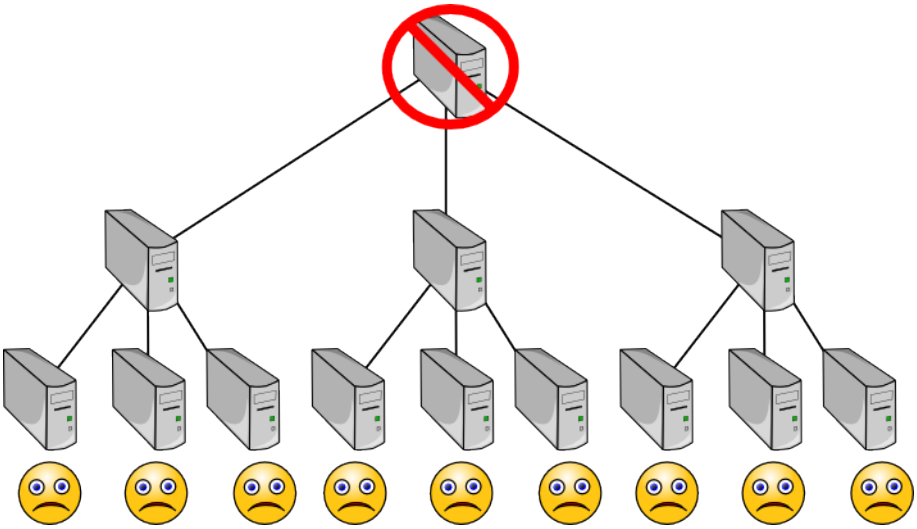

Figura 3.2: Sbilanciamento del carico su architettura gerarchica. Il ramo evidenziato in rosso sarà sovraccaricato.

infatti non trasmette i dati esclusivamente ad un singolo nodo, come avviene in una struttura gerarchica, ma per ogni evento generato diverso dai precedenti, inoltrerà un messaggio ad un nodo differente. Questo significa che anche nel caso una rete venga presa di mira e si ritrovi sotto un pesante attacco, non ci sarà il rischio di sovraccaricare un ramo o un nodo particolare, come visto invece nella sezione 3.2 riguardante le reti gerarchiche. Nel peggiore dei casi fallirà solo il nodo sotto attacco e le operazioni di analisi in corso presso di lui andranno perse, ma gli attacchi di cui era responsabile rimarranno disponibili presso i nodi che gestiscono le repliche e uno di questi subentrerà a quello fallito, esattamente come descritto nella tolleranza ai guasti. Per rendere le idee più chiare su come funzioni il bilanciamento è possibile osservare in figura. Si può notare come dal nodo sotto attacco vengano diramati i dati raccolti verso una moltitudine di nodi.

Per ritrovarsi nella situzione in cui un nodo risulta sovraccarico, sarebbe necessario che più reti subissero contemporaneamente il medesimo attacco, ma oltre ad essere un evento improbabile, risulterebbe attenuato dalla proce-

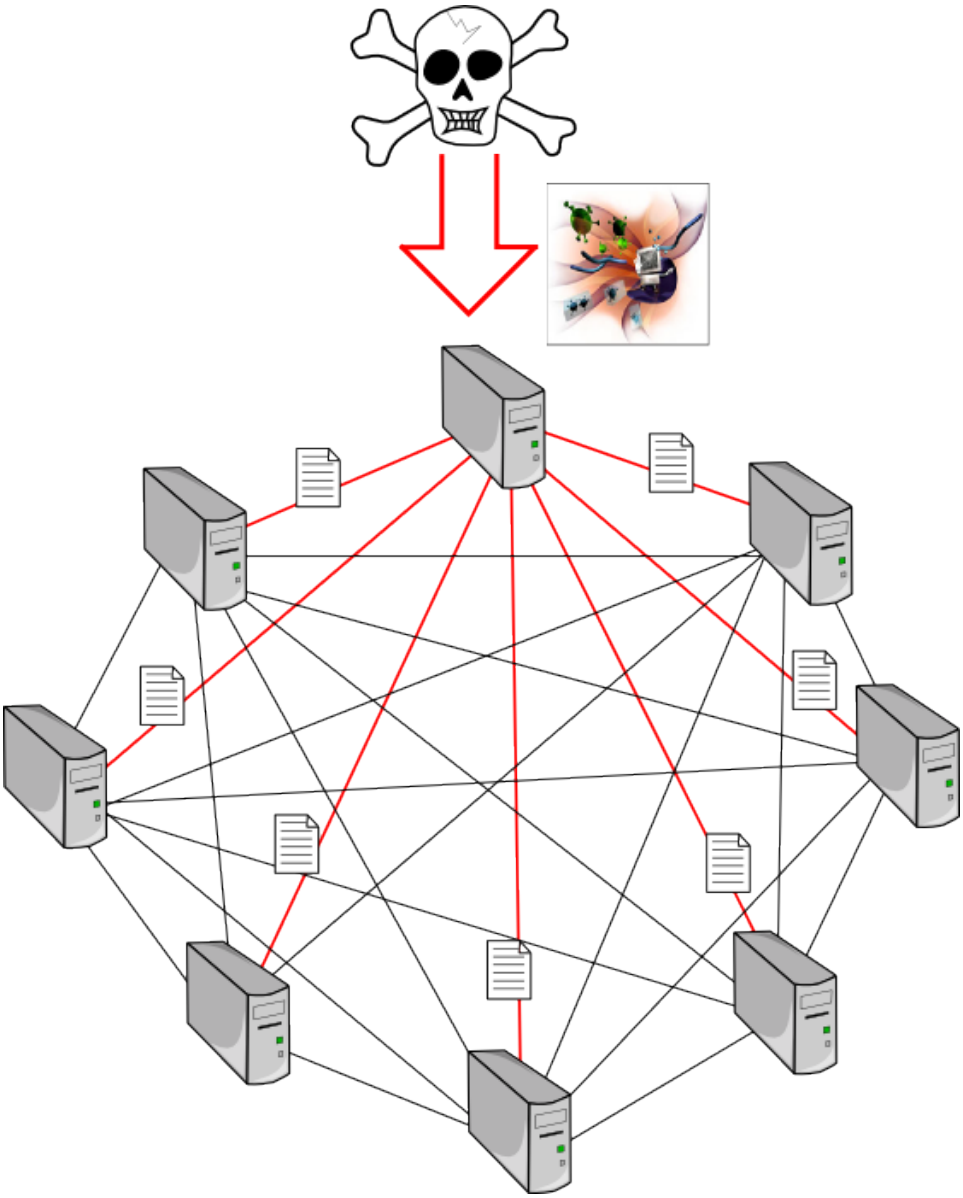

Figura 3.3: Bilanciamento del carico su architettura distribuita peer-to-peer. I rami evidenziati in rosso **non** sono sovraccaricati

dura di invio dei messaggi descritta in precedenza. Con riferimento all'esempio di invio di un malware, si evidenzia il punto 5, cioè il fatto che un nodo prima di effettuare l'inserimento di un messaggio, ne verifica la preesistenza. Questo permette un notevole risparmio di risorse, sia sotto l'aspetto della banda occupata, sia per quanto riguarda le operazioni di elaborazione richieste al nodo responsabile di quel determinato attacco.

Numero di copie memorizzate Un altro vantaggio che si ottiene dal passaggio a un'architettura peer-to-peer, risiede nel minor numero di copie dei singoli file all'interno dell'architettura stessa. Per dimostrarlo, basta valutare la crescita del numero di copie (in seguito C) in relazione al numero di nodi appartenenti all'architettura gerarchica. Considerando l'albero completo, supponiamo di trasmettere una segnalazione o un malware non ancora inviato da nessuno, in questo caso ogni manager all'interno della catena che va dal nodo foglia al collettore, memorizzerà una copia del file ricevuto, quindi avremo $C = h$ con $h = \log_k n$ dove h rappresenta l'altezza dell'albero, nè il numero di nodi foglia, mentre kè l'ordine dell'albero, ovvero il numero di diramazioni di ogni nodo. Il risultato ottenuto rappresenta il minimo valore possibile, che aumenterà per ogni nuova copia segnalata da foglie appartenenti a rami differenti, fino a raggiungere il valore massimo di $\sum_{i=0}^{h-1} k^i$.

Questo significa che la crescita del numero di copie risulta come minimo logaritmica rispetto al numero di foglie e può raggiungere valori molto elevati nonostante ogni manager non inoltri copie già inserite in precedenza. Nell'architettura proposta invece, il numero di copie è fisso e pari al fattore di replica più uno, come stabilito alla creazione della rete, quindi indipendente dal numero di nodi connessi.

Per rendere meglio l'idea si riportano in tabella alcuni esempi

	Minimo	Massimo
P2P con R=4	5	5
1000 nodi K=10	3	111
10000 nodi K=10	4	1111
100000 nodi K=10	5	11111
8000 nodi K=20	3	421
160000 nodi k=20	4	8421

Tabella 3.1: Numero di repliche dei file trasmessi all'interno della rete gerarchica

Capitolo 4

Realizzazione dell'architettura

4.1 Strumenti software utilizzati

Verranno ora descritti i software utilizzati per la realizzazione dell'architettura. Si premette che la scelta è stata focalizzata sull'utilizzo di **software libero**, a partire dal sistema operativo basato su una distribuzione GNU/Linux, di conseguenza non è richiesto l'acquisto di alcuna licenza commerciale per l'implementazione dell'architettura proposta[1].

4.1.1 Prelude

Prelude[13, 14] è un Hybrid IDS framework realizzato per permettere a tutte le applicazioni attinenti la sicurezza, sia open source che proprietarie, di inoltrare le informazioni ad un sistema centralizzato. Il raggiungimento di tale obiettivo si basa sull'utilizzo di IDMEF (Intrusion Detection Message Exchange Format) uno standard dello IETF (Internet Engineering Task Force)[15] che permette a sensori di diverso tipo di generare eventi utilizzando

[1] Un caso particolare è rappresentato da snort un IDS open source che prevede dietro pagamento l'aggiornamento costante del database delle firme. Questo abbonamento è tuttavia facoltativo, in quanto l'utente è libero di rimanere in ritardo di 30 giorni sulle firme rilasciate e di implementarne di proprie.

lo stesso linguaggio. L'idea alla base di questo prodotto è nata per rispondere all'esigenza di una maggiore sicurezza che, secondo gli autori del programma, risulta raggiungibile soltanto mediante la raccolta di informazione da più parti e di diverso tipo. Da qui il termine *Hybrid* (ibrido) volto a sottolineare questa sua caratteristica che gli permette di raccogliere informazioni da sensori sia *Host Based*, che *Network Based*, sia *Log Analyzer* che *Honeypots*.

4.1.1.1 Componenti di prelude

I componenti principali di prelude utilizzati dalla nostra architettura sono:

libprelude Sono le librerie alla base del framework. Gestiscono le comunicazioni sicure verso uno o più manager o tra i manager stessi e mette a disposizione delle API (Application Programming Interface) per creare degli eventi basati su IDMEF nel caso i sensori utilizzati non fossero già conformi allo standard e per coprire una quota maggiore di prodotti sono disponibili sia per il linguaccio C che per python e perl. Un altro servizio offerto da queste librerie è quello di *failover*, infatti nel caso il manager risultasse irraggiungibile, il sensore potrà memorizzare i dati da trasmettere in un file locale. Infine viene offerta la possibilità di creare un sensore in grado di leggere i dati ricevuti dai manager, che potrebbe essere utilizzata per esempio per realizzare un'applicazione interattiva.

Prelude-manager Il PreludeManager è un server ad alta disponibilità che raccoglie e normalizza i dati giunti dai sensori distribuiti e li memorizza nel database, o su altri mezzi in base alle preferenze dell'utente. Permette inoltre di filtrare gli eventi ricevuti, così da poter associare operazioni differenti a categorie differenti, e di inoltrarli ad altri manager, caratteristica molto importante che, come già detto in precedenza, permette la realizzazione di collegamenti gerarchici.

PreludeDB Si tratta delle librerie che permettono di accedere ai dati dei database mediante un livello di astrazione che consente di trasmettere i dati utilizzando IDMEF e senza preoccuparsi di eventuali comandi SQL e del DBMS scelto.

Prewikka Rappresenta la console di prelude e fornisce un'interfaccia grafica e funzionalità evolute. Verrà approfondita in seguito.

Esiste anche il modulo *Prelude-LML*, si tratta di un *Log analyzer*, di conseguenza rientra nella categoria degli *Host Based IDS*, ma nel nostro caso non è stato utilizzato e per quella categoria di sensori si è fatto afidamento ad un'altro prodotto.

4.1.1.2 Architettura di prelude

I modelli base di un'architettura basata su Prelude sono tre così come i componenti base da essi utilizzati. Quest'ultimi sono i sensori, i manager e la console (rispettivamente Prelude-manager e Prewikka). I sensori sono programmi che analizzano una serie di dati e che nel caso riscontrino attività sospette lo segnalano al manager mediante lo standard IDMEF che, per aumentare le prestazioni, può essere utilizzato in forma binaria invece che XML. Tornando alle architetture, eccone una breve descrizione.

Semplice Si tratta del modello base. Più sensori vengono collegati mediante connessioni TLS, in modo da garantirne un'adeguata protezione, ad un unico manager che memorizza i dati raccolti sul mezzo indicato dall'utente (in particolare database e file XML). Sarà inoltre presente la console che permetterà a quest'ultimo di monitorare lo stato del sistema.

Relaying Si tratta di una funzionalità che permette ad un Prelude-manager di inoltrare i dati ricevuti ad un altro Prelude-manager. Viene sfruttata

da tutte le architetture realizzate su più livelli e alla sua base viene adottato il modello *semplice* appena descritto.

Reverse relaying In alcune reti può risultare difficolto collegare al manager sensori provenienti da differenti sottoreti, visto che in certi casi queste vengono gestite mediante determinate policy del firewall. Si può intuire quindi l'esigenza di adottare un sistema che permetta di aggirare tali ostacoli. Per risolvere il problema Prelude utilizza il reverse relayng, dove il termine reverse sta ad indicare che le connessioni avvengo al contrario, cioè non è il manager-figlio (di livello inferiore) a collegarsi al manager-padre (di livello superiore), ma è quest'ultimo che stabilisce una connessione per prelevare i dati. Questa implementazione viene adoperata in tutti i casi in cui uno o più sensori siano dislocati all'interno della DMZ.

4.1.1.3 Lo standard IDMEF

Come già spiegato in precedenza, la capacità dei manager di Prelude di gestire diverse tipologie di sensori, si basa sull'utilizzo di un formato standard per la comunicazione dei dati: IDMEF[16]. L'acronimo sta per Intrusion Detection Message Exchange Format, che può essere tradotto come "formato per lo scambio di messaggi di intrusion detection" e le sue specifiche sono state realizzate dall'IETF. Questo formato nasce con l'intento di utilizzare un linguaggio XML, ma anche se questa versione risulta tuttora preservata per garantire la massima compatibilità, all'interno di Prelude è stato implementato un motore per la gestione di questo formato in versione binaria, in modo da ottenere maggiori prestazioni (minori dimensioni e maggiore velocità di elaborazione).

4.1.2 Prewikka

Prewikka è il componente di Prelude con cui interagisce l'utente. Può essere agganciato ad un server web già presente oppure utilizzare il proprio motore interno, in entrambi i casi si ottiene un'interfaccia grafica semplice ed efficace raggiungibile da tutti i terminali abilitati a connettersi al relativo indirizzo. Una volta collegati si possono svolgere diverse operazioni, oltre ovviamente al monitoraggio degli eventi abbiamo:

Aggregazione dei dati Prewikka effetta in automatico ed in tempo reale una correlazione sugli alert in base all'origine, alla destinazione e all'orario. Questa funzionalità consente di trasformare dati grezzi in informazioni facilmente leggibili dall'analista, che potrà inoltre intervenire personalizzando le regole alla base della correlazione.

Gestione permessi Permette di amministrare i permessi degli utenti, creandone di nuovi e definendo le funzionalità a loro disposizione.

Creazione filtri La definizione di criteri per rappresentazione degli eventi consente all'utente di evidenziare gli elementi da lui voluti, di impostare i dati da visualizzare a schermo o addirittura effettuare vere e proprie ricerche.

Monitoraggio sensori Ogni sensore deve contattare periodicamente il manager comunicandogli il proprio stato. Questi eventi vengono chiamati *hearbeats* e Prewikka offre una sezione dedicata al loro monitoraggio indicando quanti e quali agenti risultano online, offline con errori o offline senza errori.

4.1.3 Nepenthes

Nepenthes[9] è un diffuso honeypot a bassa interazione, che si distingue da altri prodotti della stessa categoria (ad esempio honeyd) perchè cerca di colmare il gap di funzionalità nei confronti di quelli ad alta interazione (es.

GenIII). Per raggiungere questo obiettivo, Nepenthes offre un emulazione dei servizi molto completa grazie ad una piattaforma modulare appositamente studiata. Di seguito verranno analizzati gli aspetti principali di questo prodotto.

Architettura

Nepenthes utilizza una struttura modulare che garantisce una buona flessibilità. Alla base di tutto troviamo il demone incaricato di gestire l'interfaccia di rete e di coordinare i diversi moduli che si suddividono in:

Vulnerability modules Sono i moduli principali che caratterizzano Nepenthes e che vanno ad attivare i meccanismi di raccolta del malware. L'idea alle loro spalle è quella che per essere infettati da un malware che si sta diffondendo autonomamente è sufficiente emulare le parti di servizi contenenti le vulnerabilità, così invece di emulare l'intero servizio si limitano ad implementare solo le parti rilevanti. Questo approccio garantisce inoltre un risparmio sul consumo delle risorse, permettendo l'installazione parallela di un numero di honeypot impensabile nel caso fossero di tipo ad alta interazione. Spesso l'emulazione di vulnerabilità risulta molto semplice, poichè si limita a restituire le risposte che si aspetta il malware lasciandogli credere di aver completato l'*exploit* e raccogliendo così l'eventuale payload da passare ai moduli successivi.

Shellcode parsing modules Questi moduli analizzano i payload ricevuti e ne estraggono automaticamente le informazioni rilevanti riguardo il tentativo di *exploit*. Uno degli approcci è quello di cercare la chiave necessaria alla decodifica dello *shellcode*, capita infatti che il payload venga criptato per sfuggire alla rilevazione degli IDS (spesso si tratta di una chiave XOR). Se trovata si procede ad analizzare il contenuto mediante pattern detection in grado di rilevare funzioni comuni, ad esempio *CreateProcess()*, o la presenza di URL. I risultati vengono analizzati

ulteriormente alla ricerca di informazioni che permettano il download del malware, per esempio credenziali, e se trovate vengono passate al modulo successivo.

Fetch modules Questi moduli hanno il compito di effettuare il download del malware dalla locazione remota. Per fare questo sono stati implementati i protocolli più diffusi trai i quali TFTP, HTTP, FTP, *csend/creceive* propri dei server IRC e periodicamente vengono aggiornati nel caso i malware implementino protocolli propri. L'operazione di download, nonostante rispecchi fedelmente il comportamento del malware, può sollevare qualche perplessità poichè una macchina sotto controllo va a contattare volontariamente un altro host, che con molte probabilità è incosapevole di quanto sta accadendo. Per questo motivo Nepenthes garantisce, all'utente che lo ritenga più opportuno, la possibilità di disabilitare questi moduli, in questo caso si otterranno comunque informazioni importanti relative all'exploit grazie ai moduli precedenti, ma risulteranno limitate.

Submission modules Questi moduli gestiscono i download completati con successo e si dividono in quattro categorie:

- quelli che memorizzano i contenuti scaricati in una determinata posizione del filesystem
- quelli che inoltrano i file ad un database centrale
- quelli che li inoltrano ad altri sensori Nepenthes permettendo la realizzazione di soluzioni gerarchiche
- infine quelli che spediscono il conenuto a servizi antivirus per ulteriori analisi

Logging modules Sono i moduli che tengono traccia degli eventi registrati e delle informazioni riguardanti i servizi emulati e aiutano a mantenere una visione generale sullo stato del sistema e sui dati raccolti.

Non si deve trascurare la possibilità che il malware non si diffonda automaticamente, ma venga caricato di volta in volta dall'attacante. In questo caso Nepenthes emula una shell di comando che mette a disposizione alcune funzioni basilari in accoppiata ad un file system virtuale in grado di tenere traccia separatemenre dei file creati in ogni sessione.

Deployment

Nepenthes fornisce una buona flessibilità in fase di deployment. La soluzione più semplice è sicuramente quella di installare un unico sensore all'interno della propria lan che operi autonomamente. Per reti più complesse è possibile installare più sensori che inoltrino il malware e log files ad un server centrale. Una seconda possibiltà per la gestione di più sensori è quella di collegarli gerarchimamente tra loro su più livelli. In questo modo ogni sensore inoltra i dati raccolti al livello superiore. Esiste infine una terza opzione che mediante l'utilizzo di VPN (Virtual Private Network) permette lo scambio di dati tra reti distribuite geograficamente.

Cattura di nuovi exploit

L'utilizzo di un honeypot permette in molti casi il rilevamento e la risposta ad attacchi cosiddetti *zero day* (*0day*), cioè quelli basati su vulnerabilità non ancora note o per le quali non sono ancora disponibili correzzioni (*patch*). Nepenthes cerca di migliorare questa potenzialità rispetto agli altri honeypots a bassa interazione, mediante l'utilizzo di due moduli particolari: *portwatch* e *bridging*. Il primo permette la raccolta del traffico di rete destinato a determinate porte, in questo modo nel caso il si tratti di tentativi di exploit sarà possibile ottenere tutte le informazioni a partire dal primo pacchetto, cosa che non sarebbe possibile basandosi su un Network IDS. L'altro modulo invece permette una completa gestione degli attacchi 0day tipica degli honeypot ad alta interazione appoggiandosi appunto ad uno di essi. Il funzionamento è abbastanza semplice: lo sfruttamento di nuove vulnerabilità

molto probabilmente uscirà dagli schemi previsti dagli emulatori dei servizi, in questo casto caso tali moduli reindirizzeranno il traffico in tempo reale verso l'honeypot d'appoggio che, grazie ad un implementazione completa del servizio e alla condivisione dello stato del modulo chiamante, sarà in grado di gestire il nuovo tentativo di exploit. Questa possibilità permette di installare diversi honeypot a bassa interazione che si appoggiano ad uno unico ad alta interazione, offrendo quindi un notevole risparmio di risorse.

In conclusione Nepenthes offre diversi vantaggi rispetto agli honeypot tradizionali, inoltre la sua modularità facilita un costante aggiornamento e permette all'utente esperto di integrare funzionalità aggiuntive, in modo da poterlo personalizzare secondo le proprie esigenze. Nepenthes rappresenta quindi un'ottima soluzione per la raccolta del malware e per questo motivo è stato scelto come honeypot alla base della nostra architettura.

4.1.4 Ossec

Ossec[11] è un Host-based IDS open source, scalabile e multipiattaforma. In particolare quest'utlima particolarità lo rende molto interessante poichè supportando molti sistemi operativi, tra i quali Windows, MacOSX, Linux, HP-UX, Solaris, FreeBSD e OpenBSD, può essere adoperato in quasi tutte le reti coprendo la maggior parte degli host. Il suo motore interno gli permette di monitorare diversi elementi, dai file di log a quelli di registro (Windows), l'integrità dei file (mediante hashing MD5 o SHA1) e rilevamento rootkit. Inoltre è possibile personalizzare il set di regole utilizzato mediante un linguaggio di scripting proprietario. La sua architettura è di tipo Agent/server, cioè gli agenti vengono installati sugli host da controllare e inoltrano i dati al server centrale che effettua l'aggregazion, in questo modo risulta scalabile anche nel caso venga gestito separatamente da altri prodotti. Un'altra caratteristica di questo prodotto è la possibilità di intervenire direttamente in caso

di attacco agendo per esempio tramite chiamate di sistema o modifica delle regole del firewall.

Altri aspetti importanti, ma non attinenti alle specifiche tecniche, che rendono appetibile questo programma sono:

- un team di sviluppo attivo che si pone come obiettivo il rilascio di nuove release ogni 3/4 mesi, aiutato dalla collaborazione della sempre vasta comunità open source;
- un'ampia diffusione in costante crescita, che lo vede impiegato in diversi ambiti, dagli enti governativi agli ambienti aziendali e accademici;
- supporto commerciale fornito dalla compagnia che sta alle spalle del progetto (Third Brigade Inc.), prezioso soprattuto in quegli ambienti dove non è possibile approfondire certi aspetti del programma per mancanza di tempo.

4.1.5 Snort

Snort[17] è un Network Intrusion Detection and Prevention System open source, basato su un linguaggio rule-driven, ovvero che permette di descrivere le regole da controllare. Questa sua caratteristica gli permette di combinare i benefici dei sistemi basati su signature e quelli di protocol e anomaly detection. Nato in origine come HostIDS è andato via via orientandosi sempre più verso la rete fino a raggiungere il suo stato attuale, in cui si auto definisce network-based nonostante sia ancora possibile utilizzarlo a livello host. La licenza GPL, il suo codice multipiattaforma scritto in C e le buone prestazioni, hanno decretato il suo successo, tanto da poterlo definire una specie di standard de facto nel campo dell'intrusion detection/prevention e non solo perchè lo dice il produttore. Altri punti di forza sono:

- adattabilità delle regole mediante l'utilizzo del linguaggio proprietario, con la possibilità opzionale di sottoscrivere un abbonamento per ricevere tutti gli aggiornamenti
- possibilità di utilizzarlo come Network-based o come Host-based in base alle proprie esigenze
- leggerezza derivante dalla struttura a plugin, che permette inoltre una notevole personalizzazione del funzioni disponibili
- possibilità di modificare il codice sorgente nel caso si voglia personalizzare determinati aspetti

Struttura a plugin La struttura a plugin utilizzata, permette una notevole personalizzazione del prodotto, senza gravare eccessivamente sulle risorse del sistema, poichè verranno installati soltanto i moduli necessari. Riassumendo brevemente le caratteristiche dei plugin, è possibile dividerli in tre categorie:

preprocessor plugins Un preprocessore è un plugin che opera per la rilevazione degli attacchi ed agisce prima del detection engine. Vengono invocati una sola volta per ogni pacchetto e servono per rilevare gli attacchi che tentano di mascherare la loro azione (ad esempio, frammentando richieste insidiose) oppure che non sono facilmente definibili con una signature (ad esempio, spp_portscan per i port scanning). Possono essere utilizzati anche per impedire che un pacchetto venga analizzato dal detection engine

detection plugins Sono il cuore del detection engine. Possono essere richiamati diverse volte per ogni pacchetto con differenti parametri. Il loro scopo è rilevare gli attacchi che non tentano di mascherare la loro azione oppure che sono facilmente definibili con una signature e per farlo esaminano i singoli pacchetti, cercando il valore definito in una rule, ad es. i

flag dell'header TCP alla ricerca di determinate sequenze di bit definite nelle rules .

output modules Creano ed inviano i messaggi di allarme (ad un file oppure al syslog) e provvedono ad avviare il logging dei pacchetti che li hanno generati.

4.1.6 MySQL

MySQL[18] è uno dei più famosi e utilizzati RDBMS open source al mondo[19], conta milioni di installazioni ed è uno dei quattro elementi costituenti la nota piattaforma LAMP (Linux Apache MySQL PHP) che si trova alla base di molti servizi web.

Un DBMS (Data Base Management System) ha capacità di gestire grandi moli di dati in un ambiente multiutente. Dal punto di vista fisico, ovvero del sistema operativo, anche un database consiste di file. Tuttavia un DBMS consente elaborazione concorrente, mantiene la consistenza delle informazioni minimizzando la ridondanza, offre supporto per test e prototipazione, permette indipendenza del software dalla organizzazione fisica e logica delle strutture dati. Un DBMS è costruito sopra il sistema operativo ed amplia la gamma delle strutture di accesso messe a disposizione dal file system. In sintesi, in una base di dati la gestione delle informazioni è logicamente centralizzata in una rappresentazione integrata e non ridondante. Dal punto di vista utente un database e' visto come una collezione di dati che modellano una certa porzione della realta' di interesse.

L'utilizzo di questo sistema permette una migliore gestione delle informazioni, garantendo un'ottima accessibilità che si traduce in minori tempi d'attesa per il recupero delle informazioni desiderate. Si capisce quindi l'importanza di potersi affidare ad uno strumento prestante ed affidabile per la memorizzazione dei dati raccolti dagli altri elementi dell'architettura.

MySQL Connettor/J I programmi utilizzati contengono al loro interno moduli per poter interagire correttamente con MySQL o direttamente o mediante chiamate a librerie. Per poter avere libero accesso da programmi scritti in linguaggio Java[35], viene messo a disposizione Connettor/J, cioè un driver JDBC di tipo 4. Questo componente funge da intermediario tra il programma Java e il DBMS e ne esistono diverse versioni a seconda del sistema utilizzato. Lo scopo è quello di rendere il programma indipendente dal DBMS, che potrà quindi essere cambiato senza modificare il codice originale, a patto di mantenere la medesima struttura interna.

Così come esistono diversi connettori messi a disposizione di Java dai diversi DBMS, MySQL mette a disposizione diversi connettori per piattaforme di programmazione differenti.

4.1.7 Pastry

Pastry[25, 26, 28, 27] è un sottostrato scalabile ed efficiente per applicazioni peer-to-peer. I nodi di Pastry formano un overlay network decentralizzato, in grado di auto organizzarsi e tollerante ai guasti, che si appoggia ad Internet. Pastry fornisce efficienti richieste di routing, locazione deterministica degli oggetti e bilanciamento del carico in maniera indipendente dalle applicazioni, ma a quest'ultime fornisce anche un supporto specifico per la replicazione degli oggetti, il caching e la tolleranza ai guasti. Vediamo ora la sua architettura.

Ogni nodo Pastry ha un nodeId unico di 128 bit. Il set di nodeId esistenti è uniformemente distribuito e questo grazie alla loro generezione che può essere casuale o basata su alcuni criteri come l'hash sicuro dell'indirizzo IP o della chiave pubblica, in entrambi i casi il risultato finale è lo stesso. A questo punto, dato un messaggio e una chiave Pastry lo indirizzerà verso il nodo con il nodeId numericamente più vicino alla chiave, scelto tra tutti i nodi attivi in quell'istante. Assumendo che la rete Pastry consista in N nodi, Pastry sarà in grado in media di indirizzare ogni messaggio verso un nodo

qualsiasi in meno di $\log_{2^b} N$ passi, dove b è un parametro di configurazione il cui valore di default è 4. Inoltre anche in caso di guasti simultanei di più nodi, la consegna è comunque garantita finchè $l/2$ o più nodi con nodeId adiacente non falliscono, dove l è un parametro con valore tipico di 16.

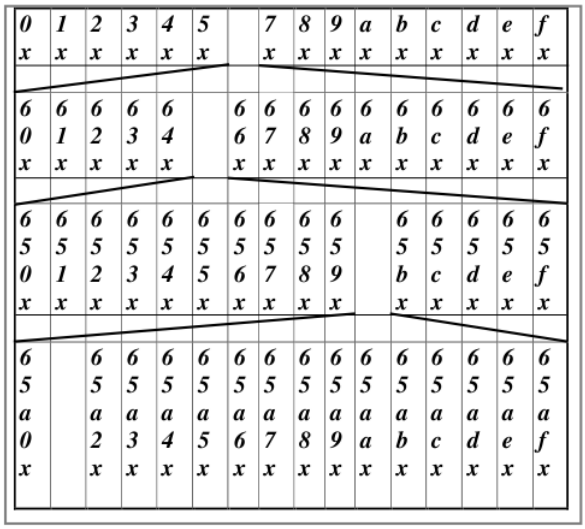

Figura 4.1: Esempio di routing table in Pastry per un nodo con nodeId $65a1x$ e $b = 4$. Le cifre sono in base 16 e x rappresenta il suffisso. Non sono mostrati gli indirizzi IP associati ad ogni entry.

La tabella di indirizzamento presente in ogni nodo avrà solo $(2^b - 1) * [\log_{2^b} N] + l$ entries e ognuna di esse mapperà un nodeId all'indirizzo IP associato al nodo. Inoltre dopo il guasto di un nodo o l'arrivo di uno nuovo, i dati presenti in tutte le routing table coinvolte possono essere aggiornati scambiando $O(\log_{2^b} N)$ messaggi. Considerando il routing, i nodeId e le chiavi sono pensati come una sequenza di caratteri con base 2^b. La routing table di un nodo è organizzata in $[\log_{2^b} N]$ righe con $2^b - 1$ entries ciascuna. Ogni entry alla riga n della routing table si riferirà ad un nodo il cui nodeId combacierà con quello del nodo presente per i primi n caratteri, ma che differirà nell' $n + 1$-esimo carattere presentando uno degli altri $2^b - 1$ possibili valori. La distribuzione uniforme dei nodeId garantisce una pari popolazione dello

spazio, nonostante questo solo $[\log_{2^b} N]$ livelli saranno popolati all'interno della routing table. Ogni entry si riferirà ad uno dei potenziali nodi il cui nodeId contiene l'appropriato prefisso, tra questi verrà scelto il più vicino a quello presente (per esempio basandosi sul *RTT-round trip time*).

In aggiunta alla routing table ogni nodo manterrà una lista aggiornata con gli indirizzi IP dei nodi presenti nel proprio *leaf set*, cioè il set dei nodi numericamente più vicini, in particolare $l/2$ con nodeId più alto e $l/2$ con nodeId più basso.

Il meccanismo di routing è abbastanza semplice, ogni nodo inoltra un messaggio ricevuto ad un altro nodo il cui nodeId combaci con la chiave del messaggio per un maggior numero di caratteri, o in alternativa ad un nodo che condivida lo stesso numero di caratteri ma sia numericamente più vicino. La ricerca avviene ovviamente all'interno della routing table e del proprio leaf set e nel caso non trovi alcun nodo che risponda alle specifiche significa che il proprio nodeId è il più vicino alla chiave, quindi il messaggio andrà memorizzato e non inoltrato.

Aggiunta e perdita di nodi

Uno dei punti chiave nella realizzazione di Pastry è stato implementare un sistema dinamico ed efficiente per tenere traccia dello stato dei nodi, nella routing table e nel leaf set, in presenza di guasti dei nodi, recovery e nuove entrate.

Nel caso di un nuovo arrivo, si ipotizzi che il nodeId scelto sia X e che per l'inizializzazione venga contattato il nodo A, che già appartiene alla rete. A inoltrerà uno speciale messaggio utilizzando X come chiave, questo attraverserà la rete fino ad arrivare al nodo Z il cui nodeId risulta quello numericamente più vicino ad X[2]. A questo punto X otterrà il leaf set da Z e la i-esima riga della routing table dall'i-esimo nodo incontrato lungo il percorso

[2]Nello sfortunato caso in cui il nodeId X sia già presente all'interno dell'anello, verrà chiesto al nodo entrante di scegliere un nuovo valore e di ricominciare da capo l'operazione d'ingresso.

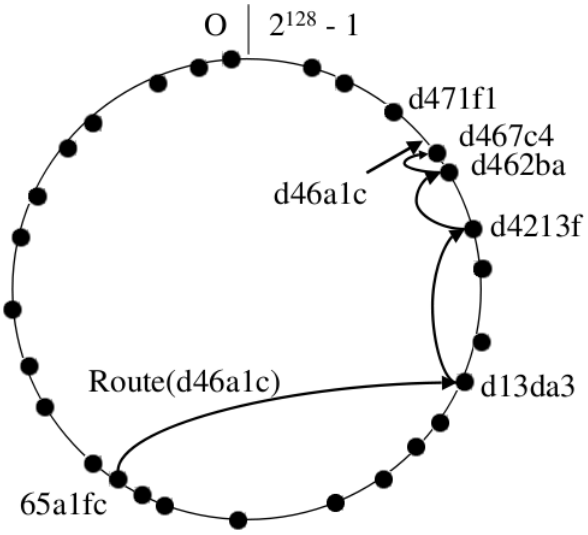

Figura 4.2: Indirizzamento di un messaggio con chiave $d46a1c$ a partire dal nodo $65a1fc$. I punti sul cerchio rappresentano i nodi attivi all'interno del dominio circolare di Pastry

da A a Z, in questo modo X potrà inizializzare il proprio stato correttamente e notificare l'evento ai nodi interessati dal suo arrivo.

Per gestire il fallimento di un nodo invece, i nodi vicini all'interno dello spazio del nodeId (si consideri che i vicinati sono parzialmente sovraposti l'un l'altro e che ogni nodo rientra nel leaf set di più nodi) si scambiano periodicamente messaggi di *keep-alive*, ovvero che segnalano il proprio stato di attività. Se un nodo non risponde per un periodo T si presume che esso sia fallito. Tutti i nodi del relativo leaf set verranno notificati e di conseguenza aggiorneranno i propri leaf set. Dal momento che i leaf set sono parzialmente sovrapposti questo aggiornamento risulta abbastanza leggero. Nel caso un nodo dovesse riprendere la propria attività, contatterà i nodi dell'ultimo leaf set conosciuto ottenenendo i loro, dopo di chè procederà all'aggiornamento del proprio e informerà i propri vicini della sua presenza. Si fa presente che la procedura di riparazione delle entries delle routing table che fanno riferimento a nodi persi è abbastanza pigra.

Implementazione

Al momento esistono due implementazioni di Pastry reperibili online: FreePastry[20] della Rice University e SimPastry/VisPastry[21] della Microsoft Research. Le release iniziali supportano API sementicamente simili e attualmente i due team di sviluppo stanno collaborando per rilasciare in futuro delle API comuni indipendenti dal linguaggio. L'implementazione da me scelta è FreePatry. Studiato per essere utilizzato all'interno di Internet, è scritto in in linguaggio Java 5 e questo gli permette di essere adoperato su più piattaforme differenti. La limitazione principale è che nella versione attuale le funzioni rivolte alla sicurezza sono minime[3], anche se il team ha garantito di porre maggiore attenzione a riguardo nelle future release.

4.1.7.1 PAST

PAST[29, 30] è un applicazione di storage su larga scala basata su un'architettura peer-to-peer, che fornisce scalabilità, disponibilità, sicurezza e condivisione collaborativa delle risorse. I file in PAST sono immutabili e possono essere condivisi a discrezione del proprietario. La sua costruzione è basata sul sottostrato di Pastry descritto in precedenza.

Architettura e funzionamento PAST è composto da nodi connessi a Internet che formano un overlay network in grado di autogestirsi, ogni nodo è in grado di inizializzare e indirizzare le richieste dei client per l'inserimento o il recupero di files e può inoltre contribuire condividendo uno spazio di storage col sistema. I file inseriti vengono replicati su una moltitudine di nodi in modo da garantire persistenza e disponibilità, inoltre con buona probabilità, il set

[3]Nelle pubblicazioni contenenti le basi teoriche di Pastry, PAST e Scribe, sono presenti sezioni specifiche riguardanti la sicurezza dei protocolli di comunicazione. Freepastry al momento attuale non implementa tutte le funzioni previste da tali pubblicazioni. Un esempio chiarificatore può essere riferito a PAST. A livello teorico sono previste interazioni basate sull'utilizzo di certificati sicuri, mentre a livello pratico questa possibilità non è ancora prevista.

di nodi sui quali il file viene replicato è eterogeneo in termini di locazione geografica, proprietà, amministrazione, connettività ad internet e legislazione e questo non può che rafforzare le due caratteristiche citate. In aggiunta le copie più popolari possono essere inserite nella cache di un qualsiasi nodo al fine di bilanciare il carico di richieste. Oltre agli ottimi valori di persistenza e disponibilità, un utility di storage globale facilita la condivisione di spazio di memorizzazione e di banda, così da permettere a gruppi di nodi di pubblicare contenuti che altrimenti eccederebbero le capacità individuali.

Anche se viene fornito un servizio di storage persistente, la semantica di accesso risulta differente rispetto ad un convenzionale file system. I file memorizzati in PAST sono associati ad un quasi-unico fileId che viene generato nel momento dell'inserimento, inoltre i file una volta inseriti risultano immutabili di conseguenza un file non può essere inserito più volte con lo stesso fileId. La condivisione può avvenire a discrezione del proprietario del file, che potrà scegliere a chi e come distribuire il fileId necessario per recuperare il file, che altrimenti risulterebbe difficilmente reperibile. Altra differenza importante è la mancanza di un'operazione di delete, al suo posto viene fornità la possibilità al proprietario di reclamare lo spazio occupato dal file in questione, ma in questo caso non si garantisce che il file non sia più disponibile. Il motivo alla base di questa scelta è principalmente la volontà di voler snellire il protocollo utilizzato per lo storage, una funzione di delete richiederebbe infatti complesse operazioni per l'accertamento dell'avvenuta cancellazione di ogni replica. Ecco una descrizione più dettagliata delle operazioni di insert e di lookup (rispettivamente inserimento e ricerca).

Insert All'atto dell'inserimento il client dovrà passare un contenuto con la relativa chiave ovvero il fileId, in caso contrario questo verrà calcolato sul momento. Il fileId ha una dimensione di 160 bit di cui solo i primi 128 vengono utilizzati per l'indirizzamento[4]. Può essere generato

[4]Questo avviene poichè il routing avviene in base ai nodeId dei nodi di pastry che sono a 128 bit

utilizzando una funzione hash sicura sul suo contenuto, oppure è possibile impostarlo manualmente. Nel primo caso è possibile applicare l'hash solo ad una parte del contenuto, cioè quella che si vuole utilizzare come chiave (per esempio il nome del file, la data e la dimensione), nel secondo caso invece occorre fare molta attenzione perchè se non si tiene conto di tutti i possibili casi si rischia di avere un notevole incremento di collisioni all'atto dell'inserimento o una distribuzione non uniforme. A questo punto il messaggio dotato di una chiave appropriata verrà inserito nella rete PAST con destinazione il nodo con nodeId più vicino alla chiave, verranno inoltre memorizzate delle repliche sui nodi vicini in base al fattore di replicazione impostato per quella determinata istanza di PAST. Per ogni inserimento effettuato con successo il nodo risonderà al client con un messaggio di acknowledgment, così che questo possa verificare se l'operazione richiesta sia stata portata a termine con successo o nel caso contrario possa essere informato sul tipo di impedimento che l'ha fatta fallire (per esempio spazio insufficiente o messaggio perso). - Nel caso vengano implementate particolari procedure di sicurezza all'interno di PAST, quest'operazione aumenta di complessità e richiede l'utilizzo di certificati. Non viene descritto questo caso poichè la versione attuale di FreePastry non lo supporta.

Lookup In caso di lookup il client procede mandando un messaggio speciale contenente il fileId richiesto. Quando il messaggio verrà ricevuto da un nodo contente quel determinato fileId, non verrà più inoltrato e il nodo procederà a inviare il file richiesto. L'efficiente schema di routing utilizzato da Pastry garantisce che le richieste di recupero di un file vengano indirizzate verso il nodo più vicino a quello del client richiedente scelto tra i diversi nodi contenenti le repliche. Il risultato è che, nonostante il numero di nodi attraversati risulti comunque logaritmico rispetto al totale dei nodi appartenenti a PAST, l'invio del file avverrà col nodo più vicino, riducendo così i tempi di trasmissione ed il consumo di banda.

Storage management Nonostante non sia presente un'unità di controllo centralizzata e la differente dimensione dei file memorizzati possa essere notevole, uno schema di gestione dello storage presente in PAST assicura il massimo sfruttamento della capacità disponibile fino a sfiorare il 100%, . Per raggiungere tale obiettivo in un sistema distribuito su nodi senza garanzie è necessario un meccanismo che bilanci la domanda e l'offerta di storage. Per questo motivo PAST implementa un sistema di gestione delle quote in cui è richiesto ad ogni partecipante di mettere a disposizione una quota minima prefissata, con la possibilità di limitare lo spazio utilizzabile proporzionalmente a quello messo a disposizione.

Guardando però ai due compiti della gestione dello storage, cioè la maggior utilizzazione possibile e la memorizzazione delle repliche sui nodi con nodeId vicino al fileId, si può notare come essi contrastino. Per risolvere il problema son state introdotte le tecniche di *file diversion* e *replica diversion*. La prima è molto semplice, entra in gioco quando il nodo responsabile per la memorizzazione non ha spazio libero sufficiente e si limita a chiedere al client di ricalcolare il fileId in modo da spostare la sua localizzazione su un'altra area dello spazio dei nodeId. La seconda invece svolge la propria attività in automatico e risulta trasparente al client. Quando un nodo non è in grado di memorizzare la replica che gli spetta, questa viene memorizzata da uno dei nodi presenti all'interno del suo leaf set che abbia sufficiente spazio. Per tenere traccia di quest'operazione, il nodo responsabile memorizzerà un seplice riferimento alla copia presente nel nodo "offertosi volontario". Occorre fare attenzione a non abusare di questa tecnica, poichè a lungo andare le prestazioni della rete e la disponibilità dei file potrebbero risentirne.

4.1.7.2 Scribe

Scribe[31, 32] è un'infrastruttura multicast costruita a livello applicativo sopra a Pastry. Qualsiasi nodo di Scribe può creare un gruppo, può entrare in uno esistente e può mandare messaggi multicast agli altri nodi appartenenti.

Scribe fornisce una consegna di tipo best-effort dei messaggi multicast e non garantisce l'ordine d'arrivo, comunque garanzia di consegna e mantenimendo dell'ordine possono essete implementati da un livello superiore a Pastry. I nodi possono creare e spedire messaggi e possono iscriversi a diversi gruppi, quest'ultimi a loro volta possono avere più sorgenti al loro interno e molti membri. In particolare Scribe può supportare contemporaneamente un elevato numero di gruppi di grosse dimensioni e un alto tasso di turnover dei membri, cioè dei nodi. Scribe usa Pastry per gestire la creazione dei gruppi, le iscrizioni e la creazione di alberi di multicast (*multicast tree*) per ogni singolo gruppo, usati per inoltrare i messaggi multicast tra i membri. Così come Pastry è completamente decentralizzato lo è anche Scribe, tutte le decisioni avvengono in base alle informazioni locali e ogni nodo ha identiche capacità, perciò può fungere da sorgente, da root o da nodo dei *multicast tree*, da semplice membro e gestire ogni ruolo relativamente al gruppo di appartenenza (per esempio un nodo può essere la radice di un gruppo senza esserne membro, ed essere nodo e sorgente in un altro).

Implementazione Un sistema Scribe consiste in una rete di nodi Pastry dove ognuno di essi esegue Scribe sottoforma di applicazione software. Quest'ultima metterà a disposizione due metodi: *forward* e *deliver*, che vengono invocati ogni volta che il nodo riceve un messaggio. Il primo viene invocato quando un messaggio viene indirizzato attraverso il nodo, il secondo invece viene chiamato quando un messaggio arriva al nodo con nodeId numericamente più vicino alla chiave del messaggio, o quando un messaggio viene indirizzato direttamente a quel messaggio mediante l'operazione *send*. I tipi di messaggio sono quattro.

CREATE Serve per creare un gruppo. Viene inviato dal nodo che vuole crearlo indirizzandolo al nodo con nodeId numericamente più vicino al groupId, cioè alla chiave che rappresenta il gruppo. Questo, una

volta ricevuto il messaggio, dventerà il rendez-vous point, ovvero la root dell'albero di multicast relativo al gruppo.

JOIN Viene utilizzato dai nodi per partecipare ai gruppi. Il nodo deve conoscere la chiave del gruppo, dopo di chè invia un messaggio di JOIN indirizzato alla root. Ogni nodo che verrà attraversato da questo messaggio verificherà se il gruppo indicato è tra quelli di cui è membro, in caso affermativo il messaggio non verrà inoltrato ulteriormente e il nodo ricevente inserirà il richiedente tra i suoi *child*. Questo schema permette una notevole scalabilità, poichè nonostante esista una sola root per gruppo non riceverà tutto il traffico relativo alla gestione del gruppo.

LEAVE Questo messaggio è complementare a quello di JOIN e viene inviato quando un nodo desidera lasciare un gruppo di cui è membro. Il messaggio verrà indirizzato al nodo superiore a livello di *multicast tree* e verrà poi inoltrato fino a quando tutti i nodi aventi il richiedente tra i propri *child* non l'avranno rimosso. Il rischio di loop infiniti non esiste poichè ogni inoltro viene effettuato verso un livello più alto, cioè con un nodo che abbia il prefisso più lungo in comune con la chiave del gruppo, nel caso peggiore il messaggio arriverà alla root.

MULTICAST Serve per ottenere l'indirizzo IP del *rendez-vous point*, in modo da poter poi a lui inviare tutti i messaggi multicast. Questi verranno poi inoltrati dalla root a tutti i membri del gruppo. La scelta di far riferimento alla root per tutti i messaggi da inviare è legata all'intenzione di permettere al nodo rendez-vous di effettuare un controllo sugli accessi al canale multicast.

4.2 Realizzazione dell'architettura distribuita p2p

L'idea alla base di questo progetto è quella di sfruttare tutti i vantaggi derivanti dalla distribuzione dei nodi su una rete peer-to-peer e di renderla facilmente implementabile, ovvero compatibile con la maggior parte dei sistemi in circolazione. In questo caso la compatibilità è riferita a 360° a tutti i componenti di sistema, questo per permettere ad un amministratore di partecipare all'architettura distribuita mantenendo la propria struttura interna il più possibile inalterata. Per questo motivo son stati scelti componenti già diffusi, in più l'utilizzo di Prelude consente la realizzazione di un ulteriore livello di astrazione, grazie alla sua ampia compatibilità con diversi prodotti.

I vantaggi ottenuti appoggiandosi ad una rete peer-to-peer, sono già stati accennati in precedenza, ma ora possiamo andare a descriverli più nel dettaglio con riferimento agli strumenti scelti e alla loro integrazione. Per semplicità si farà riferimento ai singoli nodi che partecipano alla rete senza citare la struttura delle sotto-reti alle loro spalle, che in parte è già stata descritta nel capitolo precedente.

Ogni nodo parteciperà alla rete collaborativa grazie ad un'applicazione software realizzata utilizzando le API messe a disposizione da FreePastry. Questo programma avrà principalmente quattro funzioni:

- instaurazione e gestione della rete
- immisione dei dati
- elaborazione dei dati
- comunicazione dei risultati

Instaurazione e gestione della rete FreePastry mette a disposizione classi e metodi che permettono con pochi passaggi di creare dei nodi Pastry e di connetterli tra loro. Tutte le operazioni per la gestione della rete Pastry possono essere eseguite autonomamente, senza che il programmatore

se ne debba preoccupare. Questo permette di concentrarsi sul cuore dell'applicazione fin dall'inizio e di posticipare eventuali personailizzazioni che potranno prendere la forma di modifiche al codice sorgente, *override* di metodi già esistenti o, come avviene spesso in informatica, realizzazione di un livello soprastante che meterà a disposizione le funzioni desiderate. Nel nostro caso per evitare spiacevoli inconvenienti con gli aggiornamenti futuri delle API, si è scelto di lasciare intatto il codice sorgente e si è fatto ricorso solo alle altre due possibilità. La prima, ovvero l'override, è stata sfruttata per modificare l'atteggiamento di PAST all'atto dell'inserimento di un file. Se questo infatti è già presente, o più precisamente lo è la sua chiave, all'interno della DHT non verrà segnalato errore (atteggiamento di default di PAST), poichè per il nostro utilizzo sarà un evento frequente e del tutto normale, basti pensare all'inserimento dello stesso malware raccolto da più nodi.

Per quanto riguarda la realizzazione di un livello soprastante, ci si riferisce alla creazione di una piccola applicazione che permette ai nodi di scambiarsi messaggi unicast, che potranno essere utilizzati per diversi scopi.

Immissione dei dati Per la condivisione dei dati raccolti si utilizza l'implementazione di PAST presente all'interno di FreePastry, che mette a disposizione classi e metodi sufficienti solo in parte per il nostro scopo. Per l'inserimento ed il recupero dei dati non ci sono problemi, ma non sono previsti metodi per far svolgere al nodo ricevente determinate operazioni in base al contenuto ricevuto, il nodo infatti è inconsapevole dei contenuti che vengono memorizzati presso di esso. Questo non è un difetto di PAST, ma una scelta voluta e presa sulla base di un tipico ambito di utilizzo, nel quale i nodi partecipanti non devono elaborare i dati ricevuti, ma solo archiviarli.

Per risolvere il problema è stata realizzata una semplice applicazione per la comunicazione unicast che permette al nodo mittente di segnalare al ricevente il contenuto appena inviato che andrà elaborato.

Elaborazione dei dati Ogni nodo dovrà implementare funzioni proprie per l'elaborazione dei dati ricevuti o, in alternativa, appoggiarsi a serivizi di analisi esterni. Al momento attuale all'interno del nostro prototipo l'analisi dei dati ricevuti viene solo simulata

Comunicazione dei risultati Questo è l'ultimo passagio, comunicare le informazioni ricevute dall'analisi dei dati. Questa comunicazione andrà fatta su larga scale, poichè tutti i nodi saranno potenzialmente interessati ai risultati, per questo motivo è stata realizzata un semplice applicazione basata su Scribe che permette di mandare messaggi multicast o, nel caso tutti i nodi abbiano sottoscritto un determinato argomento, broadcast.

Passando ad una descrizione più approfondita del progetto, possiamo dividerlo in due parti: quella relativa all'architettura di una sottorete, o rete interna, e quella relativa al programma software realizzato per far collaborare i nodi.

4.2.1 Architettura

Distinguendo i componenti in base all'host su cui erano installati, ecco la loro disposizione.

Un computer con sistema operativo Windows® XP Professional, dotato di firewall e connesso mediante servizio di natting fornito dal router, su cui è stato installato un agente OSSEC.

Un computer con sistema operativo Ubuntu 8.04, con porte aperte sul firewall e reindirizzamento diretto da parte del router delle porte più importanti (ovvero le *well know ports*, più quelle predefinite di diversi servizi). Su questo host son stati installati:

- un server OSSEC, che oltre ad effettuare il compito di sensore, raccoglieva i dati dall'agente installato sull'altro host;

- Nepenthes, che nonostante emulasse più servizi, ha mostrato come la stragrande maggioranza del malware fosse destinata alla porta 135 (servizio netbios di Windows®);
- Snort, che ha rilevato alcuni tentativi di exploit basati su vecchie vulnerabilità;
- Prelude, che ha svolto il compito di manager per i sensori appena citati concentrando tutte le informazioni raccolte ed inserendole all'interno del database creato appositamente;
- Prewikka, che ha permesso di monitorare in tempo reale gli alert raccolti dal manager attraverso la sua interfaccia grafica;
- MySQL, su cui sono stati attivati più utenti relativi ai componenti abilitati ad inserire i dati, in questo caso prelude, prewikka e snort, i quali hanno creato i propri schemi e immesso i dati raccolti.

Tutti e tre i tipi di sensori, fanno parte dell'elenco presente sul sito di Prelude riferito ai sensori compatibili. Per il loro utilizzo col manager è bastato abilitare le opzioni o all'atto dell'installazione o modificando i file di configurazione, dopo di ché è stato necessario effettuare la registrazione. In questo caso tutti i componenti si trovano sul medesimo computer di conseguenza il traffico non può essere intercettato, ma risulta comunque necessaria la registrazione ed inoltre rispecchia fedelmente le operazioni tipiche di casi in cui i sensori siano su host differenti.

Dopo aver installato tutti gli strumenti si è proceduto con la regolazione delle impostazioni del firewall, per garantire ai sensori l'arrivo del traffico (in)desiderato. Infine per verificare il corretto funzionamento è stato avviato prewikka mediante il proprio server web, che ha permesso di visualizzare gli alert raccolti dal manager utilizzando un comune browser web.

Per quanto riguarda l'installazione del DBMS, che deve avvenire per prima, si sottolinea come la scelta del suo utilizzo, al posto della memorizzazione

su semplici file, è stata fatta sia per i principali benefici che si ottengono sull'accesso ai dati (es. velocità, filtraggio, aggregazione), sia per la possibilità di accedere ad essi da remoto. Quest'ultima operazione permette di installare il software per la realizzazione dell'architettura peer-to-peer su un host differente da quello su cui è presente il DBMS, o su cui è presente il Prelude-manager. L'amministratore di rete potrà quindi installare il programma su un host indipendente, mantenendo inalterate le restanti macchine.

4.2.2 Implementazione

Vista l'esigenza di compatibilità, è stato scelto come linguaggio di programmazione Java 6. La principale critica a questo linguaggio è relativa alla scarsa efficienza nell'utilizzo delle risorse a causa della sua esecuzione all'interno di macchine virtuali, le JVM (Java Virtual Machine). Allo stesso tempo questa sua particolarità permette di eseguire il bytecode generato (una specie di codice binario) su qualsiasi piattaforma implementi una JVM adeguata. Questo significa che con i dovuti accorgimenti[5], sarà possibile adoperare il programma tanto in ambiente Unix quanto in ambiente Windows. Ovviamente la scelta del linguaggio è stata fatta insieme a quella relativa alle librerie da utilizzare per la realizzazione di una rete peer-to-peer, che nel nostro caso è risultata FreePastry.

Il progetto è costituito da tre package più un file di configurazione XML.

malwarepastcontent

Questo package contiene le classi utilizzate per il trasferimento dei dati tra i nodi. Al suo interno è presente una classe generica con altre tre che ne ereditano i metodi comuni e l'adattano a casi più specifici. Questa struttura

[5]Una delle differenze principali si riscontra nell'utilizzo dei percorsi dei file. Come separatore i sistemi Unix utilizzano "/" (*slash*), mentre Windows utilizza "\" (*back-slash*); anche la radice è differente così come i caratteri a disposizione. Per questo motivo ogni utilizzo di percorsi all'interno del programma fa riferimento ad un file di configurazione esterno che potrà essere modificato dall'utente mediante un semplice editor di testo.

consente una discreta flessibilità poiché è possibile programmare con riferimento alla classe madre riservando casi specifici in parti di codice attivate a tempo di esecuzione solo se necessario. Inoltre si ha a disposizione uno schema solido per eventuali classi aggiuntive che volessero specificare casi non ancora trattati. Infine è presente una classe che svolge il compito di deserializzare i contenuti ricevuti. Verranno ora analizzate le singole classi.

GenericMalwareContent Come si intuisce dal nome, rappresenta la classe generica che contiene i campi e i metodi fondamentali. Tra i primi troviamo il campo *tipo*, che sarà differente per ogni classe e che viene utilizzato dal deserializzatore, e l'*id*, che rappresenta la chiave utilizzata per l'inserimento all'interno di PAST. Per offrire la possibilità di inserire messaggi con contenuto non strutturato, è stato aggiunto anche il campo *stringContent*, che non è altro che una stringa che potrà essere riempita arbitrariamente. I metodi presenti servono invece per leggere il contenuto dei campi appena descritti (che una volta inizializzati dal costruttore non potranno essere modificati), per serializzare il contenuto in modo più efficiente rispetto alla serializzazione standard di Java e per modificare il comportamento di PAST nel caso si tenti di inserire dati già presenti nella DHT.

Malware Questa classe viene utilizzata per inserire i dati relativi al malware raccolto da Nepenthes. I campi aggiuntivi sono:

- il codice binario, costituito da un array di byte
- il nome originale del file scaricato
- la dimensione, ridondante rispetto alla lunghezza dell'array
- l'hash del codice binario, che nel nostro caso viene usato anche come prefisso per l'id
- l'indirizzo IP dell'attaccante

- l'indirizzo IP da cui è stato scaricato il malware, che spesso coincide con quello dell'attaccante

SegnalazioniPrelude Viene usata per identificare delle generiche segnalazioni provenienti da Prelude, in particolare è stata utilizzata senza ulteriori dettagli per le connessioni ricevute dall'honeypot. Al suo interno sono presenti i campi:

- IP dell'attaccante
- IP della rete attaccata
- porta di destinazione della connessione
- protocollo utilizzato, questo campo può assumere tre valori: TCP, UDP e UNKNOW

SegnalazioniSnort Questa classe estende SegnalazioniPrelude con alcuni campi relativi ai dati raccolti da snort[6], più precisamente:

- sigId, indica l'Id della signature che ha generato l'alert
- sigClass, indica la classe della signature

GenericMalwareContentDeserializer Questa classe viene utilizzata per richiamare le funzioni di deserializzazione delle classi appena descritte, che non sono altro che costruttori appositamente modificati. Ad ogni nuova applicazione PAST creata da un nodo, viene passata un'istanza di questa classe che andrà a sostituire il deserializzatore di default. Il deserializzatore andrà a leggere il campo *tipo* per sapere che metodo invocare, il valore 0 è riservato alla serializzazione standard di java.

[6]I dati raccolti da snort potranno essere prelevati dal database di prelude, specificando snort come sorgente dei dati, o dal database creato da snort stesso. Ovviamente la scelta dovrà essere coerente con le impostazioni specificate all'atto dell'installazione. Il vantaggio consiste nella riduzione di complessità della query da eseguire sul database, lo svantaggio nella ridondanza delle informazioni raccolte, con conseguente spreco di spazio di storage.

commonpast

Questo package contiene tutte le classi che costituiscono la struttura base dell'applicazione. Oltre al Main, troviamo due classi di tipo *continuation*[7], un'interfaccia contenente tutti i metodi che l'applicazione dovrà necessariamente implementare e due sotto-applicazioni utilizzate per le comunicazioni, rispettivamente unicast e multicast/broadcast, assieme ad una classe ciascuna per i messaggi da inviare. Andiamo ora a descrivere più nel dettaglio le singole classi.

Main Si tratta della classe utilizzata per lanciare l'esecuzione, in particolare contiene al suo interno le funzionalità necessarie per convertire opportunamente i parametri d'ingresso che andranno utilizzati per l'utilizzo del programma. Più precisamente di tratta del numero di porta da associare all'applicazione e indirizzo (IP + numero di porta) di un nodo preesistente, per mezzo del quale effettuare il join della rete a cui esso appartiene. Nel caso quest'ultimo non venga trovato, ne verrà avviata una nuova.

InsertSigContinuation Questa classe viene istanziata all'atto dell'inserimento di un contenuto all'interno della DHT e avrà il compito di gestire il risultato ritornato. I metodi presenti sono i due previsti dalla classe Continuation di FreePastry, ovvero receiveResult(Object result) e receiveException(Exception ex). Il primo verrà chiamato passando come oggetto il risultato dell'operazione, che in questo caso sarà un vettore di variabili booleane, una per ogni replica inserita (inclusa la copia primaria). Nel caso il vettore contenga esclusivamente valori *false* verrà segnalato l'errore all'utente e si procederà con un nuovo tentativo. Il

[7]Una continuation è una specie di *callback*, o con riferimento a Java, ad un *Listener*, con la differenza che tipicamente viene usato una sola volta. In ambito informatico le continuation trovano diversi utilizzi, in FreePastry servono per gestire i ritardi della rete, più precisamente per rendere i metodi legati ad essi non bloccanti. Questo significa che la programmazione basata su tali metodi risulta asincrona.

secondo metodo invece viene chiamato in caso d'errore e come nel caso descritto segnalerà l'eccezione sollevata e ritenterà l'inserimento.

SigNodeContiuation Questa classe viene istanziata sempre durante l'operazione di inserimento, ma in una fase successiva rispetto alla classe precedente. Il suo compito è quello di verificare quali nodi abbiano memorizzato le diverse repliche e sceglierne uno a cui comunicare di procedere con l'elaborazione del file ricevuto. Perché vi sia uniformità il criterio utilizzato per la scelta dev'essere sempre applicabile e non ambiguo, nel nostro caso viene scelto il nodeId più alto. La comunicazione potrà avvenire mediante l'utilizzo dell'applicazione PastMessenger descritta in seguito, in particolare è possibile indicare la chiave del contenuto appena inserito ed eventualmente il tipo.

CommonPastApplicationMethod Questa interfaccia elenca tutti i metodi che la classe, o le classi, del package *test* dovranno implementare. Il suo scopo è quello di creare un livello di astrazione tra le classi alla base dell'architettura appartenenti a commonpast e quelle che andranno ad implementare l'applicazione soprastante, in questo modo sarà possibile modificare quest'ultima senza preoccuparsi di aggiornare le classi esterne. Il problema nasce dalla chiamata di metodi propri dell'applicazione all'interno delle classi base. Ad esempio la classe InsertSigContinuation descritta in precedenza, nel momento in cui verifica che l'inserimento è avvenuto con successo, dovrà invocare un metodo dell'applicazione soprastante che effettui la segnalazione, che a sua volta istanzierà una classe SigNodeContinuation. L'interfaccia creata viene utilizzata come riferimento per conoscere i metodi messi a disposizione, allo stesso tempo fornirà delle linee guida per la realizzazione dell'applicazione soprastante. I metodi più significativi sono tre.

insertLookSig(GenericMalwareContent mal) che serve per inserire un contenuto all'interno della DHT implementata da PAST. Come

si nota, il contenuto fa riferimento alla classe generica descritta nel package precedente.

sigNode(Id key, int copy) che viene invocato dopo l'inserimento effettuato con successo e che prende in ingresso la chiave del contenuto inserito e il numero di copie (quella primaria più le repliche).

lookUpId(Id key) utilizzato per recuperare un contenuto inserito in precedenza, basandosi sulla relativa chiave.

PastMessenger Si tratta della classe che fornisce uno strumento di comunicazione unicast. Il funzionamento e la sua implementazione sono abbastanza semplici, in particolare si evidenziano tre metodi: routeMyMsg, che inserisce i messaggi nella rete in base all'id indicato come destinatario, routeMyMsgDirect, che invia il messaggio direttamente ad un determinato nodo (utilizzabile solo se si conosce il nodo e non solo il nodeId), infine il metodo deliver, che contiene tutte le operazioni da svolgere quando si riceve un messaggio.

MyMsg Rappresenta la classe su cui sono basati i messaggi di PastMessenger. I campi sono ridotti a: id mittente, id destinatario, stringa del contenuto e il tipo. Quest'ultimo è un intero e ad ogni valore può essere associato un particolare comportamento, mentre il contenuto sotto forma di stringa garantisce una certa flessibilità.

MulticastClient Come dice il nome stesso, si tratta dell'applicazione che consente l'invio e la ricezione di messaggi multicast ed è basata su Scribe. Ogni nodo potrà sottoscrivere diversi argomenti in modo da ricevere tutti i messaggi ad essi appartenenti e nel caso tutti i nodi sottoscrivessero un determinato *topic*, la comunicazione risulterebbe broadcast. Scribe prevede anche la comunicazione Anycast, ma in questo progetto non viene utilizzata.

MyScribeContent Rappresenta la classe su cui sono basati i messaggi di MulticastClient. I campi sono ridotti a: mittente, tipo e contenuto. Quest'ultimi sono identici per tipo e utilità a quelli di MyMsg. Il tipo infatti può essere utilizzato come "selettore" delle operazioni da svolgere all'atto del ricevimento e permette anche un ulteriore filtraggio oltre alla sottoscrizione di topic differenti. Il contenuto sottoforma di stringa garantisce invece una certa flessibilità.

test

In questo package troviamo la classe principale di questo progetto, quella che implementa i metodi dell'interfaccia descritta in precedenza e che all'interno del proprio costruttore contiene le procedure per la creazione dei nodi su cui istanziare PAST e le altre applicazioni di supporto (PastMessenger e MulticastClient). La scelta di dedicare un intero package a questa sezione è stata fatta per garantire un maggiore ordine anche nel caso futuri aggiornamenti prevedano la creazione di più classi.

Al momento attuale la classe presente è una sola e si chiama ApplicationTestMultinode, il cui nome deriva dall'integrazione di tutte le caratteristiche necessarie per generare, o simulare, più nodi. Oltre al costruttore, nel quale i nodi vengono inizializzati e connessi alla rete, troviamo un metodo per la lettura dei parametri principali dal file di configurazione, i metodi per la lettura dei dati raccolti all'interno del database e ovviamente tutti i metodi descritti nell'interfaccia *CommonPastApplicationMethod*, supportati da altre funzioni secondarie.

Una breve descrizione dei passaggi svolti può essere riassunta come segue:

- l'applicazione viene lanciata dal Main col passaggio di alcuni parametri
- il costruttore legge i parametri ricevuti e legge i restanti dal file di configurazione
- vengono creati i nodi (o il nodo) e li si connette alla rete

- se la connessione avviene con successo vengono avviate le applicazioni PAST (utilizzando le classi standard contenute da FreePastry), PastMessenger e MulticastClient.

Con queste operazioni il costruttore termina dando inizio alle operazioni di lettura ed inserimento dei dati che avvengono ad intervalli prefissati (polling). Più precisamente verrà invocata la lettura del database per controllare la presenza di nuove segnalazioni, in caso affermativo i dati vengono letti e passati ai metodi che ne gestiranno il corretto inserimento all'interno della rete.

Rimangono da implementare le funzioni di analisi ed elaborazione dei dati ricevuti dagli altri nodi, che al momento vengono solo simulate, mentre sono già presenti quelle per effettuare la comunicazione broadcast/multicast dei risultati ottenuti.

XML

Per mantenere il programma il più adattabile possibile, molte funzioni fanno riferimento a parametri che vengono letti da un file di configurazione. Quest'ultimo è stato realizzato basandosi sulla versione 1.0 dello standard XML[36], questa scelta, assieme a quella di utilizzare nomi significativi, permette all'utente di modificare tale documento in modo da personalizzarlo in base alle proprie esigenze. I parametri principali utilizzati dal programma sono:

- nome dell'istanza dell'applicazione PAST[8]
- numero di repliche
- directory dedicata allo storage dei contenuti ricevuti

[8]Per poter collaborare tutti i nodi devono utilizzare lo stesso nome identificativo dell'istanza di PAST, allo stesso tempo nomi differenti permettono a più nodi appartenenti a reti PAST differenti di convivere sulla stessa macchina ed anche sulla stessa porta (questa possibilità non viene attualmente utilizzata all'interno dell'applicazione)

- spazio messo a disposizione per lo storage (in MB)
- spazio messo a disposizione per il caching (in KB)
- directory in cui reperire i binari dei malware
- stringa di riferimento per il driver JDBC
- url del database
- query per la lettura dei dati dal database[9]

[9]Le query non potranno essere modificate liberamente, il loro inserimento all'interno di questo file serve a garantire la possibilità di adattare la sintassi al DBMS utilizzato.

Capitolo 5

Sperimentazione dell'architettura distribuita

In questo capitolo vengono presentati i risultati ottenuti dalla sperimentazione dell'architettura, divisi in due sezioni. La prima mostrerà il corretto funzionamento dei singoli componenti, mentre la seconda riporterà i risultati ottenuti mediante ripetute simulazioni.

5.1 Validazione dei singoli componenti

Come primo passaggio nella sperimentazione dell'architettura si è proceduto con la validazione dei singoli componenti, cioè dei sensori e del manager. La disposizione dei vari elementi è già stata descritta nella sezione 4.2.1 e riassumendo brevemente consiste in un agente OSSEC su sistema operativo Windows XP Professional protetto da firewall, insieme agli altri sensori installati su sistema operativo Ubuntu 8.04 non protetto da firewall, entrambi collegati ad una comune linea ADSL domestica. Per verificare il corretto funzionamento è stato sufficiente lanciare in esecuzione i programmi e attendere qualche minuto prima che cominciassero ad arrivare le prime segnalazioni. Descriviamo ora più nel dettagli i risultati di ogni singolo componente.

5.1.1 Sensori

OSSEC Questo componente era l'unico installato su entrambe le macchine e svolgeva il compito di HIDS. Considerando l'ambito di simulazione, in cui gli utenti erano fidati, e i sistemi protetti, questo sensore avrebbe dovuto generare esclusivamente alert informativi. I risultati confermavano le aspettative e le segnalazioni riguardavano normali attività di sistema (livello *info*) come l'avvio del sistema stesso e la rotazione del file di log, così si è proceduto con azioni mirate a generare notifiche più rilevanti, in particolare sono stati effettuati diversi tentativi di autenticazione come utente root sbagliando volontariamente e ripetutamente la password. OSSEC ha reagito segnalando distintamente i due casi, più precisamente con un solo tentativo fallito è stato emesso un alert di livello *low*, mentre per tre tentativi falliti consecutivamente è stato emesso un alert di livello *medium*. Oltre al livello e alla descrizione generica dell'evento (nel primo caso *User authentication failure* mentre nel secondo *Three failed attempts to run sudo*), sono stati riportati i comandi che hanno causato la segnalazione, in questo modo è stato possibile verificare che l'utente aveva cercato di lanciare una shell di root, quindi un tentativo potenzialmente molto pericoloso.

Sul sistema Windows invece è stato rilevato un malware durante il tentativo di modifica del file HOSTS. Questo ha permesso di ripristinare il file originale senza incorrere in spiacevoli inconvenienti.

Snort Snort aveva il compito di monitorare il traffico in transito all'interno della rete con la funzione di NIDS. Nel nostro caso non erano presenti servizi accessibili dall'esterno della rete (ad eccezione di quelli simulati dall'honeypot), di conseguenza il numero di segnalazioni è risultato inferiore a quello che potrebbe essere un valore medio in un ambiente reale, questo a causa di tutti quegli attacchi che richiedono un certo grado di interazione da parte del servizio. Ciò nonostante, le segnalazioni generate son state sufficienti a validare tanto il sensore quanto il manager che le ha raccolte.

Nepenthes Questo sensore riporta sotto forma di segnalazioni ogni attività che lo coinvolga. Il motivo è che, come già descritto nella sezione 2.2, ogni connessione avvenuta verso di esso può essere considerata malevola, anche se la caratteristica principale rimane quella di poter scaricare il codice binario dei malware proposti, così da poterli successivamente analizzare. Le aspettative riguardanti la sua esecuzione consistevano nell'ottenimento di qualche esemplare differente di malware e di un numero discreto di connessioni ricevute, così da poter raccogliere un numero sufficiente di dati da poter successivamente utilizzare all'interno delle simulazioni.

Una volta lanciato in esecuzione con le porte del firewall aperte, i risultati sono andati oltre le aspettative. Nepenthes ha infatti cominciato a raccogliere una mole impressionante di dati. Il flusso di segnalazioni non era costante, ma su periodi di un'ora son stati raggiunti valori medi di 2 connessioni ricevute al minuto e considerando che ogni connessione poteva generare più di un evento si comprende come il numero di segnalazioni abbia sorpreso. Per quanto riguarda il download di malware i valori erano inferiori con una proporzione rispetto alle connessioni ricevute (cioè un malware scaricato ogni x connessioni ricevute) che in media oscillava tra 1/2 e 1/9, che rappresenta comunque un valore decisamente elevato. In questa statistica non son stati differenziati i malware raccolti, di conseguenza rientrano nel conteggio anche le repliche di esemplari già raccolti, mentre con riferimento agli hash unici il numero è stato di diverse decine raccolte nell'arco di qualche giorno.

Una nota interessante riguarda la distribuzione del malware che nella gran parte dei casi avveniva mediante la porta 135, cioè quella utilizzata dal servizio Netbios di Windows, a dimostrazione di come questi software cerchino di sfruttare le vulnerabilità dei servizi più diffusi.

5.1.2 Manager

I manager utilizzati all'interno dell'architettura sono due: quello di OSSEC e il prelude-manager. Il corretto funzionamento del primo dei due è sta-

to verificato in precedenza, nella sottosezioni dei sensori. Le segnalazioni rilevate provenivano da entrambi i computer utilizzati per il test, se ne deduce che il manager ha svolto il proprio lavoro come previsto. La scelta di utilizzare questo meccanismo, invece che collegare entrambi i sensori direttamente al prelude-manager, è stata presa per inserire un ulteriore livello nell'architettura locale.

Prelude manager Il manager di prelude aveva il compito di raccogliere tutte le segnalazioni generate dai sensori e di memorizzarle all'interno del database, inoltre aveva l'incarico di verificare lo stato di attività dei sensori ad esso collegati mediante la rilevazioni degli *heartbeats*. Questo termine inglese indica le pulsazioni del cuore e viene qui utilizzato come riferimento ai segnali inviati a cadenza regolare dai sensori, in particolare nel nostro caso è stata utilizzata l'impostazione di default, ovvero una cadenza di dieci minuti. Inoltre verificando la regolarità di questi segnali risulta possibile riscontrare eventuali malfunzionamenti. L'assenza del segnale si può verificare in due casi: quando il nodo risulta sconnesso e di conseguenza non può inviare gli hearbeats, oppure quando il sensore viene fermato o spento. Per distinguere i due casi si possono controllare i segnali di eventuali altri sensori presenti sullo stesso nodo, se l'assenza di informazioni è generale si tratterà di un problema di connessione.

Per quanto riguarda la verifica del corretto funzionamento della raccolta dei dati, si è proceduto sia controllando il contenuto del database creato all'interno di MySQL, sia mediante la più comoda interfaccia grafica messa a disposizione da Prewikka. Osservando prima i dati presenti sul database è stato possibile verificare l'efficacia di Prewikka come filtro, in grado di mostrare in modo razionale le informazioni più importanti. In particolare l'utilizzo di schermate differenti che accompagnano l'utente dalla visione panoramica fino al massimo dettaglio, garantiscono una migliore efficienza, lasciando decidere all'utente quali alert sia meglio approfondire. Anche l'utilizzo dei

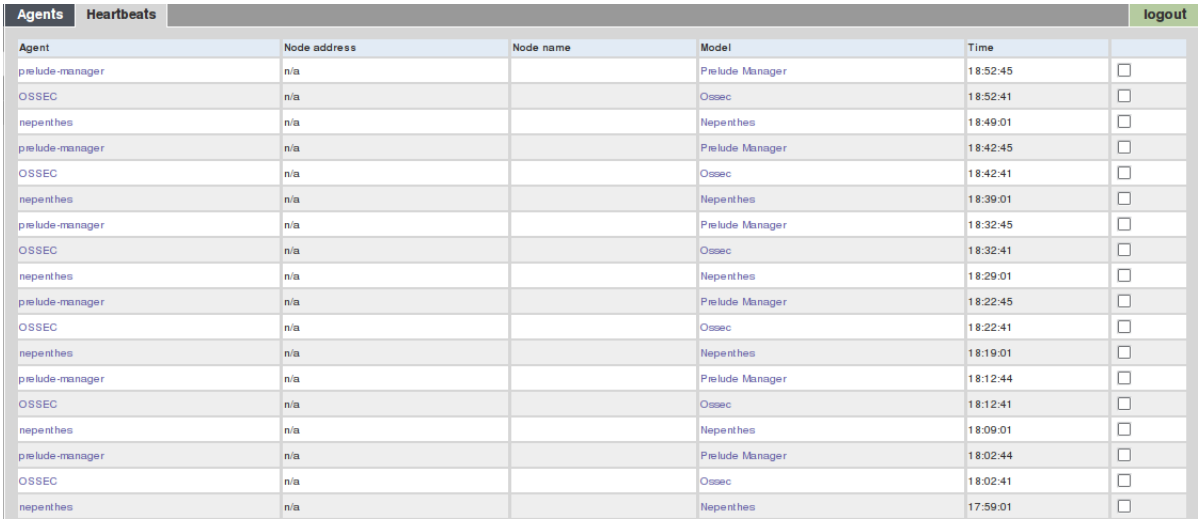

Figura 5.1: Prewikka - Schermata degli Heartbeats

colori e l'aggregazione in tempo reale rendono il monitoraggio degli eventi più semplice e veloce.

Con riferimento alla figura, si notano i raggruppamenti effettuati in base all'indirizzo sorgente, che mette in risalto eventuali host particolarmente aggressivi, allo stesso tempo la prima colonna riporta la classificazione degli eventi evidenziando con colori differenti i vari livelli di criticità.

Si prenda come esempio la quinta riga, riportante una serie di attacchi rilevati da Nepenthes e aggregati da Prewikka sulla base degli indirizzi IP di sorgente e destinazione. Una macchina collegata alla rete internet con indirizzo IP 87.18.92.135 ha effettuato 53 connessioni verso l'honeypot, riuscendo per 19 volte a offrire il proprio malware mediante due shellcode, che hanno portato al download di 18 esemplari di due tipi differenti[1]. Si notano anche i colori che all'aumentare del rischio passando dall'azzurro al rosso, passando per verde e arancione. Si sottolinea inoltre come questa serie di minacce

[1]La presenza di due nomi differenti (*OCwg+a* e *e7FX1A*) porta a distinguere i file scaricati in due esemplari, uno per tipo. Una futura analisi potrebbe mostrare che si tratti dello stesso malware che semplicemente si propaga con nomi differenti.

Figura 5.2: Prewikka - Schermata degli eventi

sia stata registrata tra le 19:17 e le 22:50, il ché significa una media di un malware scaricato ogni 12 minuti da un solo indirizzo IP, un valore molto elevato.

5.2 Validazione dell'intera architettura

Dopo aver verificato il corretto funzionamento dei singoli componenti, passiamo alla validazione dell'intera architettura mediante la simulazione di un largo numero di nodi. I dati utilizzati per le comunicazioni e le impostazioni delle singole simulazioni, variano di volta in volta in base alle caratteristiche che si vogliono osservare. In particolare varieranno il numero di nodi simulati e la quantità dei contenuti inseriti, compreso il fattore di replica, mentre rimarranno invariati i parametri utilizzati da FreePastry per la gestione del layer peer-to-peer.

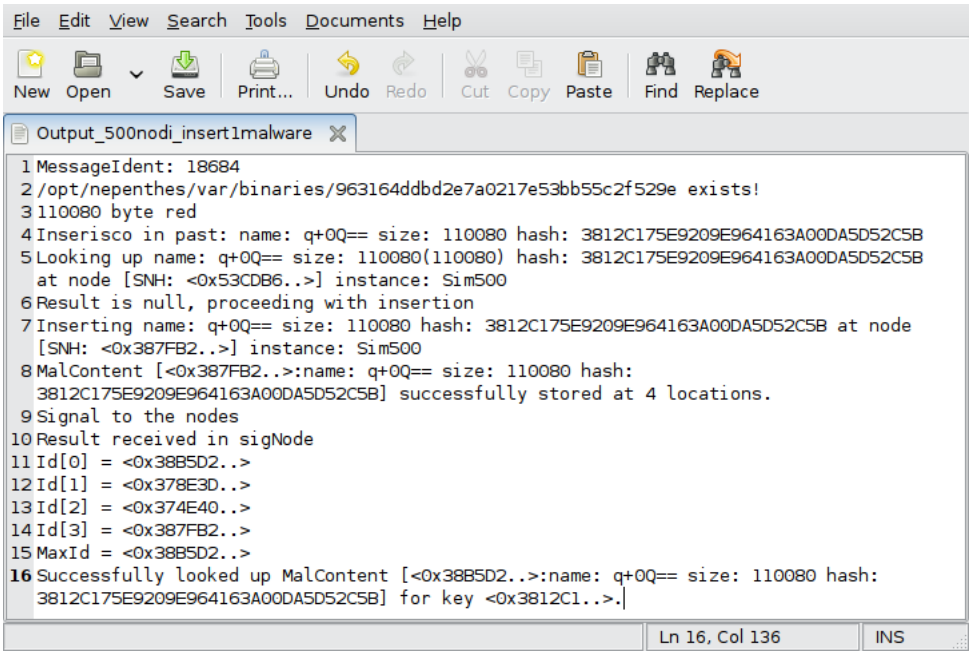

Figura 5.3: Output della lettura ed inserimento dei dati del relativi al malware

5.2.1 Analisi di un singolo attacco

Come prima simulazione è stato analizzato il comportamento dell'architettura in conseguenza ad un singolo attacco. Per rendere il trattamento più interessante si è scelto di utilizzare un caso registrato da Nepenthes dal quale risultino due segnalazioni utili per lo stesso evento: una connessione all'honeypot e il conseguente download di un malware. La conseguenza di questa scelta è la possibilità di verificare come la segnalazione degli eventi relativi ad un singolo nodo e ad un singolo attacco venga distribuita su più nodi appartenenti all'architettura.

Dopo aver lanciato la simulazione di 500 nodi è stato comandato ad uno di essi di effettuare la lettura dei dati dal database, che era stato appositamente predisposto. In figura 5.3 è stato riportato l'output relativo alla lettura e all'inserimento dei dati relativi al malware

La schermata potrebbe risultare ostica ad una prima occhiata, ma seguendo la descrizione seguente verranno chiariti i vari passaggi.

1. La prima riga riporta l'ident della tupla inserita nel database, le due seguenti indicano che il file indicato esiste e che sono stati letti circa 110KB di dati, che non sono altro che il codice binario del malware. Infine la quarta riga riporta i dati che saranno inseriti nel pacchetto, come nome, dimensione e hash, tutti letti dal database.

2. Alla riga 5 si verifica l'eventuale preesistenza di un pacchetto con la medesima chiave. Il nodeId indicato, *cioè <0x53CDB6..>*, è quello del nodo che ha effettuato la lettura è che sta per inserire i dati, si ricorda che la sua scelta è stata casuale.

3. La riga 6 riporta che il risultato della ricerca è nullo, quindi si procede con l'inserimento (riga 7). Si evidenzia come il nodo di destinazione abbia un prefisso vicino alla chiave del pacchetto da inserire.

4. La riga 8 riporta che il contenuto è stato inserito con successo 4 volte, cioè la copia primaria più le tre repliche previste dalla nostra configurazione

5. Dalla riga 9 alla 15, si verifica quali nodi abbiano ricevuto una copia del pacchetto e si procede a segnalare a quello col nodeId maggiore di effettuare l'elaborazione dei dati ricevuti.

6. La riga 16 ci conferma che il nodo responsabile, *cioè <0x38B5D2..>* ha ricevuto l'ordine di elaborazione e per dimostrarlo pubblica i dati del pacchetto

Il nodo ha effettuato correttamente la lettura di tutti i dati e ha provveduto alla creazione dei pacchetti da inserire all'interno della DHT. Per poter

verificare la loro corretta memorizzazione si è fatto ricorso all'analisi dei contenuti di tutti i nodi, tenendo presente che gli unici valori inseriti erano i due pacchetti di nostro interesse, caratterizzati dalle chiavi descritte di seguito.

L'hash del codice binario del malware era: *3812C175E9209E964163A00-DA5D52C5B*, per l'inserimento di questi dati viene utilizzata la classe malware che utilizzerà come chiave l'hash stesso del binario con un padding definito a priori. Il motivo è da ricercarsi nelle caratteristiche di PAST descritto nella sezione 4.1.6.1, la chiave da esso utilizzata è di 160 bit mentre l'hash utilizzato da Nepenthes per memorizzare i malware è di 128 bit, da qui l'esigenza di colmare quel vuoto con un suffisso prestabilito, che nel nostro caso è stato 000CAFFE. La chiave risultante è stata quindi: *3812C175E9209E964163A00DA5D52C5B000CAFFE.*

Per quanto riguarda la connessione all'honeypot, è stata utilizzata la classe SegnalazioniPrelude, che chiamando la funzione messa a disposizione da FreePastry per la generazione delle chiavi, utilizzando l'IP sorgente come base, ha generato l'ID: *920E4B38999638OC5D1754A950065A65376A8679.*

L'analisi è stato possibile perché trattandosi di una simulazione, tutti i nodi memorizzavano i dati sullo stesso disco più precisamente all'interno della cartella indicata nel file di configurazione, utilizzando ognuno la propria sottocartella nominata col prefisso del proprio nodeId. Elencando il contenuto di tutte le cartelle si è ottenuto il risultato riportato nelle figure 5.4 e 5.5.

Come si può notare, in mezzo ad un elenco di nodi vuoti, compaiono 2 gruppi di 4 elementi ciascuno, contenenti i file cercati. Il numero di copie è pari alla principale più le tre previste dal fattore di replica impostato all'interno del file di configurazione. Si può notare come i prefissi dei nodeId e delle chiavi inserite abbiano al massimo tre cifre in comune, questo a causa del vasto spazio di dominio dei nodi e del numero relativamente basso di quest'ultimi. Ciò nonostante, il risultato è sicuramente positivo, il numero delle copie risulta infatti quello previsto e anche la loro collocazione coincide col funzionamento descritto nel capitolo precedente.

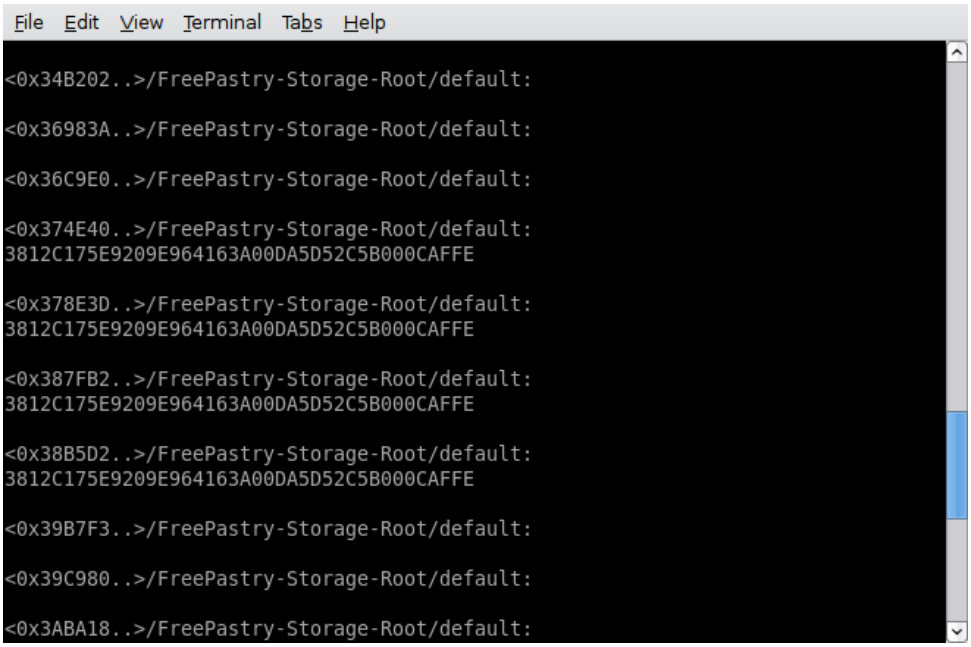

Figura 5.4: Cartelle dei nodi contenenti il file relativo al malware

5.2.2 Analisi di un attacco a più vittime dal medesimo attaccante

Dopo aver visto nel dettaglio la gestione di un singolo attacco, passiamo ora a verificare il caso un cui un unico nodo attaccante vada a colpire più reti appartenenti alla nostra architettura. La situazione potrebbe trovare riscontro in un host infetto dal quale il malware (anche polimorfico) cerca di propagarsi sul maggior numero di destinazioni possibili, incontrando più honeypot sotto il nostro controllo. Un altro caso potrebbe essere quello di un attaccante che effettua ripetute scansioni su larga scala alla ricerca di macchine vulnerabili. In entrambi i casi il risultato è lo stesso, ci si ritrova con più nodi sotto attacco che raccoglieranno le medesime informazioni relative all'attaccante e andranno ad inserirle all'interno della rete. Il comportamento atteso è quello limitare il traffico dati alla prima segnalazione, mentre gli altri nodi verificheranno che le informazioni sono già state inserite e si

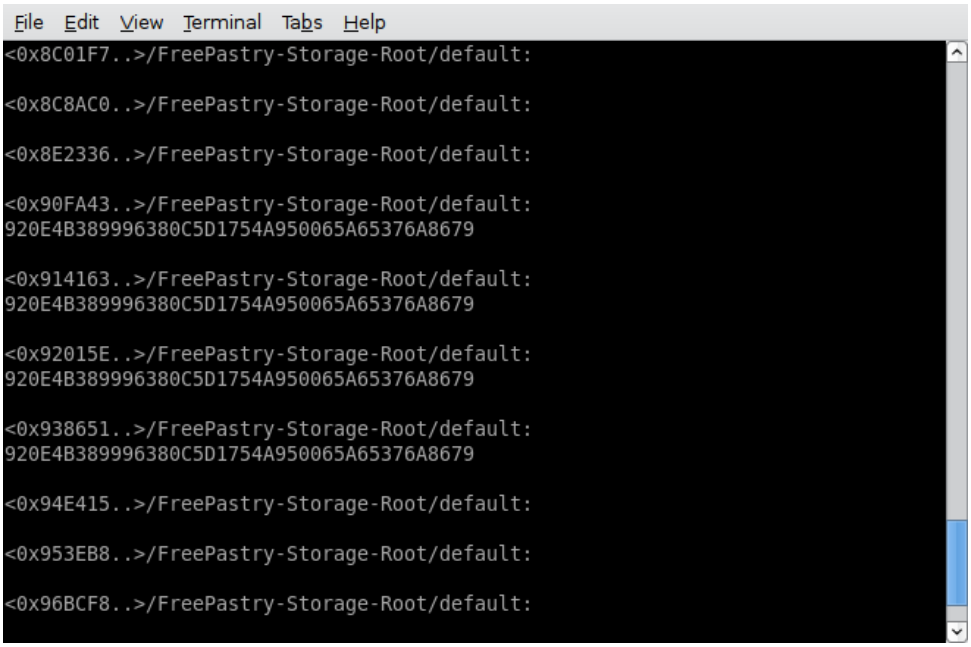

Figura 5.5: Cartelle dei nodi contenenti il file relativo alla connessione avvenuta tra l'host attaccante e l'honeypot

limiteranno a notificare che sono rimasti vittime del medesimo attaccante. Il nodo responsabile dell'elaborazione dei dati inseriti, avrà il compito di tenere traccia del ripetersi dell'evento e raggiunta una certa soglia potrà diramare un avviso per informare tutti i nodi della rete dell'elevata attività malevola proveniente da un determinato indirizzo IP che potrà essere inserito in una blacklist locale temporanea.

Per la simulazione sono stati generati 500 nodi e a 10 di essi presi casualmente è stato comandato di leggere una serie di eventi dal database caratterizzati dal medesimo indirizzo IP sorgente. Nella figura 5.6 viene riportato l'output della simulazione.

Le prime righe sono simili a quelle della simulazione precedende, quelle di maggiore interesse sono le seguenti:

- 3, 17 e 30 dalle quali si può vedere chi ha effettuato la lettura dal database, in particolare <0xCD837A..>, <0xB80E49..> e <0x1E0F55..>
- 4, 18, 31 dalle quali si vede che nel primo caso il controllo dell'eventuale preesistenza restituisce un valore nullo e si procede con l'inserimento, negli altri due invece il risultato è affermativo, di conseguenza non sarà necessario reinserire i dati, sarà sufficiente effettuare la segnalazione
- 26 e 39, dove il nodo responsabile oltre a pubblicare i dati relativi al messaggio riporta il numero di notifiche ricevute. Questo non avveniva nella simulazione precedente poiché il malware inserito era la prima segnalazione.

Infine è stato effettuato il controllo del contenuto dei diversi nodi relativo all'indirizzo IP segnalato, per verificare che il numero di dati inseriti fosse pari al solo numero delle repliche inserite dal primo nodo. Vedi figura 5.7

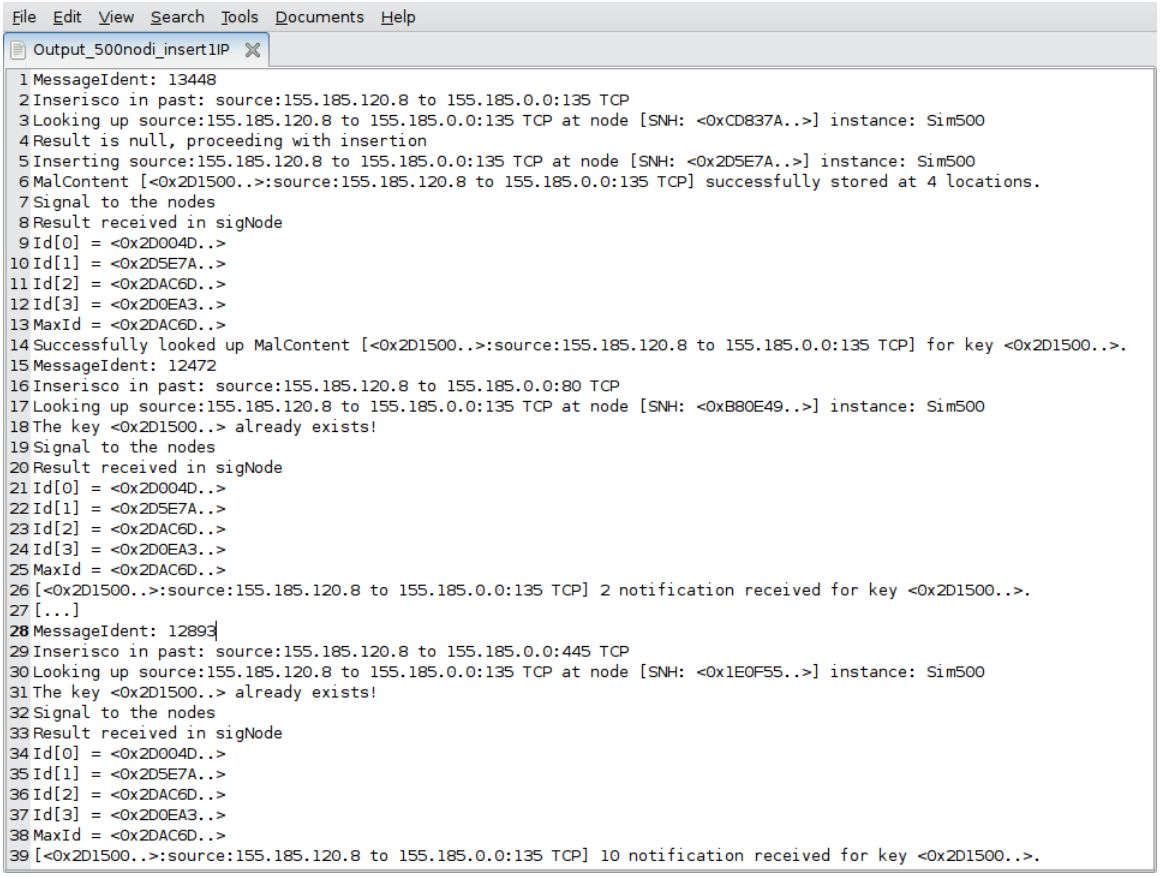

Figura 5.6: Output relativo all'inserimento di 10 eventi provocati dal medesimo IP sorgente

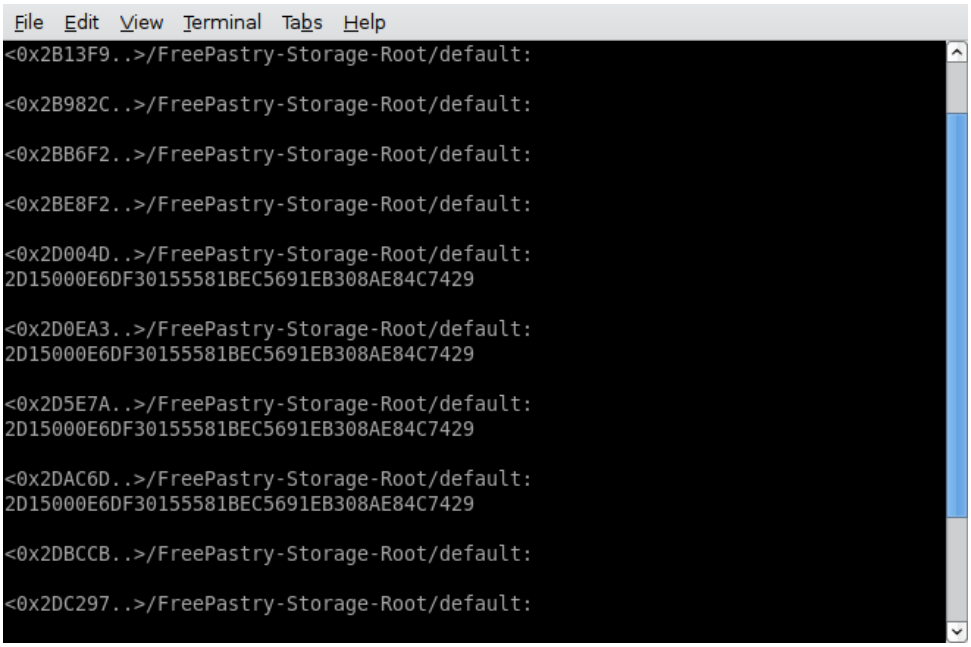

Figura 5.7: Cartelle dei nodi contenenti il file relativo all'indirizzo IP rilevato e segnalato da 10 nodi differenti

5.2.3 Analisi della diffusione di una tipologia d'attacco

Questo caso rappresenta un'evoluzione delle due simulazioni precedenti e si viene a verificare quando un numero diffuso di sottoreti segnala la rilevazione di attacchi della medesima tipologia provenienti da nodi differenti. Un riferimento ad una situazione reale di questo tipo può essere il diffondersi del malware su larga scala, oppure il diffondersi di una tecnica di attacco basata su una determinata vulnerabilità. Questo secondo caso può verificarsi per esempio quando viene annunciata una nuova vulnerabilità per la quale non sono ancora disponibili patch. Il nutrito gruppo di cracker e blackhat in generale troverà maggiore spinta nel cercare di sfruttare tale problema, consapevole che l'efficacia su larga scala sarà maggiore, inoltre questa maggiore spinta porterà ad una maggiore concentrazione degli attacchi in questa direzione. Fortunatamente l'aggiornamento delle signature degli IDS potrà avvenire senza particolari ritardi e proprio in questi casi mostra la sua importanza: poter avvertire l'amministratore di attacchi potenzialmente molto dannosi.

La simulazione di questa tipologia d'evento è stato organizzata come segue. Dopo aver lanciato 500 nodi è stato comandato a 10 di essi di accedere al database per leggere determinati eventi. Quest'ultimi sono stati appositamente selezionati per le loro caratteristiche, più precisamente riportavano la stessa tipologia d'attacco, sferrata da indirizzi IP differenti, dove per tipologia si è fatto riferimento alla signature di snort generatrice dell'evento.

Osservando la figura 5.8, le righe di maggior interesse sono le seguenti:

- 1-3, 15-17 e 28-30, dalle quali è possibile osservare che i dati letti differiscono per ip sorgente, ma si riferiscono alla stessa signature. Si può osservare inoltre il nodeId dei nodi che leggono i dati dal database e che controllano l'eventuale preesistenza

- 4, 18 e 31, sono i risultati del controllo di preesistenza. Nel primo caso il risultato è null, quindi viene effettuato l'inserimento, mentre negli

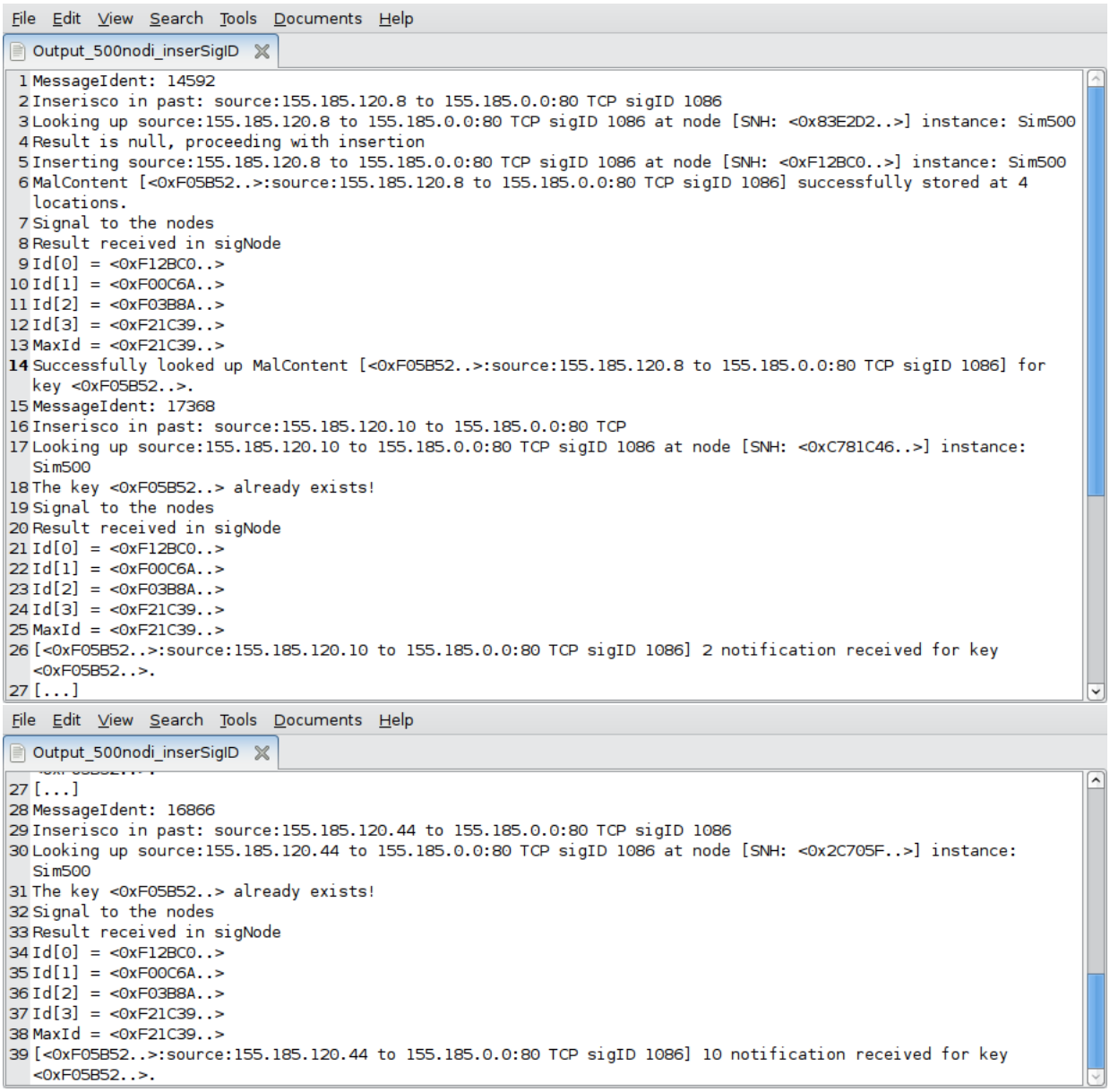

Figura 5.8: Output relativo all'inserimento di 10 eventi generati dalla medesima signature

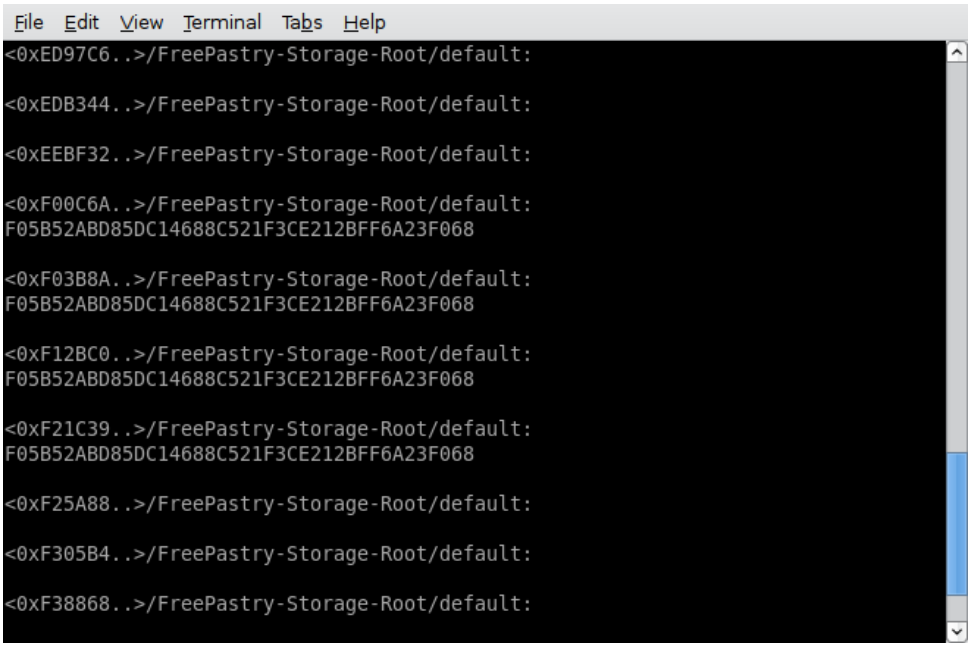

Figura 5.9: Cartelle dei nodi contenenti il file relativo alla signature segnalata da 10 nodi differenti

agli due il risultato è positivo, quindi eseguono solo la segnalazione

- 14, 25 e 39, da quali si vedono i risultati dell'inserimento prima e delle notifiche poi

Anche in questo caso è stato effettuato il controllo del contenuto dei diversi nodi relativo alla signature segnalata, per verificare che il numero di dati inseriti fosse pari al solo numero delle repliche inserite dal primo nodo. Vedi figura 5.9

5.2.4 Analisi bilanciamento del carico

Uno dei vantaggi principali di quest'architettura, discusso nelle sezione 3.3.2, è quello del bilanciamento del carico. L'utilizzo di funzioni hash per la generazione delle chiavi assicura infatti una distribuzione uniforme dei vari mes-

saggi sullo spazio dei nodi. Nelle pagine precedenti è stato mostrato l'inserimento di un numero limitato di contenuti ed è stato possibile osservare nel dettaglio i meccanismi di distribuzione (vicinanza dei prefissi nodo-chiave). Per quanto riguarda invece, test volti a verificare il bilanciamento del carico, risulta fondamentale simulare un alto numero di nodi su cui inserire un ancor più elevato numero di messaggi. Per avere un'idea dei valori cui ci si sta riferendo si parla dell'ordine di grandezza delle centinaia, o meglio ancora delle migliaia, per i nodi e delle decine di migliaia per i messagi.

Vengono riportate di seguito alcune immagini raffiguranti i diagrammi CDF (Cumulative Distribution Function - Funzioni Cumulative Distribuite) conosciute in italiano come funzioni di ripartizione. Tale funzione rappresenta la sommatoria delle probabilità di ogni singole valore: $F(x) = P(X \leq x) = \sum_{x_i<x} P(X = x_i) = \sum_{x_i<x} p(x_i)$ Il suo utilizzo permette di rendersi conto di come siano distribuite le varie probabilità. Tutte le immagini ad eccezione dell'ultima, hanno come ordinate la sommatoria delle probabilità espressa in percentuale e sulle ascisse il rapporto tra il numero dei messaggi memorizzati e la media, si ricorda inoltre che il fattore di replica indica il numero di repliche presenti oltre la copia primaria. L'ultima riporta sulle ascisse il valore assoluto del numero di messaggi memorizzati da ogni nodo.

Si evidenzia in particolare l'ultima immagine, dove vengono riportate sullo stesso diagramma due simulazioni con identiche impostazioni ad eccezzione del fattore di replica. Con k=3, la media è 40 e la deviazione standard 15.25, mentre con k=4 la media è 50 e la deviazione standard 17,48.

Nonostante il numero di simulazioni efettutate non sia stato sufficiente a permettere un analisi statistica su tale fattore, è comunque possibile affermare che l'aumentare del numero di nodi e del numero dei messaggi scambiati migliora l'uniformità della ditribuzione del carico, ovvero del suo bilanciamento.

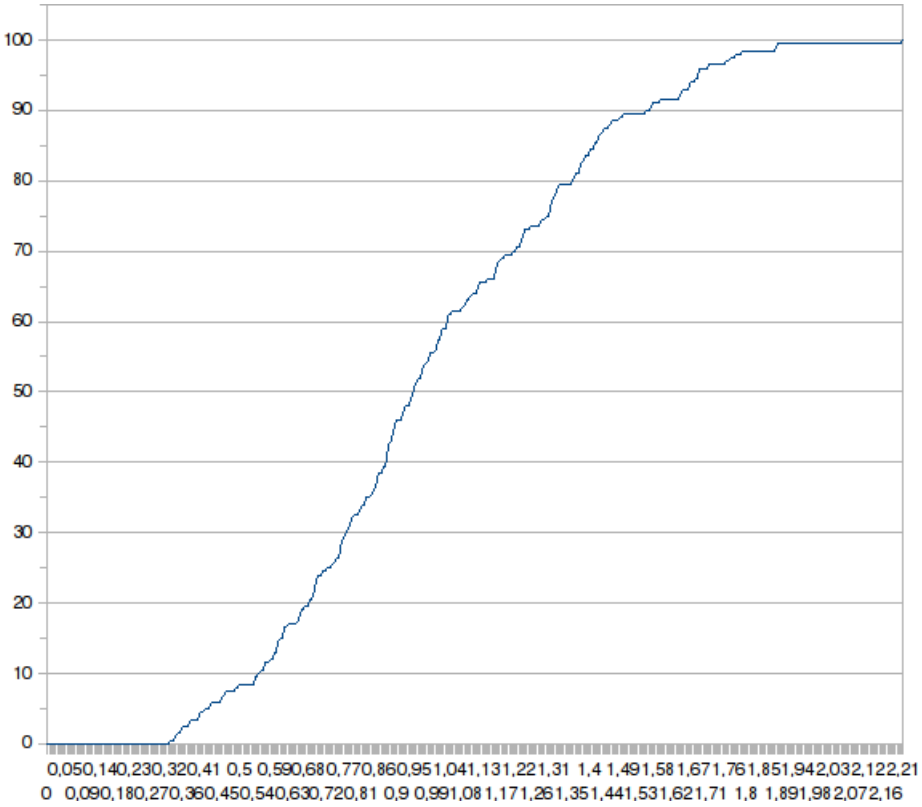

Figura 5.10: CDF di 200 nodi con 40000 messaggi memorizzati e fattore di replica = 3. Media = 200, deviazione standard = 75,56

5.2.5 Analisi tolleranza ai guasti

Per l'analisi della tolleranza ai guasti son state effettuate alcune simulazioni, con differrente numero dei nodi e differente numero di guasti. L'esecuzione consisteva nel lancio di un numero di nodi prestabilito (compreso tra 100 e 1000) e nell'inserimento di un numero di messaggi pari a 10000 con fattore di replica k=4. Completato l'inserimento dei messaggi, si effettuava la scansione dei singoli nodi per ottenere il numero di messaggi contenuti, dopo di chè venivano uccisi casualmente un numero variabile di nodi inferiore o uguale

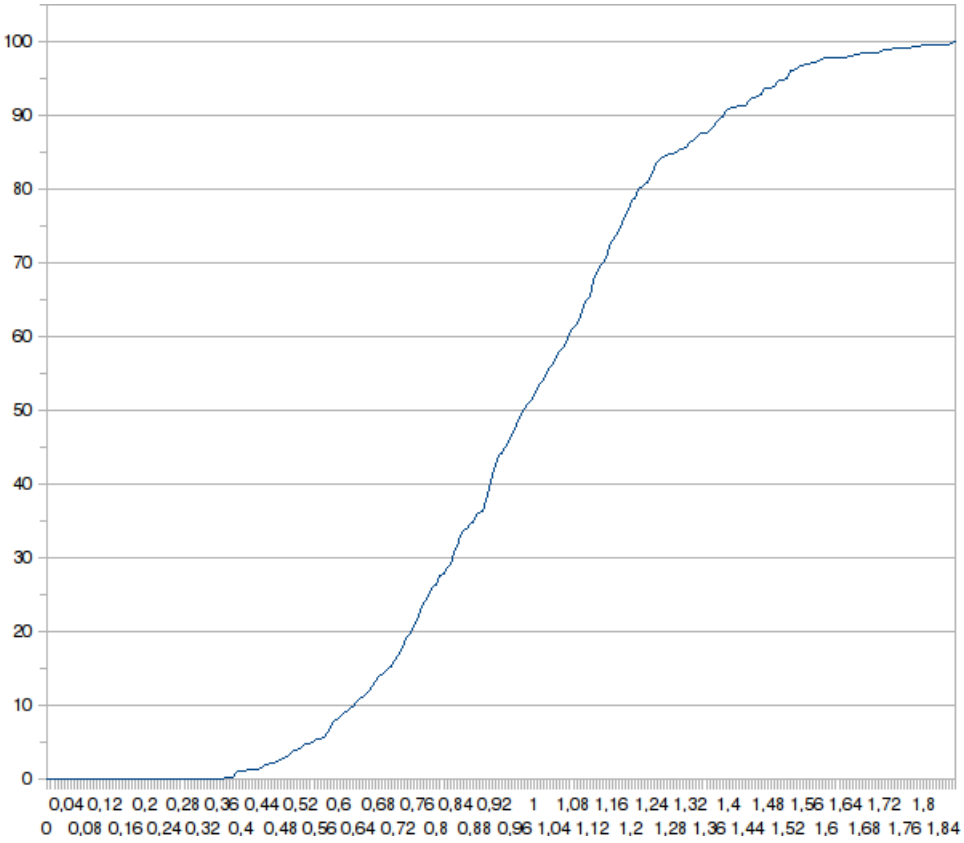

Figura 5.11: CDF di 500 nodi con 75000 messaggi memorizzati con fattore di replica = 4. Media 150, deviazioni standard 43,4

al 5% del totale[2]. Terminata l'operazione, veniva lasciato il tempo ai nodi sopravissuti di aggiornare le proprie routing table e di ristabilire il numero di repliche previsto dalla configurazione.

In tutti i casi è stato possibile ripristinare correttamente il numero di roliche previsto dalla configurazione, a dimostrazione della solidità dell'architettura.

[2]Anche se il numero di failure comandate potrebbe sembrare limitato, si sottolinea come nella realtà risulti assai difficile perdere il 5% dei nodi partecipanti nello stesso momento, o comunque in un lasso di tempo inferiore alla periodicità con cui vengono ristabilite le repliche.

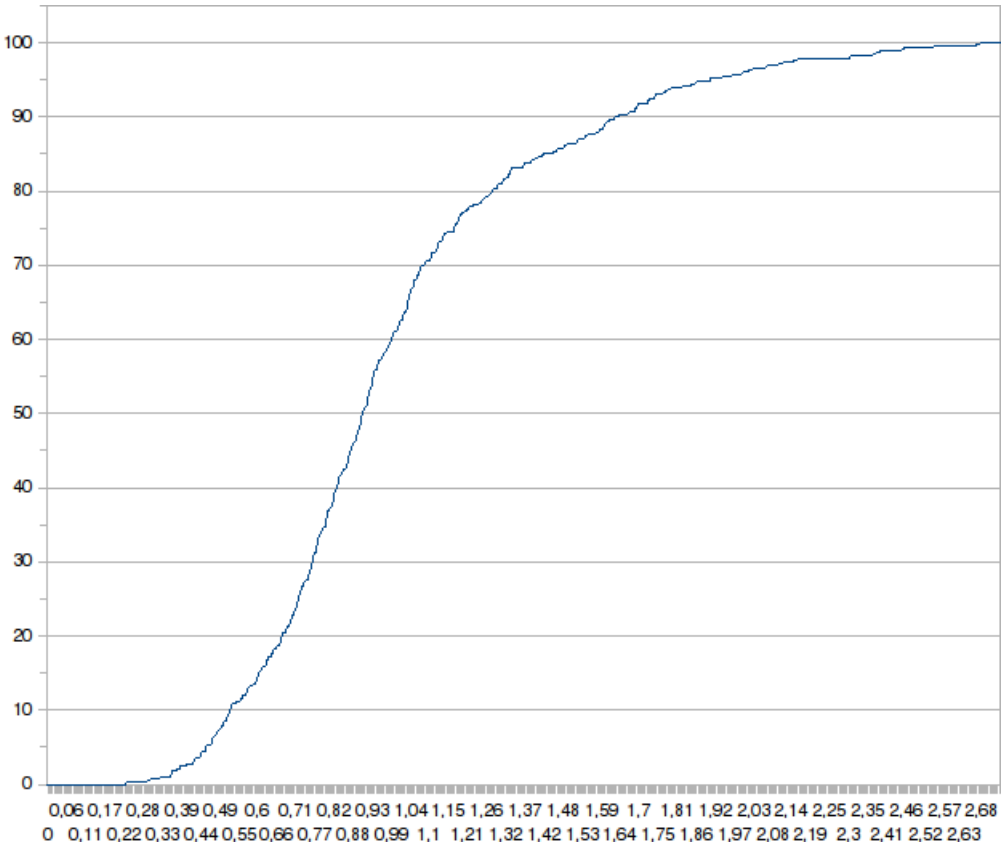

Figura 5.12: CDF di 500 nodi con 100000 messaggi memorizzati e fattore di replica = 4. Media 200, deviazione standard = 71,61

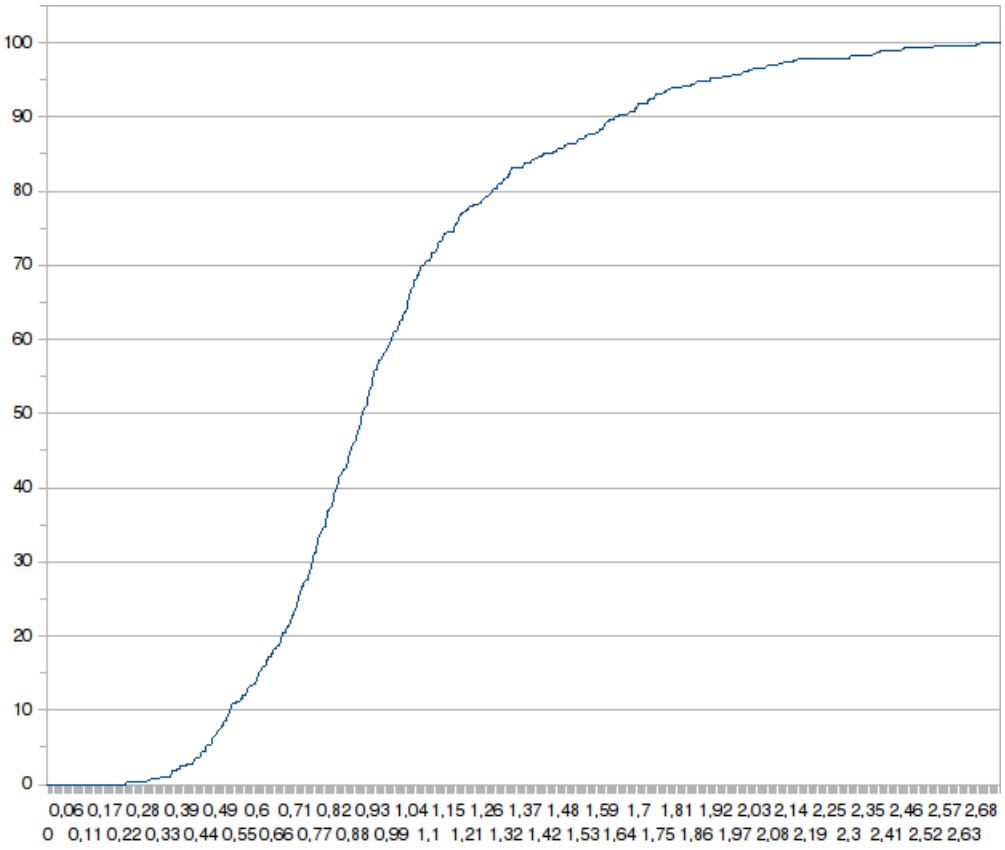

Figura 5.13: CDF di 1000 nodi con 160000 messaggi memorizzati e fattore di replica = 3. Media 160, deviazione standard 76,38

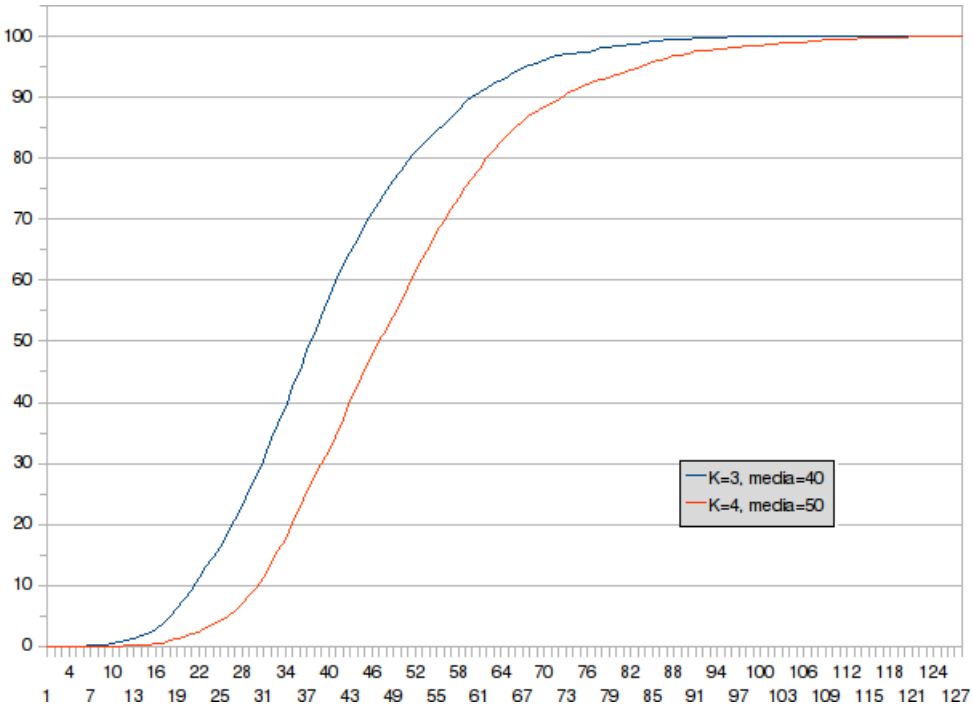

Figura 5.14: CDF di 3000 nodi con numero messaggi = 30000 e due differenti fattori di replica

Capitolo 6

Conclusioni

In questa tesi è stata presentata un'architettura collaborativa peer-to-peer progettata per fornire un sistema di difesa dalle minacce provenienti dalla rete, più efficace di quelli attualmente disponibili. La sua realizzazione ha richiesto mesi di tempo, durante i quali sono stati inizialmente analizzati i punti deboli sia dei prodotti già presenti sul mercato sia dei prototipi e dei diversi approcci proposti nella recente letteratura. Dopo di ché si è cercato di capire come risolverli, quindi si è passati allo studio di come far interagire i nodi e di quali dati considerare rilevanti per gli scopi prefissati. Successivamente è stato realizzato un prototipo minimale, ma perfettamente funzionante, utilizzato per dimostrare la fattibilità dell'approccio proposto mediante validazioni sperimentali. Tali esperimenti son stati effettuati utilizzando segnalazioni di attività intrusive ed esemplari di malware attualmente circolanti in rete. Infine, dopo lo studio dell'ambiente di simulazione messo a disposizione dalle librerie utilizzate, il programma realizzato è stato modificato in modo da poter implementare le relative funzioni. Questo ha consentito lo svolgimento prove con migliaia di nodi simulati, che altrimenti non sarebbe stato possibile effettuare a causa delle ingenti risorse richieste. Tutto questo ha permesso di gettare le basi per quella che si crede la migliore soluzione possibile.

I vantaggi portati da questa nuova architettura sono notevoli. Innanzitutto vengono risolti i principali problemi delle architetture gerarchiche, con l'introduzione di un'elevata tolleranza ai guasti e del bilanciamento del carico. I confronti effettuati hanno infatti mostrato come questo nuovo approccio fornisse una valida soluzione su entrambi gli argomenti.

Per quanto riguarda la tolleranza ai guasti, la mancanza di un nodo centrale e l'equivalenza di tutti i nodi in termini di funzioni e ruolo, garantisce già una buona base di partenza, nonché una prima soluzione al problema. L'utilizzo inoltre di tale sistema con una gestione delle repliche di tutti i contenuti inseriti, aumenta notevolmente la tolleranza ai guasti, così come hanno dimostrato l'analisi teorica prima e le simulazioni poi.

Discorso simile per il bilanciamento del carico, che trova nella struttura adottata un ottimo punto di partenza per la distribuzione del lavoro su più nodi, che risultata ulteriormente migliorato dall'uniformità raggiunta mediante appositi meccanismi, quali l'utilizzo di funzioni hash. Anche in questo caso i risultati ottenuti dalle simulazioni svolte hanno confermato quanto previsto a livello teorico, in particolare si sottolinea il trend positivo che si ottiene con l'aumentare del numero dei nodi e del numero di contenuti inseriti all'interno della DHT. Ciò significa che una maggiore partecipazione renderebbe più uniforme il bilanciamento del carico, il ché rappresenta un'ottima base per la nascita di un circolo virtuoso.

In secondo piano rispetto ai primi due, ma sempre degno di nota, è l'ottimizzazione della gestione dello spazio di storage, legata al quantitativo fisso del numero di repliche dei contenuti inseriti e di particolare interesse per quanto riguarda la raccolta di malware, viste le dimensioni maggiori dei relativi file rispetto a quelli riguardanti semplici segnalazioni.

Nella realizzazione di quest'architettura si è inoltre cercato di venire incontro alle esigenze dei potenziali partecipanti, adottando un approccio che, oltre essere orientato all'utilizzo di software libero, garantisse un certo margine di flessibilità, partendo proprio dagli elementi che si trovano alla

base e che è compito dei partecipanti gestire, ovvero i sensori e i manager. I primi rappresentano l'elemento fondamentale per la raccolta dei dati e il loro deployment rappresenta un elemento fondamentale per una buona collaborazione. Nonostante quelli presi in considerazione all'interno del progetto siano molto diffusi, rappresentano solo una parte di quelli attualmente disponibili. Per questo motivo risulta molto importante la scelta del manager a cui affidare il compito di fornire un punto di aggregazione, che nel nostro caso ha visto in Prelude un'ottima soluzione, grazie alla sua elevata interoperabilità con diversi sensori, legata all'utilizzo di un sistema di comunicazione basato sullo standard IDMEF.

Grazie a queste considerazioni è stato possibile realizzare un componente software compatibile con un vasto numero di reti già esistenti, senza richiedere particolari risorse, tanto in termini hardware quanto in termini di tempo da investire.

Le principali difficoltà riscontrare durante la realizzazione di questo progetto, hanno avuto a che fare con le simulazioni. Questo progetto infatti, oltre a richiedere uno studio e una progettazione finalizzati alla simulazione, quindi con un lavoro aggiuntivo rispetto all'applicazione base, ha richiesto un elevato numero di risorse per tutta la parte riguardante l'analisi del bilanciamento del carico. Infatti, le numerose simulazioni hanno assorbito notevoli risorse in termini di tempo, ma anche di capacità di storage, il ché ha richiesto uno studio preliminare per ogni simulazione, per ottenere il giusto compromesso tra raggiungimento dei limiti e confrontabilità dei risultati.

La flessibilità dell'architettura presentata, insieme all'estensibilità delle funzioni offerte aprono diverse strade ad eventuali sviluppi futuri. Infatti, la struttura del software realizzato è stata pensata appositamente per garantire ampie possibilità di espansione in termini di funzionalità offerte. Vengono inoltre sottolineati quelli che potrebbero essere i due principali punti su cui lavorare:

- integrazione delle funzioni di analisi dei dati raccolti, visto che attualmente vengono solo simulate
- implementazione di maggiori funzioni volte alla gestione della sicurezza delle comunicazioni, anche se per questo punto si rende necessario uno studio per stabilire se intervenire al livello applicativo più alto o attendere l'uscita di nuove versioni delle librerie utilizzate

Bibliografia

[1] http://www.steampowered.com/status/survey.html

[2] http://seclists.org/fulldisclosure/2007/Aug/0520.html

[3] http://www.secureworks.com/media/press_releases/20080922-attacks/

[4] http://money.cnn.com/2000/02/08/technology/yahoo/

[5] http://asert.arbornetworks.com/2008/01/church-of-scientology-ddos-statistics/

[6] http://www.computerworld.com/securitytopics/security/holes/-story/0,10801,105484,00.html

[7] Elias Athanasopoulos, Andreas Makridakis, Spyros Antonatos, Demetres Antoniades, Sotiris Ioannidis, Kostas G. Anagnostakis, and Evangelos P. Markatos. Antisocial Networks: Turning a Social Network into a Botnet. In Proceedings of the 11th Information Security Conference (ISC 2008). September 2008, Taipei, Taiwan.

[8] http://photography.nationalgeographic.com/photography/photo-of-the-day

[9] http://nepenthes.mwcollect.org/

[10] http://www.mwcollect.org/

Colajanni and Francesca Mazzoni. HoneySpam: Honeypots fighting spam at the source. Proceedings of the Steps to Reducing Unwanted Traffic on the Internet on Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI'05). July 2005, Cambridge, MA

[11] http://www.ossec.net/

[12] Ptacek, Thomas H. and Newsham, Timothy N.. Insertion, Evasion and Denial of Service: Eluding Network Intrusion Detection

[13] https://trac.prelude-ids.org/

[14] http://www.prelude-ids.com

[15] http://www.ietf.org

[16] http://www.ietf.org/rfc/rfc4765.txt

[17] http://www.snort.org/

[18] http://www.mysql.com/

[19] http://www.sun.com/aboutsun/pr/2008-01/sunflash.20080116.1.xml

[20] http://www.freepastry.org/FreePastry/

[21] http://research.microsoft.com/~antr/Pastry/download.htm

[22] Ion Stoica and Robert Morris and David Karger and M. Frans Kaashoek and Hari Balakrishnan Ý, Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications, 2001

and Oskar Sandberg and Brandon Wiley, Protecting Free Expression Online with Freenet, IEEE Internet Computing, January-February 2002, pages 40–49

[23] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. 2001. A scalable content-addressable network. SIGCOMM Comput. Commun. Rev. 31, 4 (August 2001), 161-172.

[24] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. 2001. Tapestry: an Infrastructure for Fault-Tolerant Wide-Area Location and. Technical Report. University of California at Berkeley, Berkeley, CA, USA.

[25] Antony Rowstron and Peter Druschel, Pastry: {Scalable,} distributed object location and routing for large-scale peer-to-peer systems, IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Nov 2001, pages "329-350", Heidelberg, Germany

Kermarrec and Antony Rowstron, One ring to rule them all: {Service} discover and binding in structured peer-to-peer overlay networks, SIGOPS European Workshop, Sep 2002, France

[26] Miguel Castro and Peter Druschel and Y. Charlie Hu and Antony Rowstron, Exploiting Network Proximity in Distributed Hash Tables, International Workshop on Future Directions in Distributed Computing (FuDiCo), Editors: Ozalp Babaoglu and Ken Birman and Keith Marzullo, pages "52-55", Jun 2002, Bertinoro, Italy

[27] Miguel Castro and Peter Druschel and Ayalvadi Ganesh and Antony Rowstron and Dan S. Wallach, Security for structured peer-to-peer overlay networks, 5th Symposium on Operating Systems Design and Implementaion (OSDI'02), Dec 2002, Boston, MA, USA

[28] Ratul Mahajan and Miguel Castro and Antony Rowstron, Controlling the Cost of Reliability in Peer-to-peer Overlays, IPTPS'03, Berkeley, CA, USA, Feb 2003

[29] Peter Druschel and Antony Rowstron, {PAST: A} Persistent and Anonymous Store, HotOS VIII, May 2001, Schoss Elmau, Germany

[30] Antony Rowstron and Peter Druschel, Storage management and caching in {PAST}, a large-scale, persistent peer-to-peer storage utility, Oct 2001, 18th ACM Symposium on Operating Systems Principles (SOSP'01), Chateau Lake Louise, Banff, Canada, pages 188-201

[31] Antony Rowstron and Anne-Marie Kermarrec and Miguel Castro and Peter Druschel, SCRIBE: The design of a large-scale event notification infrastructure, 3rd International Workshop on Networked Group Communication (NGC2001), UCL, London, UK, November 2001

[32] Miguel Castro and Michael B. Jones and Anne-Marie Kermarrec and Antony Rowstron and Marvin Theimer and Helen Wang and Alec Wolman, An Evaluation of Scalable Application-level Multicast Built Using Peer-to-peer overlays, Infocom'03, San Francisco, CA, USA, Apr 2003

[33] V. N. Padmanabhan and H. J. Wang and P. A. Chou and K. Sripanidkulchai. Distributing Streaming Media Content Using Cooperative Networking. ACM NOSSDAV, Miami Beach, FL, USA May 2002

[34] http://www.reuters.com/article/technologyNews/idUSL3026621820080131

[35] http://java.sun.com/javase/6/

[36] http://www.w3.org/XML/